



PurpleGenesis

JOLlyTr0LLz tutorial

AP Security.

У большинства исследователей программного обеспечения первые шаги одинаковые. Речь идет о превичном анализе бинарного файла. Каждый раз повторится один и тот же алгоритм:

1. Узнать что за файл
2. Узнать какой у него хеш
3. Какие защиты стоят (Canary, PIE, NX, RELRO)

В данной статье будет рассмотрена утилита для реверс-инжиниринга и бинарной эксплуатации бинарного файла под названием **JOLlyTroLLz**.

Дисклеймер: Все данные, предоставленные в данной статье, взяты из открытых источников, не призывают к действию и являются только лишь данными для ознакомления, и изучения механизмов используемых технологий.

Запуск

После скачивания программы необходимо запкстить файл `setuptools.sh`:

```
./setuptools.sh
```

После этого начнется утсановка всех необходимых библиотек и пакетов.

Когда успешная установка пакетов пройдена, в директории `font` нужно установить шрифт `MODES____.TTF` и запускать

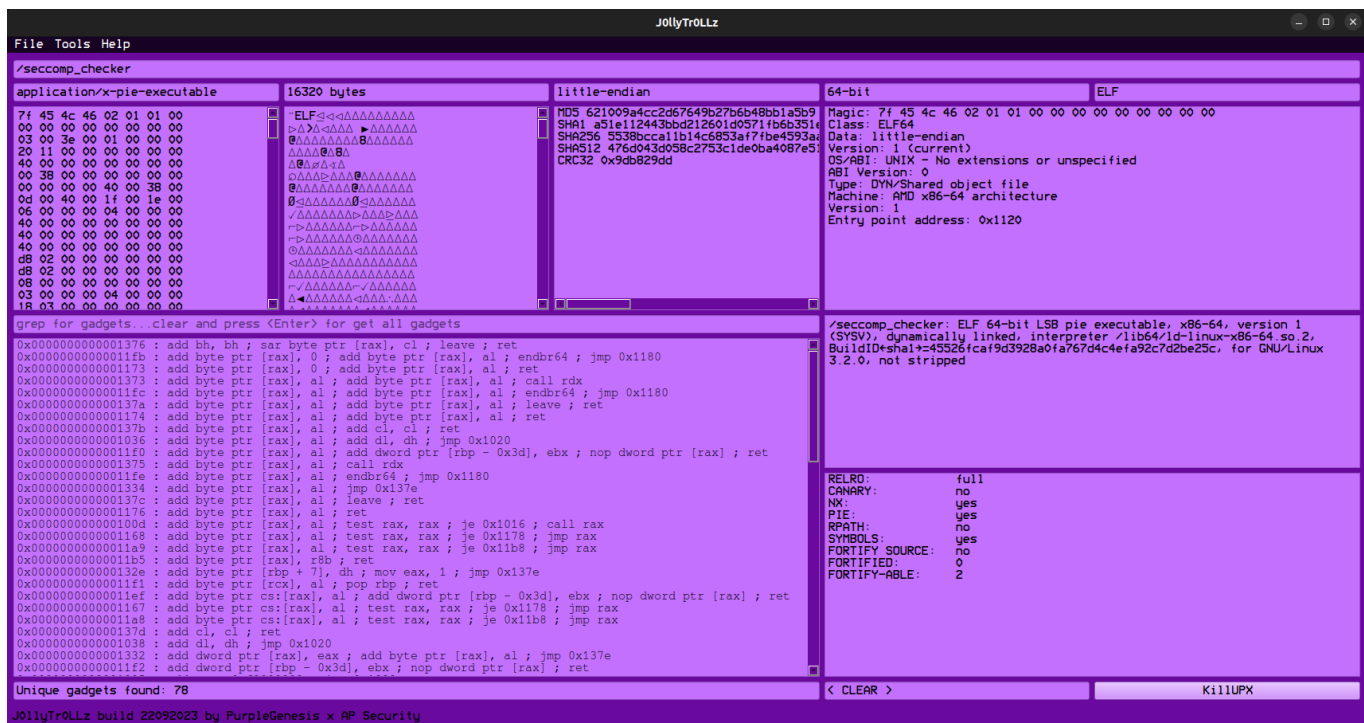
```
python3 main.py
```

Обзор

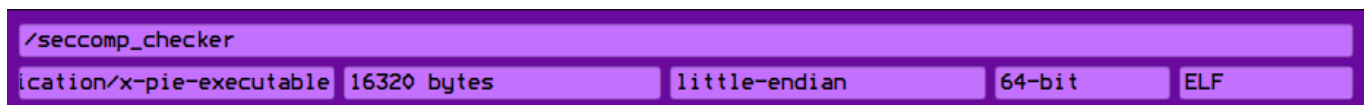
Называется так программа не случайно. Она включает в себя целый комплекс утилит:

1. Информация о бинарном файле (`readelf`)
2. Общая информация (`file`)
3. Представление файла в шестнадцатичном представлении и ASCII
4. Информация о разрядности и типе файла
5. Защиты программы (`checksec`)
6. Детектор UPX и избавление от него
7. Поиск ROP-гаджетов
8. Строки программы (`strings`)
9. Справочная таблица о системных вызовах (x86, x86_64, arm32, arm64)
10. Определение защит для шеллкодов (seccomp-tools)
11. Дизассемблер шеллкодов для различных архитектур и разрядности
12. Простой сканнер потенциально уязвимых функций и подходящих гаджетов, например, для попадания в стек
13. Дизассемблер программы и представление его в виде графа

Сначала главные экран и откроем программу для анализа нажав комбинацию `Ctrl+O`



Здесь можно увидеть поле, в котром будет отображен полный путь до программы, тип файла(исполняемый, скрипт итд), его размер, порядок следования байт, архитектура, разрядность и тип(ELF, PE, Mach-O).



Ниже два окна с представлением в шестнадцатеричном виде и его переводе в ASCII.

После этого набор хешей программы:

1. MD5
2. SHA1
3. SHA256
4. SHA512
5. CRC32



После поле, в котором информация аналогичная `readelf`, ниже поле с функционалом программы `file`. В окне, находящемся в правом нижем углу, можно будет увидеть защиты программы, аналогично `checksec`, детектор и уничтожитель упаковщика `UPX`.

```
Magic: 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
Class: ELF64
Data: little-endian
Version: 1 (current)
OS/ABI: UNIX - No extensions or unspecified
ABI Version: 0
Type: DYN/Shared object file
Machine: AMD x86-64 architecture
Version: 1
Entry point address: 0x1120

|/seccomp_checker: ELF 64-bit LSB pie executable, x86-64, version 1
(SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.
2, BuildID(sha1)=45526fcaf9d3928a0fa767d4c4efa92c7d2be25c, for
GNU/Linux 3.2.0, not stripped

RELRO:                full
CANARY:                no
NX:                    yes
PIE:                   yes
RPATH:                 no
SYMBOLS:               yes
FORTIFY_SOURCE:        no
FORTIFIED:              0
FORTIFY-ABLE:          2

< CLEAR >                KillUPX
```

Самое большое поле - это отображение ROP-гаджетов с выводом количества гаджетов и поиском.

```
grep for gadgets...clear and press <Enter> for get all gadgets
0x0000000000001376 : add bh, bh ; sar byte ptr [rax], cl ; leave ; ret
0x00000000000011fb : add byte ptr [rax], 0 ; add byte ptr [rax], al ; endbr64 ; jmp 0x1180
0x0000000000001173 : add byte ptr [rax], 0 ; add byte ptr [rax], al ; ret
0x0000000000001373 : add byte ptr [rax], al ; add byte ptr [rax], al ; call rdx
0x00000000000011fc : add byte ptr [rax], al ; add byte ptr [rax], al ; endbr64 ; jmp 0x1180
0x000000000000137a : add byte ptr [rax], al ; add byte ptr [rax], al ; leave ; ret
0x0000000000001174 : add byte ptr [rax], al ; add byte ptr [rax], al ; ret
0x000000000000137b : add byte ptr [rax], al ; add cl, cl ; ret
0x0000000000001036 : add byte ptr [rax], al ; add dl, dh ; jmp 0x1020
0x00000000000011f0 : add byte ptr [rax], al ; add dword ptr [rbp - 0x3d], ebx ; nop dword ptr [rax]
0x0000000000001375 : add byte ptr [rax], al ; call rdx
0x00000000000011fe : add byte ptr [rax], al ; endbr64 ; jmp 0x1180
0x0000000000001334 : add byte ptr [rax], al ; jmp 0x137e
0x000000000000137c : add byte ptr [rax], al ; leave ; ret
0x0000000000001176 : add byte ptr [rax], al ; ret
0x000000000000100d : add byte ptr [rax], al ; test rax, rax ; je 0x1016 ; call rax
0x0000000000001168 : add byte ptr [rax], al ; test rax, rax ; je 0x1178 ; jmp rax
0x00000000000011a9 : add byte ptr [rax], al ; test rax, rax ; je 0x11b8 ; jmp rax
0x00000000000011b5 : add byte ptr [rax], r8b ; ret
0x000000000000132e : add byte ptr [rbp + 7], dh ; mov eax, 1 ; jmp 0x137e
0x00000000000011f1 : add byte ptr [rcx], al ; pop rbp ; ret
0x00000000000011ef : add byte ptr cs:[rax], al ; add dword ptr [rbp - 0x3d], ebx ; nop dword ptr [rax]
0x0000000000001167 : add byte ptr cs:[rax], al ; test rax, rax ; je 0x1178 ; jmp rax
0x00000000000011a8 : add byte ptr cs:[rax], al ; test rax, rax ; je 0x11b8 ; jmp rax
0x000000000000137d : add cl, cl ; ret
0x0000000000001038 : add dl, dh ; jmp 0x1020
0x0000000000001332 : add dword ptr [rax], eax ; add byte ptr [rax], al ; jmp 0x137e
0x00000000000011f2 : add dword ptr [rbp - 0x3d], ebx ; nop dword ptr [rax] ; ret

Unique gadgets found: 78
J0llyTr0LLz build 22092023 by PurpleGenesis x AP Security
```

Поиск осуществляется очень просто:

1. Попробуем найти гаджеты `jmp rax`. Набираем и нажимаем `Enter`

```
jmp rax
0x0000000000001168 : add byte ptr [rax], al ; test rax, rax ; je 0x1178 ; jmp rax
0x00000000000011a9 : add byte ptr [rax], al ; test rax, rax ; je 0x11b8 ; jmp rax
0x0000000000001167 : add byte ptr cs:[rax], al ; test rax, rax ; je 0x1178 ; jmp rax
0x00000000000011a8 : add byte ptr cs:[rax], al ; test rax, rax ; je 0x11b8 ; jmp rax
0x0000000000001166 : jbe 0x1196 ; add byte ptr [rax], al ; test rax, rax ; je 0x1178 ; jmp rax
0x000000000000116d : je 0x1178 ; jmp rax
0x00000000000011ae : je 0x11b8 ; jmp rax
0x000000000000116f : jmp rax
0x000000000000116b : test eax, eax ; je 0x1178 ; jmp rax
0x00000000000011ac : test eax, eax ; je 0x11b8 ; jmp rax
0x000000000000116a : test rax, rax ; je 0x1178 ; jmp rax
0x00000000000011ab : test rax, rax ; je 0x11b8 ; jmp rax

Unique gadgets found: 12
J0llyTr0LLz build 22092023 by PurpleGenesis x AP Security
```

2. Для возврата всех гаджетов удаляем строку и жмем Enter

```
grep for gadgets...clear and press <Enter> for get all gadgets

0x0000000000001376 : add bh, bh ; sar byte ptr [rax], cl ; leave ; ret
0x00000000000011fb : add byte ptr [rax], 0 ; add byte ptr [rax], al ; endbr64 ; jmp 0x1180
0x0000000000001173 : add byte ptr [rax], 0 ; add byte ptr [rax], al ; ret
0x0000000000001373 : add byte ptr [rax], al ; add byte ptr [rax], al ; call rdx
0x00000000000011fc : add byte ptr [rax], al ; add byte ptr [rax], al ; endbr64 ; jmp 0x1180
0x000000000000137a : add byte ptr [rax], al ; add byte ptr [rax], al ; leave ; ret
0x0000000000001174 : add byte ptr [rax], al ; add byte ptr [rax], al ; ret
0x000000000000137b : add byte ptr [rax], al ; add cl, cl ; ret
0x0000000000001036 : add byte ptr [rax], al ; add dl, dh ; jmp 0x1020
0x00000000000011f0 : add byte ptr [rax], al ; add dword ptr [rbp - 0x3d], ebx ; nop dword ptr [rax]
0x0000000000001375 : add byte ptr [rax], al ; call rdx
0x00000000000011fe : add byte ptr [rax], al ; endbr64 ; jmp 0x1180
0x0000000000001134 : add byte ptr [rax], al ; jmp 0x137e
0x000000000000137c : add byte ptr [rax], al ; leave ; ret
0x0000000000001176 : add byte ptr [rax], al ; ret
0x000000000000100d : add byte ptr [rax], al ; test rax, rax ; je 0x1016 ; call rax
0x0000000000001168 : add byte ptr [rax], al ; test rax, rax ; je 0x1178 ; jmp rax
0x00000000000011a9 : add byte ptr [rax], al ; test rax, rax ; je 0x11b8 ; jmp rax
0x00000000000011b5 : add byte ptr [rax], r8b ; ret
0x000000000000132e : add byte ptr [rbp + 7], dh ; mov eax, 1 ; jmp 0x137e
0x00000000000011f1 : add byte ptr [rcx], al ; pop rbp ; ret
0x00000000000011ef : add byte ptr cs:[rax], al ; add dword ptr [rbp - 0x3d], ebx ; nop dword ptr [ra
0x0000000000001167 : add byte ptr cs:[rax], al ; test rax, rax ; je 0x1178 ; jmp rax
0x00000000000011a8 : add byte ptr cs:[rax], al ; test rax, rax ; je 0x11b8 ; jmp rax
0x000000000000137d : add cl, cl ; ret
0x0000000000001038 : add dl, dh ; jmp 0x1020
0x0000000000001332 : add dword ptr [rax], eax ; add byte ptr [rax], al ; jmp 0x137e
0x00000000000011f2 : add dword ptr [rbp - 0x3d], ebx ; nop dword ptr [rax] ; ret

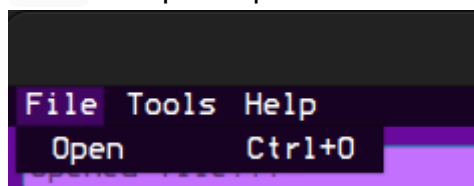
Unique gadgets found: 78

JollyTr0LLz build 22092023 by PurpleGenesis x AP Security
```

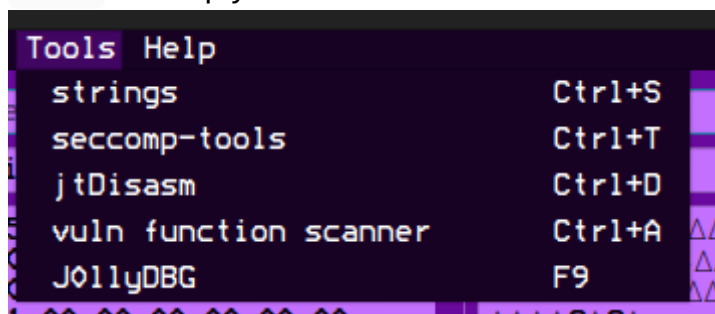
Так же, есть подсказки прям на полях, что является очень удобным.

На верхней панели можно увидеть:

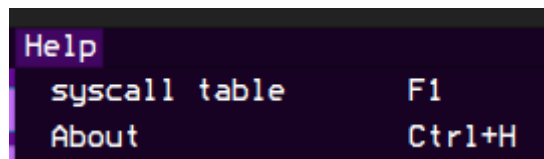
1. File - открыть файл



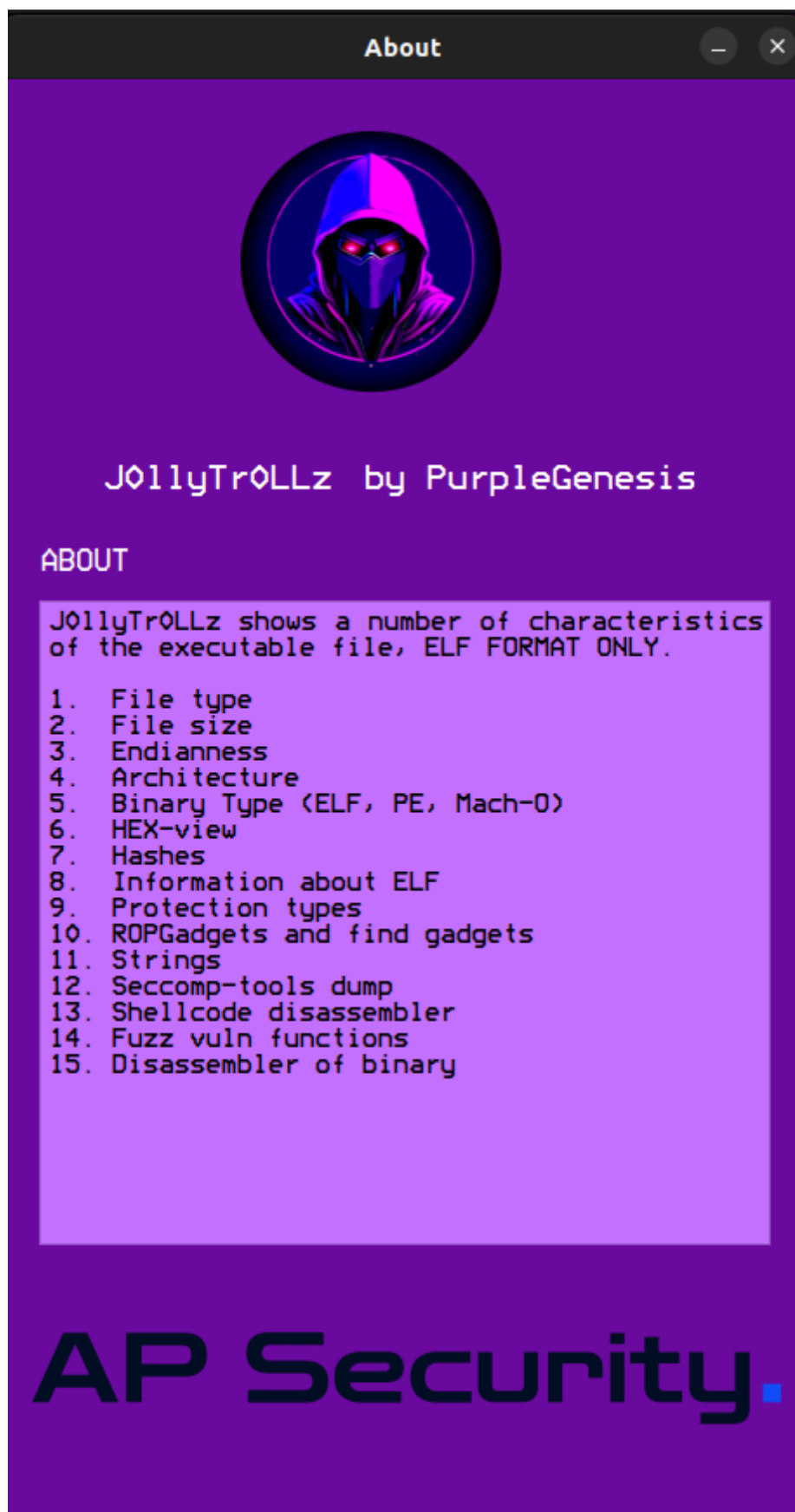
2. Tools - набор утилит



3. Help - содержится информация о файле и справка по системным вызовам



Дальше в обзоре идет окно About, в которое можно попасть нажав комбинацию Ctrl+H

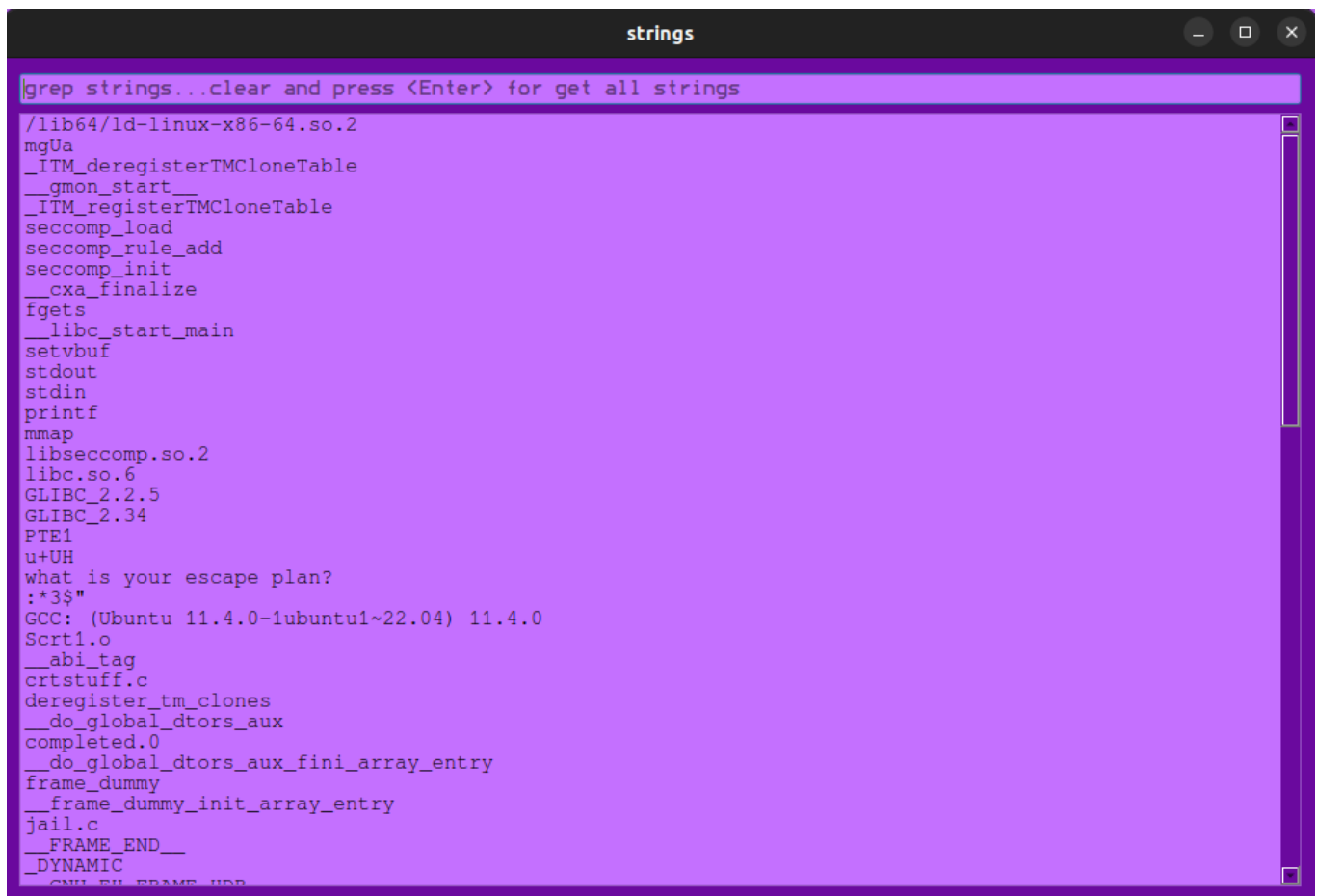


Здесь общая информация о разработке и функционале программы.

После узнаем, что хранится в справочной информации о системных вызовах - F1

Linux System Call Table									
x86	x86_64	arm_32_bit_EABI	arm64						
NR	syscall name	references	rax	arg0 (rdi)	arg1 (rsi)	arg2 (rdx)	arg3 (r10)	arg4 (r8)	arg5 (r9)
0	restart_syscall	man/ cs/	0x00	-	-	-	-	-	-
1	exit	man/ cs/	0x01	int ...	-	-	-	-	-
2	fork	man/ cs/	0x02	-	-	-	-	-	-
3	read	man/ cs/	0x03	unsigned ...	char *buf	size_t ...	-	-	-
4	write	man/ cs/	0x04	unsigned ...	const char*	size_t ...	-	-	-
5	open	man/ cs/	0x05	const char*	int flags	umode_t ...	-	-	-
6	close	man/ cs/	0x06	unsigned ...	-	-	-	-	-
7	waitpid	man/ cs/	0x07	pid_t pid	int ...	int ...	-	-	-
8	creat	man/ cs/	0x08	const char*	umode_t ...	-	-	-	-
9	link	man/ cs/	0x09	const char*	const char*	-	-	-	-
10	unlink	man/ cs/	0x0a	const char*	-	-	-	-	-
11	execve	man/ cs/	0x0b	const char*	const char*	const char*	-	-	-
12	chdir	man/ cs/	0x0c	const char*	-	-	-	-	-
13	time	man/ cs/	0x0d	time_t ...	-	-	-	-	-
14	mknod	man/ cs/	0x0e	const char*	umode_t ...	unsigned ...	-	-	-
15	chmod	man/ cs/	0x0f	const char*	umode_t ...	-	-	-	-
16	lchown	man/ cs/	0x10	const char*	uid_t user	gid_t ...	-	-	-
17	break	man/ cs/	0x11	?	?	?	?	?	?
18	oldstat	man/ cs/	0x12	?	?	?	?	?	?
19	lseek	man/ cs/	0x13	unsigned ...	off_t ...	unsigned ...	-	-	-
20	getpid	man/ cs/	0x14	-	-	-	-	-	-

Посмотрим какие строки нашла программа. Для этого нужно нажать на комбинацию клавиш **Ctrl+S**

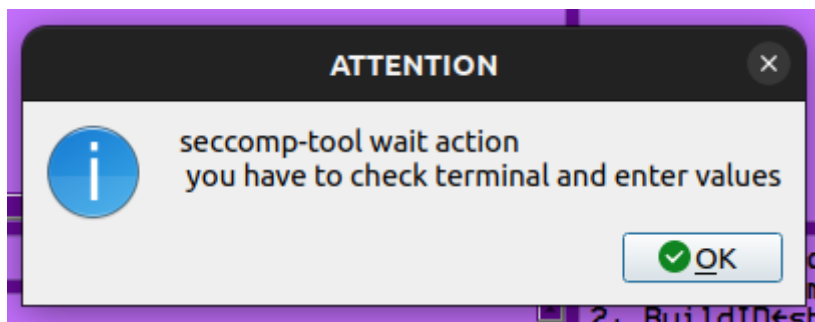


```
strings
grep strings...clear and press <Enter> for get all strings

/lib64/ld-linux-x86-64.so.2
mgUa
__ITM_deregisterTMCloneTable
__gmon_start__
__ITM_registerTMCloneTable
seccomp_load
seccomp_rule_add
seccomp_init
__cxa_finalize
fgets
__libc_start_main
setvbuf
stdout
stdin
printf
mmap
libseccomp.so.2
libc.so.6
GLIBC_2.2.5
GLIBC_2.34
PTE1
u+UH
what is your escape plan?
.*3$
GCC: (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0
Scrt1.o
__abi_tag
crtstuff.c
deregister_tm_clones
__do_global_dtors_aux
completed.0
__do_global_dtors_aux_fini_array_entry
frame_dummy
__frame_dummy_init_array_entry
jail.c
__FRAME_END__
_DYNAMIC
GNU_EH_FRAME_HDR
```

В данном окне отображаются все строки, которые находятся в бинарном файле и присутствует аналогичная строка поиска.

Посмотрим, какие защиты от шеллкода в программе, комбинацией `Ctrl+T`. Будет высвечено сообщение, что необходимо перейти в терминал и ввести какие-нибудь данные, чтобы программа смогла определить защиту

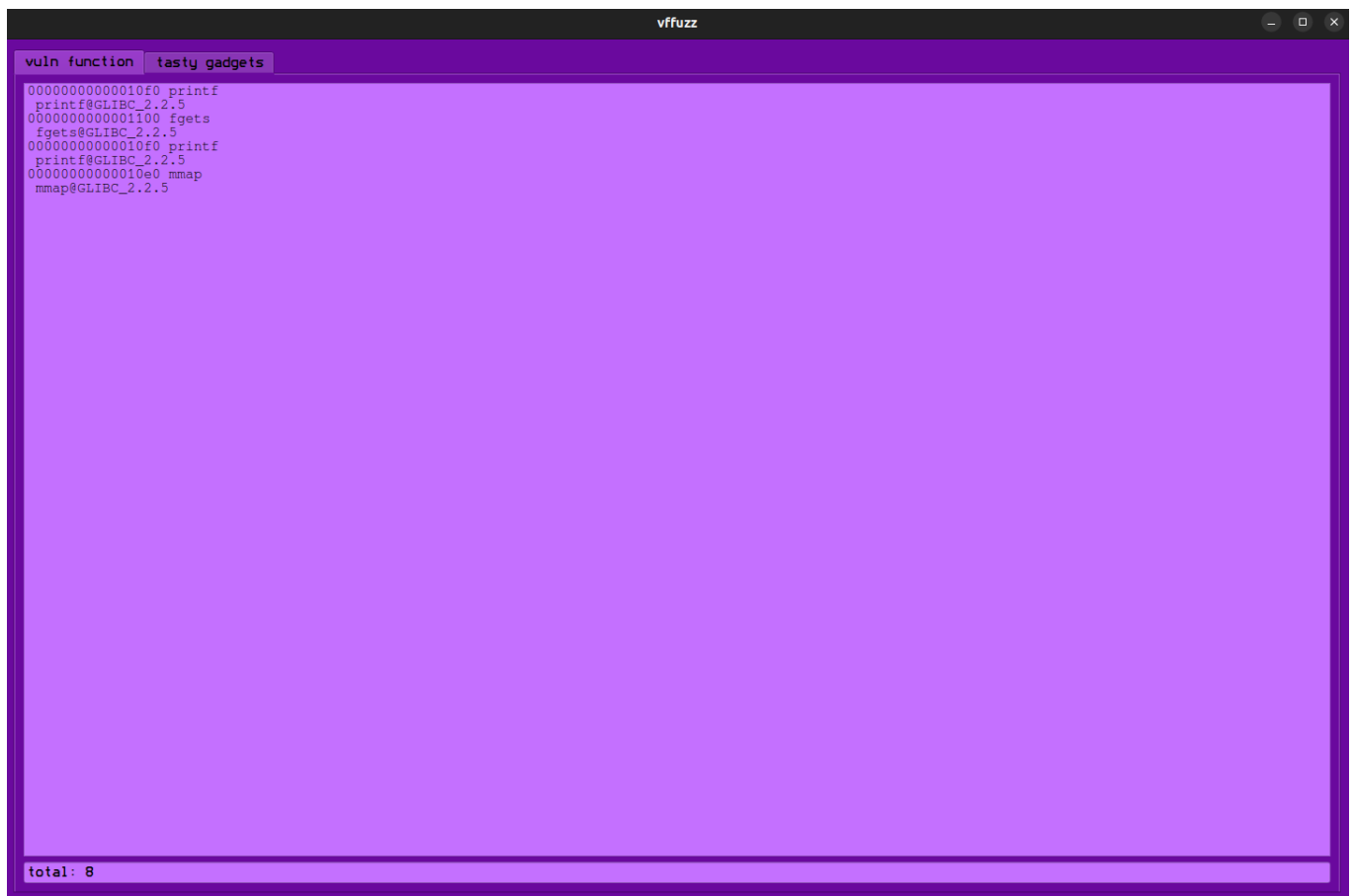


```
[*] seccomp-tool wait action
purplegenesis x apsecurity
```

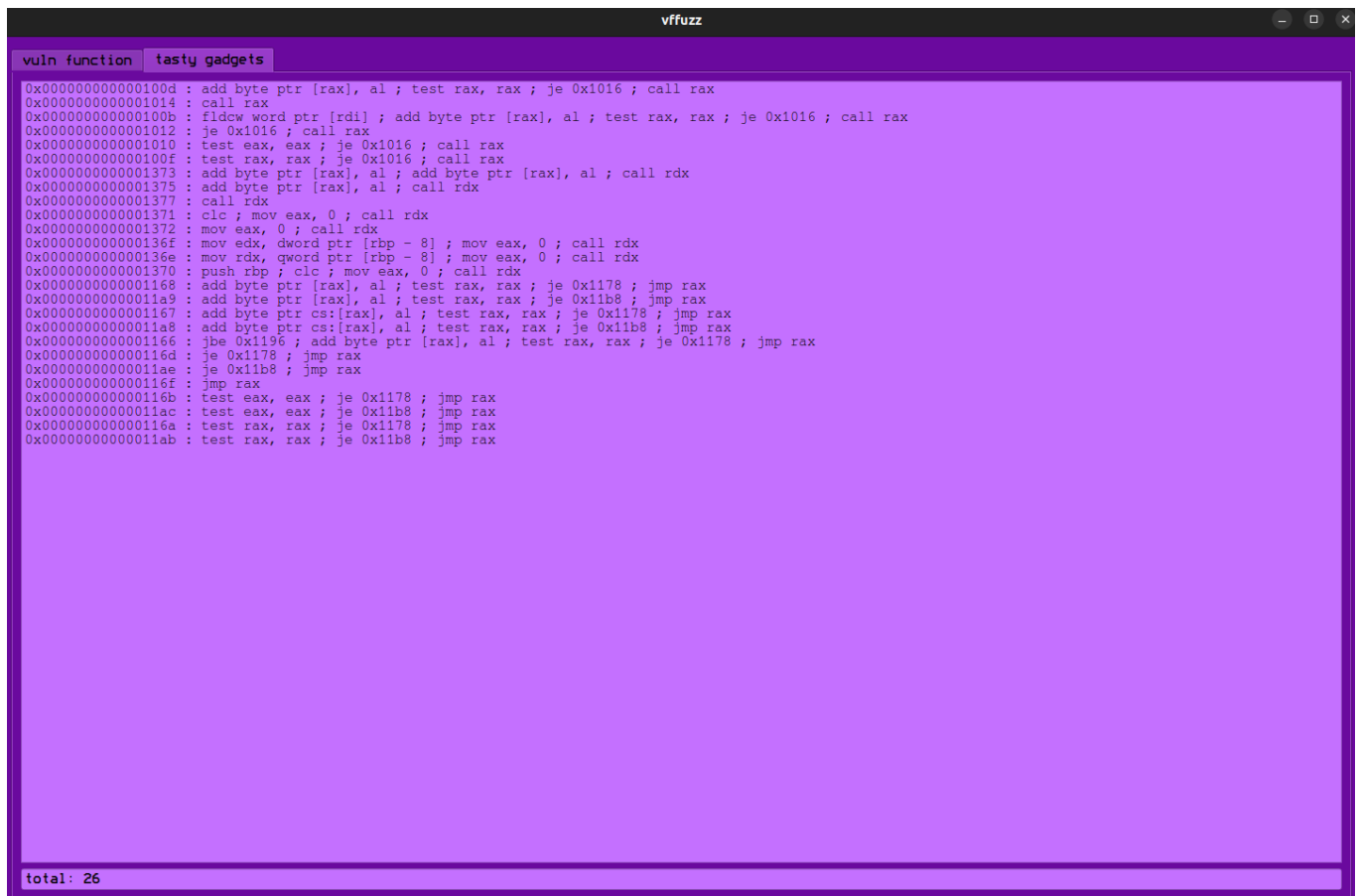
Результат выполнения будет отображено в отдельном окне

```
seccomp-tools
line  CODE  JT   JF      K
0000: 0x20  0x00  0x00  0x00000004  A = arch
0001: 0x15  0x00  0x08  0xc000003e  if (A != ARCH_X86_64) goto 0010
0002: 0x20  0x00  0x00  0x00000000  A = sys_number
0003: 0x35  0x00  0x01  0x40000000  if (A < 0x40000000) goto 0005
0004: 0x15  0x00  0x05  0xffffffff  if (A != 0xffffffff) goto 0010
0005: 0x15  0x03  0x00  0x00000000  if (A == read) goto 0009
0006: 0x15  0x02  0x00  0x00000023  if (A == nanosleep) goto 0009
0007: 0x15  0x01  0x00  0x0000003c  if (A == exit) goto 0009
0008: 0x15  0x00  0x01  0x00000101  if (A != openat) goto 0010
0009: 0x06  0x00  0x00  0x7fff0000  return ALLOW
0010: 0x06  0x00  0x00  0x00000000  return KILL
```

Дальше идет сканнер потенциально уязвимых функций и поиск подходящих гаджетов для попадания в стек - `Ctrl+A`. В первой вкладке набор уязвимых функций

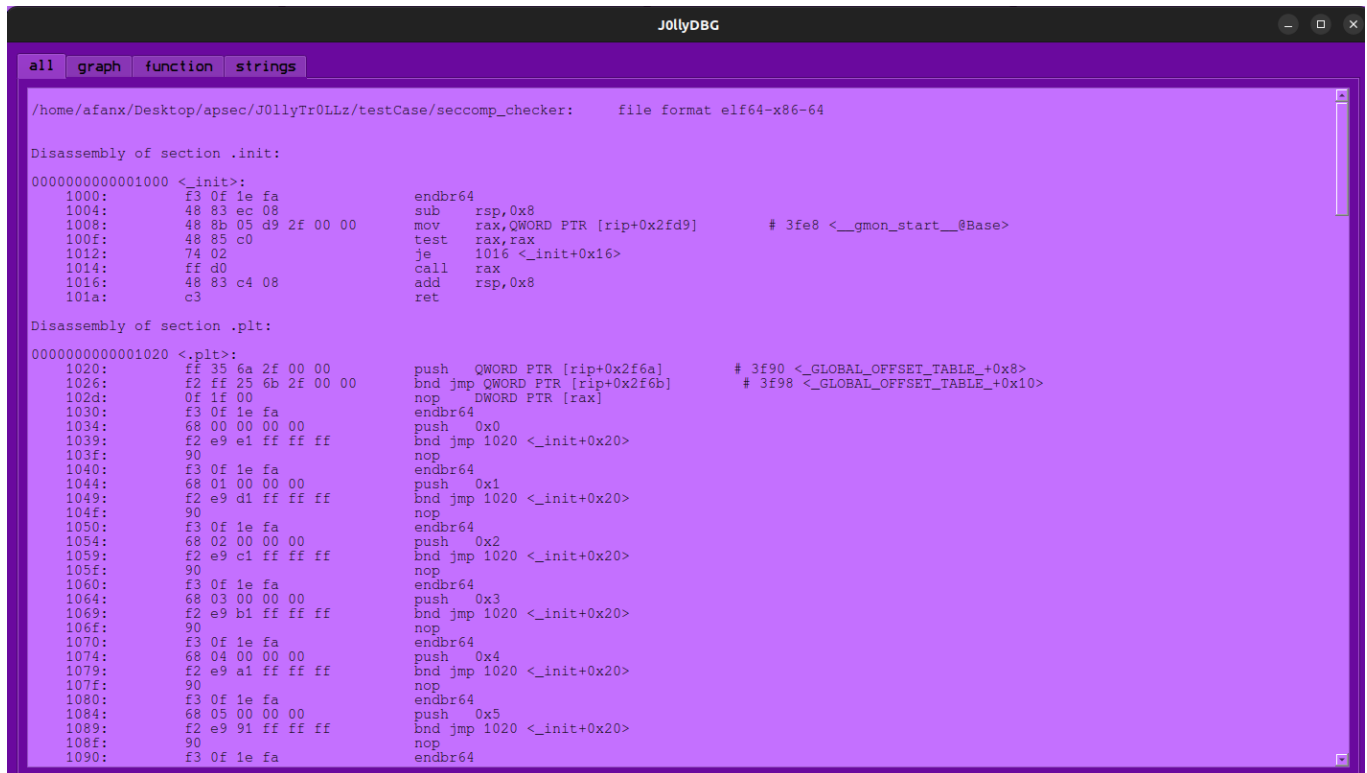


Во второй гаджеты

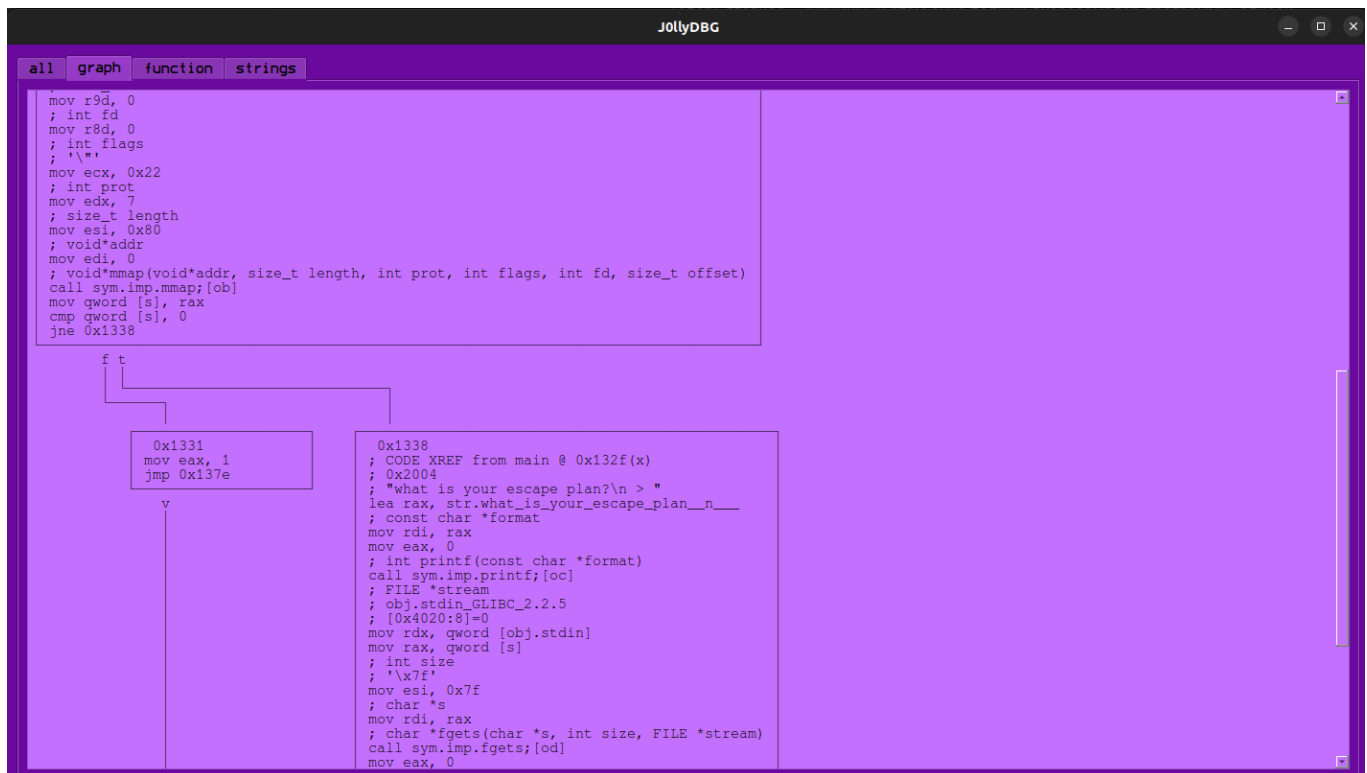


Теперь главное - JollyDBG. В этом окне полный дизассемблер программы, список функций и графы к ним. Для попадания в него нажимаем **F9**

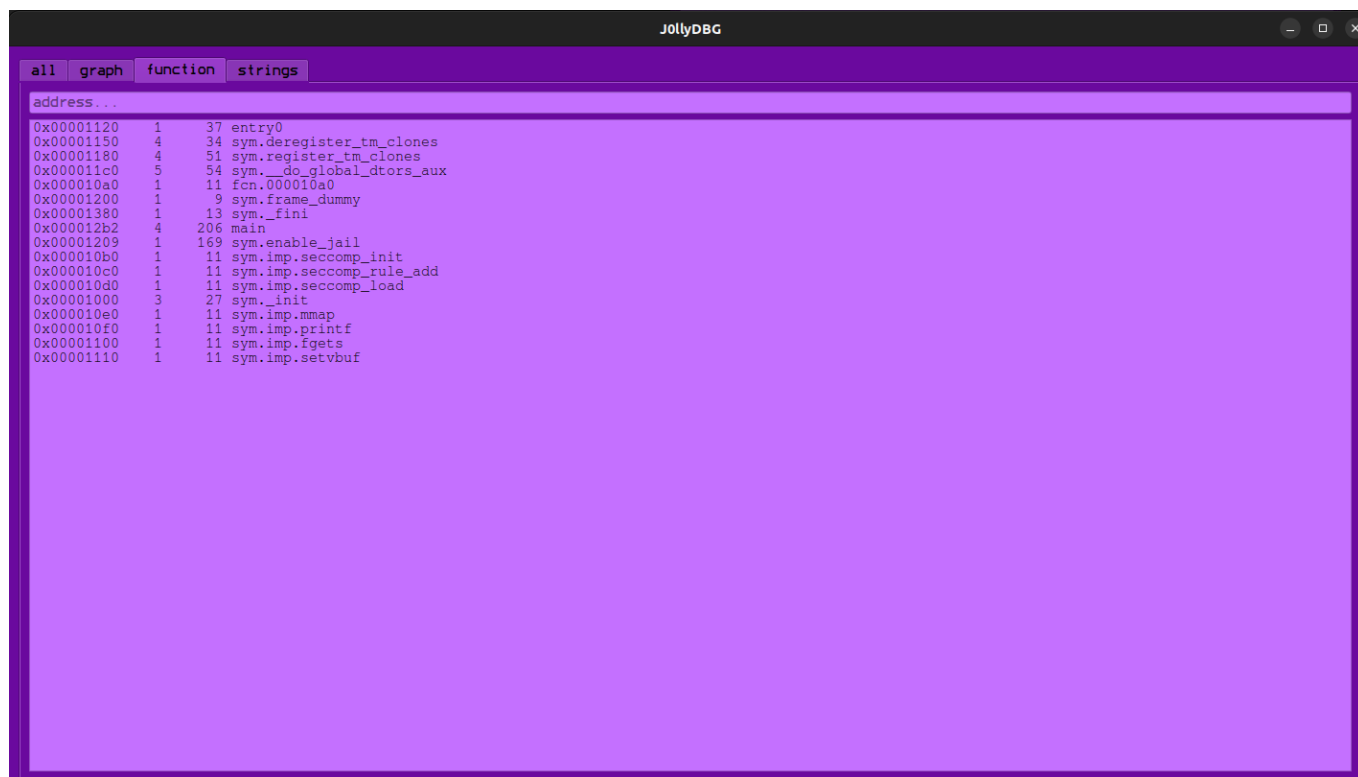
Вкладка **All** хранит всю программу со всеми секциями и кодом



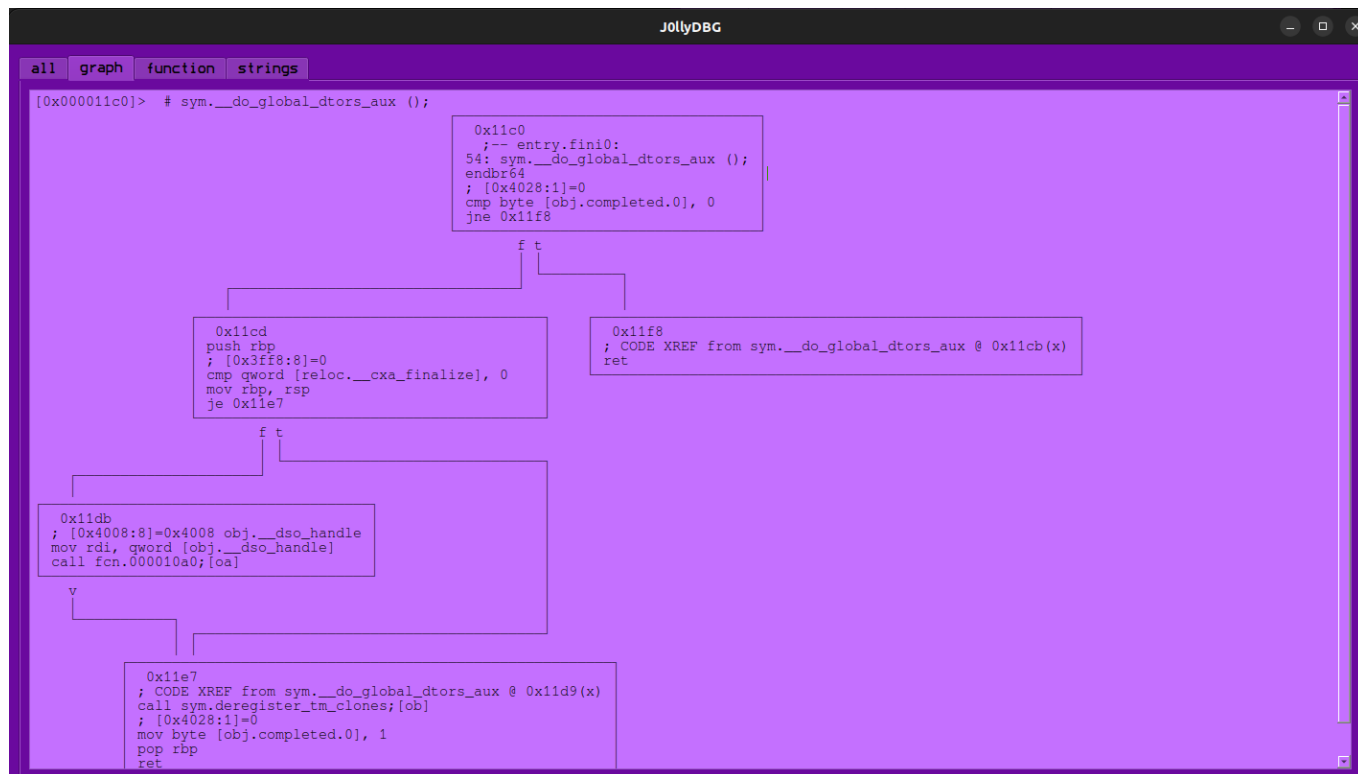
Вкладка **Graph** - отображение функции в виде графа. По умолчанию - функция `main()`



Вкладка `function` содержит все функции, которые есть в программе



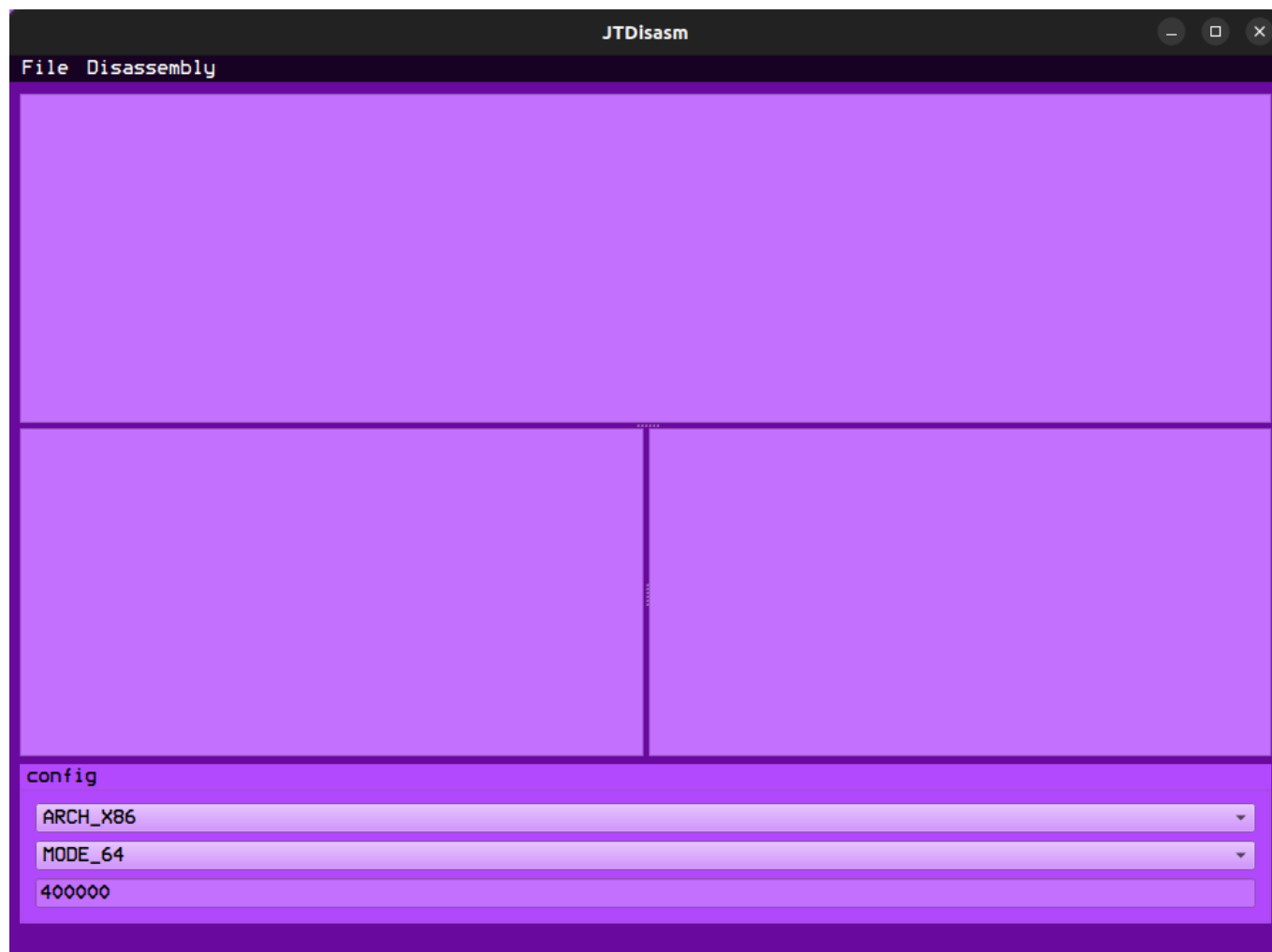
Чтобы посмотреть ее в виде графа, нужно нажать дважды по функции. Посмотрим граф функции `sym.__do_global_dtors_aux`. Два раза нажимаем на него и попадаем в граф



Вкладка `strings` аналогична той, которая была ранее.

Последнее, что осталось - дизассемблер шеллкода. Очень удобная штука для анализа шеллкода. Больше подойдет для людей, которые занимаются форензикой. Для создания шеллкода, у AP Security есть программа [KillerQueen](#).

Чтобы запустить, нужно нажать `Ctrl+D`



Для теста откроем классический шеллкод - вызов командной оболочки. Открыть файл - `Ctrl+O`

config

ARCH_X86

MODE_64

100

При изменении адреса, меняется адрес и в дизассемблере

```
0x100: 4831c0 xor    rax, rax
0x103: 4831ff xor    rdi, rdi
0x106: 4831d2 xor    rdx, rdx
0x109: 4831f6 xor    rsi, rsi
0x112: 488d3d0900000000 lea     rdi, [rip + 9]
0x119: 4883c03b add    rax, 0x3b
0x123: 0f05 syscall
0x125: 0000 add    byte ptr [rax], al
0x127: 002f add    byte ptr [rdi], ch
```