Reverse Engineering

Reverse Engineering (Реверс) - процесс восстановления внутренней структуры/вида предмета/программы из конечного продукта человеческой деятельности, интуитивно конструируя внутреннюю механику по выходным данным. Два основных вида – hardware и software reverse. Нас интересует software, так что на «железячном» останавливаться не будем.

Software Reverse

Реверсинг ПО - восстановление исходного кода программы для исследования и/или создания аналогичного ПО. Часто применяется для:

- Анализа вредоносного кода с целью создания средств защиты.
- Поиска недокументированных возможностей в закрытых программах с целью создания вредоносного кода.
- Создания описаний для форматов данных/протоколов, используемых в программах и устройствах.

Не следует путать взлом (crack) ПО с его реверсингом: для взлома достаточно разобрать принцип работы лицензионной проверки, а реверсинг - это полный разбор программы по кирпичикам.

Для успешного реверса требуется знать ассемблер, иметь общее представление о криптографии и знать английский, чтобы ~~курить мануалы~~ изучить stackoverflow, официальную документацию по функциям конкретного языка/ устройства. Ну и матан - например, теорию графов для анализа ветвлений. Но можно и не знать, главное — уметь быстро найти, что почитать, и резво разобраться.

Основные инструменты

На данный момент must have состоит из следующих программ:

- **Process Monitor**: Очень удобный инструмент для исследования внешнего поведения подопытной программы. Позволяет отслеживать задачи, выполняемые программой. Следит за Registry Access, File System Access, Network Access, Thread Management и т.д. Так же реализованы фильтры по названию / поведению. Ссылка - https://docs.microsoft.com/ru-ru/sysinternals/downloads/procmon
- **JetBrains dotPeek**: Позволяет декомпилировать .Net assembly.

- **Ida Pro**: Дизассемблер. По нынешним временам – основной инструмент реверс-инжинера (виндусятника, хотя на линуксе можно использовать через wine). Поставляется вместе с (псевдо) декомпилятором hexrays, который переводит ассемблерный код в исполняемый С. (~~можно найти в интернете по фразе ida pro leak, но я вам этого не говорил~~)
- **Radare2**: Фреймворк для реверс-инжиниринга. Изначально hex-редактор включает в себя дизассемблер, декомпилилятор. Мультиплатформенный.

Для успешного использования любого дизассемблера, нужно знать ~~спасибо, кэп~~ язык ассемблера.

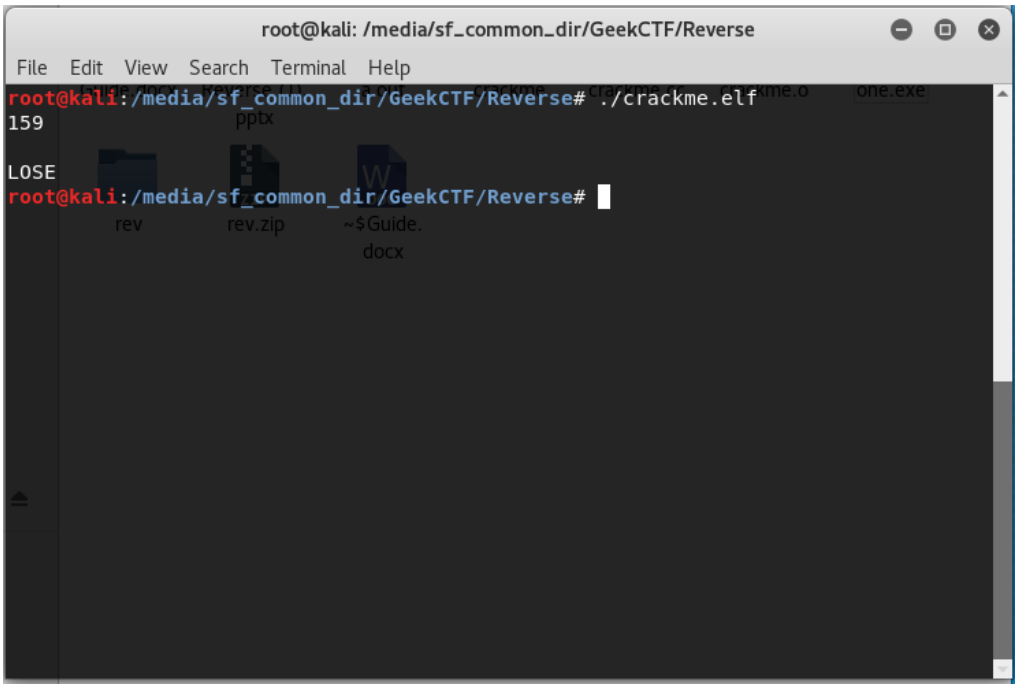
Язык ассемблера (ассемблер / асм / asm) — простейший способ записи машинного кода с помощью английских сокращений, называемых мнемониками.

Смысла писать очередной самоучитель нет, так что вот пару полезных ссылок на гайды по асму:

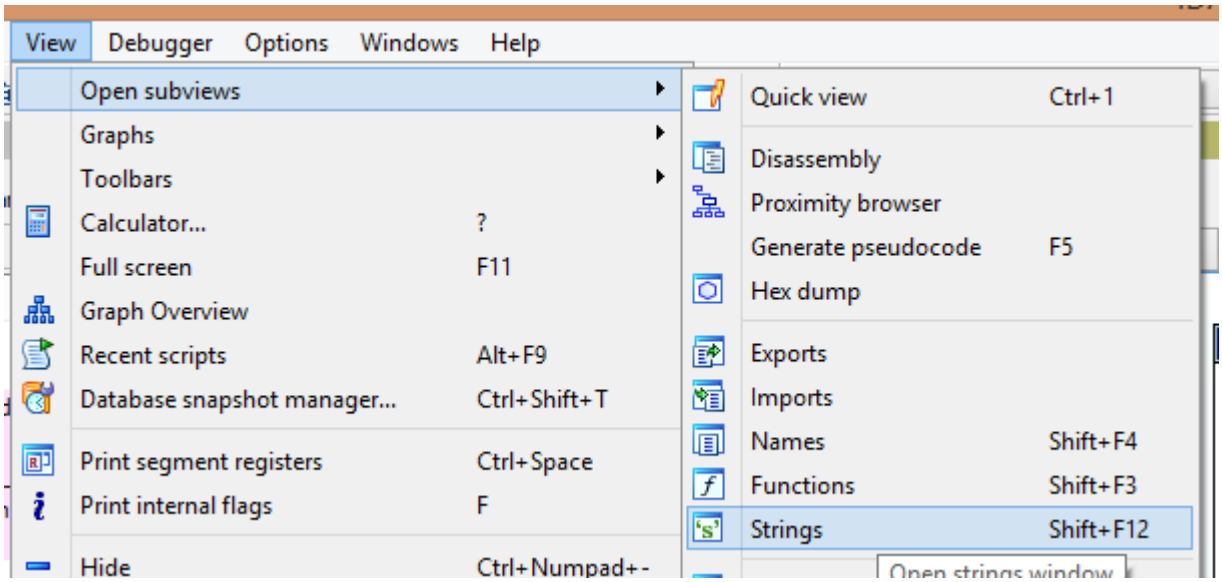
- http://osinavi.ru/asm/1.html
- http://asmworld.ru/uchebnyj-kurs/001-neobxodimye-instrumenty/

Глянем простейший crack-me. Исходники давно утеряны, но мы тут все-таки реверсом занимаемся, так что приступим.

Для начала запускаем наш elf файл (желательно в виртуальной машине):

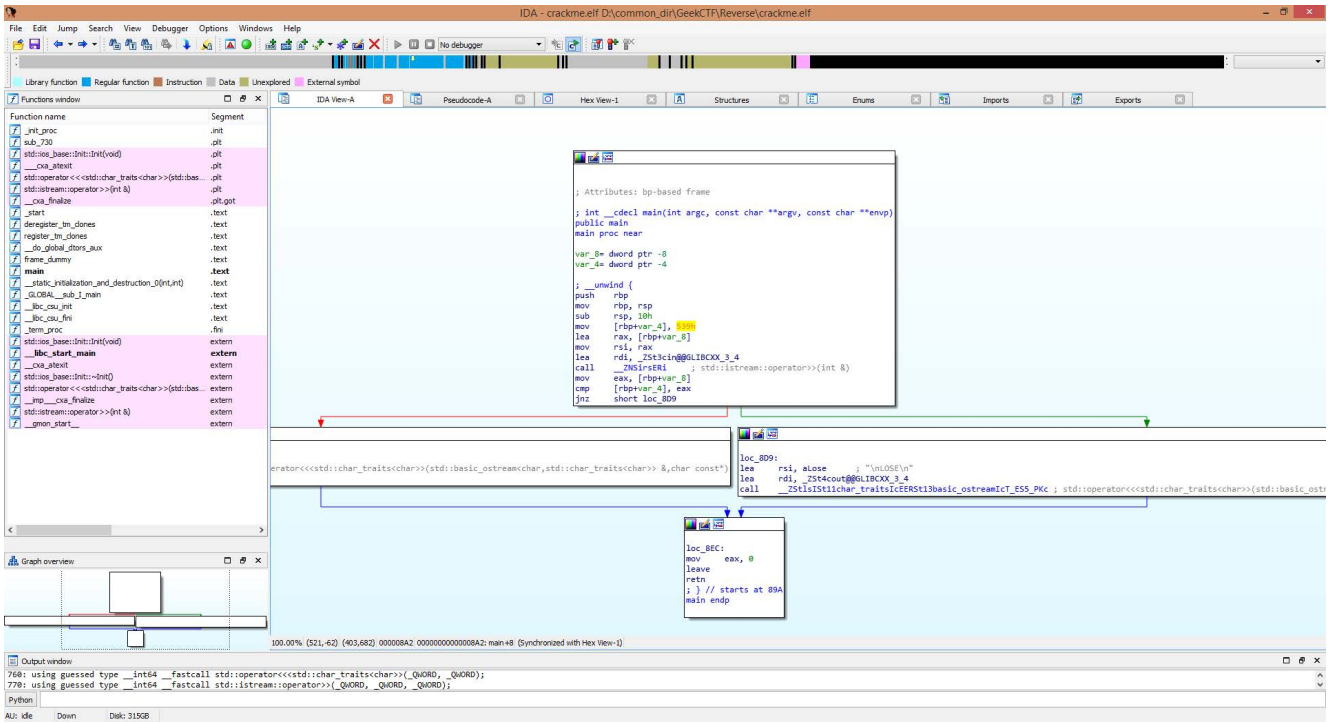


Вводим случайные данные и получаем сообщение об ошибке. Используя это сообщение мы сможем найти кусок кода, который отвечает за проверку пароля. (С помощью поиска по строкам, в данный момент нам это не пригодится, так как программа маленькая)



Для этого открываем наш любимый IDA Pro 7.0 x32 и открываем в нем наш файл.

Видим следующую картину:

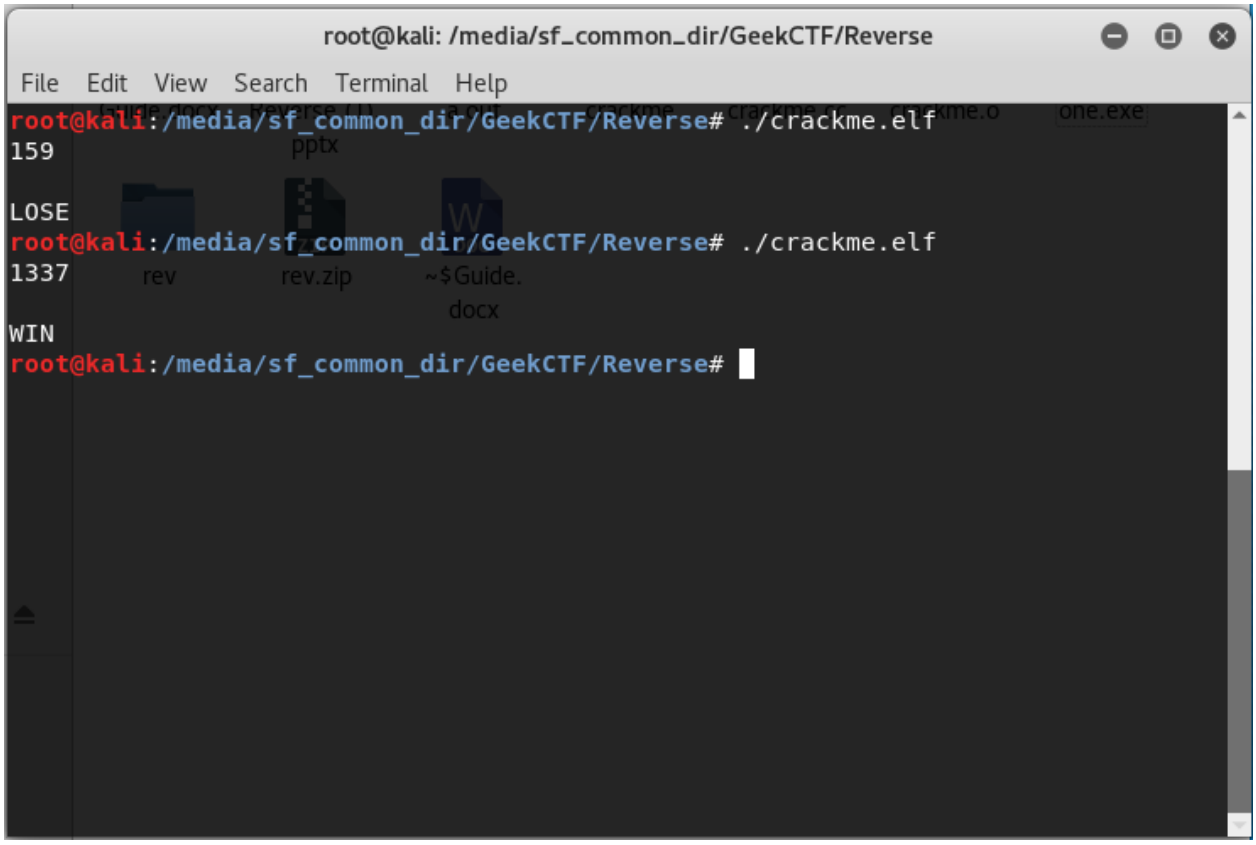


Да, тут не особо что видно, но нам нужно в основном окошко кода. Для начала посмотрим псевдокод. Нажимаем F5 и смотрим на чудо:

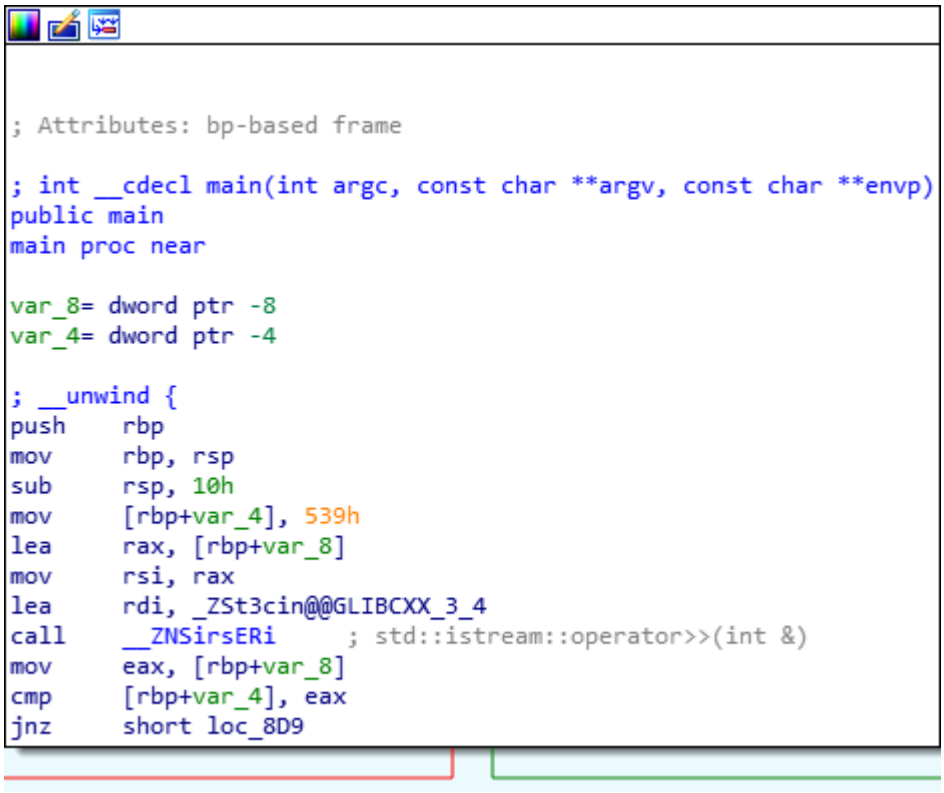
IDA View-A | Pseudocode-B | Hex View-1

```
1  int __cdecl main(int argc, const char **argv, const char **envp)
2  {
3    int v4; // [rsp+8h] [rbp-8h]
4    int v5; // [rsp+Ch] [rbp-4h]
5
6    v5 = 1337;
7    std::istream::operator>>(&std::cin, &v4, envp);
8    if ( v5 == v4 )
9      std::operator<<<std::char_traits<char>>(&std::cout, "\nWIN\n");
10   else
11     std::operator<<<std::char_traits<char>>(&std::cout, "\nLOSE\n");
12   return 0;
13 }
```

Нечто ужасное свернулось в 13 строк кода. 3, 4 строки – объявление переменных, 7 – ввод в переменную v4, 8 – сравниваем v4 и v5. Значит, если мы введем 1337 в программу, то получим сообщение “WIN”. Проверим:



Да, так оно и есть. Теперь рассмотрим подробнее ассемблерный код:



```
; Attributes: bp-based frame

; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near

var_8= dword ptr -8
var_4= dword ptr -4

; __unwind {
push    rbp
mov     rbp, rsp
sub     rsp, 10h
mov     [rbp+var_4], 539h
lea     rax, [rbp+var_8]
mov     rsi, rax
lea     rdi, _ZSt3cin@@GLIBCXX_3_4
call    __ZNSirsERi     ; std::istream::operator>>(int &)
mov     eax, [rbp+var_8]
cmp     [rbp+var_4], eax
jnz     short loc_8D9
```

var_8 =… , var_4 = … - объявление переменных. Далее смотрим, где они используются в коде.

Видим строку mov [rbp+var_4], 539h. – происходит запись по адресу rbp+var_4 (в нашу переменную var_4) шестнадцатеричного числа 0x539 = 1337.

Далее видим какую-то движуху с переменной var_8, которую ida любезно подписывает как std::istream::operator>>(int &).

/* Для использования функции в асм, необходимо загрузить в стек/ нужные регистры входные данные, в нашем случае – адрес переменной var_8 и после этого вызвать функцию – операция call */

mov eax, [rbp+var_8] – переписываем в регистр eax переменную var_8

cmp [rbp+var_4], eax – сравниваем var_4 и eax (var_8).

jnz short loc_8D9 – если не равны, то прыгаем на кусок кода, выводящий LOSE, если равны – прыгаем на WIN.

Это был простой и корявый гайд по основам реверса. Есть прекрасный курс, находящийся сейчас в стадии перевода: https://wasm.in/blogs/vvedenie-v-reversing-s-nulja-ispolzuja-ida-pro-chast-1.3/