ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
Национальный исследовательский университет
«Высшая школа экономики»

Московский институт электроники и математики им. Тихонова А.Н.
Национального исследовательского университета
«Высшая школа экономики»

Департамент прикладной математики

Кафедра «Компьютерная безопасность»

ОТЧЕТ ПО ЗАДАНИЮ 2 (часть 1)
по дисциплине «Защита программ и данных»

Выполнил:
студент группы СКБ161
Воинов Н. В.

МОСКВА 2020

1 Задание

Дизассемблировать первую предложенную программу и восстановить алгоритм работы. После восстановления алгоритма написать программу «кейген» для генерации пароля (ключа) по введённым данным. Составить отчёт по проведённым исследованиям.

2 Используемые методы

В ходе работы был применен статический метод восстановления алгоритма. А именно были произведены декомпиляция и дезассемблирование. В данном случае этого метода хватило для восстановления алгоритма.

3 Используемые программы

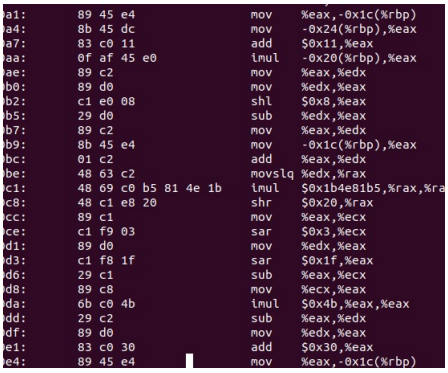
Для проведения анализа была использованы:

- objdump.
- IDA. Бесплатная версия, удобно использовать в качестве дизассемблера + построить некоторое представление в виде графа.
- Cutter. Использовал в качестве декомпилятора.

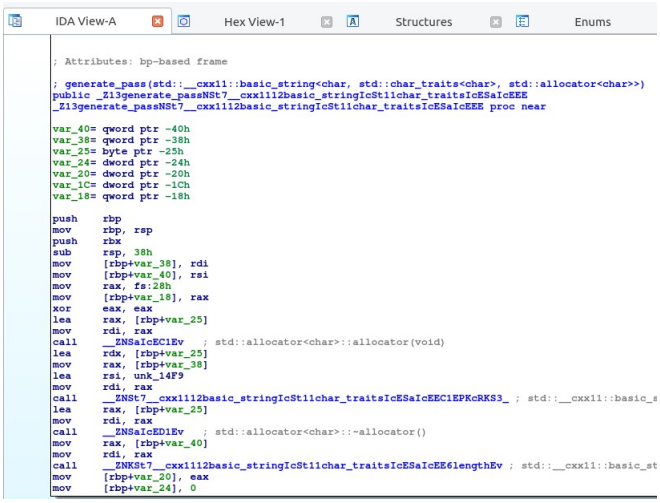
4 Ход работы

Анализ начал с самого простого шага - с помощью objdump дизассемблировал исходный код. Быстро “пробежался” по коду и заметил имена функций *...generate_pass...* и *...check...*.

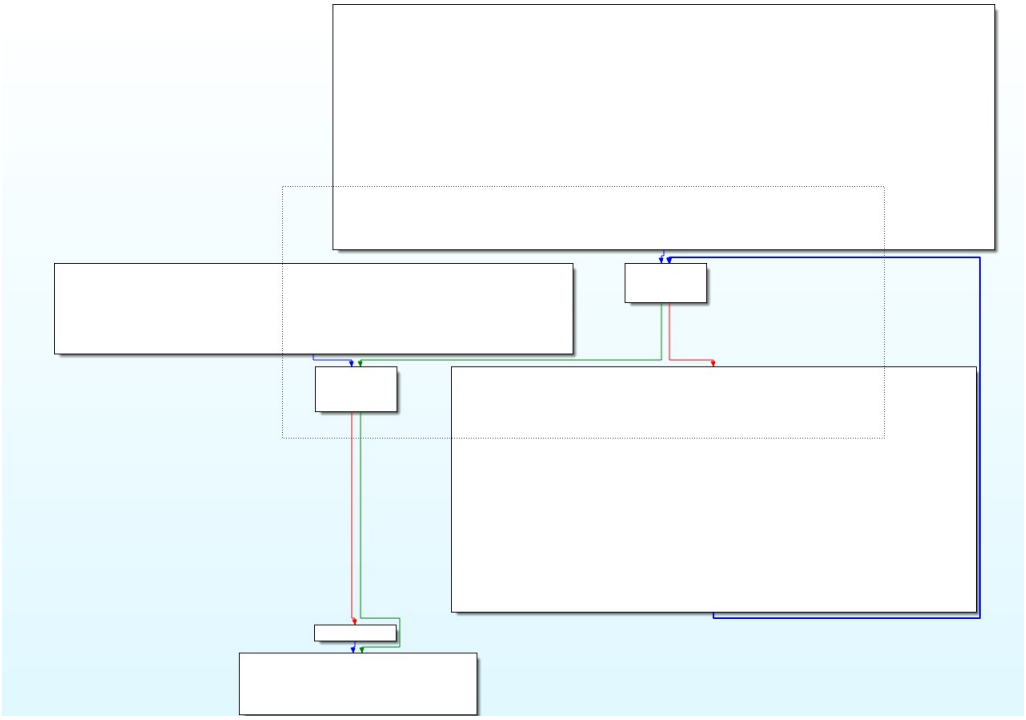
Название *generate_pass* очень привлекло внимание и я решил посмотреть “что внутри”. Внутри оказались некоторые математические операции, например add, imul, sub.



Рассматривать вывод objdump-a полезно, но не наглядно, поэтому решил воспользоваться IDA. Перешел к найденной функции *...generate_pass...*.



IDA построила представление этой функции в виде графа. Из него стало видно, что в функции есть цикл. В этом цикле производятся некоторые арифметические вычисления. Рассмотрел его подробнее.



```
mov     eax, [rbp+var_24]
movsxd  rdx, eax
mov     rax, [rbp+var_40]
mov     rsi, rdx
mov     rdi, rax
call    __ZNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEixEm ; st
movzx   eax, byte ptr [rax]
movsx   eax, al
mov     [rbp+var_1C], eax
mov     eax, [rbp+var_24]
add     eax, 11h
imul    eax, [rbp+var_20]
mov     edx, eax
mov     eax, edx
shl     eax, 8
sub     eax, edx
mov     edx, eax
mov     eax, [rbp+var_1C]
add     edx, eax
movsxd  rax, edx
imul    rax, 1B4E81B5h
shr     rax, 20h
mov     ecx, eax
sar     ecx, 3
mov     eax, edx
sar     eax, 1Fh
sub     ecx, eax
mov     eax, ecx
imul    eax, 4Bh
sub     edx, eax
mov     eax, edx
add     eax, 30h ; '0'
mov     [rbp+var_1C], eax
mov     eax, [rbp+var_1C]
movsx   edx, al
mov     rax, [rbp+var_38]
mov     esi, edx
mov     rdi, rax
call    __ZNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEpLEc ; st
add     [rbp+var_24], 1
jmp     loc_107A
```

Заметил, что этот цикл проходит по всем элементам параметра функции. И последовательно производит следующие операции:

1. Сложение счетчика с 11h
2. Умножение полученного значения на значение $[rbp + var\_20]$, которое является результатом выполнения (т.е. длиной логина)

   `_ZNKSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEE6lengthEv ;`

`std::__cxx11::basic_string<char,std::char_traits<char>, std::allocator<char>>::length(void)`

3. Сдвиг на 8 влево и вычитание этого значения (Это реализация умножения на $0xff$)
4. Некоторое вычисление (Его я не понял сразу, решил посмотреть декомпилированную версию)
5. Прибавление 30h

Дальше я произвел декомпилирование с помощью Cutter. Оно подтвердило все догадки и добавило информации об алгоритме.

```
std::basic_istream<char, std::char_traits<char> >& std::operator>><char, std::char_traits<char> >(std::basic_
          (reloc.std::cin, &var_70h);
arg3 = _reloc.strcmp;

std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >::basic_string(std::__cxx11::b
          (&var_b0h, &var_d0h, &var_d0h);
generate_pass(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >)
          ((int64_t)&var_90h, (int64_t)&var_b0h);
arg2 = std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >::c_str() const
                    (&var_90h);
check(char*, char const*, int (*)(char const*, char const*))((int64_t)&var_70h, arg2, arg3);
std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >::~basic_string()(&var_90h);
std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >::~basic_string()(&var_b0h);
```

Функция $generate\_pass(...)$:

```
int64_t generate_pass(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >)
                (int64_t arg1, int64_t arg2)
{
    char *pcVar1;
    int64_t in_FS_OFFSET;
    int64_t var_40h;
    int64_t var_38h;
    int64_t var_25h;
    int32_t var_1ch;
    int64_t canary;

    // generate_pass(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >)
    canary = *(int64_t *)(in_FS_OFFSET + 0x28);
    std::allocator<char>::allocator()(&var_25h);

    std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >::basic_string(char const*, std::allocator<char> const&)
              (arg1, 0x14f9, &var_25h);
    std::allocator<char>::~allocator()(&var_25h);
    *(int32_t *)0x0 =
         std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >::length() const(arg2);
    var_25h._1_4_ = 0;
    while (var_25h._1_4_ < stack0xfffffffffffffd8) {
        pcVar1 = (char *)
                  std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >::operator[](unsigned long)
                            (arg2, (int64_t)var_25h._1_4_, (int64_t)var_25h._1_4_);
        var_1ch = ((var_25h._1_4_ + 0x11) * stack0xfffffffffffffd8 * 0xff + (int32_t)*pcVar1) % 0x4b + 0x30;
        std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >::operator+=(char)
                  (arg1, (int32_t)(char)var_1ch, (int32_t)(char)var_1ch);
        var_25h._1_4_ = var_25h._1_4_ + 1;
    }
    if (canary != *(int64_t *)(in_FS_OFFSET + 0x28)) {
    // WARNING: Subroutine does not return
        __stack_chk_fail();
    }
    return arg1;
}
```

Функция $check(...)$:

```
void check(char*, char const*, int (*)(char const*, char const*))(int64_t arg1, int64_t arg2, int64_t arg3)
{
    int32_t iVar1;
    int64_t var_18h;
    int64_t var_10h;
    int64_t var_8h;

    // check(char*, char const*, int (*)(char const*, char const*))
    iVar1 = (*(code *)arg3)(arg1, arg2, arg2, arg3);
    if (iVar1 == 0) {
        puts("Access granted");
    } else {
        puts("Access denied");
    }
    return;
}
```

Данная функция проверяет совпадают ли введенный пароль и пароль, полученный из логина. И, если совпадают, выводит "Access granted". Иначе - "Access denied". Параметрами функции являются два пароля, введенный и вычисленный, а так же функция сравнения (В данном случае использована функция $\_reloc.strcmp$ (по всей видимости - strcmp)).

5 Восстановленный алгоритм

1. Ввод логина
2. Цикл пока true (Бесконечный)

   Если длина логина $<= i$

   Ввод пароля

   $generate\_pass(...)$

   $check(...)$

   Выйти

   Иначе

   Если i-й символ является буквой

   $i++$

   Иначе

   Вывести ошибку

   Выйти

Функция $generate\_pass$ вычисляет значение по следующему алгоритму:

1. Цикл от $i = 0$ до len-1(len=“Длина логина”)

   S - i-й символ логина

   Результат$[i] = ((i + 0x11) * len * 0xff + (int32_t)S)\%0x4b + 0x30;$
2. Вернуть Результат

6 Генератор пароля

```
#include <iostream>
#include <string>
int main(int argc,char* argv[])
{
    if(argc!=2) {std::cout<<"usage: gen_name login\n"; return 1;}
```

```
    std::string arg2=argv[1],a={};
    auto arg1=&a; auto len =arg2.length();
    auto i = 0;
    for(int j=0;j<len;j++) if(!isalpha(arg2[j])) {std::cout<<"Only a-Z!\n";
        return 1;}
    while (i < len) {
        auto S = arg2[i];
        auto v = ((i + 0x11) * len * 0xff + (int32_t)S)%0x4b + 0x30;
      a+=v;
      i = i + 1;
    }
    std::cout<<a;
    return 0;
}
```

Примеры работы: