

**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....*

```
0000000000001015 <_Z13generate_passNSt7__cxx1112basic_stringIcSt11char_traitsIcESa1cEEE>:
1015: 55          push    %rbp
1016: 48 89 e5    mov     %rsp,%rbp
1019: 53          push    %ebx
101a: 48 83 ec 38 sub     $0x38,%rsp
101e: 48 89 7d c8 mov     %rdi,-0x38(%rbp)
1022: 48 89 75 c0 mov     %rsi,-0x40(%rbp)
1026: 64 48 8b 04 25 28 00 mov     %fs:0x28,%rax
102d: 00 00
```

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

```

a1: 89 45 e4      mov     %eax,-0x1c(%rbp)
a4: 8b 45 dc      mov     -0x24(%rbp),%eax
a7: 83 c0 11      add     $0x11,%eax
aa: 0f af 45 e0   inu1    -0x20(%rbp),%eax
ae: 89 c2        mov     %eax,%edx
b0: 89 d0        mov     %edx,%eax
b2: c1 e0 08      shl     $0x8,%eax
b5: 29 d0        sub     %edx,%eax
b7: 89 c2        mov     %eax,%edx
b9: 8b 45 e4      mov     -0x1c(%rbp),%eax
bc: 01 c2        add     %eax,%edx
be: 48 63 c2      movslq  %edx,%rax
c1: 48 09 c0 b5 81 4e 1b inu1    $0x1b4e81b5,%rax,%rax
c8: 48 c1 e8 20      shr     $0x20,%rax
ce: 89 c1        mov     %eax,%ecx
ce: c1 f9 03      sar     $0x3,%ecx
d1: 89 d0        mov     %edx,%eax
d3: c1 f8 1f      sar     $0x1f,%eax
d6: 29 c1        sub     %eax,%ecx
d8: 89 c8        mov     %ecx,%eax
da: 6b c0 4b      inu1    $0x4b,%eax,%eax
dd: 29 c2        sub     %eax,%edx
df: 89 d0        mov     %edx,%eax
e1: 83 c0 30      add     $0x30,%eax
e4: 89 45 e4      mov     %eax,-0x1c(%rbp)

```

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

```

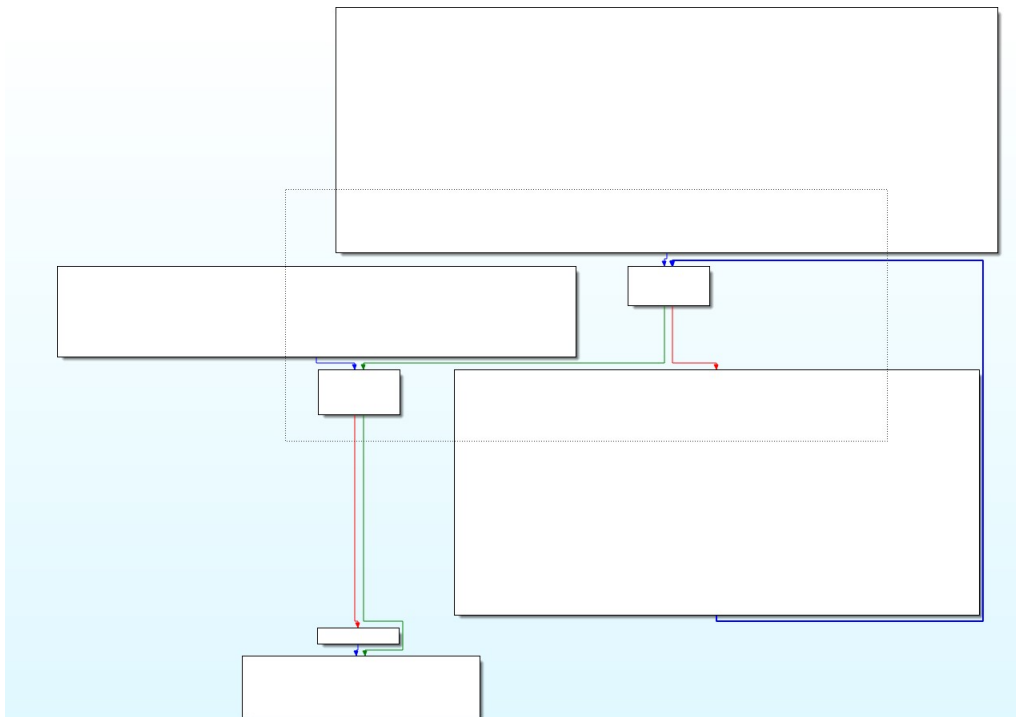
; Attributes: bp-based frame
; generate_pass(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>)
public __Z13generate_passNSt7__cxx112basic_stringIcSt11char_traitsIcESaIcEEE
__Z13generate_passNSt7__cxx112basic_stringIcSt11char_traitsIcESaIcEEE proc near

var_40= qword ptr -40h
var_38= qword ptr -38h
var_25= byte ptr -25h
var_24= dword ptr -24h
var_20= dword ptr -20h
var_1c= dword ptr -1Ch
var_18= qword ptr -18h

push    rbp
mov     rbp, rsp
push    rbx
sub     rsp, 38h
mov     [rbp+var_38], rdi
mov     [rbp+var_40], rsi
mov     rax, fs:28h
mov     [rbp+var_18], rax
xor     eax, eax
lea     rax, [rbp+var_25]
mov     rdi, rax
call    __ZNSt11__ZNSaIcE1Ev ; std::allocator<char>::allocator(void)
lea     rdx, [rbp+var_25]
mov     rax, [rbp+var_38]
lea     rsi, unk_14f9
mov     rdi, rax
call    __ZNSt7__cxx112basic_stringIcSt11char_traitsIcESaIcEEE6RKKS3_ ; std::__cxx11::basic_s
lea     rax, [rbp+var_25]
mov     rdi, rax
call    __ZNSt11__ZNSaIcE1Ev ; std::allocator<char>::~allocator()
mov     rax, [rbp+var_40]
mov     rdi, rax
call    __ZNKSt7__cxx112basic_stringIcSt11char_traitsIcESaIcEEE6lengthEv ; std::__cxx11::basic_s
mov     [rbp+var_20], eax
mov     [rbp+var_24], 0

```

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    _ZNKSt7__cxx112basic_stringIcSt11char_traitsIcESaIcEE1xEa ; 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     ecx, eax
mov     eax, ecx
shl     eax, 8
sub     ecx, eax
mov     ecx, eax
mov     eax, [rbp+var_1C]
add     ecx, eax
movsxd  rax, ecx
imul    rax, 1B4E81B5h
shl     rax, 20h
mov     ecx, eax
sar     ecx, 3
mov     eax, ecx
sar     eax, 1Fh
sub     ecx, eax
mov     ecx, eax
imul    eax, 4Bh
sub     ecx, eax
mov     ecx, eax
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    _ZNKSt7__cxx112basic_stringIcSt11char_traitsIcESaIcEE6lengthEv ; 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__cxx112basic_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::t
    (&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_h0h);
```

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_ < stack0xffffffffffffd8) {
        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) * stack0xffffffffffffd8 * 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  $\leq i$

        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  $\text{len}-1$  ( $\text{len}$ ="Длина логина")

$S$  -  $i$ -й символ логина

    Результат[ $i$ ] =  $((i + 0x11) * \text{len} * 0xff + (\text{int}32_t)S) \% 0x4b + 0x30$ ;

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:

```

nick@nick-virtual-machine: ~/Desktop/Disasn/Task_1
nick@nick-virtual-machine:~/Desktop/Disasn/Task_1$ ./keygen 012
Only a-Z!
nick@nick-virtual-machine:~/Desktop/Disasn/Task_1$ ./keygen Hello
xJQQT
nick@nick-virtual-machine:~/Desktop/Disasn/Task_1$ ./keygen password
ssIXkr9:
nick@nick-virtual-machine:~/Desktop/Disasn/Task_1$

nick@nick-virtual-machine:~/Desktop/Disasn/Task_1$ ./1
Enter login: 012
строка содержит цифру или другой неподходящий символ
nick@nick-virtual-machine:~/Desktop/Disasn/Task_1$ ./1
Enter login: Hello
Enter password: xJQQT
Access granted
nick@nick-virtual-machine:~/Desktop/Disasn/Task_1$ ./1
Enter login: Hello
Enter password: xJQQt
Access denied
nick@nick-virtual-machine:~/Desktop/Disasn/Task_1$ ./1
Enter login: password
Enter password: ssIXkr9
Access denied
nick@nick-virtual-machine:~/Desktop/Disasn/Task_1$ ./1
Enter login: password
Enter password: ssIXkr9:
Access granted
nick@nick-virtual-machine:~/Desktop/Disasn/Task_1$

```