

**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 2)  
по дисциплине «Защита программ и данных»**

**Выполнил:  
студент группы СКБ161  
Воинов Н. В.**

**МОСКВА 2020**

## 1 Задание

Дизассемблировать 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 дизассемблирование. В данном случае этого метода хватило для восстановления алгоритма.

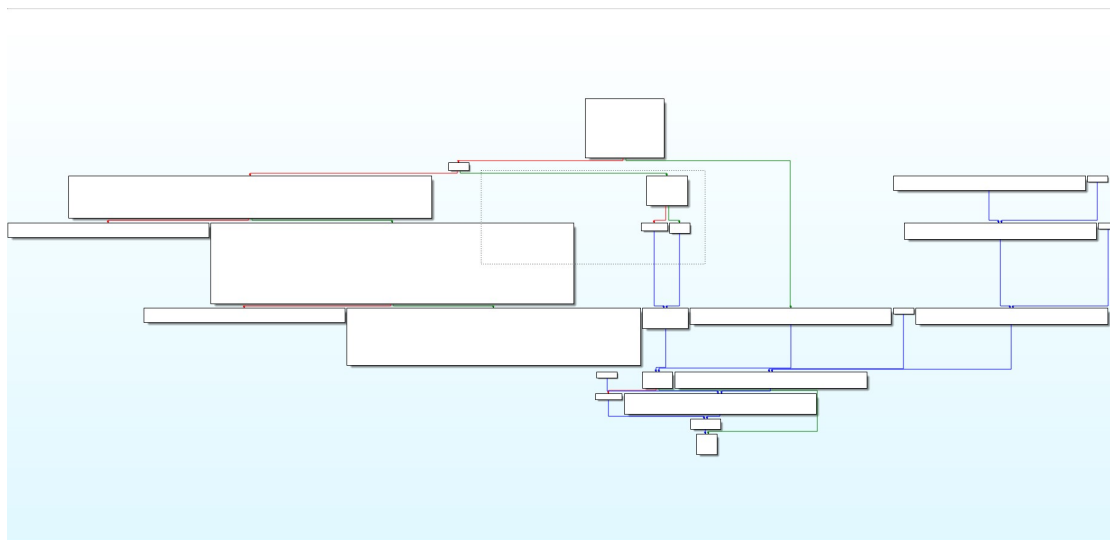
## 3 Используемые программы

Для проведения анализа была использованы:

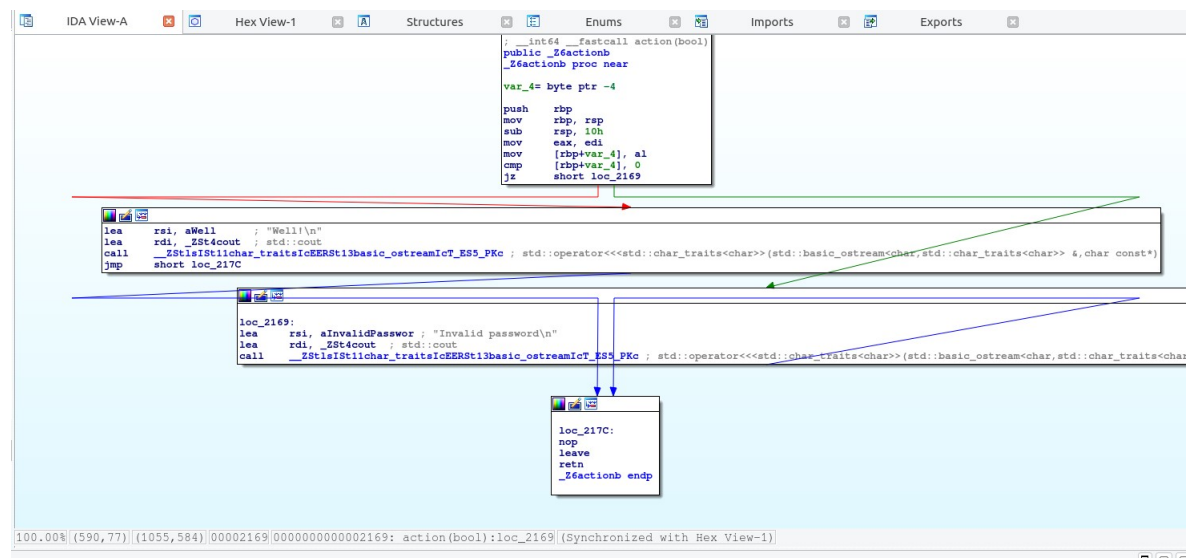
- IDA.
- Cutter.

## 4 Ход работы

Анализ начал с дизассемблирования исходного кода с помощью IDA. Рассмотрел граф вызовов.



Из построенного графа и кода видно, что почти в самом начале происходит форк процесса. Необходимо рассматривать отдельно родителя, отдельно потомка. Бегло просмотрев вызываемый код потомка, увидел из “интересных” вызовов и функций - вызов `action(bool)`. Она по полученному аргументу выводит сообщение об успехе/неудаче.



Перешел к рассмотрению родителя. В этом процессе происходят операции ввода/вывода и два “интересных” вызова - `get_secret` и `cmp`

```

mov     rdi, rax
call    _Z10get_secretNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEE ;
lea     rdx, [rbp+var_A0]
lea     rax, [rbp+var_40]
mov     rsi, rdx
mov     rdi, rax
call    _ZNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEC2ERKS4_ ; std::
lea     rdx, [rbp+var_60]
lea     rax, [rbp+var_40]
mov     rsi, rdx
mov     rdi, rax
call    _Z3cmpNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEES4_ ; cmp(
movzx   eax, al
mov     edi, eax ; status
call    _exit

```

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

1. Умножение на `getpid`
2. Деление на `geteuid`
3. Циклический сдвиг (`rot`)

Далее произвел декомпилирование с помощью Cutter.

```

// get_secret(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >)
canary = *(int64_t *) (in_FS_OFFSET + 0x28);
var_1dh = 0;
var_19h = 0;
var_28h = arg2;
var_38h = std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >::begin()(arg2);
var_30h = std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >::end()(var_28h);
while( true ) {
    cVar1 =
        bool __gnu_cxx::operator!=(const char*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > >)(__gnu_cxx::
            ((int64_t)&var_38h, (int64_t)&var_30h));
    if (cVar1 == '\0') break;
    pcVar4 = (char *)
        __gnu_cxx::__normal_iterator<char*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > >::
            ((int64_t)&var_30h));
    cVar1 = *pcVar4;
    iVar2 = getpid();
    uVar3 = geteuid();
    uVar3 = var_1dh + (uint32_t)(iVar2 * (cVar1 + -0x21)) / uVar3;
    var_1dh = uVar3 - (uVar3 * 0x2000 | uVar3 >> 0x13);
    __gnu_cxx::__normal_iterator<char*, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > >::operator++()
        ((int64_t)&var_30h);
}
var_1dh = 0x41482214;
std::allocator<char>::allocator()(&var_30h);

```

Тут показан восстановленный алгоритм, однако есть важное замечание. Видимо, при лексической обфускации я, случайно, вместо оператора “|” оставил только “=” и в этом случае, очень неприятном, пароль генерируется для всех значений логина одинаковый - 1095246356.

Обнаружил это я в тот момент, когда не увидел в дизассемблированном коде каких-либо логических операций. А вместо этого - просто присваивание этого значения.



```

loc_200C:
lea     rax, [rbp+var_1D]
mov     dword_ptr [rax], 41482214h

```

## 5 Восстановленный алгоритм

Так как в написанной программе допущена ошибка, алгоритм вычисления пароля - получение константного значения 1095246356. А все остальные действия - не имеют смысла.

Естественно, алгоритм без этой обидной ошибки был посложнее.

## 6 Выводы

1. Лексически анализировать исходный код без изменений сложно. Но, можно воспользоваться средствами, которые улучшат читаемость, раскрыть макрос. В таком случае, код анализировать становится очень легко.
2. Если под “защитыми” переменными подразумеваются явно заданные значения в коде - то они могут располагаться в качестве параметров явно, как в случае с 41482214h, или, например, как строки, которые находятся в сегменте данных только для чтения

```

LOAD:0000000000003DA9
.rodata:0000000000003DB0 ; =====
.rodata:0000000000003DB0 Segment type: DATA data
.rodata:0000000000003DB0 Segment permissions: Read
.rodata:0000000000003DB0 Segment alignment 'qword' can not be represented in assembly
.rodata:0000000000003DB0 _rodata segment para public 'CONST' use64
.rodata:0000000000003DB0 assume cs:_rodata
.rodata:0000000000003DB0 ;org 3DB0h
.rodata:0000000000003DB0 public _IO_stdin_used
.rodata:0000000000003DB0 db 1
.rodata:0000000000003DB1 db 0
.rodata:0000000000003DB2 db 2
.rodata:0000000000003DB3 db 0
.rodata:0000000000003DB4 unk_3DB4 db 0AEh ; DATA XREF: std:;__detail:;__from_cha;
.rodata:0000000000003DB5 db 0E7h ; std:;__detail:;__from_chars_alpha_to;
.rodata:0000000000003DB6 db 0FFh
.rodata:0000000000003DB7 db 0FFh

.rodata:0000000000003E2F db 0
.rodata:0000000000003E2F db 0
.rodata:0000000000003E30 sWell db 'Well',0Ah,0 ; DATA XREF: action(bool)+13*o
.rodata:0000000000003E31 aInvalidPassword db 'Invalid password',0Ah,0 ; DATA XREF: action(bool):loc_2169*o
.rodata:0000000000003E32 aInputLogin db 'Input login:',0Ah,0 ; DATA XREF: main+40*o
.rodata:0000000000003E33 aErrorTryAgain db 'Error! Try again',0Ah,0 ; DATA XREF: main+82*o
.rodata:0000000000003E34 aInputPassword db 'Input password: ',0
.rodata:0000000000003E35 asc_3EEB db ', ',0 ; DATA XREF: main+AE*o
.rodata:0000000000003E36 asc_3EEE db ' ',0Ah,0 ; DATA XREF: main+D3*o
.rodata:0000000000003E37 aError db 'Error!',0Ah,0 ; DATA XREF: main+132*o
.rodata:0000000000003E38 aForkError db 'Fork error!',0 ; DATA XREF: main:loc_232A*o
.rodata:0000000000003E39 align 8
.rodata:0000000000003E3A ; char aBasicStringMCr[]
.rodata:0000000000003E3B db 'basic_string::_M_construct null not valid',0
.rodata:0000000000003E3C ; DATA XREF: std:;__cxx11::basic_string<
.rodata:0000000000003E3D ; std:;__cxx11::basic_string<char,std:ch
.rodata:0000000000003E3E aBasicStringMCr db 'basic_string::_M_create',0
.rodata:0000000000003E3F ; DATA XREF: std:;__cxx11::basic_string<
.rodata:0000000000003E40 _rodata ends
.oexcpt_table:0000000000003F4A : =====

```

3. Анализировать программу лично мне было очень легко, я знал где и что нужно искать. Также я не думаю, что анализ этой программы вызовет большие сложности. IDA достаточно подробно строит граф, который явно показывает структуру. Но, в любом случае, придется потратить время, так как в итоге (хотя я этого не хотел) поиск сводится к нахождению лишь одного значения среди всего кода.