

Міністерство освіти і науки України
Національний технічний університет «ХПІ»
Навчально-науковий інститут комп'ютерних наук та інформаційних
технологій
Кафедра комп'ютерної інженерії та програмування

ЗВІТ

з лабораторної роботи № 5
з дисципліни «Сучасні технології безпечного програмування»
«[WINDOWS] ДОСЛІДЖЕННЯ ДИНАМІЧНИХ БІБЛІОТЕК»

Виконав:
студент гр. КН-Н9226
Кулик Д.І.

Перевірів:
Бульба С. С.

Харків – 2022

Мета роботи: Отримати навички основ реверс-інженірингу "на практиці".

Індивідуальне завдання

1. Визначити мову програмування, на якій була написана бібліотека (C/C++, C#, Delphi, Java). Визначення мови програмування дозволить найбільш ефективно використовувати "декомпілятор". При цьому, рекомендуються наступні декомпілятори:
 - для Java - jdgui, або нативний декомпілятор від IntelliJ IDEA
 - для C# - dotPeek
 - для Delphi - DeDe
 - для C/C++ - Ghidra, IDA Pro
2. Визначити функції та їх прототипи, з яких складається динамічна бібліотека.
3. Створити додаток, що підключає дану бібліотеку та визначити, що роблять функції getIV, getK.
4. Декомпілювати функцію CRC_16_IBM. Судячи з її назви Вам буде не важко це зробити (бо її алгоритм завідомо відомий), але треба бути підібрати кректні коефіцієнти. Необхідно довести коректність реалізованого алгоритму через порівняння результатів з результатами роботи функції динамічної бібліотеки.
5. Декомпілювати та переписати функції dec та enc. Який алгоритм вони використовують? Підказка - це скорочення від encode, decode. Більшість алгоритмів шифрування використовують табличні дані, на базі яких використовується кодування. Знавши це - найбільш швидкий для вас варіант - визначити таблицю, що використовується, та знайти алгоритм, що її використовує. Реалізувати алгоритм на мові високого рівня, та довести коректність реалізованого алгоритму через порівняння результатів з результатами роботи функції динамічної бібліотеки.

Хід роботи

Встановлено Ghidra для декомпіляції бібліотеки

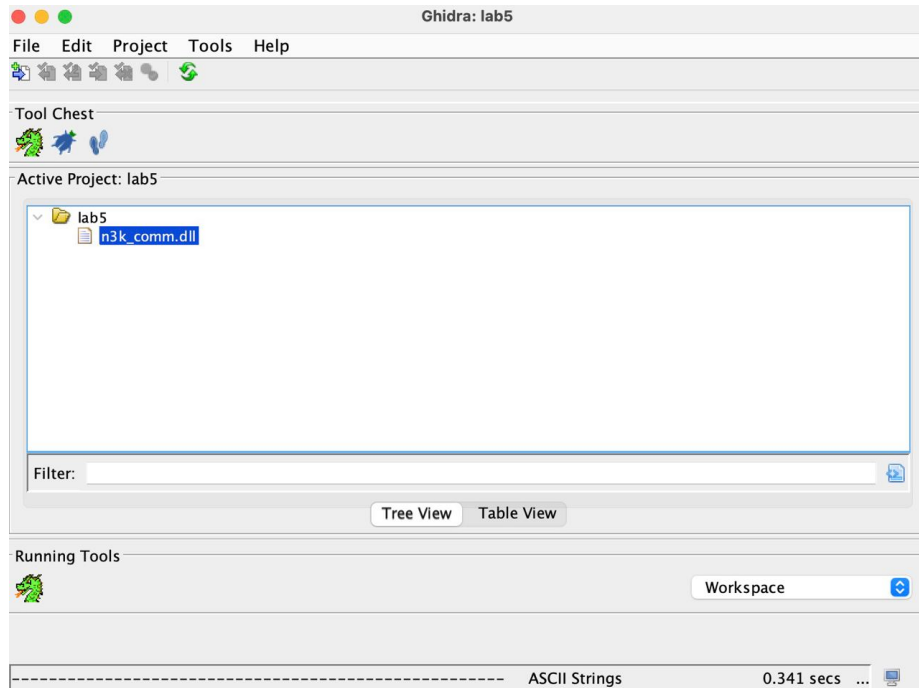


Рисунок 1 – Ghidra



Рисунок 2 – Знайдені функції бібліотеки

```
Decompile: CRC_16_IBM - (n3k_comm.dll)
1
2 int __cdecl CRC_16_IBM(int param_1,int param_2)
3
4 {
5     uint uVar1;
6     int iVar2;
7
8     /* 0x21d0 1 CRC_16_IBM */
9     uVar1 = 0;
10    iVar2 = 0;
11    if (0 < param_1) {
12        do {
13            uVar1 = uVar1 ^ *(byte *)(iVar2 + param_2);
14            if ((uVar1 & 1) != 0) {
15                uVar1 = uVar1 ^ 0x14002;
16            }
17            uVar1 = uVar1 >> 1;
18            if ((uVar1 & 1) != 0) {
19                uVar1 = uVar1 ^ 0x14002;
20            }
21            uVar1 = uVar1 >> 1;
22            if ((uVar1 & 1) != 0) {
23                uVar1 = uVar1 ^ 0x14002;
24            }
25            uVar1 = uVar1 >> 1;
26            if ((uVar1 & 1) != 0) {
27                uVar1 = uVar1 ^ 0x14002;
28            }
29            uVar1 = uVar1 >> 1;
30            if ((uVar1 & 1) != 0) {
31                uVar1 = uVar1 ^ 0x14002;
32            }
33            uVar1 = uVar1 >> 1;
34            if ((uVar1 & 1) != 0) {
35                uVar1 = uVar1 ^ 0x14002;
36            }
37            uVar1 = uVar1 >> 1;
```

Рисунок 3 – Декомпільована функція CRC_16_IBM

```
undefined4 __cdecl getIV(int param_1)
{
    char *pcVar1;
    char cVar2;
    int iVar3;

    /* 0x2420 6 getIV */
    iVar3 = 0;
    do {
        pcVar1 = (char *)(iVar3 + param_1);
        cVar2 = (char) iVar3;
        iVar3 = iVar3 + 1;
        *pcVar1 = (pcVar1[(int)&DAT_1000b040 - param_1] - cVar2) + -1;
    } while (iVar3 < 0x10);
    return 1;
}
```

Рисунок 4 – Декомпільована функція getIV

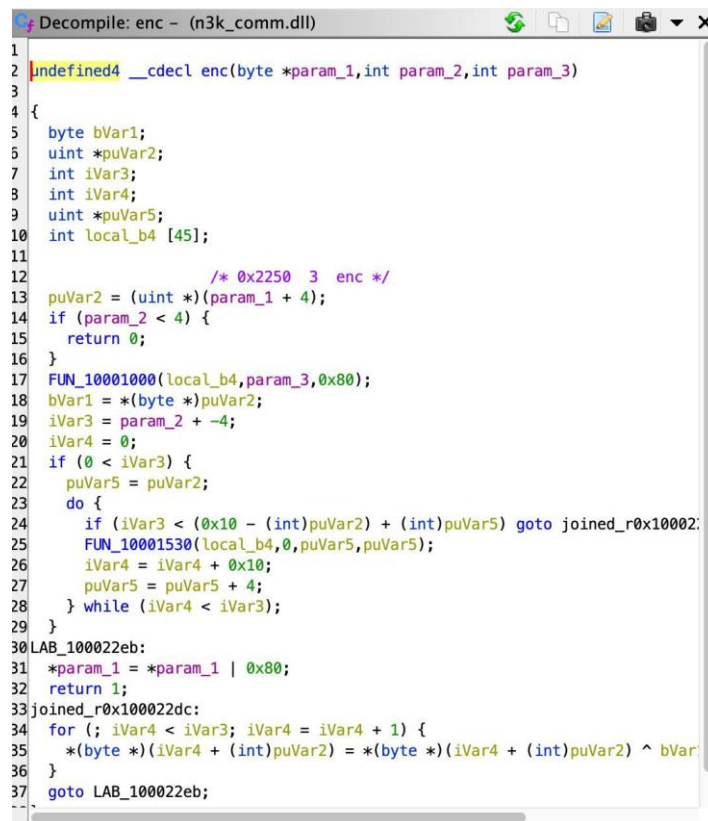
```

undefined4 __cdecl getK(int param_1)
{
    char *pcVar1;
    char cVar2;
    int iVar3;

    /* 0x23f0 8 getK */
    iVar3 = 0;
    do {
        pcVar1 = (char *)(iVar3 + param_1);
        cVar2 = (char)iVar3;
        iVar3 = iVar3 + 1;
        *pcVar1 = pcVar1[(int)&DAT_1000b030 - param_1] + cVar2 + '\x01';
    } while (iVar3 < 0x10);
    return 1;
}

```

Рисунок 5 – Декомпільована функція getK



```

1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
1  undefined4 __cdecl enc(byte *param_1,int param_2,int param_3)
2  {
3  {
4  {
5      byte bVar1;
6      uint *puVar2;
7      int iVar3;
8      int iVar4;
9      uint *puVar5;
10     int local_b4 [45];
11
12     /* 0x2250 3 enc */
13     puVar2 = (uint *)(param_1 + 4);
14     if (param_2 < 4) {
15         return 0;
16     }
17     FUN_10001000(local_b4,param_3,0x80);
18     bVar1 = *(byte *)puVar2;
19     iVar3 = param_2 + -4;
20     iVar4 = 0;
21     if (0 < iVar3) {
22         puVar5 = puVar2;
23         do {
24             if (iVar3 < (0x10 - (int)puVar2) + (int)puVar5) goto joined_r0x100022dc;
25             FUN_10001530(local_b4,0,puVar5,puVar5);
26             iVar4 = iVar4 + 0x10;
27             puVar5 = puVar5 + 4;
28         } while (iVar4 < iVar3);
29     }
30 LAB_100022eb:
31     *param_1 = *param_1 | 0x80;
32     return 1;
33 joined_r0x100022dc:
34     for (; iVar4 < iVar3; iVar4 = iVar4 + 1) {
35         *(byte *)(iVar4 + (int)puVar2) = *(byte *)(iVar4 + (int)puVar2) ^ bVar1;
36     }
37     goto LAB_100022eb;

```

Рисунок 6 – Декомпільована функція enc

```

Decompile: dec - (n3k_comm.dll)
1
2 undefined4 __cdecl dec(byte *param_1,int param_2,int param_3)
3
4 {
5     int iVar1;
6     uint *puVar2;
7     int iVar3;
8     uint *puVar4;
9     int local_b4 [45];
10
11     /* 0x2310 2 dec */
12     if (-1 < (char)*param_1) {
13         return 0;
14     }
15     FUN_10001270(local_b4,param_3,0x80);
16     puVar2 = (uint *) (param_1 + 4);
17     if (param_2 < 4) {
18         return 0;
19     }
20     iVar1 = param_2 + -4;
21     iVar3 = 0;
22     if (0 < iVar1) {
23         puVar4 = puVar2;
24         do {
25             if (iVar1 < (0x10 - (int)puVar2) + (int)puVar4) goto joined_r0x100023a6;
26             FUN_10001530(local_b4,1,puVar4,puVar4);
27             iVar3 = iVar3 + 0x10;
28             puVar4 = puVar4 + 4;
29         } while (iVar3 < iVar1);
30     }
31 LAB_100023b9:
32     *param_1 = *param_1 & 0x7f;
33     return 1;
34 joined_r0x100023a6:
35     for (; iVar3 < iVar1; iVar3 = iVar3 + 1) {
36         *(byte *) (iVar3 + (int)puVar2) = *(byte *) (iVar3 + (int)puVar2) ^ *(byte *) (iVar3 + (int)puVar2);
37     }
38 }

```

Рисунок 7 – Декомпільована функція dec

```

*****
*                               FUNCTION                               *
*****
uint __cdecl CRC_16_IBM(int param_1, int param_2)
    assume FS_OFFSET = 0xffdf000
    EAX:4    <RETURN>
    Stack[0x4]:4  param_1
    Stack[0x8]:4  param_2
    0x21d0 1  CRC_16_IBM
    Ordinal_1
    CRC_16_IBM
    PUSH     ESI
    100021d0 56
    XREF[1]: 100021d1(R)
    XREF[1]: 100021dd(R)
    XREF[2]: Entry Point(*), 1000a

```

Рисунок 8 – Адреса функції CRC_16_IBM

```

int main()
{
    auto hdl = LoadLibraryA("n3k_comm.dll");
    if (hdl)
    {
        auto crc16Func = reinterpret_cast<CRC_16_IBM>(GetProcAddress(hdl, "CRC_16_IBM"));
        if (crc16Func) {
            auto dllResult = crc16Func(16, 0x1000b040);
            auto myResult = MY_CRC_16_IBM(16, 0x1000b040);
            std::cout << "DLL CRC_16_IBM result = " << dllResult << std::endl;
            std::cout << "My CRC_16_IBM result = " << myResult << std::endl;
        }
        else {
            std::cout << "Function CRC_16_IBM not found!" << std::endl;
        }

        auto getIVFunc = reinterpret_cast<getIV>(GetProcAddress(hdl, "getIV"));
        if (getIVFunc) {
            char ivStr[17] = { 0 };
            auto r = getIVFunc(ivStr);
            char ivStr2[17] = { 0 };
            auto r2 = My_getIV(ivStr2);
            std::cout << "DLL getIV result = " << ivStr << std::endl;
            std::cout << "My getIV result = " << ivStr2 << std::endl;
        }
        else {
            std::cout << "Function getIV not found!" << std::endl;
        }

        auto getKFunc = reinterpret_cast<getK>(GetProcAddress(hdl, "getK"));
        if (getKFunc) {
            char kStr[17] = { 0 };
            auto r = getKFunc(kStr);
            std::cout << "Dll getK result = " << kStr << std::endl;
        }
        else {
            std::cout << "Function getK not found!" << std::endl;
        }

        auto encFunc = reinterpret_cast<enc>(GetProcAddress(hdl, "enc"));
        auto decFunc = reinterpret_cast<dec>(GetProcAddress(hdl, "dec"));
        if (encFunc && decFunc) {
            byte str[16] = { 'H', 'E', 'L', 'L', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0', '\0' };
            int key = 0x1000b040;
            auto r = encFunc(str, 16, key);
            std::cout << "DLL enc result = " << str << std::endl;
            auto r2 = decFunc(str, 16, key);
            std::cout << "DLL dec result = " << str << std::endl;
        }
        else {
            std::cout << "Function enc and dec not found!" << std::endl;
        }

        FreeLibrary(hdl);
    }
    else {
        std::cout << "Library not found!" << std::endl;
    }
}

```

Рисунок 9 – Код мовою C++ для виклику функції з dll

```

unsigned int MY_CRC_16_IBM(int len, int address)
{
    unsigned int result = 0;

    for (int i = 0; i < len; i++)
    {
        result ^= *(byte*)(i + address);
        for (int j = 0; j < 8; j++)
        {
            if ((result & 1) != 0)
                result ^= 0x14002;
            result >>= 1;
        }
    }

    return result;
}

```

Рисунок 10 – Реалізована функція CRC_16_IBM

```

DLL CRC_16_IBM result = 31352
My CRC_16_IBM result = 31352
DLL getIV result = Wiegand1997csncl
My getIV result = Wiegand1997csncl
Dll getK result = WGMMASTERWgmaster
DLL enc result = LELL
DLL dec result = HELL

```

Рисунок 11 – Результат виконання програми

Функції дес та енс викликають підпрограми що мають вбудовані таблиці даних. Таблиця нижче співпадає з таблицею AES S-Box алгоритму AES розробленого у 2 лабораторній роботі.

Listing: n3k_comm.dll

Address	Disassembly	Comment
10008110 63	??	63h c
10008111 7c	??	7Ch
10008112 77	??	77h w
10008113 7b	??	7bh {
10008114 f2	??	F2h
10008115 6b	??	6Bh k
10008116 6f	??	6Fh o
10008117 c5	??	C5h
10008118 30	??	30h 0
10008119 01	??	01h
1000811a 67	??	67h g
1000811b 2b	??	2Bh +
1000811c fe	??	FEh
1000811d d7	??	D7h
1000811e ab	??	ABh
1000811f 76	??	76h v
10008120 ca	??	CAh
10008121 82	??	82h
10008122 c9	??	C9h
10008123 7d	??	7Dh }
10008124 fa	??	FAh
10008125 59	??	59h Y
10008126 47	??	47h G
10008127 f0	??	F0h
10008128 ad	??	ADh
10008129 04	??	04h
1000812a a2	??	A2h
1000812b af	??	AFh
1000812c 9c	??	9Ch
1000812d a4	??	A4h
1000812e 72	??	72h r
1000812f c0	??	C0h

Decompile: FUN_10001000 - (n3k_comm.dll)

```

11  if (param_3 == 0x80) {
12      *param_1 = 10;
13  }
14  }
15  else if (param_3 == 0xc0) {
16      *param_1 = 0xc0;
17  }
18  else {
19      if (param_3 != 0x100) {
20          return;
21      }
22      *param_1 = 0xe;
23  }
24  iVar4 = param_3 >> 5;
25  puVar1 = (uint *) (param_1 + 1);
26  if (0 < iVar4) {
27      puVar2 = puVar1;
28      do {
29          *puVar2 = *(uint *) ((param_2 - (int) puVar1) + (int) puVar2);
30          puVar2 = puVar2 + 1;
31          iVar4 = iVar4 - 1;
32      } while (iVar4 != 0);
33  }
34  if (*param_1 == 10) {
35      puVar2 = &DAT_10008110;
36      do {
37          uVar3 = puVar2[-1] ^
38              CONCAT31(CONCAT21(CONCAT11((&DAT_10008110)[puVar1[3] & 0xff],
39              (&DAT_10008110)[*(byte *) ((int) puVar1 + 0)],
40              (&DAT_10008110)[*(byte *) ((int) puVar1 + 0)] ^ *
41              (&DAT_10008110)[*(byte *) ((int) puVar1 + 0xd)])) ^ *
42              uVar5 = puVar1[1] ^ uVar3;
43              uVar5 = puVar1[2] ^ uVar6;
44              puVar1[4] = uVar3;
45              puVar1[6] = uVar5;
46              uVar5 = uVar5 ^ puVar1[3];
47              puVar1[7] = uVar5;

```



```

"""таблиця констант, що отримана у вигляді перетворень поля GF(2^8)"""
s_box = (
    0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,
    0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,
    0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,
    0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,
    0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,
    0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,
    0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,
    0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,
    0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,
    0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB,
    0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79,
    0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,
    0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,
    0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E,
    0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF,
    0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16,
)

```

Рисунок 12 – Таблиця S-Box для AES алгоритму

Висновки: в результаті виконання лабораторної роботи було отримано навичок основ реверс-інженірингу "на практиці".