

Лабораторна робота №5. Windows Дослідження динамічних бібліотек

Мета : отримати навички основ реверс-інженерінгу “на практиці”

**Завдання**

Використовуючи [бібліотеку зовнішнього вендора](#), виконати наступні дії:

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 крєктні коефіцієнти. Необхідно довести корєктність реалізованого алгоритму через порівняння результатів з результатами роботи функції динамічної бібліотеки.
5. Декомпілювати та переписати функції dec та enc. Який алгоритм вони використовують? Підсказка - 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 корєктність реалізованого алгоритму через

порівняння результатів з результатами роботи функції динамічної бібліотеки.

### Хід роботи

Після завантаження бібліотеки спробуємо дослідити її за допомогою програми Detect it Easy.

Бачимо, що компіляція здійснювалась Microsoft Visual C/C++ 2003. Тобто, мова програмування саме C/C++.

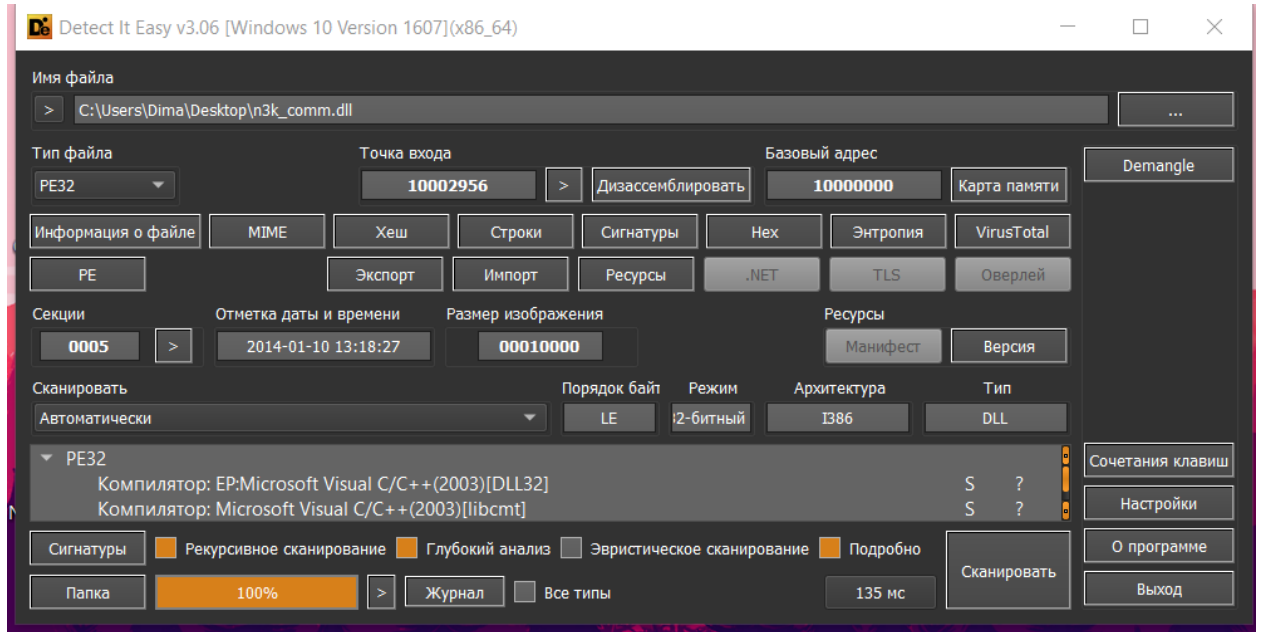


Рис. 1 – Дослідження бібліотеки за допомогою DIE

Далі визначимо функції та її прототипи через CodeBrowser.



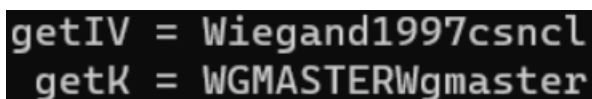
Рис. 2 – Дослідження публічних функцій.

Підключаємо бібліотеку. Нижче бачимо декомпільований код функції getIV

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

Перепишемо його у більш лаконічний та практичний вигляд.

```
int getIV(int param_1)
{
    char* dest;
    for (int i = 0; i < 16; i++)
    {
        dest = (char*)(i + param_1);
        *dest = (dest[(int)&DAT_1000b040 - param_1] - i) - 1;
    }
    return 1;
}
```



```
getIV = Wiegand1997csncl
getK = WGMASrERWgmaster
```

Рис. 3 – Результат виконання функції.

Функція getIV копіює 16 байт з пам'яті даних бібліотеки з адресою 0x1000b040 у область пам'яті за змінною param\_1.

Нижче представлена декомпіляція функції CRC\_16\_IBM

```
uint __cdecl CRC_16_IBM(int param_1,int param_2)
{
    uint uVar1 = 0;
    int iVar2 = 0;
    if (0 < param_1) {
        do {
            uVar1 = uVar1 ^ *(byte *)(iVar2 + param_2);
            if ((uVar1 & 1) != 0) {
                uVar1 = uVar1 ^ 0x14002;
            }
            uVar1 = uVar1 >> 1;
            if ((uVar1 & 1) != 0) {
                uVar1 = uVar1 ^ 0x14002;
            }
            uVar1 = uVar1 >> 1;
            if ((uVar1 & 1) != 0) {
                uVar1 = uVar1 ^ 0x14002;
            }
            uVar1 = uVar1 >> 1;
        } while (iVar2 < param_1);
    }
    return uVar1;
}
```

```

    if ((uVar1 & 1) != 0) {
        uVar1 = uVar1 ^ 0x14002;
    }
    uVar1 = uVar1 >> 1;
    if ((uVar1 & 1) != 0) {
        uVar1 = uVar1 ^ 0x14002;
    }
    uVar1 = uVar1 >> 1;
    if ((uVar1 & 1) != 0) {
        uVar1 = uVar1 ^ 0x14002;
    }
    uVar1 = uVar1 >> 1;
    if ((uVar1 & 1) != 0) {
        uVar1 = uVar1 ^ 0x14002;
    }
    uVar1 = uVar1 >> 1;
    if ((uVar1 & 1) != 0) {
        uVar1 = uVar1 ^ 0x14002;
    }
    uVar1 = uVar1 >> 1;
    iVar2 = iVar2 + 1;
} while (iVar2 < param_1);
}
return uVar1;
}

```

## Декомпільовані функції dec та inc :

```

int __cdecl enc(byte *param_1,int param_2,int param_3)
{
    byte bVar1;
    uint *puVar2;
    int iVar3;
    int iVar4;
    uint *puVar5;
    int local_b4 [45];

    puVar2 = (uint *)(param_1 + 4);
    if (param_2 < 4) {
        return 0;
    }
    FUN_10001000(local_b4,param_3,0x80);
    bVar1 = *(byte *)puVar2;
    iVar3 = param_2 + -4;
    iVar4 = 0;
    if (0 < iVar3) {
        puVar5 = puVar2;
        do {
            if (iVar3 < (0x10 - (int)puVar2) + (int)puVar5) goto joined_r0x100022dc;
            FUN_10001530(local_b4,0,puVar5,puVar5);
            iVar4 = iVar4 + 0x10;
            puVar5 = puVar5 + 4;
        } while (iVar4 < iVar3);
    }
LAB_100022eb:
    *param_1 = *param_1 | 0x80;
    return 1;
joined_r0x100022dc:
    for (; iVar4 < iVar3; iVar4 = iVar4 + 1) {
        *(byte *)(iVar4 + (int)puVar2) = *(byte *)(iVar4 + (int)puVar2) ^ bVar1;
    }
}

```

```

    }
    goto LAB_100022eb;
}
int __cdecl dec(byte *param_1,int param_2,int param_3)
{
    int iVar1;
    uint *puVar2;
    int iVar3;
    uint *puVar4;
    int local_b4 [45];
    if (-1 < (char)*param_1) {
        return 0;
    }
    FUN_10001270(local_b4,param_3,0x80);
    puVar2 = (uint *)(param_1 + 4);
    if (param_2 < 4) {
        return 0;
    }
    iVar1 = param_2 + -4;
    iVar3 = 0;
    if (0 < iVar1) {
        puVar4 = puVar2;
        do {
            if (iVar1 < (0x10 - (int)puVar2) + (int)puVar4) goto joined_r0x100023a6;
            FUN_10001530(local_b4,1,puVar4,puVar4);
            iVar3 = iVar3 + 0x10;
            puVar4 = puVar4 + 4;
        } while (iVar3 < iVar1);
    }
LAB_100023b9:
    *param_1 = *param_1 & 0x7f;
    return 1;
joined_r0x100023a6:
    for (; iVar3 < iVar1; iVar3 = iVar3 + 1) {
        *(byte *)(iVar3 + (int)puVar2) = *(byte *)(iVar3 + (int)puVar2) ^ *(byte
*)puVar2;
    }
    goto LAB_100023b9;
}

```

Напевно, судячи з їх назви, це функції шифрування та дешифрування.

Роздивимось декілька з викликів підпрограм.

10008110	63	??	63h	36	puVar2 = &DAT_100000d4;
10008111	7c	??	7Ch	37	do {
10008112	77	??	77h	38	uVar3 = puVar2[-1] ^
10008113	7b	??	7Bh	39	CONCAT31(CONCAT21(CONCAT11((&DAT_10008110)[puVar1[3] & 0xff],
10008114	f2	??	F2h	40	(&DAT_10008110)[*(byte *)((int)puVar1 + 0xf)]),
10008115	6b	??	6Bh	41	(&DAT_10008110)[*(byte *)((int)puVar1 + 0xe)]),
10008116	6f	??	6Fh	42	(&DAT_10008110)[*(byte *)((int)puVar1 + 0xd)]) ^ *puVar1;
10008117	c5	??	C5h	43	uVar6 = puVar1[1] ^ uVar3;
					uVar5 = puVar1[2] ^ uVar6;

Рис. 4 – Виклики підпрограм.

Бачимо, що це співпадає зі значеннями таблиці.

AES S-box																
	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16
The column is determined by the least significant nibble, and the row by the most significant nibble. For example, the value $9a_{16}$ is converted into $b8_{16}$ .																

Рис. 5 – Таблиця AES s-box

Висновок : під час виконання лабораторної роботи я отримав навички основ реверс-інженерінгу “на практиці”.