# Báo cáo đồ án môn An toàn và an ninh mạng

Lớp học phần: INT3307E 1
Giảng viên: TS. Nguyễn Đại Thọ

Nguyễn Tường Hùng – 23020078

December 1, 2025

**Abstract**

Your abstract.

# I. Introduction

# II. Background

# III. Initial Reconnaissance

## 3.1. Glibc Version

## 3.2. Binary Mitigations

# IV. Pseudocode Review

## 4.1. `rune` Struct

## 4.2. `main()` Function

```
int __fastcall __noreturn main(int argc, const char **argv, const char **envp)
{
  int action; // [rsp+Ch] [rbp-4h]

  setup(argc, argv, envp);
  puts(
    "This is the ultimate test!\n"
    "Do you have what it takes to master the runes?\n"
    "Are you worthy of laying your eyes on the Pharaoh's tomb?\n"
    "Only your actions will tell...");
  while ( 1 )
  {
    while ( 1 )
    {
      puts("1. Create rune\n2. Delete rune\n3. Edit rune\n4. Show rune\nAction: ");
      action = read_int();
      if ( action != 4 )
        break;
      show();
    }
    if ( action > 4 )
    {
invalid_action:
      puts("Invalid action!");
    }
    else if ( action == 3 )
    {
      edit();
    }
    else
    {
      if ( action > 3 )
        goto invalid_action;
      if ( action == 1 )
      {
        create();
      }
      else
      {
        if ( action != 2 )
          goto invalid_action;
        delete();
      }
    }
  }
}
```

Tại hàm `main()`, chương trình bắt đầu với lời gọi hàm `setup()`, sau đó đi vào vòng lặp `while` cho phép người dùng lựa chọn 1 trong 4 hành động:

1. Tạo `rune`.
2. Xóa `rune`.
3. Chỉnh sửa `rune`.
4. Hiển thị `rune`.

Tuy nhiên không cung cấp lựa chọn nào để thoát khỏi chương trình.

### 4.3. `setup()` Function

```
int setup()
{
  rune **v0; // rax
  int i; // [rsp+Ch] [rbp-4h]

  setvbuf(stdin, 0, 2, 0);
  setvbuf(stdout, 0, 2, 0);
  LODWORD(v0) = setvbuf(stderr, 0, 2, 0);
  for ( i = 0; i <= 63; ++i )
  {
    v0 = MainTable;
    MainTable[i] = &items[i];
  }
  return (int)v0;
}
```

Qua quan sát mã giả của hàm `setup()`, các cấu trúc dữ liệu chính được xác định như sau:

- `rune items[64]` - Mảng chứa 64 phần tử với kiểu dữ liệu `rune`, mỗi phần tử có kích thước 24 bytes.
- `rune *MainTable[64]` - Mảng chứa 64 phần tử với kiểu dữ liệu là con trỏ `rune *`, ban đầu được khởi tạo trỏ đến các phần tử tương ứng trong mảng `items` theo thứ tự.

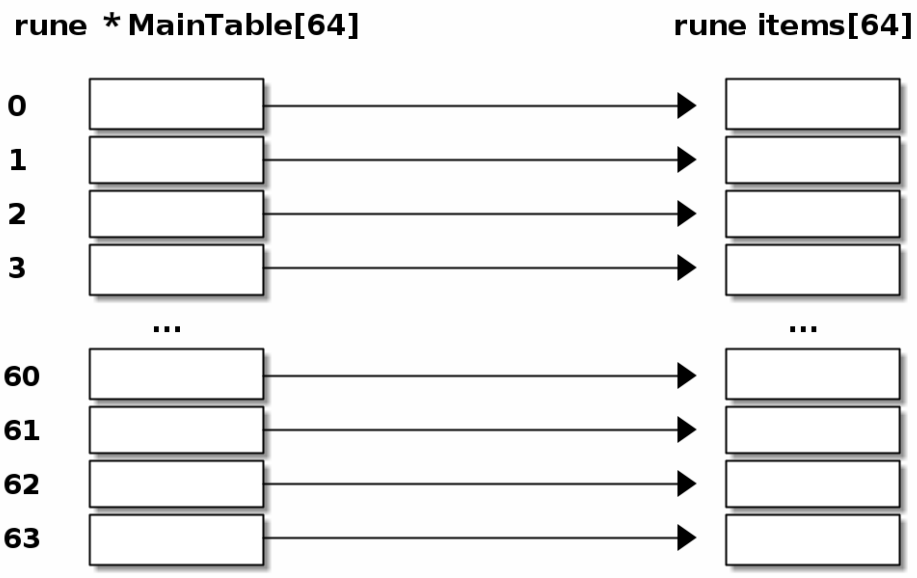


Figure 1: Hình ảnh minh họa

### 4.4. `read_int()` Function

```
int read_int()
{
  char buf[40]; // [rsp+0h] [rbp-30h] BYREF
  unsigned __int64 canary; // [rsp+28h] [rbp-8h]

  canary = __readfsqword(0x28u);
  memset(buf, 0, 32u);
  readline(buf, 31);
  return atoi(buf);
}
```

### 4.5. `create()` Function

```
unsigned __int64 create()
{
  int index; // [rsp+0h] [rbp-20h]
  unsigned int length; // [rsp+4h] [rbp-1Ch]
  char *content; // [rsp+8h] [rbp-18h]
  char name[8]; // [rsp+10h] [rbp-10h] BYREF
  unsigned __int64 canary; // [rsp+18h] [rbp-8h]

  canary = __readfsqword(0x28u);
  *(_QWORD *)name = 0;
  puts("Rune name: ");
  read(0, name, 8u);
  index = hash(name);
  if ( MainTable[(unsigned int)hash(name)]->content )
  {
    puts("That rune name is already in use!");
  }
  else
  {
    puts("Rune length: ");
    length = read_int();
    if ( length <= 0x60 )
    {
      content = (char *)malloc(length + 8);
      strcpy(MainTable[index]->name, name);
      MainTable[index]->content = content;
      MainTable[index]->length = length;
      strcpy(content, name);
      puts("Rune contents: ");
      read(0, content + 8, length);
    }
    else
    {
      puts("Max length is 0x60!");
    }
  }
  return __readfsqword(0x28u) ^ canary;
}
```

### 4.6. `hash()` Function

```
__int64 __fastcall hash(char *name)
{
  char ascii_sum; // [rsp+10h] [rbp-8h]
```

```
  int i; // [rsp+14h] [rbp-4h]

  ascii_sum = 0;
  for ( i = 0; i <= 7; ++i )
    ascii_sum += name[i];
  return ascii_sum & 0x3F; // sum & 0b111111
}
```

### 4.7. `delete()` Function

```
unsigned __int64 delete()
{
  int index; // [rsp+Ch] [rbp-14h]
  char name[8]; // [rsp+10h] [rbp-10h] BYREF
  unsigned __int64 canary; // [rsp+18h] [rbp-8h]

  canary = __readfsqword(0x28u);
  *(_QWORD *)name = 0;
  puts("Rune name: ");
  read(0, name, 8u);
  index = hash(name);
  if ( MainTable[index]->content )
  {
    free(MainTable[index]->content);
    memset(MainTable[index], 0, 20u);
    puts("Rune deleted successfully.");
  }
  else
  {
    puts("There's no rune with that name!");
  }
  return __readfsqword(0x28u) ^ canary;
}
```

### 4.8. `show()` Function

```
unsigned __int64 show()
{
  int index; // eax
  char name[8]; // [rsp+0h] [rbp-10h] BYREF
  unsigned __int64 canary; // [rsp+8h] [rbp-8h]

  canary = __readfsqword(0x28u);
  *(_QWORD *)name = 0;
  puts("Rune name: ");
  read(0, name, 8u);
  if ( MainTable[(unsigned int)hash(name)]->content )
  {
    puts("Rune contents:\n");
    index = hash(name);
    puts((const char *)MainTable[index]->content + 8);
  }
  else
  {
    puts("That rune doesn't exist!");
  }
  return __readfsqword(0x28u) ^ canary;
}
```

### 4.9. `edit()` Function

```
unsigned __int64 edit()
{
  int new_index; // eax MAPDST
  char **content_ptr; // rbx
  int old_index; // eax
  int current_index; // eax
  char *content; // [rsp+0h] [rbp-30h]
  char old_name[8]; // [rsp+8h] [rbp-28h] BYREF
  char new_name[8]; // [rsp+10h] [rbp-20h] BYREF
  unsigned __int64 canary; // [rsp+18h] [rbp-18h]

  canary = __readfsqword(0x28u);
  *(_QWORD *)old_name = 0;
  *(_QWORD *)new_name = 0;
  puts("Rune name: ");
  read(0, old_name, 8u);
  content = MainTable[(unsigned int)hash(old_name)]->content;
  if ( content )
  {
    puts("New name: ");
    read(0, new_name, 8u);
    if ( MainTable[(unsigned int)hash(new_name)]->content )
    {
      puts("That rune name is already in use!");
    }
    else
    {
      new_index = hash(new_name);
      strcpy(MainTable[new_index]->name, new_name);
      content_ptr = &MainTable[(unsigned int)hash(old_name)]->content;
      new_index = hash(new_name);
      memcpy(&MainTable[new_index]->content, content_ptr, 12u);
      strcpy(content, new_name);
      old_index = hash(old_name);
      memset(MainTable[old_index], 0, 20u);
      puts("Rune contents: ");
      current_index = hash(content);
      read(0, content + 8, MainTable[current_index]->length);
    }
  }
  else
  {
    puts("There's no rune with that name!");
  }
  return __readfsqword(0x28u) ^ canary;
}
```

# V. Exploitation

## 5.1. Leaking Heap Address

## 5.2. Leaking Libc Address

## 5.3. Remote Code Execution

# VI. Conclusion

# VII. References