Dreamhack just read memory

일단 파일을 ida에서 디컴파일해보았다.

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  _QWORD v4[8]; // [rsp+0h] [rbp-70h] BYREF
  char v5; // [rsp+40h] [rbp-30h]
  void *v6; // [rsp+48h] [rbp-28h]
  int j; // [rsp+54h] [rbp-1Ch]
  char *v8; // [rsp+58h] [rbp-18h]
  int v9; // [rsp+64h] [rbp-Ch]
  int v10; // [rsp+68h] [rbp-8h]
  int i; // [rsp+6Ch] [rbp-4h]

  memset(v4, 0, sizeof(v4));
  v5 = 0;
  printf("Input: ");
  __isoc99_scanf("%64s", v4);
  head_node = malloc(0x10uLL);
  *((_QWORD *)head_node + 1) = 0LL;
  *(_BYTE *)head_node = 0;
  tail_node = (__int64)head_node;
  for ( i = 0; i <= 63; ++i )
    append_list((unsigned int)i);
  v10 = 1;
  v9 = 0;
  v8 = (char *)*((_QWORD *)head_node + 1);
```

```
 while ( head_node )
 {
   *(_BYTE *)head_node = 0;
   v6 = (void *)*((_QWORD *)head_node + 1);
   *((_QWORD *)head_node + 1) = 0LL;
   free(head_node);
   head_node = v6;
 }
 tail_node = 0LL;
 if ( v10 )
 {
   puts("Correct");
   printf("flag is DH{");
   for ( j = 0; j <= 63; ++j )
     printf("%d", *((_BYTE *)v4 + j) % 0xAu);
   puts("}");
 }
 else
 {
   puts("Wrong");
 }
 return 0;
```

일단 main함수의 흐름을 분석했다.

1. 변수 초기화

v4[8]: 사용자 입력을 저장할 64바이트 크기의 배열.

v5, v6, v8, v9, v10 등은 각각 상태 플래그, 포인터, 인덱스, 비교 결과를 관리.

2. 입력 처리

scanf를 통해 최대 64바이트의 입력을 v4 배열에 저장.

입력 데이터는 문자열로 처리되며, 널 종료가 필요 없기 때문에 64바이트 제한을 넘어갈 경우 메모리 손상이 발생할 수 있다.

3. 연결 리스트 생성

16바이트 크기의 노드를 할당하여 연결 리스트의 헤드로 설정.

append_list((unsigned int)i) 호출을 통해 64개의 노드를 연결 리스트로 구성.

4. 비교 로직

사용자 입력(v4)과 연결 리스트에 저장된 데이터(v8)를 비교.

하나라도 다르면 v10 플래그를 0으로 설정하여 실패로 간주.

v8 포인터를 이동하며 리스트의 다음 노드로 진행.

5. 연결 리스트 메모리 해제

생성된 연결 리스트를 순회하며 각 노드를 해제.

메모리 누수를 방지하며, head_node와 tail_node를 NULL로 설정.

6. 출력 및 플래그 생성

입력 데이터와 연결 리스트가 일치하면 "Correct"를 출력하고 플래그를 생성:

플래그는 입력 값의 각 바이트를 10으로 나눈 나머지 값(mod 10)을 조합하여 생성.

플래그 형식은 DH{플래그}. 일치하지 않으면 "Wrong"을 출력.

일단 scanf로 제한 없는 %s 형식을 사용했으므로, 64바이트를 초과하는 입력이 있을 경우 버퍼 오버플로우가 발생할 수 있겠다.

또한 메모리 초기화와 관련된 취약점으로, 연결 리스트 생성 시 노드의 초기화가 충분하지 않을 경우 예상치 못한 데이터가 남을 수 있다고 생각할 수 있다.

입력 데이터(v4)가 연결 리스트의 데이터와 정확히 일치할 때만 플래그가 출력되므로 이를 이용해야한다고 생각되었다. 그런데 이때 연결 리스트의 데이터는 append_list 함수에서 특정 방식으로 생성되므로 이 방식을 알아야 하는데 이것과 메모리를 어떻게 연관지을지는 잘 모르겠다.