Dreamhack datastring Writeup

```
yuna@yuna-virtual-machine:~$  nc host3.dreamhack.games 17915
Calendar v0.1
Year: 2024
Month: 1
Day: 1
Hour: 1
Minute: 1
Second: 1
Formatted date: Mon Jan  1 01:01:01 2024
```

프로그램의 실행 예시이다. 사용자에게 6개 필드(Calendar, Month, Day, Hour, Minute, Second)를 입력 받아 문자열을 생성, 출력하는 것을 확인할 수 있다.

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  int v3; // eax
  int v5; // [rsp+18h] [rbp-78h] BYREF
  int v6; // [rsp+1Ch] [rbp-74h] BYREF
  int v7; // [rsp+20h] [rbp-70h] BYREF
  int v8; // [rsp+24h] [rbp-6Ch] BYREF
  int v9; // [rsp+28h] [rbp-68h] BYREF
  int v10; // [rsp+2Ch] [rbp-64h] BYREF
  _DWORD v11[3]; // [rsp+30h] [rbp-60h] BYREF
  int v12; // [rsp+3Ch] [rbp-54h]
  int v13; // [rsp+40h] [rbp-50h]
  int v14; // [rsp+44h] [rbp-4Ch]
  int v15; // [rsp+48h] [rbp-48h]
  char v16[28]; // [rsp+70h] [rbp-20h] BYREF
  int v17; // [rsp+8Ch] [rbp-4h]
```

```
v17 = 0;
setup(argc, argv, envp);
puts("Calendar v0.1");
printf("Year: ");
__isoc99_scanf("%d", &v8);
v14 = v8 - 1900;
printf("Month: ");
__isoc99_scanf("%d", &v9);
v9 = (v9 - 1) % 12;
v13 = v9++;
printf("Day: ");
__isoc99_scanf("%d", &v10);
v10 = (v10 - 1) % 31 + 1;
v12 = v10;
printf("Hour: ");
__isoc99_scanf("%d", &v7);
v7 %= 24;
v11[2] = v7;
printf("Minute: ");
__isoc99_scanf("%d", &v6);
v6 %= 60;
v11[1] = v6;
printf("Second: ");
__isoc99_scanf("%d", &v5);
v5 %= 60;
```

```
v11[0] = v5;
if ( v9 > 2 )
  v3 = v8 - 2;
else
  v3 = v8--;
v10 += v3;
v15 = (v8 / -100 + v8 / 4 + 23 * v9 / 9 + v10 + 4 + v8 / 400) % 7;
calendar(v16, v11);
printf("Formatted date: %s", v16);
if ( v13 == 11 && v12 == 25 && !v15 && v17 )
{
  puts("A Present for Admin!");
  flag();
}
return 0;
```

일단 ida에서 열어본 main 함수 부분이다. main에서는 다음과 같은 동작들을 수행한다.

사용자 입력받기

- Year: 연도를 입력받아 1900년을 기준으로 보정(v14 = v8 - 1900).
- Month: 월을 0부터 시작하도록 보정(v9 = (v9 - 1) % 12).
- Day: 날짜를 1일부터 시작하도록 보정(v10 = (v10 - 1) % 31 + 1).
- Hour, Minute, Second: 시간을 24시간제와 분, 초에 맞게 보정.

요일 계산

- v9 > 2: 1월과 2월은 전년도에 속하는 것으로 간주.
- 요일 계산 식은 Zeller's Congruence 알고리즘을 기반으로 요일을 도출.
- 결과 v15는 0부터 6까지의 값을 가지며, 각각 일요일(0) ~ 토요일(6)을 나타냄.

날짜를 포맷하여 출력

- calendar 함수는 날짜와 시간을 특정 형식으로 변환하여 v16에 저장.
- 이후 변환된 날짜를 printf로 출력.

조건 체크

조건을 만족하면 "A Present for Admin!" 메시지를 출력하고 flag 함수가 호출됨.

- v13 == 11: 월이 12월(0부터 시작하므로 11).
- v12 == 25: 일이 25일.

- !v15: 요일이 일요일(0).
- v17 != 0: v17 플래그가 설정되어야 함.

즉, 입력 날짜가 12월 25일 일요일이고, 추가적인 조건(v17)이 만족될 때 플래그가 출력됨.

이때 요일 계산식에서 이용되는 Zeller's Congruence는 날짜를 기반으로 요일을 계산하여 공식의 결과는 요일을 숫자로 반환하는 것인데 문제 풀이에 그렇게 중요하지는 않은 것 같아서 조금만 보고 넘어갔다.

```
int __fastcall calendar(char *a1, _DWORD *a2)
{
  _DWORD v3[12]; // [rsp+10h] [rbp-50h] BYREF
  _DWORD v4[8]; // [rsp+40h] [rbp-20h] BYREF

  v4[0] = 7238995;
  v4[1] = 7237453;
  v4[2] = 6649172;
  v4[3] = 6579543;
  v4[4] = 7694420;
  v4[5] = 6910534;
  v4[6] = 7627091;
  v3[0] = 7233866;
  v3[1] = 6448454;
  v3[2] = 7496013;
  v3[3] = 7499841;
  v3[4] = 7954765;
  v3[5] = 7238986;
  v3[6] = 7107914;
  v3[7] = 6780225;
  v3[8] = 7365971;
  v3[9] = 7627599;
  v3[10] = 7761742;
  v3[11] = 6513988;

  return sprintf(
           a1,
           "%.3s %3s%3d %.2d:%.2d:%.2d %d\n",
           (const char *)&v4[a2[6]],
           (const char *)&v3[a2[4]],
           a2[3],
           a2[2],
           a2[1],
           *a2,
           a2[5] + 1900);
}
```

main에서 언급된 calendar 함수이다.

일단 v3~v4에서의 데이터는 각각 월 정보, 요일 정보를 ASCII 코드값으로 인코딩한 것으로 보인다.

이제 sprint()를 보면

- %.3s: 요일 이름(v4[a2[6]])의 첫 3글자를 출력.
- %3s: 월 이름(v3[a2[4]])의 첫 3글자를 출력.
- %3d: 일(day)을 출력.
- %.2d: 시간(hour), 분(minute), 초(second)를 2자리로 출력.
- %d: 연도(year)을 출력 (a2[5] + 1900으로 변환).

이렇게 되어 있는데, 여기서 취약점을 찾아낼 수 있다.

sprintf()는 포맷 문자열에 따라 데이터를 특정 버퍼에 저장한다. 이 과정에서 버퍼 크기 검사를 수행하지 않기 때문에 적절한 크기의 버퍼가 제공되지 않으면 버퍼 오버플로우가 발생할 수 있다.

문제에서 함수 매개변수 a1은 결과 문자열을 저장하는 버퍼이다. 그런데 이 버퍼는 코드 내에서 크기가 명시적으로 제한되지 않았다. 즉 포맷된 문자열의 크기가 a1버퍼에 할당된 크기를 초과한다면 오버플로우가 발생하여 sprintf()가 데이터를 초과하여 쓰게 된다.

main()을 다시 살펴보면, v5~v10에 해당하는 변수들에 scanf()로 입력을 받고, 나머지 연산으로 포매팅을 거친 뒤 해당 값들을 v11~v15에 나누어 저장하는 것을 확인할 수 있다.

calendar()에 인자로 주어지는 변수가 v16, v11인데 v11의 타입이 int [3]으로 해석되는 점과 v12~v15의 값이 할당되기만 하고 사용되지 않는 것을 보면 v11은 28바이트 크기 구조체일 것이라고 가정할 수 있다.

V11을 적당히 어떻게 하면 v16의 [28] 바이트값을 v12로 밀어서 할 수 있을 것 같다.

4 바이트 값을 7개 가지는 구조체를 선언해 v11에 할당한다고 가정하자.

그럼 main()의 v12부분 선언은 char v12[28]과 int v13;이 될 것이다.

이를 이용하여 버퍼 오버플로우를 공략할 수 있을 것이다. 적당히 긴 연도 정보를 입력하는 것으로 v12의 28 바이트 크기를 넘어서 v13 값까지 침범할 수 있게 된다. v12의 크기 및 널 바이트 입력을 감안하면 1천만 이상의 값을 넣을 때 침범이 가능하다고 생각해 볼 수 있다.

따라서 익스플로잇 코드에서 해야 할 것은 **1천만 이상의 연도 중 12월 25일이 일요일이라고 프로그램이 인식하는 일자를 찾는 것**이라 할 수 있겠다. 일단 여기까진 생각해 보았고 코드에 대해서는 좀 더 생각해봐야할 것 같다.