introducción básica

Self-Modifying Code

Eso puede aumentar la dificultad del análisis estático, este es un ejemplo simple

Generar exe archivo básico

Visual Studio crea un proyecto vacío, crea un nuevo test.c en la carpeta "archivos de origen", pega el contenido de la siguiente manera:

```
#include <stdio.h>

char format[] = "%s %s\n";
char hello[] = "Hello";
char world[] = "World";

int main()
{
    __asm {
        mov eax, start
        mov ecx, end
        sub ecx, eax
        the_loop:
            mov ebx, [eax]
            xor ebx, 0x23
            mov [eax], ebx
            inc eax
            loop the_loop
        start:
            pushad
            mov  eax, offset world
            push eax
            mov  eax, offset hello
            push eax
            mov  eax, offset format
            push eax
            call printf
            pop  ebx
            pop  ebx
            pop  ebx
            popad
        end:
    }
```

```
    return 0;
}
```

El código es relativamente simple, solo di algo

Las 3 líneas de código ensamblador por encima de the_loop son para obtener el área a ser XORed, la dirección inicial es eax y la longitud es ecx;

El bucle para comenzar es la lógica de XOR, y aquí se usa 0x23 XOR;

De principio a fin es la lógica para generar "Hola mundo"

Luego compile la versión x86 del exe. Tenga en cuenta que el exe en este momento no puede generar "Hello World" normalmente, porque la parte de principio a fin no ha sido XORed. Además, hay dos problemas predeterminados del mecanismo PE, que se discutirán más adelante.

Use IDA para XOR algunos códigos

Utilizando IDAPro para abrir el archivo exe generado en el paso anterior y busca la parte que queremos procesar, de la siguiente manera:

```
.text:00411894 60                pusha
.text:00411895 B8  10 A0  41 00          mov         eax, offset aWorld ;  "world"
.text:0041189A 50                push        eax
.text:0041189B B8  08 A0  41 00          mov         eax, offset aHello ;  "Hello"
.text:004118A0 50                push        eax
.text:004118A1 B8  00 A0  41 00          mov         eax, offset aSS ;  "%s %s\n"
.text:004118A6 50                push        eax
.text:004118A7 E8  21 F8 FF FF                                       call        sub_4110CD
.text:004118AC 5B                pop          ebx
.text:004118AD 5B                pop          ebx
.text:004118AE 5B                pop          ebx
.text:004118AF 61                popa
```

Escriba la dirección de inicio y la dirección final, aquí están 0x00411894 y 004118AF,
Luego, en IDA, File -> Script file..., ejecute la siguiente secuencia de comandos de Python:

```
  # coding:utf8
```

```
from ida_bytes  import get_byte, patch_byte

def range_xor(start_addr, end_addr, xor_num):
        for the_addr  in range(start_addr, end_addr+1):
                the_byte  = get_byte(the_addr)
                patch_byte(the_addr, the_byte ^ xor_num)

range_xor(0x00411894,  0x004118AF,  0x23)
```

Edit -> Patch program -> Apply patches to input file... , y después de guardar de esta manera, obtenemos el exe después del procesamiento de XOR, pero todavía no puede ejecutarse normalmente ahora.

Abordar los problemas causados por el mecanismo PE

Los dos mecanismos de PE mencionados anteriormente hacen que el programa no se ejecute normalmente.
Uno es que el segmento de código no se puede escribir de forma predeterminada y el otro es una dirección base dinámica. Usamos DIEWin para modificar la ubicación correspondiente.

Deshabilitar la dirección base dinámica:
Primero, desmarque "solo lectura" en la esquina superior derecha IMAGE_NT_HEADERS -> IMAGE_OPTIONAL_HEADER -> DllCharacteristics desmarque "DYNAMIC_BASE".

El segmento del código de configuración se puede escribir:
Bloque -> .texto -> Haga clic derecho para editar -> desmarque la marca de "solo lectura" en la esquina superior derecha, y haga clic para marcar "MEM_WRITE" para las características
Después de esto, podemos realizar el ejemplo de smc más simple.