CÁTEDRA DE CIBERSEGURIDAD CIBERUGR, INCIBE-UGR

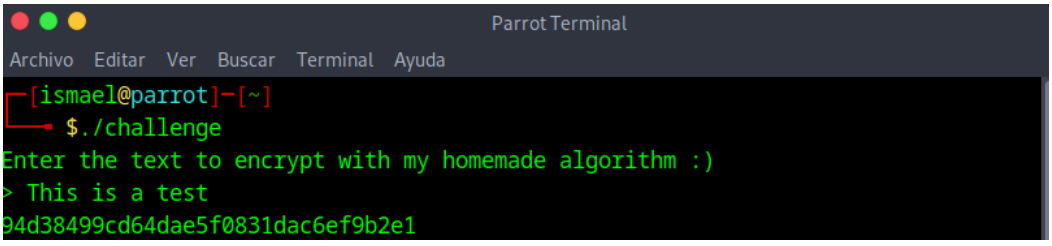
Nombre	Rounded
Categoría	Misc (Reversing + Crypto)
Dificultad	Difícil
Puntos	500

DESCRIPCIÓN DEL RETO

Una vuelta, dos vueltas, tres vueltas, cuatro vueltas, cinco vueltas, seis vueltas, siete vueltas, ocho vueltas, nueve vueltas, diez vueltas, once vueltas… ¡Anda! ¿Y esto? 5a78d931803e9b6d5cc7a3301a0d8fb7ed95b3b33652529626aa1eb85e715eec

WRITEUP

Se nos da un binario llamado “challenge”. Al ejecutar el binario se nos pide un texto de entrada y devuelve una cadena en hexadecimal. Por el texto se puede suponer que el binario cifra la entrada y devuelve la salida cifrada.



Si abrimos el binario con IDA, encontraremos la función main con todo el código del programa. Si analizamos el ensamblador se puede observar cómo se usan instrucciones relacionadas con operaciones AES (vaesenc, vaesenclast, vaeskeygenassist). Dichas instrucciones son específicas de la arquitencura x86/64 para el cifrado AES. Una búsqueda rápida en internet nos dará información de que hace cada instrucción. Por ejemplo, en el caso de “vaesenc”:

AESENC — Perform One Round of an AES Encryption Flow

Opcode/Instruction	Op/En	64/32-bit Mode	CPUID Feature Flag	Description
66 0F 38 DC /r AESENC xmm1, xmm2/m128	A	V/V	AES	Perform one round of an AES encryption flow, using one 128-bit data (state) from xmm1 with one 128-bit round key from xmm2/m128.
VEX.128.66.0F38.WIG DC /r VAESENC xmm1, xmm2, xmm3/m128	B	V/V	AES AVX	Perform one round of an AES encryption flow, using one 128-bit data (state) from xmm2 with one 128-bit round key from the xmm3/m128; store the result in xmm1.
VEX.256.66.0F38.WIG DC /r VAESENC ymm1, ymm2, ymm3/m256	B	V/V	VAES	Perform one round of an AES encryption flow, using two 128-bit data (state) from ymm2 with two 128-bit round keys from the ymm3/m256; store the result in ymm1.
EVEX.128.66.0F38.WIG DC /r VAESENC xmm1, xmm2, xmm3/m128	C	V/V	VAES AVX512VL	Perform one round of an AES encryption flow, using one 128-bit data (state) from xmm2 with one 128-bit round key from the xmm3/m128; store the result in xmm1.
EVEX.256.66.0F38.WIG DC /r VAESENC ymm1, ymm2, ymm3/m256	C	V/V	VAES AVX512VL	Perform one round of an AES encryption flow, using two 128-bit data (state) from ymm2 with two 128-bit round keys from the ymm3/m256; store the result in ymm1.
EVEX.512.66.0F38.WIG DC /r VAESENC zmm1, zmm2, zmm3/m512	C	V/V	VAES AVX512F	Perform one round of an AES encryption flow, using four 128-bit data (state) from zmm2 with four 128-bit round keys from the zmm3/m512; store the result in zmm1.

Instruction Operand Encoding

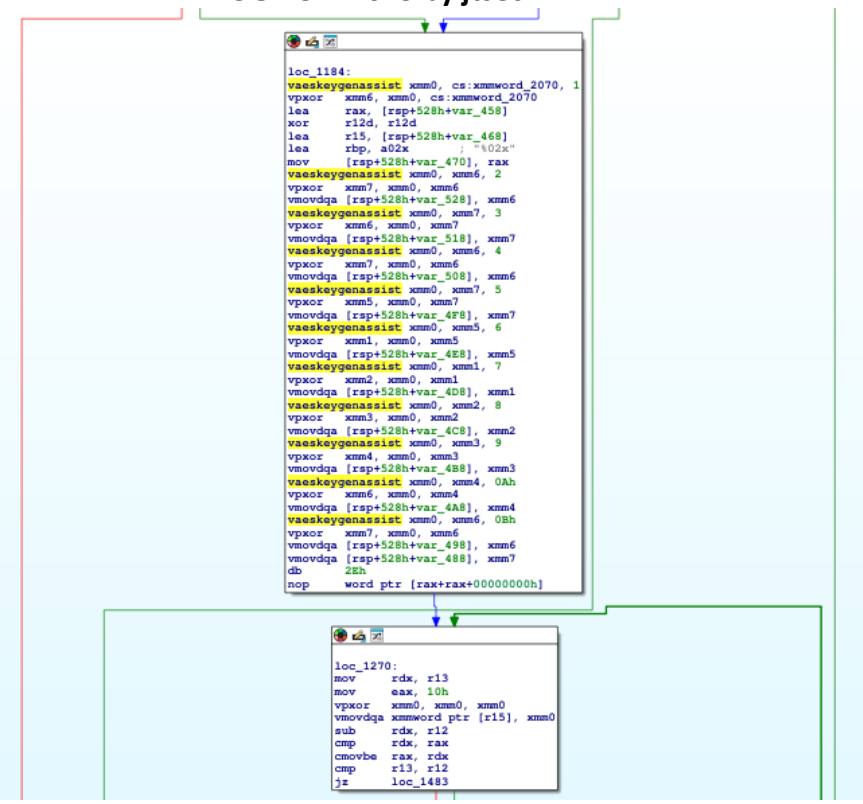
Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	N/A	ModRM:reg (r, w)	ModRM:r/m (r)	N/A	N/A
B	N/A	ModRM:reg (w)	VEX.vvvv (r)	ModRM:r/m (r)	N/A
C	Full Mem	ModRM:reg (w)	EVEX.vvvv (r)	ModRM:r/m (r)	N/A

Antes del análisis del código fuente, por tener una pequeña introducción de cómo funciona AES. AES es un cifrado simétrico por bloques. Simplificando mucho, su funcionamiento se basa en el cifrado de los datos a través de múltiples rondas usando como clave una expansión de una clave original que podrá ser de 128, 192 o 256 bits. En la siguiente web podemos ver de una manera gráfica tanto el proceso de expansión de clave de un algoritmo AES (Aunque en nuestro, la expansión será mucho más sencilla) y el proceso de cifrado basado en rondas https://formaestudio.com/rijndaelinspector/archivos/Rijndael_Animation_v4_eng-html5.html.

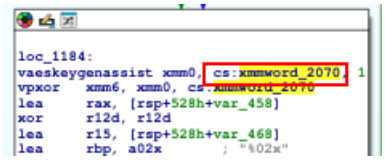
Si analizamos con detalle el código podremos separar el flujo de ejecución en varias fases.

- Fase 1: Inicio y entrada del texto a cifrar.

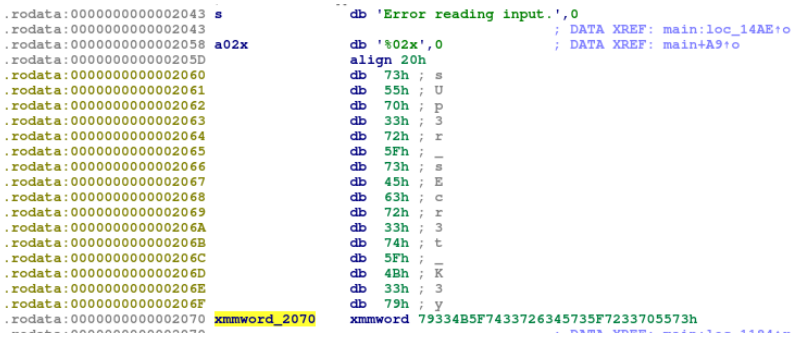
- Fase 2: Expansión de clave. Se utiliza la instrucción “vaeskeygenassist” junto a la instrucción “vpxor”. Según la documentación “vaeskeygenassist”, ignorando su funcionamiento interno, realizará junto a “vpxor” la expansión de la clave que se usará para las rondas AES. “vaeskeygenassist” necesita una clave maestra para realizar la expansión



Si miramos en los parámetros de la primera iteración podemos ver cómo se obtiene una palabra de memoria



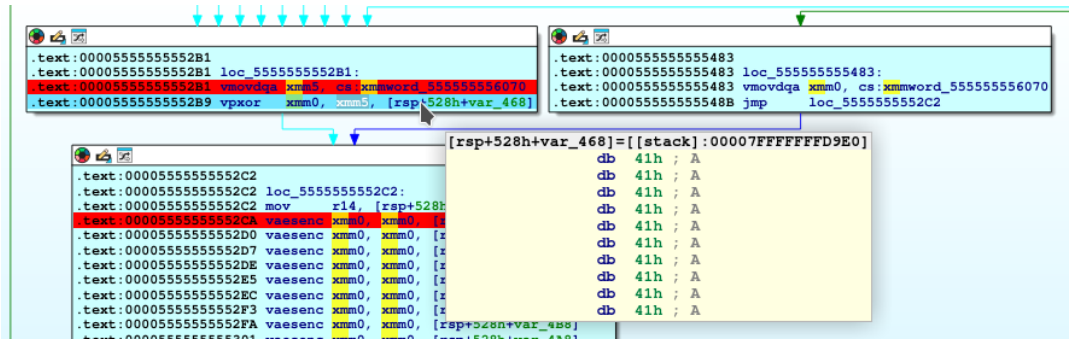
Aquí se encontrará la clave maestra “sUp3r_sEcr3t_K3y”.



- Fase 3: Cifrado por rondas. Una vez se expande la clave, se cifra el texto de entrada por rondas. En concreto se usarán 11 rondas. La última de ellas especificadas con la instrucción “vaenclast”. Por último, se imprimirá por pantalla el contenido cifrado en hexadecimal.

Con el análisis del binario finalizado, el objetivo será realizar el proceso inverso para descifrar el texto cifrado "5a78d931803e9b6d5cc7a3301a0d8fb7ed95b3b33652529626aa1eb85e715eec".

Si probamos depurarlo y parar junto antes del cifrado AES, podremos ver que la entrada "AAAAAA" que introducimos en la terminal se corresponde con el texto que se le manda a vpxor.



Con toda esta información podemos empezar a implementar un código de descifrado. Para la fase 2, la expansión de clave, se ejecuta un bucle de 11 iteraciones donde se usa "vaeskeygenassist" y "vpxor", donde se le da la clave maestra como entrada. Una posible implementación sería la siguiente:

```
const uint8_t key[16] = "sUp3r_sEcr3t_K3y";

void aes_key_expansion(__m128i *round_keys) {
    round_keys[0] = _mm_loadu_si128((const __m128i *)key);
    __m128i temp;

    for (int i = 1; i < 12; i++) {
        temp = _mm_aeskeygenassist_si128(round_keys[i - 1], i);
        round_keys[i] = _mm_xor_si128(round_keys[i - 1], temp);
    }

    for (int i = 1; i < 11; i++) {
        round_keys[i] = _mm_aesimc_si128(round_keys[i]);
    }
}
```

En este código, realizamos la expansión de clave como se ha hecho para cifrar y finalmente invertimos para el descifrado. Se puede ver un ejemplo parecido en https://gist.github.com/acapola/d5b940da024080dfaf5f.

En el caso de la fase 3, simplemente realizaremos el paso contrario usando "_mm_aesdec_si128" y "_mm_aesdeclast_si128". En este caso, se empieza desde el último bloque cifrado en el código original hasta el primero.

```
// Custom AES decryption with 11 rounds
void aes_decrypt_custom(const uint8_t *ciphertext, uint8_t *plaintext, int rounds) {
    __m128i block = _mm_loadu_si128((const __m128i *)ciphertext);
    __m128i round_keys[12];

    aes_key_expansion(round_keys);

    block = _mm_xor_si128(block, round_keys[rounds]);

    for (int i = 10; i > 0; i--) {
        block = _mm_aesdec_si128(block, round_keys[i]);
    }

    block = _mm_aesdeclast_si128(block, round_keys[0]);

    _mm_storeu_si128((__m128i *)plaintext, block);
}
```

Por último, llamaremos a la función de descifrado dos veces (una para cada bloque de 16 bytes).

```
41  int main() {
42      uint8_t ciphertext[32] = {0x5a,0x78,0xd9,0x31,0x80,0x3e,0x9b,0x6d
43      uint8_t plaintext[32];
44      int rounds = 11;
45
46      aes_decrypt_custom(ciphertext, plaintext, rounds);
47      aes_decrypt_custom(ciphertext + 16, plaintext + 16, rounds);
48
49      for (size_t i = 0; i < 32; i++) {
50          printf("%c", plaintext[i]);
51      }
52      printf("\n");
53
54      return 0;
55  }
```

Si compilamos y ejecutamos obtendremos la flag UGR_ETSIIT_CTF25{W0w_1_4m_d1Zzy}