

Desencriptado en tiempo de ejecución

Introducción

Hace algunos días me topé por causalidad con una versión antigua del programa FlashSlider (v 0.9) y tuve dificultades en llegar a la zona donde se analizaba el serial. Descubrí con sorpresa que las zonas nobles del programa eran desencriptadas en tiempo de ejecución. El mecanismo es sencillo y traté de aprovecharlo.

El Análisis

Mediante un *memory breakpoint on write* en la sección CODE es fácil encontrar la zonas donde el programa reescribe sobre su código antes y después de ejecutarlo. Todas tienen un aspecto similar:

004EF076	. 33C0	XOR EAX,EAX		
004EF078	. 55	PUSH EBP		
004EF079	. 68 25F14E00	PUSH FlashSli.004EF125		
004EF07E	. 64:FF30	PUSH DWORD PTR FS:[EAX]		
004EF081	. 64:8920	MOV DWORD PTR FS:[EAX],ESP		
004EF084	. 53	PUSH EBX		
004EF085	. BB DDF04E00	MOV EBX,FlashSli.004EF000	<--- Final	
004EF08A	. 51	PUSH ECX		
004EF08B	. B9 29000000	MOV ECX,29	<--- Longitud	
004EF090	. 50	PUSH EAX		
004EF091	. B8 F6A5EED1	MOV EAX,D1EEA5F6	<--- Semilla	
004EF096	> 3003	XOR BYTE PTR DS:[EBX],AL		
004EF098	. C1C0 05	ROL EAX,5		
004EF09B	. 05 BB80D7C5	ADD EAX,C5D780BB		
004EF0A0	. C1C8 0A	ROR EAX,0A		
004EF0A3	. 4B	DEC EBX		
004EF0A4	. 05 F802997D	ADD EAX,7D9902F8	<--- Retroceso	
004EF0A9	. 35 B36B21E4	XOR EAX,E4216BB3		
004EF0AE	. 05 02337428	ADD EAX,28743302		
004EF0B3	. ^ E2 E1	LOOPE SHORT FlashSli.004EF096	<--- Bucle	
004EF0B5	> 97	XCHG EAX,EDI		
004EF0B6	. 3B52 F7	CMP EDX,DWORD PTR DS:[EDX-9]		
004EF0B9	. C08472 5E880	ROL BYTE PTR DS:[EDX+ESI*2+870C885E],0F5		
004EF0C1	. 56	PUSH ESI		
004EF0C2	. 1D 28A5DF45	SBB EAX,45DFA528		
004EF0C7	. 6E	OUTSDX, BYTE PTR ES:[EDI]		
004EF0C8	. 2928	SUB DWORD PTR DS:[EAX],EBP		
004EF0CA	. 5C	POP ESP		
004EF0CB	. A5	MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]		
004EF0CC	. ^ 7D 10	JGE SHORT FlashSli.004EF0DE		
004EF0CE	. A8 58	TEST AL,58		
004EF0D0	. A9 B7116B78	TEST EAX,786B11B7		
004EF0D5	. ^ 7D DE	JGE SHORT FlashSli.004EF0B5		
004EF0D7	. 99	CDQ		
004EF0D8	. 57	PUSH EDI		
004EF0D9	. 0D 9DE5F4F6	OR EAX,F6F4E59D		
004EF0DE	> 53	PUSH EBX		
004EF0DF	. BB DDF04E00	MOV EBX,FlashSli.004EF000		

Se distinguen cuatro zonas (vamos a prescindir de otras cuestiones como la alteración del SEH):

1ª Zona INICIACIÓN: aquí se empujan a la pila los registros que van a ser utilizados en el proceso (preservación de registros). Se define (a) donde se va a realizar la (des)encriptación, EBX apunta al final de la zona encriptada que será recorrida hacia atrás; (b) la longitud en bytes de la zona a (des)encriptar, ECX hace su función contadora habitual; (c) una semilla inicial para el proceso en EAX.

2ª Zona ACCIÓN: aquí se realiza la alteración del código byte a byte. Contiene una sola instrucción que puede ser:

- XOR BYTE PTR DS:[EBX],AL
- ADD BYTE PTR DS:[EBX],AL
- SUB BYTE PTR DS:[EBX],AL

Debemos tener en cuenta que debe ser reversible (tener inversa).

3ª Zona GENERACIÓN: Se realizan operaciones lógicas, y/o aritméticas para cambiar la semilla de una manera rápida, reproducible pero difícilmente reconocible. También se decremento el puntero EBX para

pasar al siguiente byte a procesar. Acaba inevitablemente con la instrucción LOOPD que cierra el bucle.

4ª Zona CÓDIGO ENCRIPTADO: Contiene los bytes encriptados a procesar. Puede incluir una parte con basura inicial para despistar al desensamblador, aún después del desenscriptado.

Después de salir del bucle se obtiene:

004EF076	33C0	XOR EAX,EAX	
004EF078	55	PUSH EBP	
004EF079	68 25F14E00	PUSH FlashSli.004EF125	
004EF07E	64:FF30	PUSH DWORD PTR FS:[EAX]	
004EF081	64:8920	MOV DWORD PTR FS:[EAX],ESP	
004EF084	53	PUSH EBX	
004EF085	B8 DDF04E00	MOV EBX,FlashSli.004EF00D	
004EF08A	51	PUSH ECX	
004EF08B	B9 29000000	MOV ECX,29	
004EF090	50	PUSH EAX	
004EF091	B8 F6A5EED1	MOV EAX,D1EEA5F6	
004EF096	3003	XOR BYTE PTR DS:[EBX],AL	
004EF098	C1C0 05	ROL EAX,5	
004EF09B	05 BB8D07C5	ADD EAX,C5D78DBB	
004EF0A0	C1C8 0A	ROR EAX,0A	
004EF0A3	4B	DEC EBX	
004EF0A4	05 F802997D	ADD EAX,7D9902F8	
004EF0A9	35 B36B21E4	XOR EAX,E4216BB3	
004EF0AE	05 02337428	ADD EAX,28743302	
004EF0B3	^ E2 E1	LOOPD SHORT FlashSli.004EF096	
004EF0B5	58	POP EAX	
004EF0B6	59	POP ECX	
004EF0B7	5B	POP EBX	
004EF0B8	^ EB 10	JMP SHORT FlashSli.004EF0CA	
004EF0BA	8472 5E	TEST BYTE PTR DS:[EDX+5E],0H	
004EF0BD	8B0C87	MOV BYTE PTR DS:[EDI+EAX*4],CL	
004EF0C0	F5	CMC	
004EF0C1	56	PUSH ESI	
004EF0C2	1D 28A5DFA8	SBB EAX,45DFA528	
004EF0C7	6E	OUTS DX,BYTE PTR ES:[EDI]	
004EF0C8	2928	SUB DWORD PTR DS:[EAX],EBP	
004EF0CA	33C0	XOR EAX,EAX	
004EF0CC	E8 837DFFFF	CALL FlashSli.004E6E54	SUBROUTINA OCULTADA
004EF0D1	33C0	XOR EAX,EAX	
004EF0D3	5A	POP EDX	
004EF0D4	59	POP ECX	
004EF0D5	59	POP ECX	
004EF0D6	64:8910	MOV DWORD PTR FS:[EAX],EDX	
004EF0D9	68 2CF14E00	PUSH FlashSli.004EF12C	
004EF0DE	53	PUSH EBX	

BASURA

Se observa cómo se oculta a los ojos indiscretos la subrutina 4E6E54. Después de su ejecución, el programa vuelve a encriptar esta parte del código.

Aplicación

Crearemos una rutina en asm que llamada antes y después de un trozo de código de nuestro programa realice el truco anteriormente visto.

```

encrptador proc uses eax ebx ecx longitud:sdword
    mov ebx, dword ptr ds:[ebp+4]           ; direccion de retorno del call
    mov ecx, longitud                       ; longitud
    .if longitud < 0
        add ebx, ecx
        sub ebx, 11
        neg ecx
    .endif
    mov eax, 0d1eea5f6h                     ; semilla
    otro:
        xor byte ptr ds:[ebx],al
        rol eax,5
        add eax,0c5d78dbbh                 ; numero magico
        ror eax,0ah
        inc ebx
        add eax,028743302h                 ; numero magico
    loopd otro
    ret
encrptador endp

```

Esta rutina sólo necesitará el parámetro de la longitud del código a procesar, puesto que ya hemos dicho que la pondremos *inmediatamente antes e inmediatamente después* del código a modificar. La implementación es relativamente sencilla. Incluimos las llamadas en el código fuente. Las etiquetas inicio: y final: sirven para calcular el valor del parámetro a transferir a la subrutina. Si el parámetro transferido es positivo entiende que es la llamada primera y si es negativo entiende que es la llamada segunda.

```
start:
    mov eax, offset final - offset inicio
    invoke encriptador, eax
inicio:
    finit
    mov ecx, 2
    .repeat
        fild qword ptr ds:qFibo[8*ecx-8]
        fild qword ptr ds:qFibo[8*ecx-16]
        fadd
        fistp qword ptr ds:qFibo[8*ecx]
        inc ecx
    .until ecx == 32
    mov ebx, 1
    .repeat
        lea eax, [2*ebx]
        fild qword ptr ds:qFibo[8*eax-8]
        fild qword ptr ds:qFibo[8*eax]
        fmul
        fistp qword ptr ds:qSolu[8*ebx]
        invoke crt_sprintf, addr szTexto, addr szFormato, ebx, qword ptr ds:qSolu[8*ebx]
        invoke MessageBox, NULL, addr szTexto, addr szTitulo, MB_ICONINFORMATION+MB_OK
        inc ebx
    .until ebx == 16
final:
    mov eax, offset inicio - offset final
    invoke encriptador, eax
    invoke ExitProcess,0
end start
```

PARTE DEL PROGRAMA
A PROTEGER

Compilamos y hacemos link con la opción `/section:.text,wer` para que el programa pueda sobrescribir su propio código. Ponemos un breakpoint en el RET de la subrutina encriptadora.

00401000	55	PUSH EBP
00401001	8BEC	MOV EBP,ESP
00401003	50	PUSH EAX
00401004	53	PUSH EBX
00401005	51	PUSH ECX
00401006	3E:8B5D 04	MOV EBX,DWORD PTR DS:[EBP+4]
0040100A	8B4D 08	MOV ECX,DWORD PTR SS:[EBP+8]
0040100D	837D 08 00	CMP DWORD PTR SS:[EBP+8],0
00401011	7D 07	JGE SHORT Problema.0040101A
00401013	03D9	ADD EBX,ECX
00401015	83EB 0B	SUB EBX,0B
00401018	F7D9	NEG ECX
0040101A	B8 F6A5EED1	MOV EAX,D1EEA5F6
0040101F	3003	XOR BYTE PTR DS:[EBX],AL
00401021	C1C0 05	ROL EAX,5
00401024	05 BB8DD7C5	ADD EAX,C5D78DBB
00401029	C1C8 0A	ROR EAX,0A
0040102C	43	INC EBX
0040102D	05 02337428	ADD EAX,28743302
00401032	E2 EB	LOOPD SHORT Problema.0040101F
00401034	59	POP ECX
00401035	5B	POP EBX
00401036	58	POP EAX
00401037	C9	LEAVE
00401039	C2 0400	RETN 4

Tomamos nota del comienzo 401046 y final de la zona a encriptar 4010C8 (¡ojo!).

00401034	. S9	POP ECX	
00401035	. 5B	POP EBX	
00401036	. 58	POP EAX	
00401037	. C9	LEAVE	
00401038	. C2 0400	RETN 4	
0040103B	. B8 83000000	MOV EAX,83	
00401040	. 50	PUSH EAX	
00401041	. E8 BAFFFFFF	CALL Problema.00401000	
00401046	. 9B	WAIT	
00401047	. DBE3	FINIT	
00401049	. B9 02000000	MOV ECX,2	
0040104E	> DF2CCD F82F41	FILD QWORD PTR DS:[ECX*8+402FF8]	
00401055	. DF2CCD F02F41	FILD QWORD PTR DS:[ECX*8+402FF0]	
0040105C	. DEC1	FADD ST(1),ST	
0040105E	. DF3CCD 003041	FISTP QWORD PTR DS:[ECX*8+403000]	
00401065	. 41	INC ECX	
00401066	. 23FD 00	CMP ECX,00	
004010B7	. 68 00314000	PUSH Problema.00403100	
004010BC	. 6A 00	PUSH 0	
004010BE	. E8 1F000000	CALL <JMP.&user32.MessageBoxA>	
004010C3	. 43	INC EBX	
004010C4	. 83FB 10	CMP EBX,10	
004010C7	. 75 A7	JNZ SHORT Problema.00401070	
004010C9	. B8 7DFFFFFF	MOV EAX,-83	
004010CE	. 50	PUSH EAX	
004010CF	. E8 2CFFFFFF	CALL Problema.00401000	
004010D4	. 6A 00	PUSH 0	
004010D6	. E8 01000000	CALL <JMP.&KERNEL32.ExitProcess>	
004010DB	. CC	INT3	
004010DC	. FF25 00204000	JMP DWORD PTR DS:[<&KERNEL32.ExitProcess>	
004010E2	. FF25 10204000	JMP DWORD PTR DS:[<&user32.MessageBoxA>	
004010E8	. 00	DB 00	

Text = ""

hOwner = NULL

MessageBoxA

ExitCode = 0

ExitProcess

kernel32.ExitProcess

user32.MessageBoxA

Damos RUN para en el BP. Seleccionamos la zona cambiada y grabamos los cambios mediante los “Copy to executable” y “Save file” habituales.

Ya tenemos nuestro programa listo para funcionar y protegido de vistazos indiscretos.

Agradecimientos a todos los CracksLatinos. Espero que os guste. Orniaco