GAME: Arabian Nights
Protection: SecuROM *new* 4.48.00.0004
Author: Luca D'Amico - V1.0 - 5 Maggio 2022

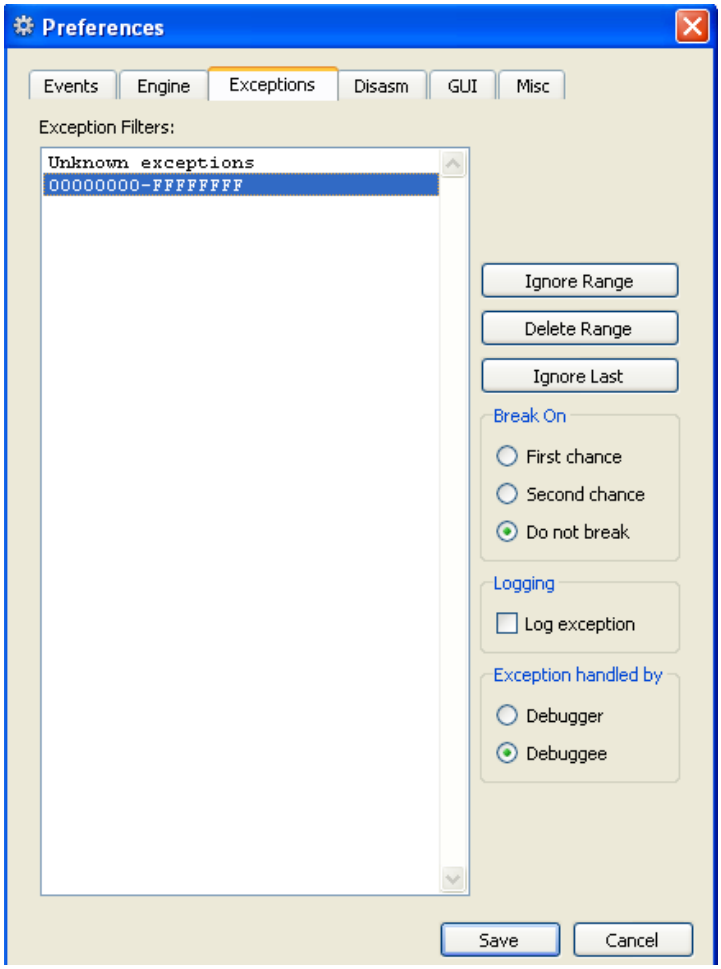
Cosa ci serve:
- Windows XP VM (ho usato VMware)
- x64dbg (x32dbg)
- CFF Explorer
- Disco di gioco originale (abbiamo bisogno del disco ORIGINALE)

Prima di iniziare:
I giochi protetti con questa versione di SecuROM purtroppo non funzioneranno su versioni di Windows più recenti di XP. Come abbiamo già visto con le prime versioni di Safedisc, una volta rimossa la protezione, il software è perfettamente godibile anche su Windows 11.
Il funzionamento di questo DRM consiste nel sostituire alcune chiamate alle varie API di Windows usate dal gioco, con una funzione che una volta effettuati i dovuti controlli, raggiungerà l'API richiesta con un jump. Il salto sarà diretto, senza passare per il thunk sulla IAT, ciò significa che quando ricostruiremo gli imports, dovremo ciclare sulla IAT alla ricerca del thunk corretto da inserire nel segmento .text patchando i bytes dove avviene la chiamata. Inoltre è presente uno strato di cifratura inziale (per questo serve il disco originale) e varie tecniche anti-debug che complicheranno il raggiungimento dell'OEP.

Iniziamo:
Installiamo il gioco selezionando installazione completa.
Carichiamo l'eseguibile (_start.exe) nel debugger e notiamo che l'entry point si trova a 0x737CFD.
Andando sulla Memory Map possiamo vedere che ci troviamo nella section .cms_t:

00400000	00001000	_start.exe		IMG	-R---	ERWC-
00401000	00062000	".text"	Executable code	IMG	ER---	ERWC-
00463000	00003000	".rdata"	Read-only initialized data	IMG	-R---	ERWC-
00466000	002C4000	".data"	Initialized data	IMG	-RWC-	ERWC-
0072A000	00002000	".idata"	Import tables	IMG	-RWC-	ERWC-
0072C000	00014000	".cms_t"		IMG	ER---	ERWC-
00740000	0002C000	".cms_d"		IMG	-RWC-	ERWC-
0076C000	00001000	".idata"	Import tables	IMG	-RW--	ERWC-
0076D000	00001000	".rsrc"	Resources	IMG	-R---	ERWC-
0076E000	00009000	".reloc"	Base relocations	IMG	-R---	ERWC-

Come possiamo immaginare il codice del gioco si trova nel segmento .text e quello che stiamo per eseguire è in realtà il loader di SecuROM.
Provando cliccare RUN sul debugger, verremo costantemente bloccati con eccezioni di vario tipo: questa è solo una delle varie tecniche usate per rallentarci. Fortunatamente possiamo risolvere il problema dicendo al debugger di ignorare tutte le eccezioni:

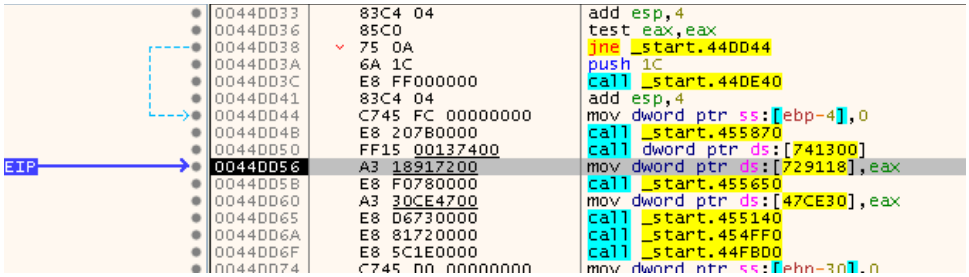


Consiglio inoltre di togliere il segno di spunta da “Log exception”, in quanto l’enorme quantità di eccezioni rallenterà moltissimo l’esecuzione. Adesso siamo pronti per iniziare.

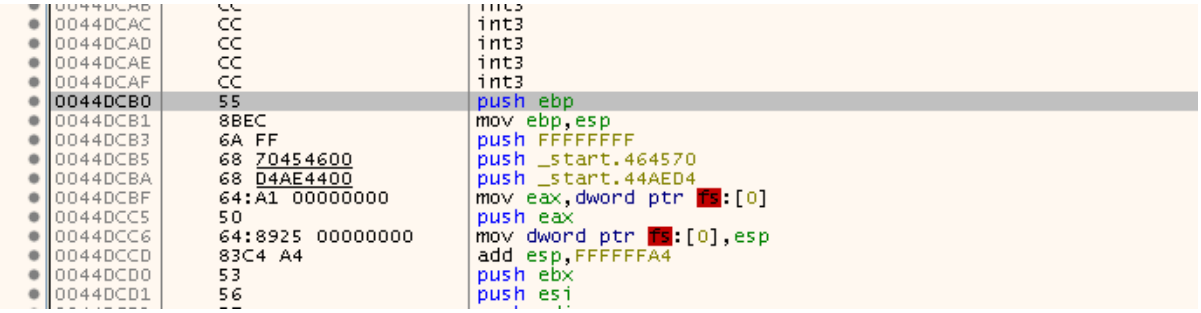
Come nella maggior parte dei casi quando si tenta di rimuovere una protezione di questo tipo, il primo passo è quello di riuscire a raggiungere l’OEP.
Il primo tentativo che ho effettuato è stato quello di settare un breakpoint hardware sull’esecuzione del segmento .text: purtroppo grazie alle varie tecniche anti-debug implementate, questa operazione verrà rilevata e causerà un loop del loader.
Ho deciso quindi di procedere in due passi:

1) Capire a che indirizzo si trova l’OEP
2) Trovare un modo per raggiungere detto indirizzo

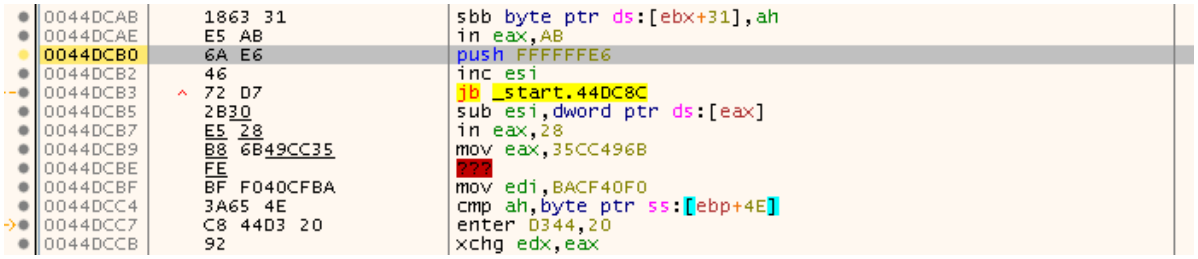
Per avvicinarmi all’OEP ho settato un breakpoint su un’API che solitamente è situata vicino all’Entry Point: GetCommandLineA.
Ogni volta che scatta il break, clicchiamo su “Run to user code” in modo tale da vedere dove si origina la chiamata. Al terzo hit, finalmente ci troviamo sul segmento .text:



Notate bene come la chiamata a GetCommandLineA si sia originata da call dword ptr ds:[741300]. Molto interessante, ma al momento restiamo concentrati sul nostro obiettivo attuale.
Poiché l'API che abbiamo breakato si trova nella funzione dove risiede l'OEP, salendo all'inizio di essa, saremo finalmente a destinazione:



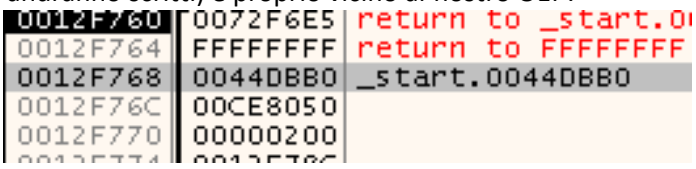
Perfetto, adesso sappiamo che 0x44DCB0 è il nostro Original Entry Point. Ora dobbiamo trovare un modo per breakare l'esecuzione proprio lì.
NB: è assolutamente obbligatorio breakare all'OEP quando si effettuano dump dalla memoria! Altrimenti l'eseguibile ottenuto non funzionerà, in quanto conterrà dati relativi all'esecuzione attuale.
Se proviamo a mettere un breakpoint a quell'indirizzo ed a riavviare l'esecuzione, il loader lo rileverà e andrà in loop. Stessa cosa se il breakpoint settato è hardware.
Se riavviamo nuovamente il debugger e ci rechiamo all'OEP prima di aver eseguito il loader del SecuROM noteremo una cosa molto interessante:



Si tratta di codice completamente diverso! È logico pensare che questa sia memoria cifrata e che verrà sovrascritta dal loader del SecuROM durante la fase di avvio.
Su Windows i processi possono modificare la memoria grazie all'API WriteProcessMemory, che ha la seguente firma:

```
BOOL WriteProcessMemory(
  [in]  HANDLE  hProcess,
  [in]  LPVOID  lpBaseAddress,
  [in]  LPCVOID lpBuffer,
  [in]  SIZE_T  nSize,
  [out] SIZE_T  *lpNumberOfBytesWritten
);
```

Ottimo, settiamo un breakpoint su WriteProcessMemory e riavviamo il debugger!
Il terzo hit è quello corretto: ne siamo sicuri poiché vediamo dallo stack che l'lpBaseAddress (l'indirizzo dove i dati andranno scritti) è proprio vicino al nostro OEP:



In 0xCE8050 c'è il buffer che verrà scritto. Clicchiamo con il destro su di esso e scegliamo "Follow DWORD in disassembler":

```
00CE8050    8CE1            mov ecx,fs
00CE8052    46              inc esi
00CE8053    0033            add byte ptr ds:[ebx],dh
00CE8055    D266 8B         shl byte ptr ds:[esi-75],cl
00CE8058    14 41           adc al,41
00CE805A    83E2 08         and edx,8
00CE805D    8955 EC         mov dword ptr ss:[ebp-14],edx
00CE8060    837D EC 00      cmp dword ptr ss:[ebp-14],0
00CE8064    74 0B           je CE8071
00CE8066    8B45 08         mov eax,dword ptr ss:[ebp+8]
```

Ci troviamo esattamente dentro al buffer che verrà scritto rimpiazzando la memoria partendo da 0x44DBB0. Trovare il nostro OEP dentro il buffer a questo punto è facile: possiamo sommare all'indirizzo attuale (0xCE8050) la differenza tra 0x44DCB0 (indirizzo dell'OEP trovato in precedenza) e 0x44DBB0 (base address dove verrà scritto il buffer): 0xCE8150.
Infatti proprio a questo indirizzo troviamo il nostro OEP dentro il buffer:

```
00CE814E    CC              int3
00CE814F    CC              int3
00CE8150    55              push ebp
00CE8151    8BEC            mov ebp,esp
00CE8153    6A FF           push FFFFFFFF
00CE8155    68 70454600     push _start.464570
00CE815A    68 D4AE4400     push _start.44AED4
00CE815F    64:A1 00000000  mov eax,dword ptr fs:[0]
00CE8165    50              push eax
00CE8166    64:8925 00000000 mov dword ptr fs:[0],esp
00CE816D    83C4 A4         add esp,FFFFFFA4
00CE8170    53              push ebx
00CE8171    56              push esi
00CE8172    57              push edi
```

Ottimo! Per bloccare l'esecuzione proprio all'OEP possiamo modificare il buffer, sostituendo i primi due byte con un EBFE (salto su sé stesso). In questo modo il buffer verrà scritto con la nostra patch e una volta che il flusso di esecuzione raggiungerà l'OEP, resterà bloccato proprio lì (saltando su sé stesso). Non resterà a quel punto che settare un breakpoint per bloccare l'esecuzione e ripristinare i byte originali. Procediamo applicando la modifica:

```
00CE814F    CC              int3
00CE8150    EB FE           jmp CE8150
00CE8152    EC              in al,dx
00CE8153    6A FF           push FFFFFFFF
00CE8155    68 70454600     push _start.464570
00CE815A    68 D4AE4400     push _start.44AED4
00CE815F    64:A1 00000000  mov eax,dword ptr fs:[0]
00CE8165    50              push eax
00CE8166    64:8925 00000000 mov dword ptr fs:[0],esp
```

Clicchiamo su RUN e lasciamo che il loader di SecuROM completi il lavoro. Possiamo rimuovere il breakpoint su WriteProcessMemory.
Aspettate qualche secondo e cliccate su PAUSE. Ci ritroveremo in un loop esattamente all'OEP:

```
0044DCAD    CC              int3
0044DCAE    CC              int3
0044DCAF    CC              int3
EIP EAX
0044DCB0    EB FE           jmp _start.44DCB0
0044DCB2    EC              in al,dx
0044DCB3    6A FF           push FFFFFFFF
0044DCB5    68 70454600     push _start.464570
0044DCBA    68 D4AE4400     push _start.44AED4
0044DCBF    64:A1 00000000  mov eax,dword ptr fs:[0]
0044DCC5    50              push eax
```

Settiamo subito un breakpoint su 0x44DCB0 e ripristiamo gli opcodes originali (558B):

```
0044DCAD    CC              int3
0044DCAE    CC              int3
0044DCAF    CC              int3
EIP EAX
0044DCB0    55              push ebp
0044DCB1    8BEC            mov ebp,esp
0044DCB3    6A FF           push FFFFFFFF
0044DCB5    68 70454600     push _start.464570
0044DCBA    68 D4AE4400     push _start.44AED4
0044DCBF    64:A1 00000000  mov eax,dword ptr fs:[0]
0044DCC5    50              push eax
```

Perfetto, siamo fermi all'OEP!

Se adesso proviamo a dumpare il binario con Scylla, ci ritroveremo con un eseguibile che crasha alla prima call (teoricamente una GetVersion). Dobbiamo quindi capire cosa SecuROM ha fatto alle API usate dal gioco (abbiamo già trovato una chiamata sospetta quando abbiamo usato un breakpoint su GetCommandLineA).
Se state usando una VM, vi consiglio di creare uno snapshot dello stato attuale, cosi dopo aver compreso cosa succede alle API, possiamo velocemente ripristinare il tutto tornando all'OEP senza dover ricominciare dall'inizio (un grazie speciale va ad Antelox per avermi consigliato questa tecnica).

Steppiamo qualche istruzione sino ad arrivare alla prima call:

```
0044DCD2    57              push edi
0044DCD3    8965 E8         mov dword ptr ss:[ebp-18],esp
0044DCD6    FF15 00137400   call dword ptr ds:[741300]
0044DCDC    A3 5CCE4700     mov dword ptr ds:[47CE5C],eax
0044DCE1    A1 5CCE4700     mov eax,dword ptr ds:[47CE5C]
```

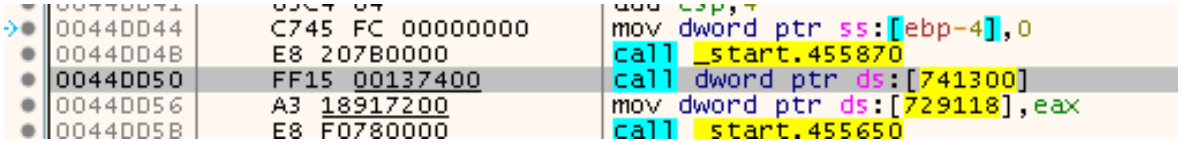
Sarebbe stato lecito aspettarsi una call a GetVersion, ma invece ci troviamo di fronte ad una call che ci porta ad una funzione situata nel segmento .cms_t.
Clicchiamo su step into per entrarci dentro e vediamone il comportamento.
Notiamo che siamo davanti ad una funzione particolare, sono anche presenti alcune chiamate a timeGetTime con probabilmente lo scopo di rilevare se stiamo trascorrendo del tempo steppando tra le istruzioni con il debugger.
Se scorriamo in basso, notiamo che prima del classico RET, c'è una istruzione molto sospetta, una jmp eax.

```
00730271    5E              pop esi
00730272    5B              pop ebx
00730273    8BE5            mov esp,ebp
00730275    5D              pop ebp
00730276    FFE0            jmp eax
00730278    5F              pop edi
00730279    5E              pop esi
0073027A    5B              pop ebx
0073027B    8BE5            mov esp,ebp
```

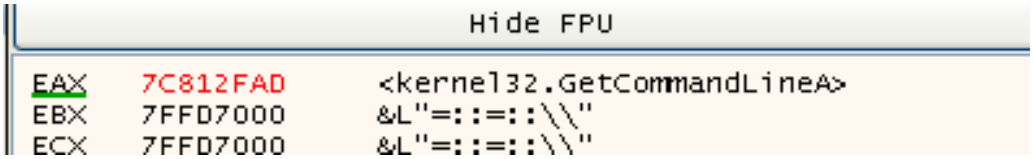
Questo già ci fa venire qualche idea su quello che sta per accadere (soprattutto se avete letto il mio documento tecnico su Laserlock!) 😊
Mettiamo un breakpoint sul jump e premiamo su RUN.
Una volta raggiunto il BP, controlliamo il registro eax:

```
Hide FPU
EAX   7C81126A   <kernel32.GetVersion>
EBX   7FFD7000   &L"=::=::\\"
ECX   00144D30
EDX   7C98B140   ntdll.7C98B140
EBP   0012F7AC
ESP   0012F730
```

Eccola qui, la chiamata che ci aspettavamo! Viene raggiunta proprio grazie a quel jump!
Possiamo fare una seconda prova cercando un'altra call a 0x00741300 subito dopo l'OEP. Se vi ricordate, mentre cercavamo l'OEP avevamo settato un breakpoint su GetCommandLineA e la chiamata dove l'esecuzione era stata bloccata, partiva da 0044DD50. Proprio a quell'indirizzo abbiamo una call a 0x00741300:



Raggiungiamo questo indirizzo e seguiamo la chiamata. Arriveremo ovviamente nuovamente nella funzione che abbiamo appena analizzato e alla fine ci ritroveremo nuovamente fermi sul jmp eax (se avete mantenuto il breakpoint attivo). Controlliamo adesso il registro eax:



Ed ecco la GetCommandLineA.

Adesso non abbiamo più alcun dubbio: SecuROM ha sostituito le API usate dal gioco con una sua funzione (situata a 0x00741300 nel segmeto .cms_t) che in base all'indirizzo di origine della chiamata, calcola l'API richiesta e la raggiunge con un salto.

Per chi avesse voglia di capire nel dettaglio come le API corrette vengono recuperate, può studiare la parte di disassemblato racchiusa dentro alla CriticalSection di quella funzione, facendo attenzione alle varie precauzioni adoperate per rendere il debugging più complicato (come la timeGetTime).

A questo punto sarebbe logico pensare di scrivere qualche riga di assembly per scorrere tutto il segmento text alla ricerca delle call di SecuROM, e dopo averle chiamarte per recuperare le giuste API, patcharle con gli indirizzi corretti appena ottenuti.

Tuttavia c'è un grosso problema: l'indirizzo contenuto in eax al momento del jump, non passa alla relativa thunk della IAT, ma è diretto sulla funzione richiesta! Non possiamo sostituirlo a quello della call a SecuROM in quanto cambierà essendo dinamico.

A noi serve l'indirizzo del thunk corrispondente nella IAT.

Quindi, l'idea è la seguente:

1) Scorriamo il segmento text alla ricerca delle call di SecuROM
2) Appena troviamo una call ci saltiamo dentro
3) Hookiamo il jump eax per tornare al nostro codice assembly (ottenendo l'indirizzo diretto della API che stiamo risolvendo)
4) Usando l'indirizzo diretto della API (contenuto in eax), scorriamo la IAT cercando il thunk corrispondente
5) Una volta trovato, patchiamo la funzione nel segmento text per chiamare l'indirizzo del thunk, sconfiggendo finalmente SecuROM.

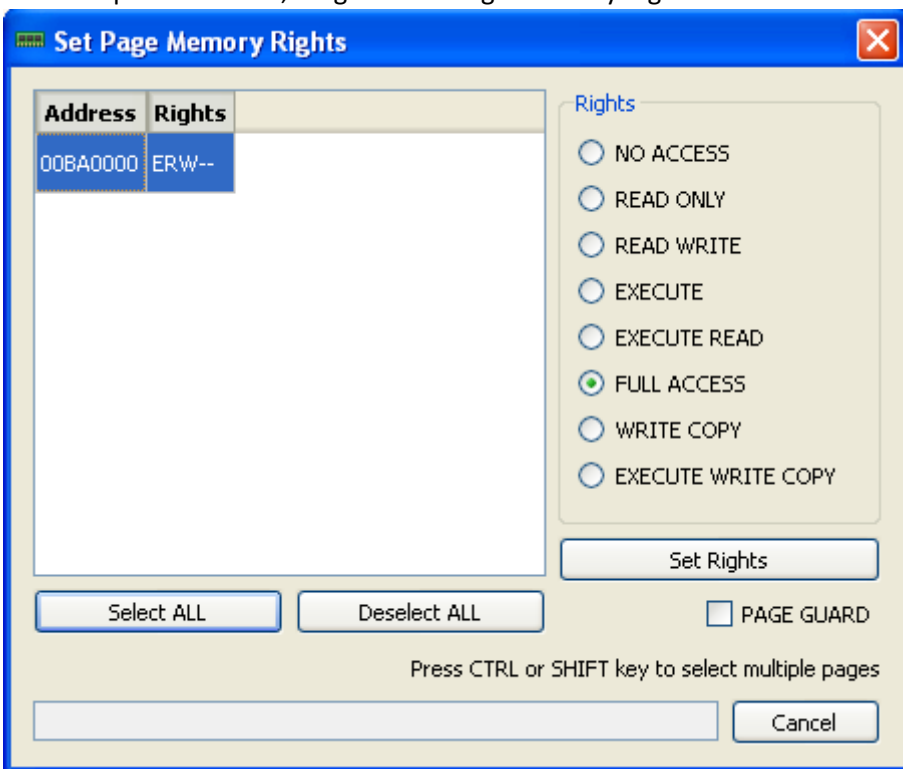
Trovare l'indirizzo di inizio della IAT è un'operazione estremamente facile: possiamo andare nella relativa sezione in Memory Map, selezionare il segmento .idata e cliccare su follow in dump e a quel punto vedere dove iniziano ad essere presenti gli indirizzi alle varie API:

```
0072A4B0 A3 CC 6F 73 1B CB 6F 73 00 00 00 00 00 00 00 00
0072A4C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0072A4D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0072A4E0 00 00 00 00 26 B1 21 72 00 00 00 00 00 00 00 00
0072A4F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0072A500 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0072A510 00 00 00 00 3B 47 E8 73 00 00 00 00 00 00 00 00
0072A520 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0072A530 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0072A540 00 00 00 00 C1 61 E4 77 69 5A E4 77 00 00 00 00
0072A550 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0072A560 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0072A570 00 00 00 00 00 00 00 00 BD FD 80 7C FA CA 81 7C
0072A580 EE 94 80 7C BF FC 80 7C 28 1A 80 7C 6B 23 80 7C
0072A590 E2 10 83 7C 12 FF 80 7C AD 2F 81 7C F5 60 83 7C
```

L'IAT inizia a 0x72A4B0 (736FCCA3 è l'indirizzo di DirectDrawCreate in questo caso). Come possiamo vedere questa IAT è un po' particolare, è piena di spazi liberi.
In alternativa avremmo potuto chiedere a Scylla di cercare l'indirizzo della IAT per noi.
Scorrendo un po' nel dump possiamo vedere che la IAT finisce all'indirizzo 0x72A87F:

```
0072A7E0 BA 7D B1 76 C1 79 B1 76 2B 84 B1 76 C2 80 B1 76
0072A7F0 92 82 B1 76 AC 7F B1 76 45 AA B1 76 A5 AD B0 76
0072A800 90 B0 B1 76 D4 02 B1 76 F8 94 B0 76 E1 95 B0 76
0072A810 B2 06 B1 76 E1 07 B1 76 E2 43 B1 76 E1 E6 B0 76
0072A820 FC FB B0 76 66 F6 B0 76 B6 5F B0 76 01 52 B0 76
0072A830 F3 05 B1 76 16 01 B1 76 00 00 00 00 00 00 00 00
0072A840 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0072A850 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0072A860 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0072A870 00 00 00 00 00 00 00 00 53 2A 4D 77 7E 05 4D 77
0072A880 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0072A890 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0072A8A0 00 00 00 00 00 00 00 00 00 00 00 00 E4 01 4D 75
0072A8B0 6C 74 69 42 79 74 65 54 6F 57 69 64 65 43 68 61
0072A8C0 72 00 64 02 53 65 74 45 72 72 6F 72 4D 6F 64 65
```

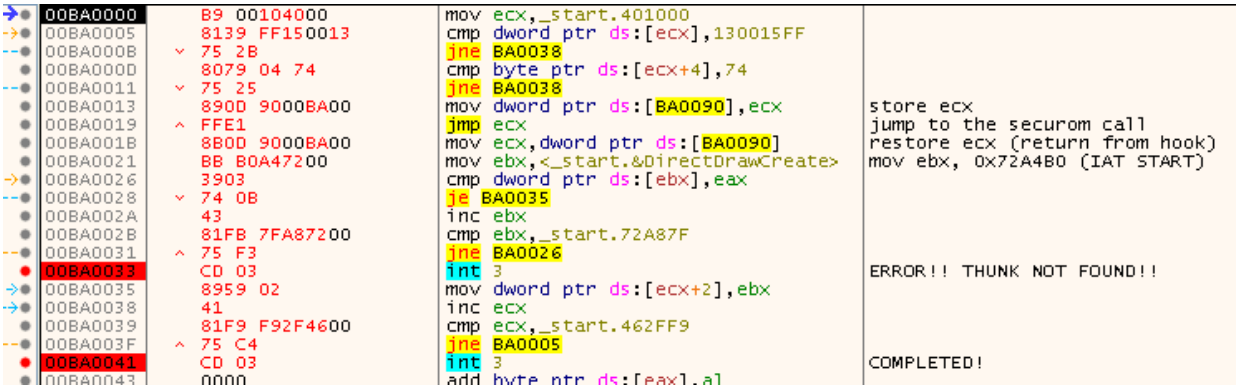
L'ultima cosa di cui abbiamo bisogno è trovare un po di spazio dove mettere il nostro codice assembly.
Dal tab MemoryMap scegliamo un'area di memoria che sia marcata a PRV, che sia libera e che sia sufficientemente grande. Io ho scelto quella che inizia d 0xBA0000. Prima di proseguire ricordatevi di cliccare sul destro su questa sezione, scegliere Set Page Memory Rights e selezionare Full Access:



Confermiamo premendo Set Rights.

Poiché andremo a patchare le call presenti nel segmento text (rimpiazzando la solita chiamata proxy di SecuROM con gli inidirizzi reali delle API), assicuriamoci che abbia il flag writeable.

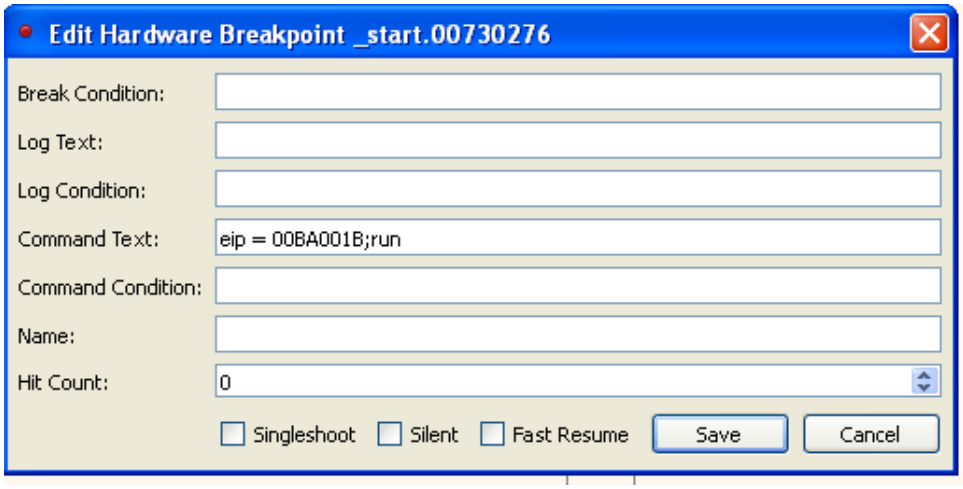
Siamo pronti per scrivere il nostro codice, posizionatevi a 0xBA0000 e copiate il seguente codice:



```
00BA0000    B9 00104000     mov ecx,_start.401000
00BA0005    8139 FF150013   cmp dword ptr ds:[ecx],130015FF
00BA000B    75 2B           jne BA0038
00BA000D    8079 04 74      cmp byte ptr ds:[ecx+4],74
00BA0011    75 25           jne BA0038
00BA0013    890D 9000BA00   mov dword ptr ds:[BA0090],ecx      store ecx
00BA0019    FFE1            jmp ecx                            jump to the securom call
00BA001B    8B0D 9000BA00   mov ecx,dword ptr ds:[BA0090]      restore ecx (return from hook)
00BA0021    BB B0A47200     mov ebx,<_start.&DirectDrawCreate> mov ebx, 0x72A4B0 (IAT START)
00BA0026    3903            cmp dword ptr ds:[ebx],eax
00BA0028    74 0B           je BA0035
00BA002A    43              inc ebx
00BA002B    81FB 7FA87200   cmp ebx,_start.72A87F
00BA0031    75 F3           jne BA0026
00BA0033    CD 03           int 3                              ERROR!! THUNK NOT FOUND!!
00BA0035    8959 02         mov dword ptr ds:[ecx+2],ebx
00BA0038    41              inc ecx
00BA0039    81F9 F92F4600   cmp ecx,_start.462FF9
00BA003F    75 C4           jne BA0005
00BA0041    CD 03           int 3                              COMPLETED!
00BA0043    0000            add byte ptr ds:[eax],al
```

Questo codice fa esattamente le seguenti operazioni:
Imposta in ecx l'indirizzo di origine della sezione text. I byte vengono quindi confrontati con FF15001374, ovvero con la call alla chiamata di SecuROM (effettuata in due cmp distinti per controllare tutti i byte relativi). Se non c'è una corrispondenza il jne verrà seguito e l'indirizzo in ecx sarà incrementato di 1 per controllare il byte successivo.
Se invece siamo in presenza di una chiamata di SecuROM, salviamo l'indirizzo attuale di ecx (che ci dice a che byte della sezione .text siamo arrivati) e ci saltiamo dentro.
Una volta che raggiungeremo l'ormai famoso jmp eax contenuto nella funzione di SecuROM imposteremo un hook (tra qualche minuto) per tornare automaticamente a 0xBA001B. A questo punto in eax abbiamo l'indirizzo diretto dell'API richiesta. Ripristiniamo il registro ecx e carichiamo nel registro ebx l'indirizzo di inizio della IAT. Scorriamo la IAT sino a quando non troviamo la thunk che punta all'indirizzo contenuto in eax (ebx verrà ovviamente incrementato di volta in volta). Se nessuna thunk contiene l'API che stiamo cercando di risolvere, allora siamo in grossi guai (INT 3 a 0xBA0033), ma ovviamente questo NON dovrebbe accadere. Se la thunk è stata trovata, salteremo all'indirizzo 0xBA0035, dove sostituiremo i byte relativi alla call di securom, con quelli del thunk corretto. A questo punto si può tornare a controllare il segmento .text cercando le restanti call. Quando il segmento .text finirà (ecx sarà 462FF9, ovvero l'ultimo address del segmento –6, che è la grandezza dei byte della call), abbiamo finito e possiamo procedere con il dump.

Prima di avviare il codice ricordatevi di cliccare con il tasto destro a riga BA0000 e scegliere Set New Origin Here. Adesso settiamo il nostro hook: ci spostiamo a 0x730276 (indirizzo dove è presente il jump eax) e clicchiamo con il destro e scegliamo Breakpoint->Set Hardware on Execution.
Spostiamoci nel tab Breakpoints, clicchiamo con il destro sul breakpoint hardware appena creato e scegliamo Edit. Configuriamolo cosi:



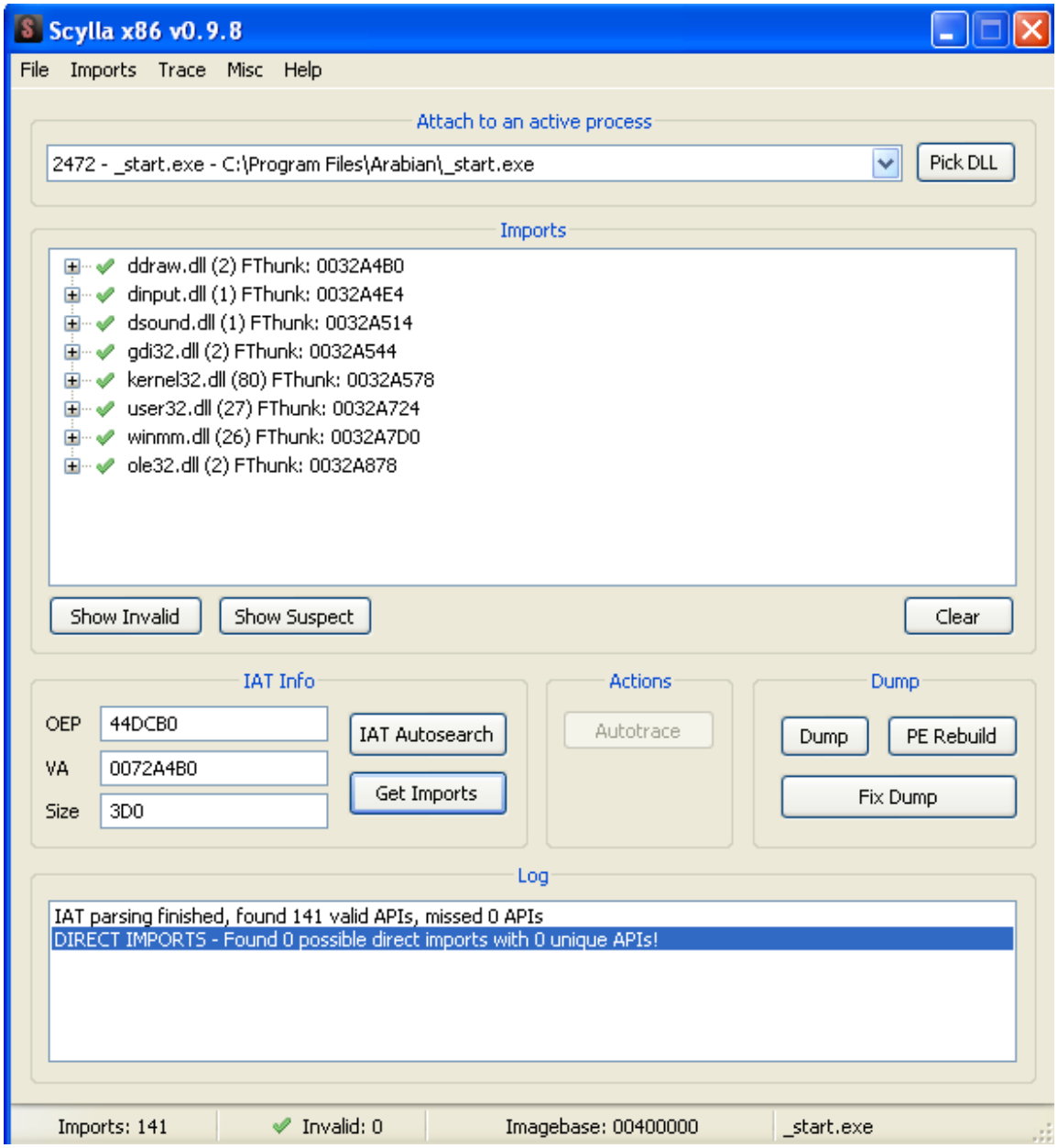
In questo modo una volta scattato il BP, torneremo automaticamente al nostro codice (all'indirizzo 0xBA001B).
NB: usare un breakpoint hardware in questo caso, altrimenti il programma andrà in crash (verrà rilevata la presenza di breakpoint software).

Siamo pronti per avviare il nostro codice, spostiamoci a 0xBA0000 e clicchiamo su RUN.

Una volta completata l'esecuzione saremo fermi a 0xBA0041:

```
00BA0000    B9 00104000          mov ecx,_start.401000
00BA0005    8139 FF150013        cmp dword ptr ds:[ecx],130015FF
00BA000B  v 75 2B                jne BA0038
00BA000D    8079 04 74           cmp byte ptr ds:[ecx+4],74
00BA0011  v 75 25                jne BA0038
00BA0013    890D 9000BA00        mov dword ptr ds:[BA0090],ecx      store ecx
00BA0019  ^ FFE1                 jmp ecx                            jump to the secu
00BA001B    8B0D 9000BA00        mov ecx,dword ptr ds:[BA0090]      restore ecx (ret
00BA0021    BB B0A47200          mov ebx,<_start.&DirectDrawCreate> mov ebx, 0x72A4E
00BA0026    3903                 cmp dword ptr ds:[ebx],eax
00BA0028  v 74 0B                je BA0035
00BA002A    43                   inc ebx
00BA002B    81FB 7FA87200        cmp ebx,_start.72A87F
00BA0031  ^ 75 F3                jne BA0026
00BA0033    CD 03                int 3                              ERROR!! THUNK NO
00BA0035    8959 02              mov dword ptr ds:[ecx+2],ebx
00BA0038    41                   inc ecx
00BA0039    81F9 F92F4600        cmp ecx,_start.462FF9
00BA003F  ^ 75 C4                jne BA0005
00BA0041    CD 03                int 3                              COMPLETED!
00BA0043    0000                 add byte ptr ds:[eax],al
00BA0045    0000                 add byte ptr ds:[eax],al
00BA0047    0000                 add byte ptr ds:[eax],al
00BA0049    0000                 add byte ptr ds:[eax],al
```

Ci siamo quasi! Apriamo Scylla, scegliamo il processo giusto (_start.exe), settiamo l'OEP, l'indirizzo della IAT e la sua dimensione:



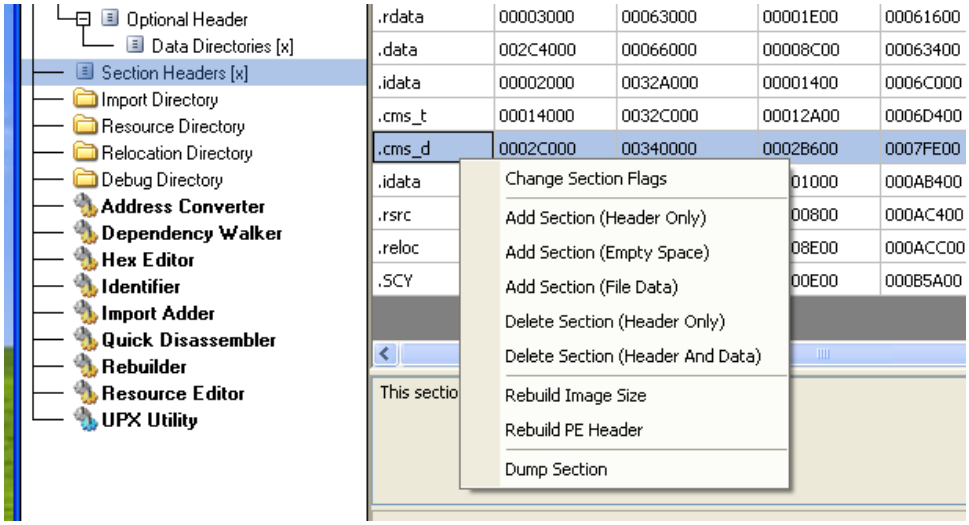
Clicchiamo quindi su Get Imports, poi su Dump e per finire su Fix Dump.

Avremo un eseguibile libero da SecuROM 😀

Completiamo il lavoro:

Il nostro eseguibile funziona perfettamente e SecuROM è solo un ricordo. Tuttavia se vogliamo essere dei veri perfezionisti, possiamo renderlo leggermente più piccolo rimuovendo quelle due sections usate dal loader di SecuROM e che adesso occupano solo spazio inutile.

Carichiamo il nostro binario in CFF Explorer e spostiamoci nel tab Section Headers. Selezioniamo le sezioni .cms_t e .cms_d, clicchiamo con il destro e scegliamo Delete Section (Header And Data):



Salviamo il nuovo eseguibile (sarà nettamente più piccolo) e abbiamo FINITO! :D

Credits:

Vorrei nuovamente ringraziare il mitico Antelox per avermi suggerito la tecnica degli snapshots su VM. Indubbiamente un ottimo sistema per risparmiare un sacco di tempo.

Conclusioni:

Come abbiamo visto SecuROM *new* 4.48.00.004 condivide concettualemente qualcosa con Laserlock.
In definitiva è un DRM molto didattico dal quale sicuramente abbiamo imparato qualcosa.
Grazie per aver letto questo documento 😊
Luca