

C.9 DEZASAMBLARE SI PATCHING CU X96DBG

PAUL A. GAGNIUC

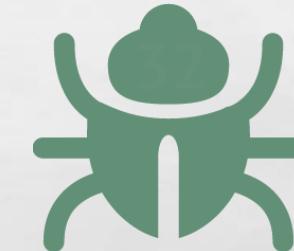


Academia Tehnică Militară „Ferdinand I”

PRINCIPALELE PĂRȚI ALE PREZENTĂRII

C.9 Dezasamblare și Patching cu X64dbg:

- **9.1 DEZASAMBLARE ȘI PETICIRE (CALL VS JMP)**
- **9.2 EXECUȚIA DE COD ÎN SECȚIUNEA .DATA**
- **9.3 MODIFICAREA STRINGURILOR IN OBIECTE**
- **9.4 MODIFICAREA STRINGURILOR IN MESAJE**
- **9.5 SCHIMBAREA MASINII VIRTUALE**



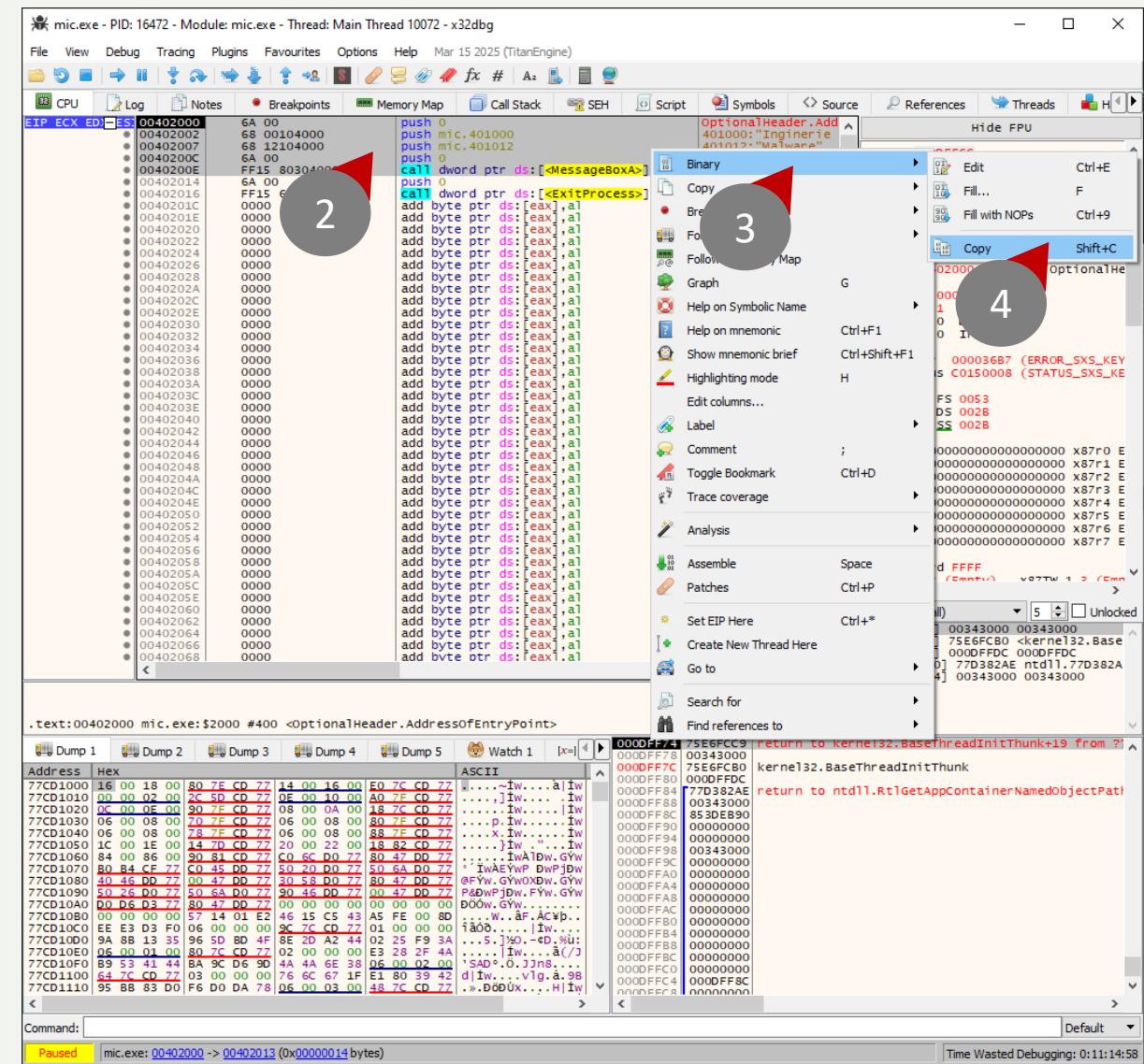
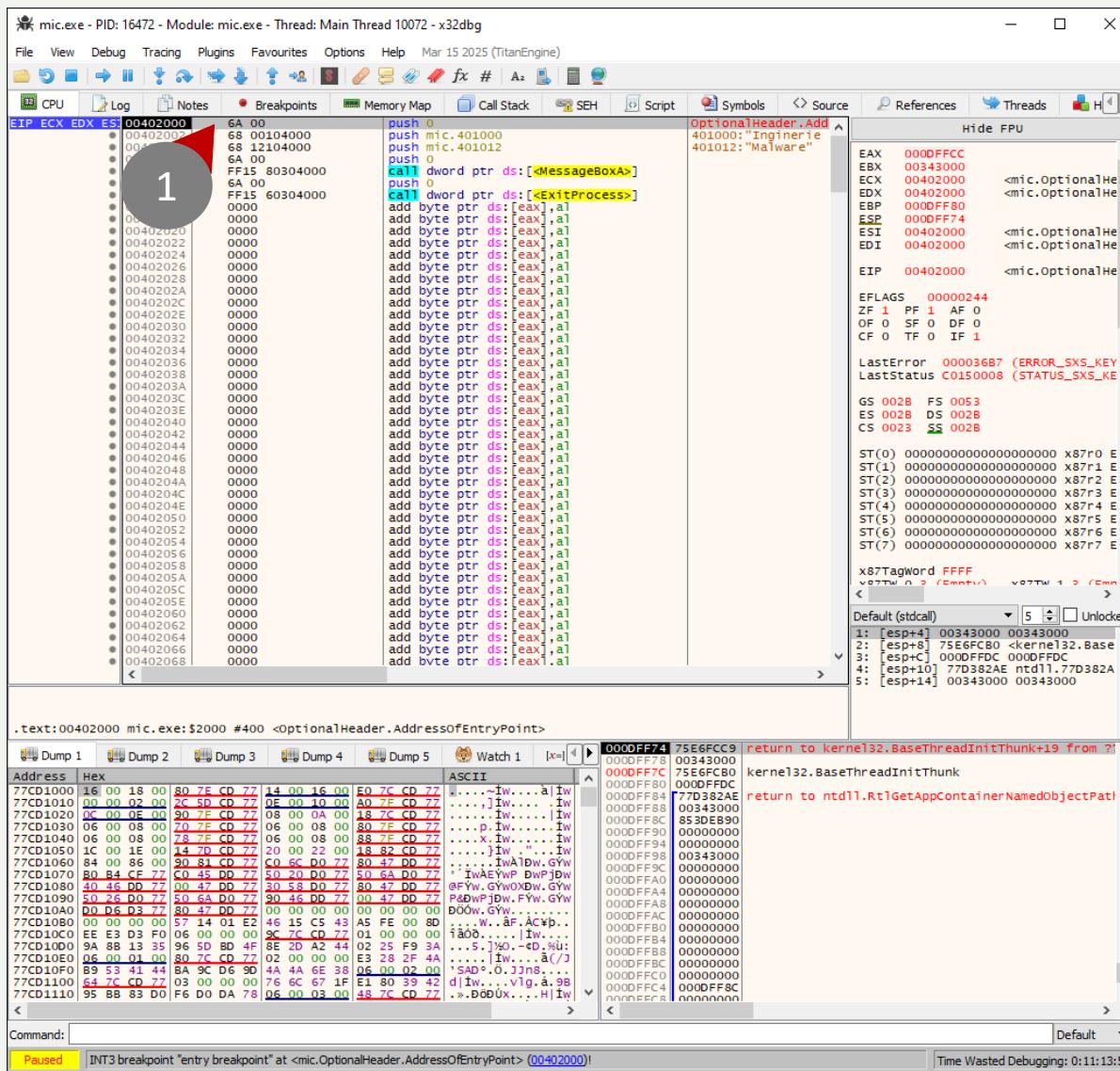
9.1

DEZASAMBLARE ȘI PETICIRE (CALL VS JMP)



1. Fișierul mic.exe este încărcat și oprit la adresa Entry Point (00402000), unde se observă începutul execuției: instrucțiuni standard de inițializare. Aceasta este zona de start a codului propriu al programului.

2. Se selectează manual blocul de cod dorit.
3. Click dreapta pe blocul selectat, de unde se alege opțiunea „Binary”.
4. Se apasă „Copy” pentru a salva codul în format binar.



Este prezentată prima linie din Entry Point în mic.exe!

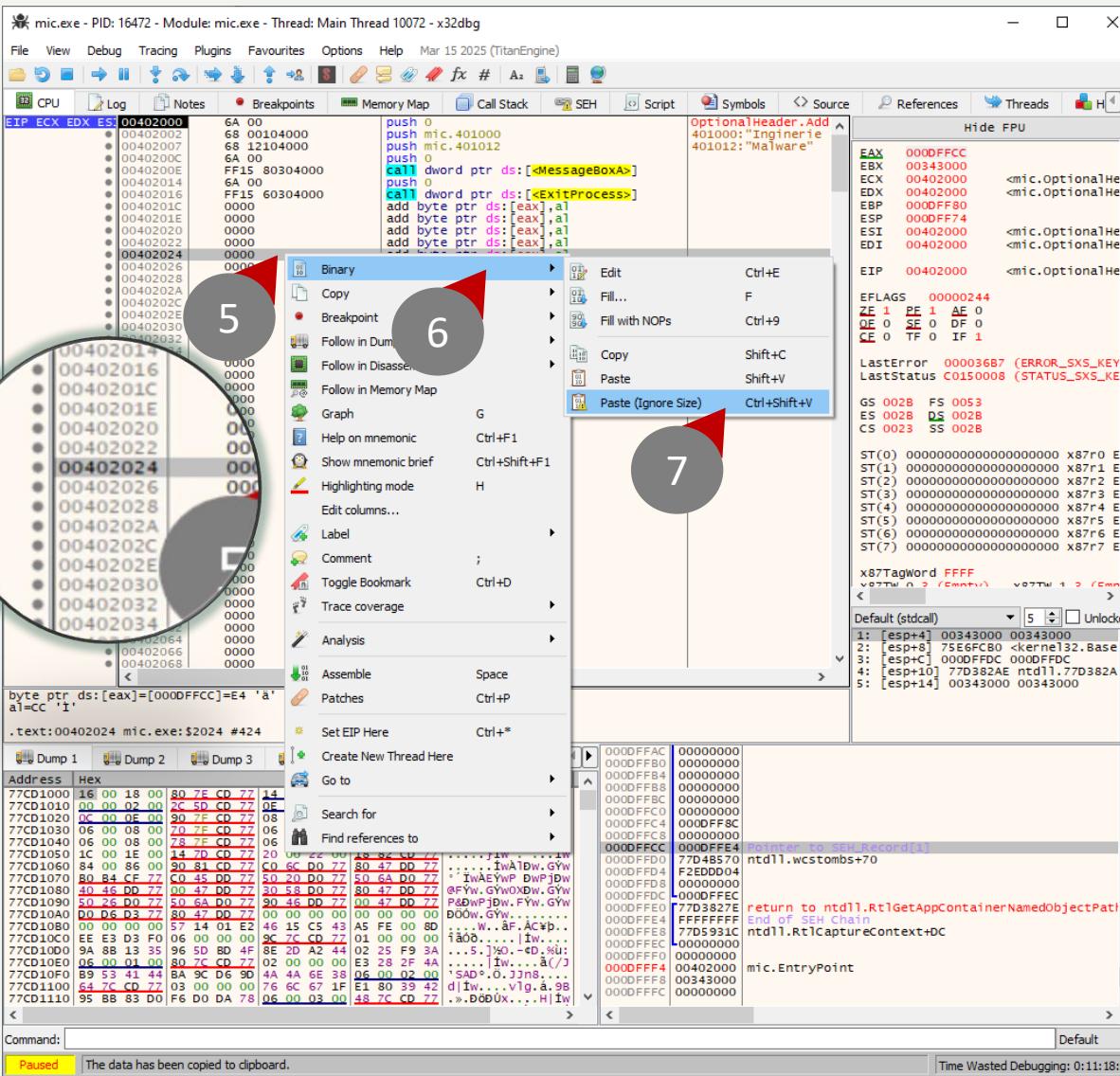
Selectăm o secțiune de cod universal funcțional și îl copiem!

5. Ne poziționăm într-o zonă liberă din secțiunea .text, după codul existent.

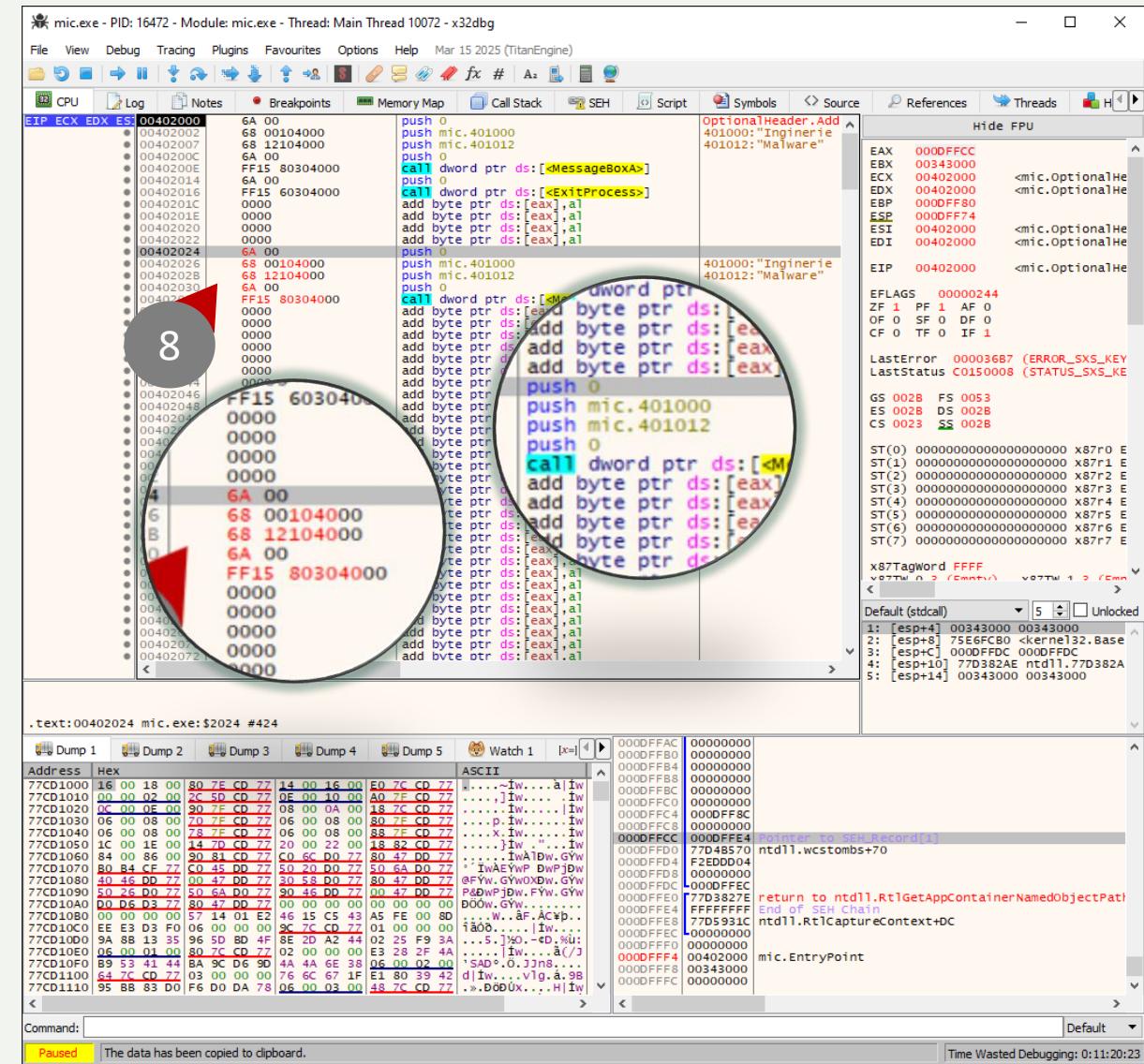
6. Click dreapta pentru a deschide meniul contextual.

7. Alegem opțiunea Paste (Ignore size) din submenuul „Binary” – pentru a lipi codul fără a ține cont de dimensiunea inițială a instrucțiunilor.

8. Codul selectat anterior este duplicat corect în regiunea aleasă, exact cum a fost copiat.



Apoi, codul copiat îl introducem într-o zonă liberă...



După apăsarea submeniului Paste...

The screenshot shows the x32dbg debugger interface. The assembly window displays the following code:

```

EIP ECX EDX ES 00402000
00402002 6A 00 push 0
00402007 68 00104000 push mic.401000
0040200C 68 12104000 push mic.401012
0040200E FF15 80304000 call dword ptr ds:[<MessageBoxA>]
00402014 6A 00 push 0
00402016 FF15 60304000 call dword ptr ds:[<ExitProcess>]
0040201C 0000 add byte ptr ds:[eax],al
0040201E 0000 add byte ptr ds:[eax],al
00402020 0000 add byte ptr ds:[eax],al
00402022 0000 add byte ptr ds:[eax],al
00402024 6A 00 push 0

```

A context menu is open at address 00402024, with the following steps highlighted:

- Facem click dreapta pe prima instrucțiune din codul duplicat (nou introdus).
- Din meniu alegem opțiunea Copy.
- Apoi selectăm Address, pentru a copia adresa exactă la care începe noua secvență de cod.

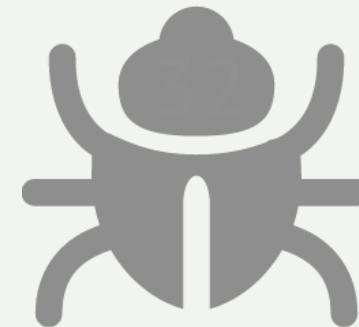
The memory dump window shows the following data starting at address 00402024:

Address	Hex	Description
77CD1000	16 00 18 00 80 7E	
77CD1010	00 00 02 00 2C 5D	
77CD1020	0C 00 0E 00 90 7F	
77CD1030	06 00 08 00 70 7E	
77CD1040	06 00 08 00 78 7E	
77CD1050	1C 00 1E 00 14 7D	
77CD1060	84 00 86 00 90 81 CD 77 C0 6C D0 77 80 47 DD 77	In AEW_PDW.GYW
77CD1070	80 84 CF 77 C0 45 DD 77 50 20 D0 77 50 6A D0 77	In AEYWP_DWPjDW
77CD1080	40 46 DD 77 00 47 DD 77 30 58 D0 77 80 47 DD 77	@FYw.GYwOXDW.GYW
77CD1090	50 26 D0 77 50 6A D0 77 90 46 DD 77 00 47 DD 77	P&DWjDW.FYW.GYW
77CD10A0	DD D6 D3 77 80 47 DD 77 00 00 00 00 00 00 00 00	D0DW.GYW
77CD10B0	00 00 00 00 57 14 01 E2 46 15 C5 43 A5 FE 00 8D	...W..AF.AC#P...
77CD10C0	EE E3 D3 F0 00 00 00 9C 7C CD 77 01 00 00 00 00 00	iæo.].Iw...å(/J
77CD10D0	9A 88 13 35 95 5D BB 4F 8E 2D A2 44 02 25 F9 3A ... 5.]æo.-cD.ºù;	
77CD10E0	06 00 01 00 80 7C CD 77 02 00 00 00 E3 28 2F 4A	
77CD10F0	B9 53 41 44 BA 9C D6 9D 4A 4A 6E 38 06 00 02 00	SAD.º.O.JJns...
77CD1100	64 7C CD 77 03 00 00 00 76 6C 67 1F E1 80 39 42	d Iw...vlg.å.98
77CD1110	95 BB 83 D0 F6 D0 DA 78 06 00 03 00 48 7C CD 77	»DöDÜx...H Iw...

The command bar at the bottom shows: Paused mic.exe: 00402024 -> 00402025 (0x00000002 bytes)

- Facem click dreapta pe prima instrucțiune din codul duplicat (nou introdus).
- Din meniu alegem opțiunea Copy.
- Apoi selectăm Address, pentru a copia adresa exactă la care începe noua secvență de cod.

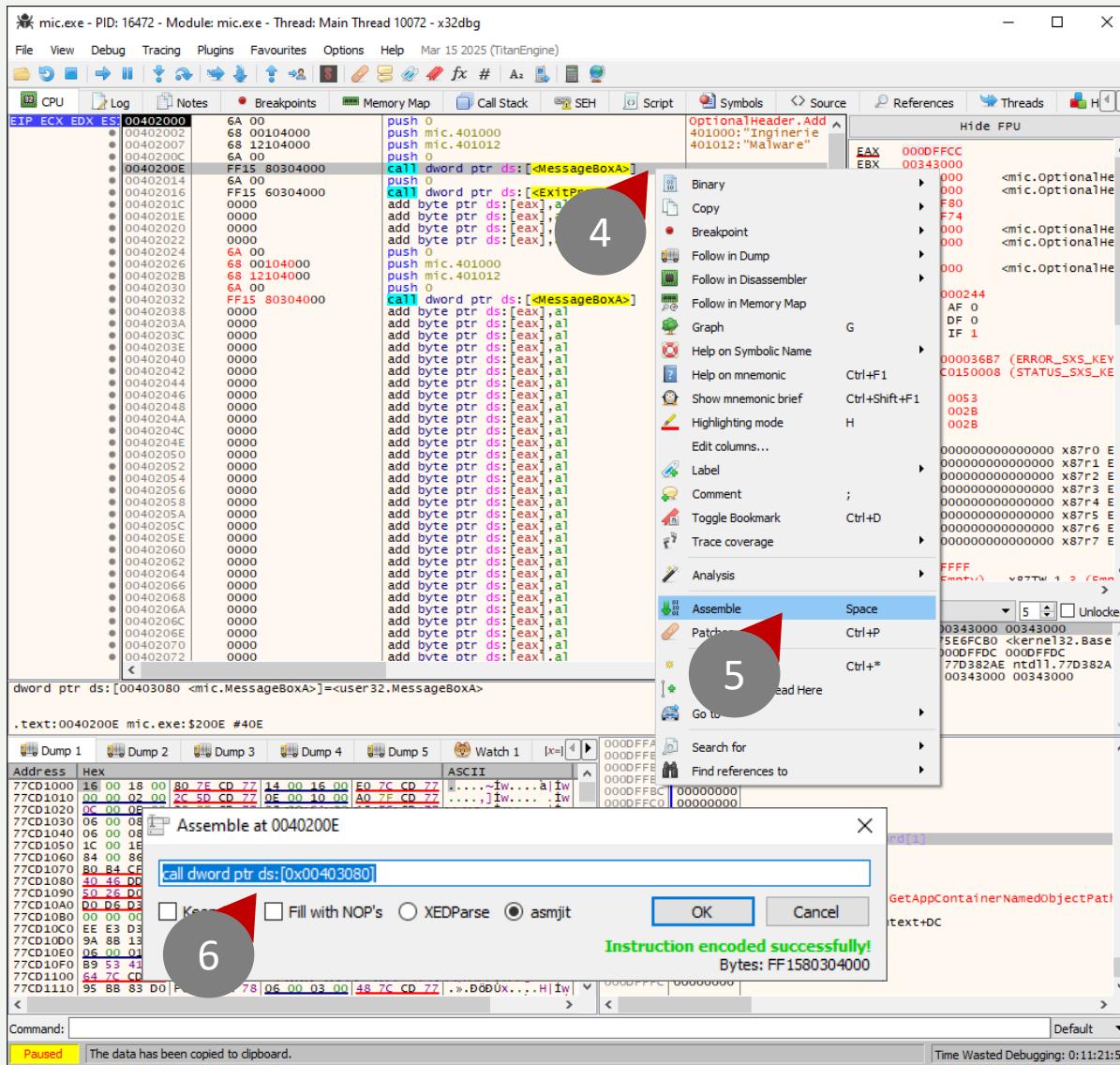
Copiem adresa de început a codului duplicat, după care vom folosi această adresă pentru a redirectiona fluxul de execuție. Astfel, programul va sări către noul cod injectat.



4. Se face click dreapta pe instrucțiunea CALL din codul original.

5. Se alege opțiunea Assemble pentru a edita instrucțiunea.

6. În fereastra de editare, se înlocuiește instrucțiunea (ex: call dword ptr ds:[0x00403080]) cu un call către adresa unde am copiat codul.

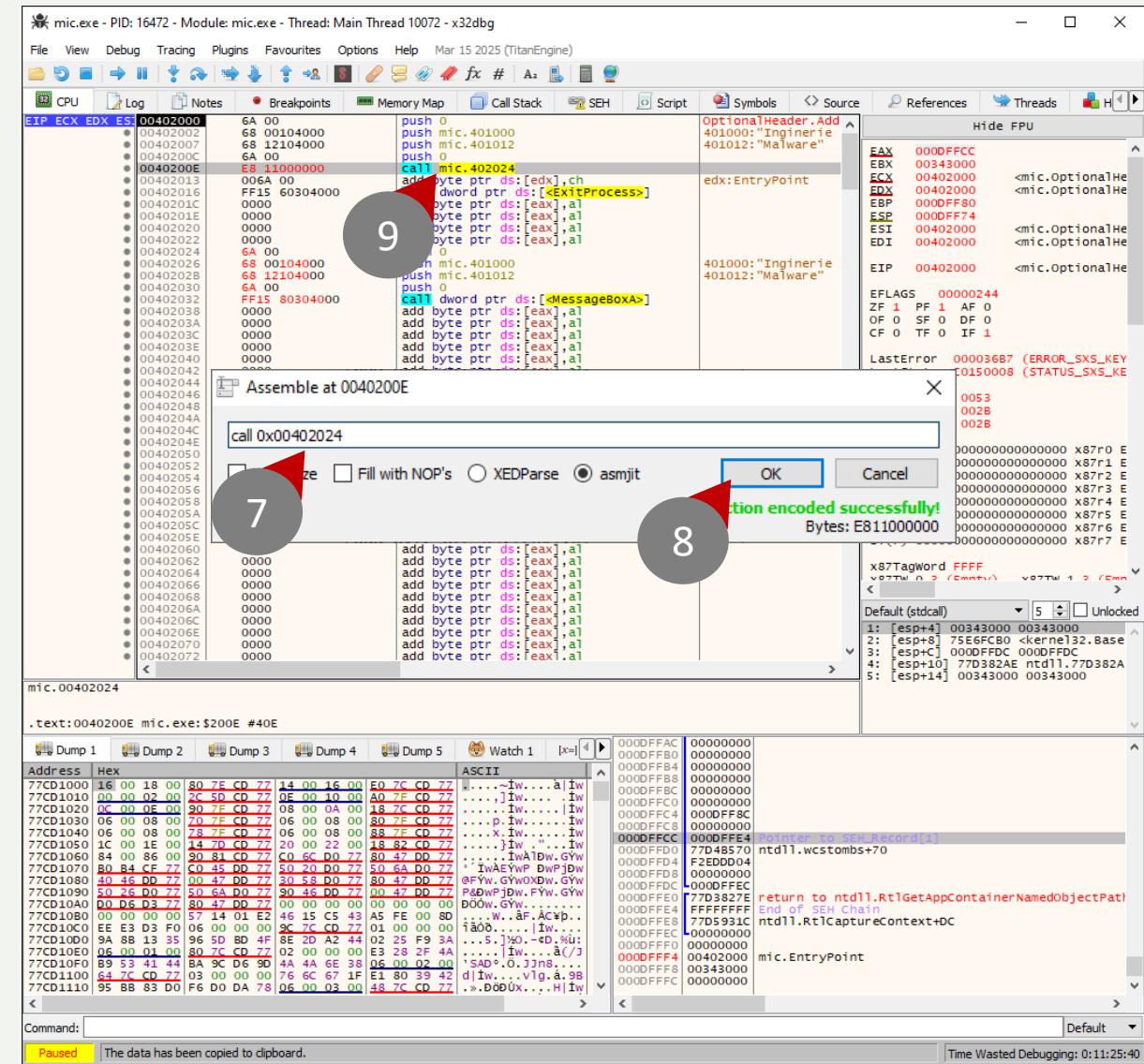


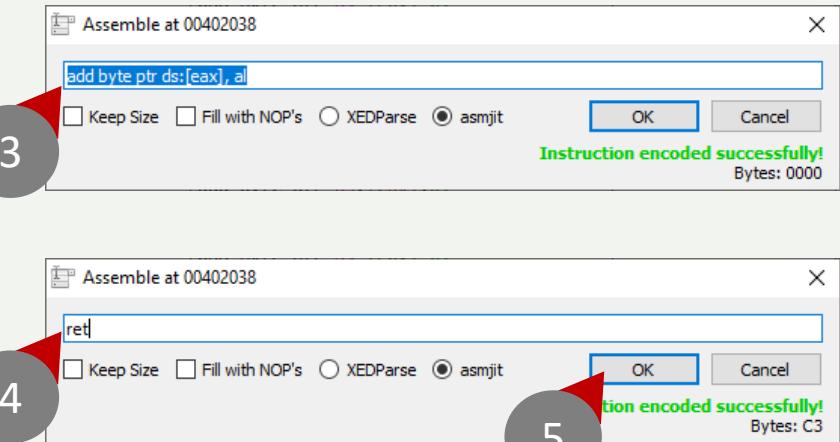
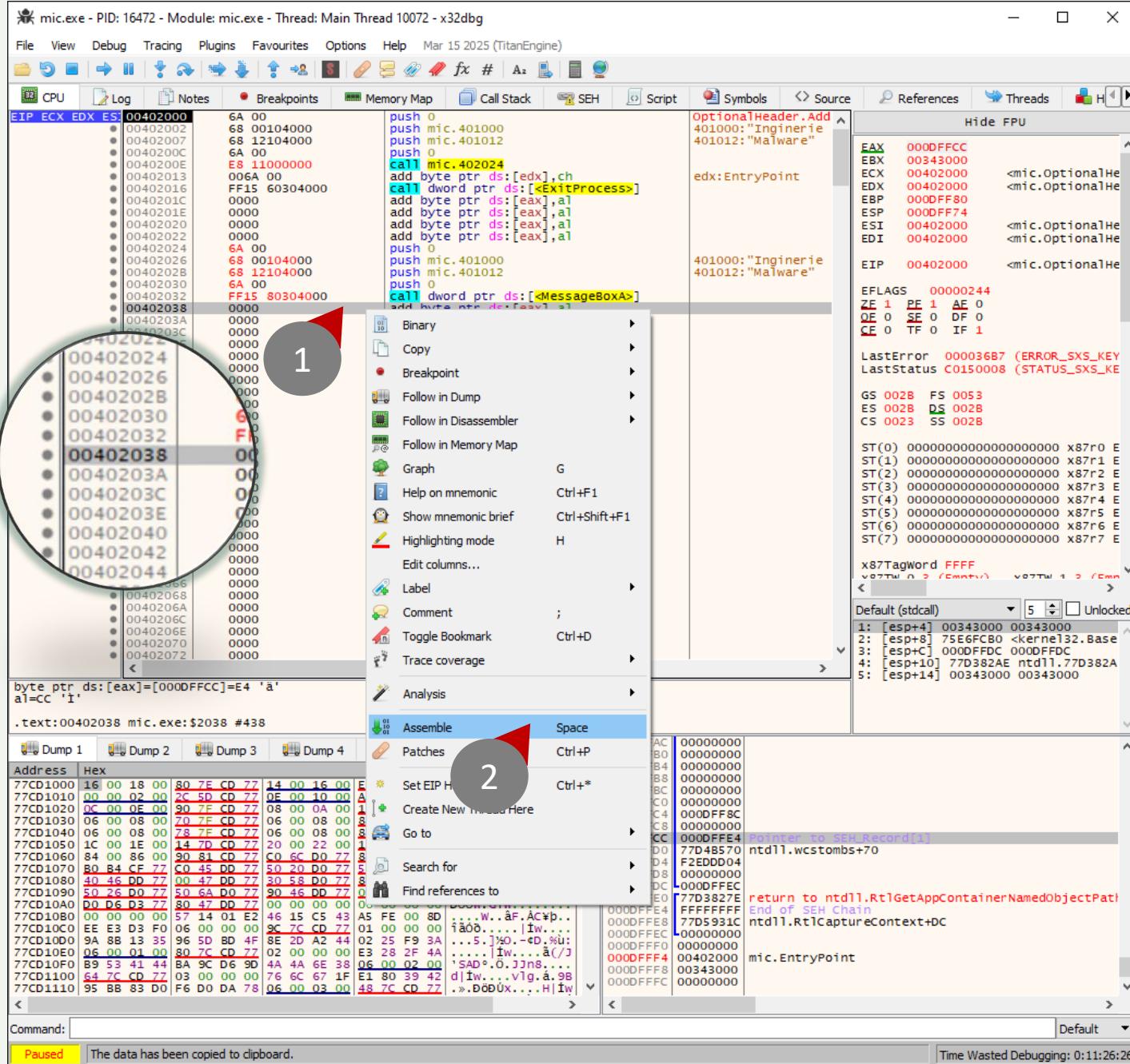
Valoarea 0x00403080 este adresa de sub eticheta “<MessageBoxA>” ...

7. Se scrie adresa codului duplicat (ex: 0x00402024) în câmpul de editare.

8. Se apasă OK – instrucția este validată și înlocuită cu succes.

9. Fluxul de execuție va fi redirectonat către codul introdus manual.





3

4

5

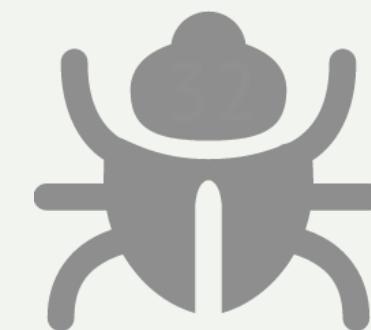
- Facem click dreapta pe adresa imediat după codul duplicat.
- Alegem opțiunea Assemble pentru a o modifica.
- Apare în clar valoarea de la acea adresă (ex: add byte ptr ds:[eax], al).
- Instrucțiunea „add byte ptr ds:[eax]” este înlocuită cu ret — pentru a reveni la fluxul inițial de execuție.
- Apăsăm OK pentru a salva modificarea.

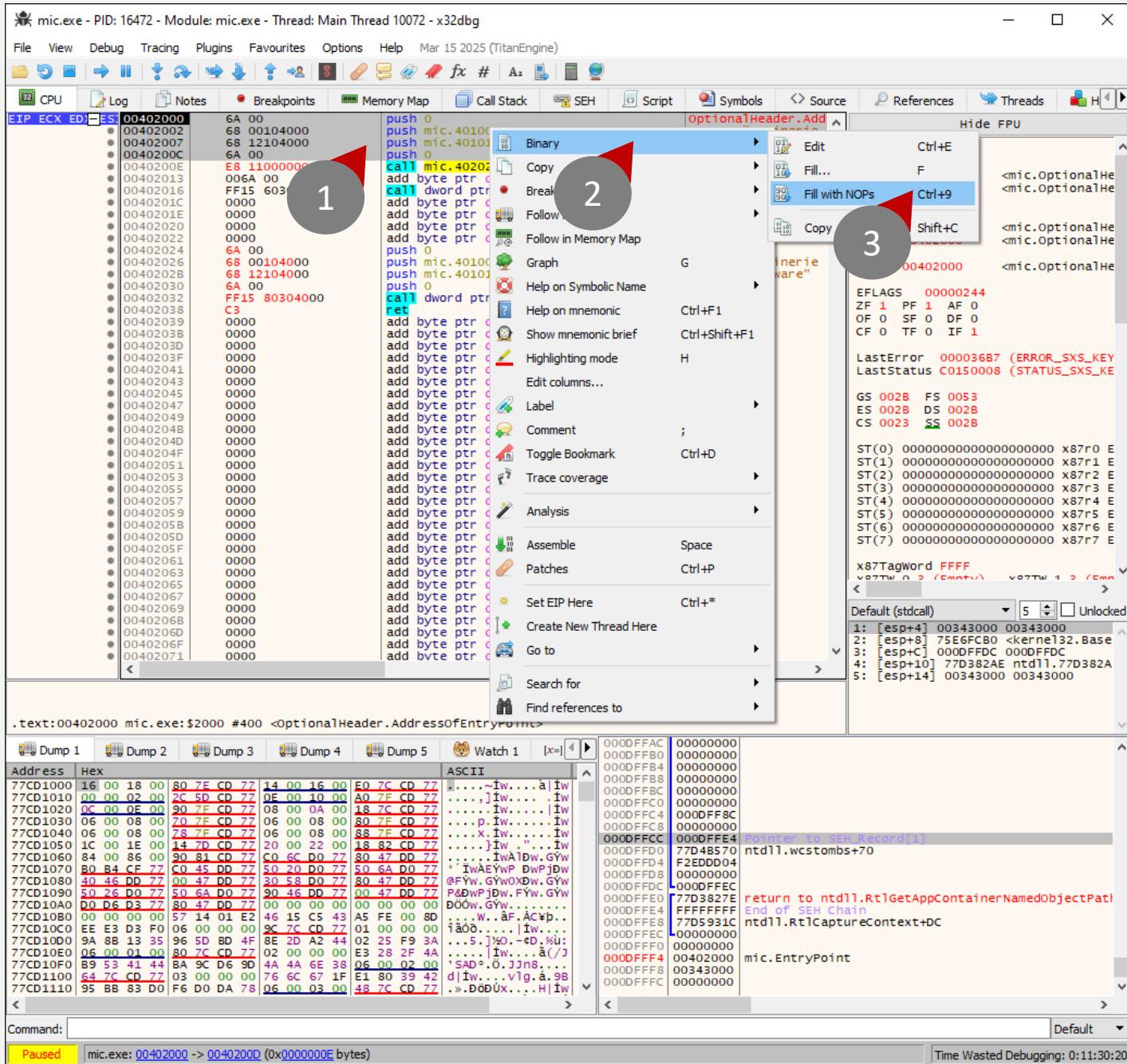
În acest pas final, ne asigurăm că execuția codului injectat se încheie corect. Adăugarea instrucțiunii ret permite revenirea la punctul de unde a fost făcut apelul (call), menținând stabilitatea programului.

The screenshot shows the x32dbg debugger interface. The CPU register pane at the top displays the EIP, ECX, EDX, and ES registers. The assembly pane shows the instruction flow starting from address 00402000, which contains a series of pushes and calls to internal functions like mic.401000 and mic.401012. A specific instruction at address 00402024 is highlighted in yellow, labeled 'call mic.402024'. The Registers pane shows various CPU flags and memory pointers. The Stack pane at the bottom shows memory dump sections for Dump 1 through Dump 11, with the current dump being Dump 1. The Dump 1 section shows memory addresses from 77CD1000 to 77CD1110, containing various ASCII and hex values.

6. Execuția a fost redirecționată cu succes către codul duplicat, care este acum activ. Instrucțiunea call plasată anterior la Entry Point duce exact aici, la adresa 00402024. Instrucțiunile din zona duplicată se execută normal, iar la final se revine cu ret.

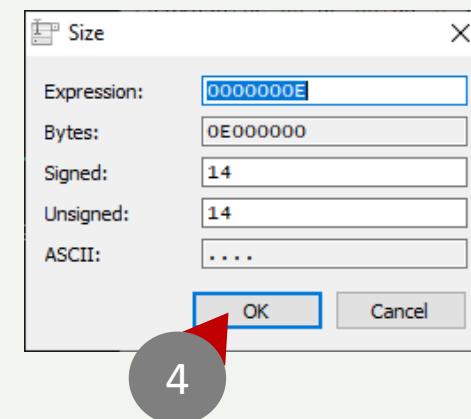
Execuția a fost deturnată cu succes. Programul sare de la Entry Point către codul introdus manual (care conține funcțiile MessageBoxA & ExitProcess). Acest mecanism este util pentru testare, injectare de cod sau bypass-uri simple. Exemplul este model clasic de bootstrapping (este rulat primul) în analiza malware.





Selectarea și copierea blocului de cod

1. Se selectează zona de cod care trebuie neutralizată.
 2. Click dreapta pe selecție → Binary.
 3. Se alege Fill with NOPs pentru a umple blocul de cod cu instrucțiuni NOP (No Operation).



4. Apare fereastra „Size”, în care se confirmă dimensiunea blocului. Este afișată dimensiunea exactă (în bytes) a blocului selectat. Se confirmă cu OK, iar codul este înlocuit în memorie cu instrucțiuni NOP (0x90), care nu au niciun efect la execuție.

5

```

mic.exe - PID: 16472 - Module: mic.exe - Thread: Main Thread 10072 - x32dbg
File View Debug Tracing Plugins Favourites Options Help Mar 15 2025 (TitanEngine)
CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols References Threads Help
EIP ECX EDX ES 00402000 90 nop
00402001 90 nop
00402002 90 nop
00402003 90 nop
00402004 90 nop
00402005 90 nop
00402006 90 nop
00402007 90 nop
00402008 90 nop
00402009 90 nop
0040200A 90 nop
0040200B 90 nop
0040200C 90 nop
0040200D 90 nop
0040200E E8 11000000 call mic.402024
00402013 006A 00 add byte ptr ds:[edx],ch
00402016 FF15 60304000 call dword ptr ds:[<ExitProcess>]
0040201C 0000 add byte ptr ds:[eax],al
0040201E 0000 add byte ptr ds:[eax],al
00402020 0000 add byte ptr ds:[eax],al
00402022 0000 add byte ptr ds:[eax],al
00402024 6A 00 push 0
00402026 68 12104000 push mic.401000
00402028 68 12104000 push mic.401012
00402030 6A 00 push 0
00402032 FF15 80304000 call dword ptr ds:[<MessageBoxA>]
00402038 C3 ret
00402039 0000 add byte ptr ds:[eax],al
0040203B 0000 add byte ptr ds:[eax],al
0040203D 0000 add byte ptr ds:[eax],al
0040203F 0000 add byte ptr ds:[eax],al
00402041 0000 add byte ptr ds:[eax],al
00402043 0000 add byte ptr ds:[eax],al
00402045 0000 add byte ptr ds:[eax],al
00402047 0000 add byte ptr ds:[eax],al
00402049 0000 add byte ptr ds:[eax],al
0040204D 0000 add byte ptr ds:[eax],al
0040204F 0000 add byte ptr ds:[eax],al
00402051 0000 add byte ptr ds:[eax],al
00402053 0000 add byte ptr ds:[eax],al
00402055 0000 add byte ptr ds:[eax],al
00402057 0000 add byte ptr ds:[eax],al
00402059 0000 add byte ptr ds:[eax],al
0040205B 0000 add byte ptr ds:[eax],al
0040205D 0000 add bvte ptr ds:[eax],al
OptionalHeader.Add Hide FPU
EAX 000DFFCC <mic.OptionalHe
EBX 00343000 <mic.OptionalHe
ECX 00402000 <mic.OptionalHe
EDX 00402000 <mic.OptionalHe
EBP 000DFF80 <mic.OptionalHe
ESP 000DFF74 <mic.OptionalHe
ESI 00402000 <mic.OptionalHe
EDI 00402000 <mic.OptionalHe
EIP 00402000 <mic.OptionalHe
EFLAGS 00000244
ZF 1 PF 1 AF 0
OF 0 SF 0 DF 0
CF 0 TF 0 IF 1
LastError 000036B7 (ERROR_SXS_KEY)
LastStatus C0150008 (STATUS_SXS_KE
GS 002B FS 0053
ES 002B DS 002B
CS 0023 SS 002B
ST(0) 00000000000000000000000000000000 x87r0 E
ST(1) 00000000000000000000000000000000 x87r1 E
ST(2) 00000000000000000000000000000000 x87r2 E
ST(3) 00000000000000000000000000000000 x87r3 E
ST(4) 00000000000000000000000000000000 x87r4 E
ST(5) 00000000000000000000000000000000 x87r5 E
ST(6) 00000000000000000000000000000000 x87r6 E
ST(7) 00000000000000000000000000000000 x87r7 E
x87TagWord FFFF
Default (stdcall) ▾ 5 Unlocked
1: [esp+4] 00343000 00343000
2: [esp+8] 75E6FCB0 <kernel32.Base
3: [esp+C] 000DFFDC 000DFFDC
4: [esp+10] 77D382AE ntdll.77D382A
5: [esp+14] 00343000 00343000
Information 20/20 patch(es) applied!
OK
Select All Deselect All Restore Selected
Import Export Pick Groups Patch File

```

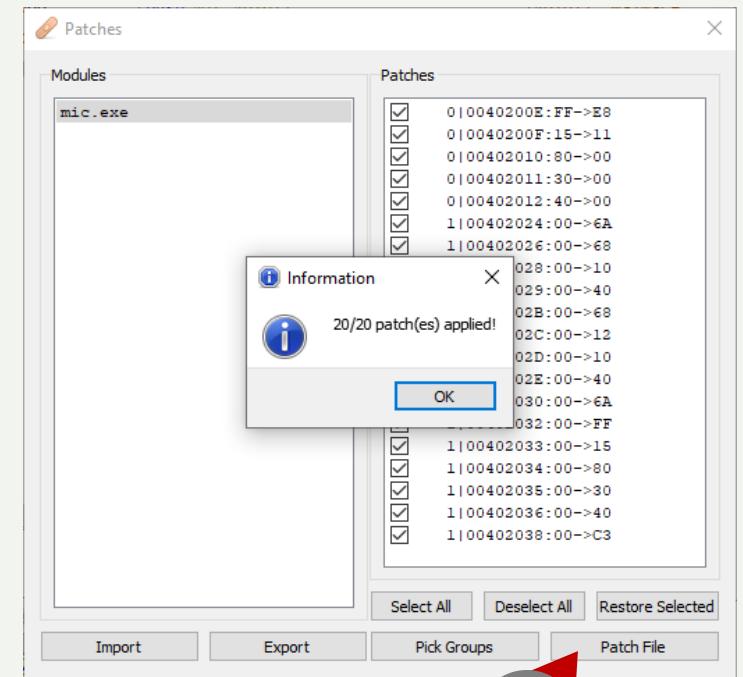
.text:00402005 mic.exe:\$2005 #405

Address	Hex	ASCII
77CD1000	16 00 18 00	80 7E CD 77 14 00 16 00
77CD1010	00 00 02 00	2C 5D CD 77 0E 00 10 00
77CD1020	00 00 0E 00	90 7F CD 77 08 00 0A 00
77CD1030	06 00 08 00	70 7F CD 77 06 00 08 00
77CD1040	06 00 08 00	78 7F CD 77 06 00 08 00
77CD1050	1C 00 1E 00	14 7D CD 77 20 00 22 00
77CD1060	84 00 86 00	90 81 CD 77 C0 6C D0 77
77CD1070	80 84 CF 77	80 47 DD 77 50 6A DD 77
77CD1080	40 46 DD 77	50 47 DD 77 50 58 D0 77
77CD1090	50 26 D0 77	50 46 DD 77 00 47 DD 77
77CD10A0	D0 D6 D3 77	80 47 DD 77 00 00 00 00
77CD10B0	00 00 00 00	57 14 01 E2 46 15 C5 43
77CD10C0	EE E3 F0	A5 FE 00 8D 5... .W. àC#P. .
77CD10D0	9A 88 13 35	95 5D B0 4F 8E 2D A2 44
77CD10E0	06 00 01 00	02 25 F9 3A 5...]à. .cd. %ù.
77CD10F0	B9 53 41 44	E3 28 2F 4A 5... W. à(/J
77CD1100	64 7C CD 77	BA 9C D6 90 4A 46 E6 38 06 00 02 00
77CD1110	95 BB 83 D0	SAD. O. JjNs. 48 7C CD 77

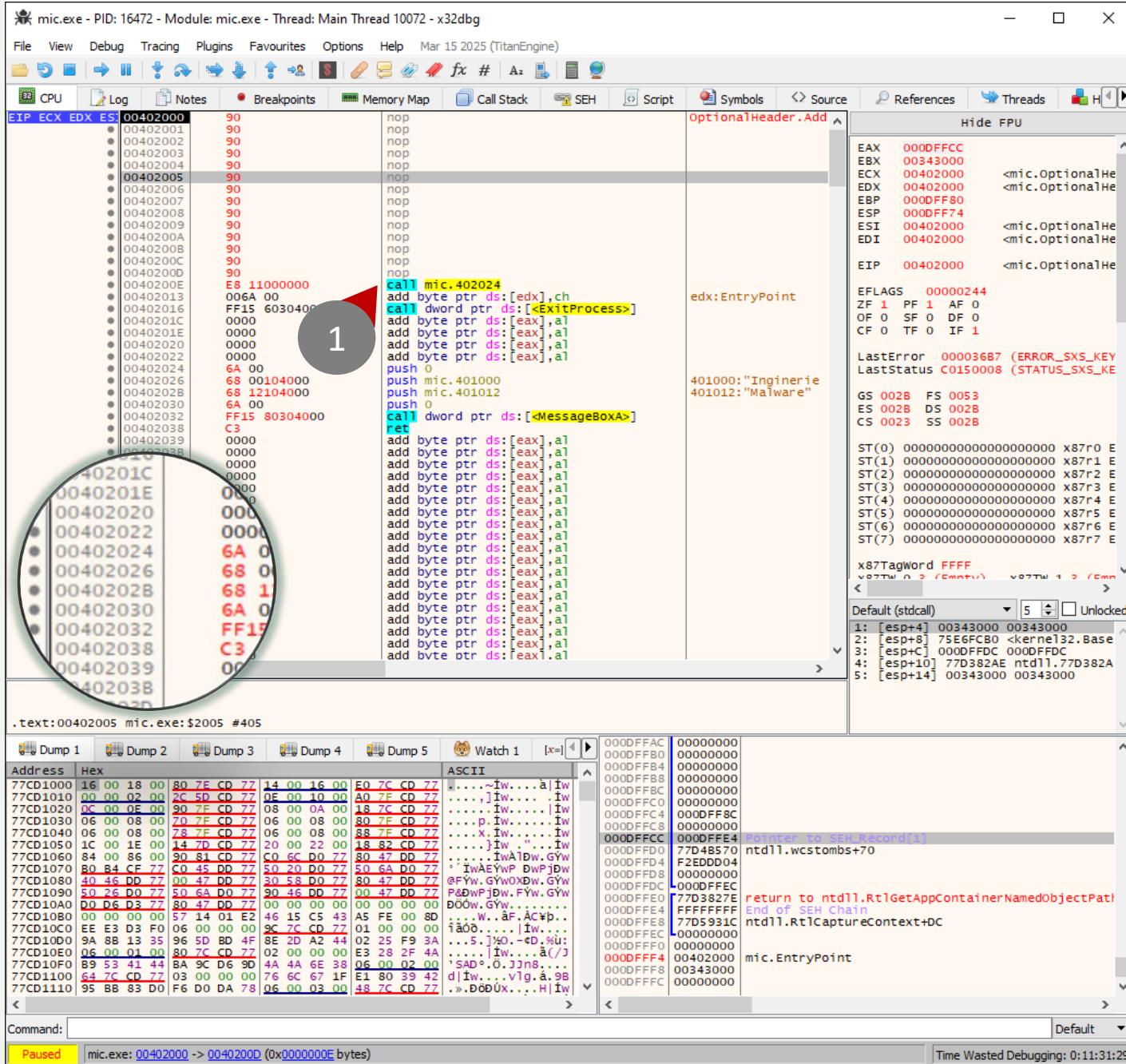
Command: Default
Paused mic.exe: 00402000 -> 0040200D (0x0000000E bytes)
Time Wasted Debugging: 0:11:31:29

5. Toate modificările realizate în memoria procesului sunt vizibile, anume, codul injectat este activ, iar codul remanent a fost înlocuit cu instrucțiuni NOP. Aceasta este forma finală a codului care urmează a fi salvat în fișier.

6. Se aplică patch-urile în fișierul original folosind butonul Patch File.



Patch-ul a fost aplicat cu succes. Modificările făcute în memoria procesului sunt acum salvate permanent în fișierul mic.exe.



Echivalare

1. Instrucția `ret` marchează finalul codului duplicat. La execuție, procesorul va reveni automat la adresa salvată anterior de instrucțiunea `call`.

Această revenire este crucială pentru ca programul să nu se prăbușească.

- a) Explicație internă: când CPU întâlnește call, face în realitate două operații: → push adresa următoarei instrucțiuni (00402013) → jmp către 00402024 (codul injectat). Aceasta este logica ascunsă în spatele unui call.

- b) Putem testa această logică înlocuind call cu push și jmp. Rezultatul este identic la execuție, dar mult mai flexibil pentru shellcode sau patching.

call mic.402024

1

CPU face intern:
push 00402013
jmp 00402024

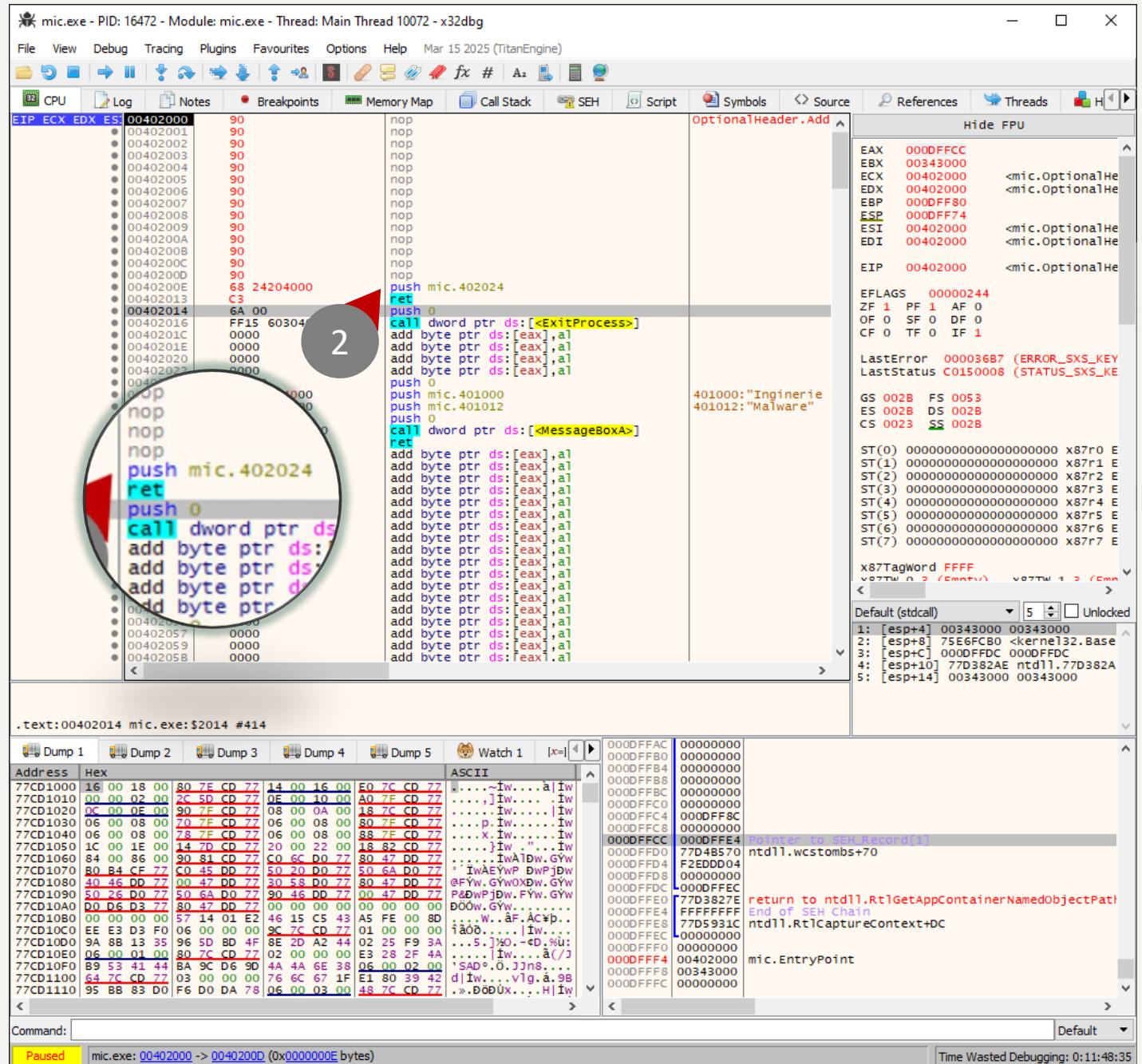
Echivalare

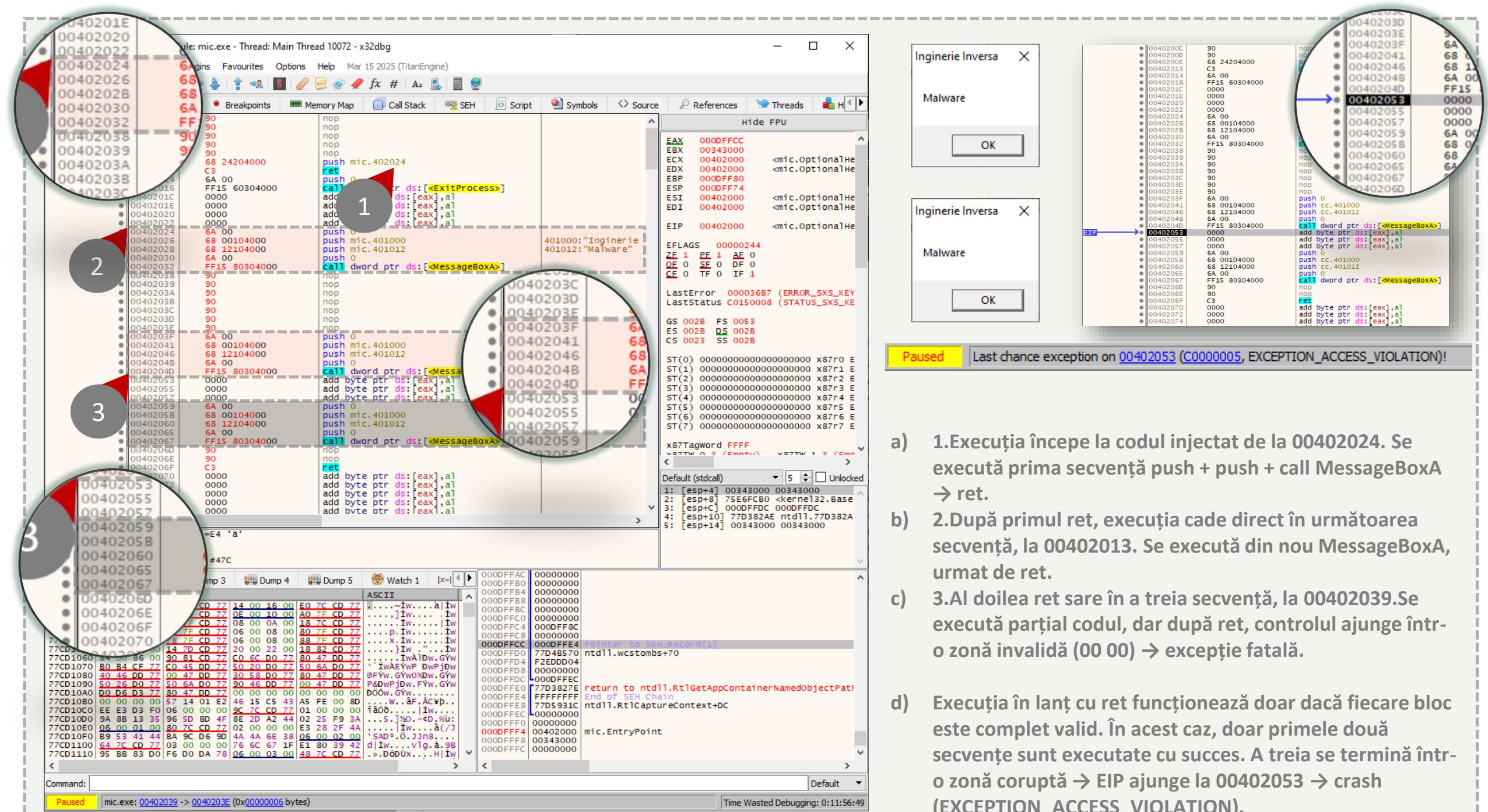
2. Instrucțiunea call este echivalentă cu o combinație internă de push + jmp. După execuția codului injectat, ret revine exact unde trebuie – la adresa salvată. Astfel, fluxul programului nu este întrerupt și pare natural.

De ce să folositi push + ret în loc de call?

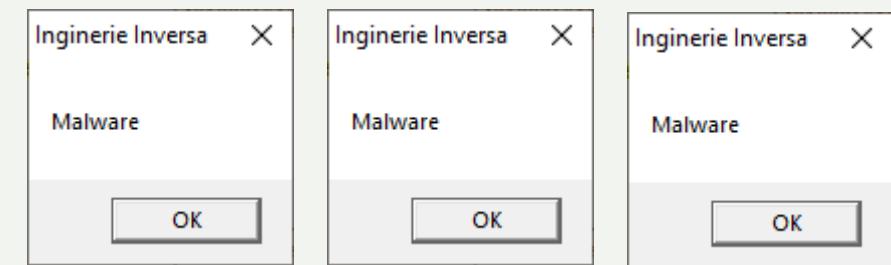
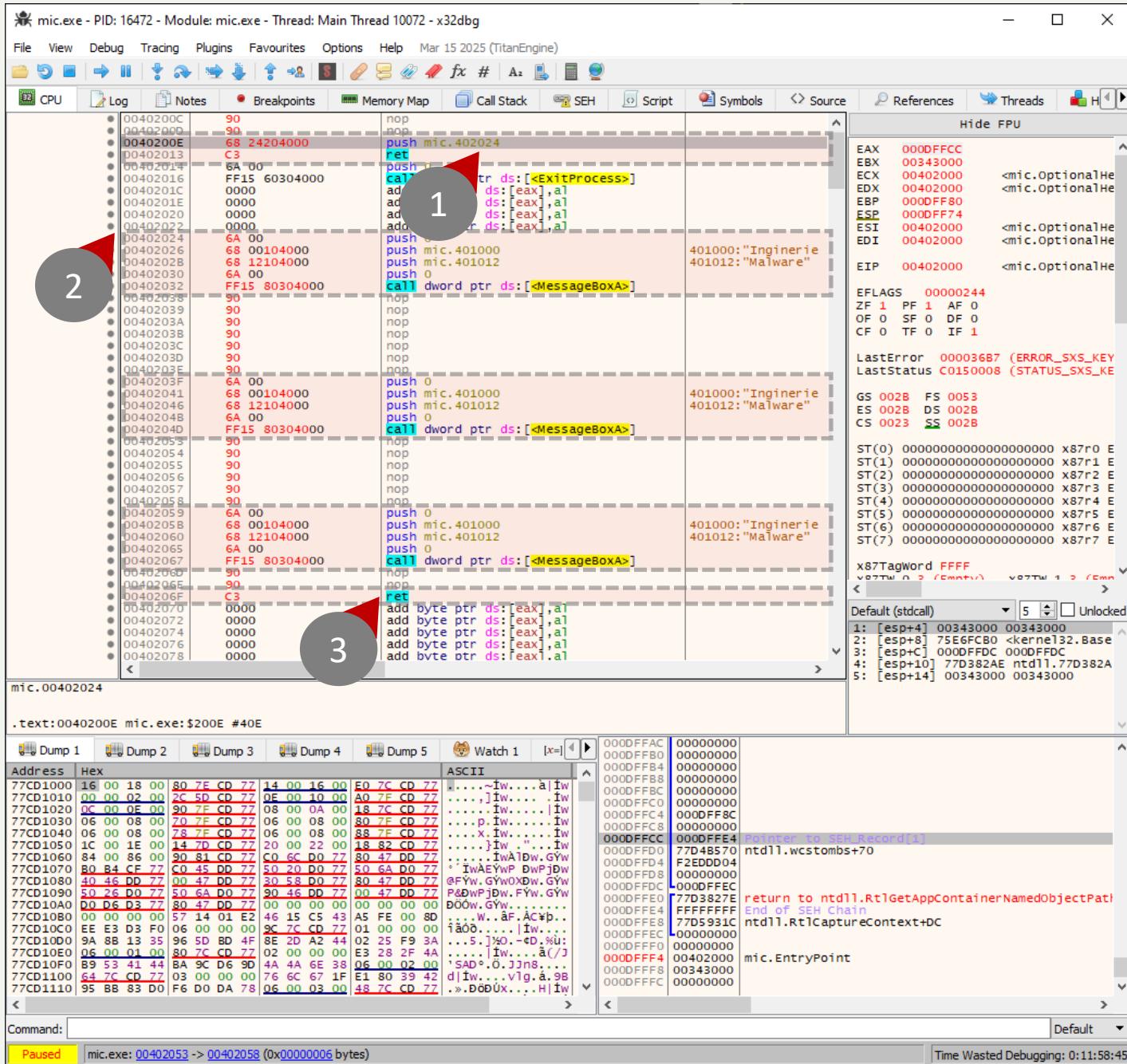
- Ofuscare – poate păcăli dezasamblorul sau analiza statică (nu este un call obișnuit).
 - Stil shellcode / malware – metode de execuție indirekte, mai greu de detectat.
 - Control precis asupra stivei – poți manipula manual adresele și fluxul.
 - Constrângeri de spațiu – uneori push + ret începe mai bine decât un call rel32, mai ales când lucrezi într-un patch sau code cave limitat.

Nota: Este un caz clasic de "call spoofing" sau return-oriented programming style – adică apelul funcției nu se face cu instrucțiunea call, ci cu push + ret.



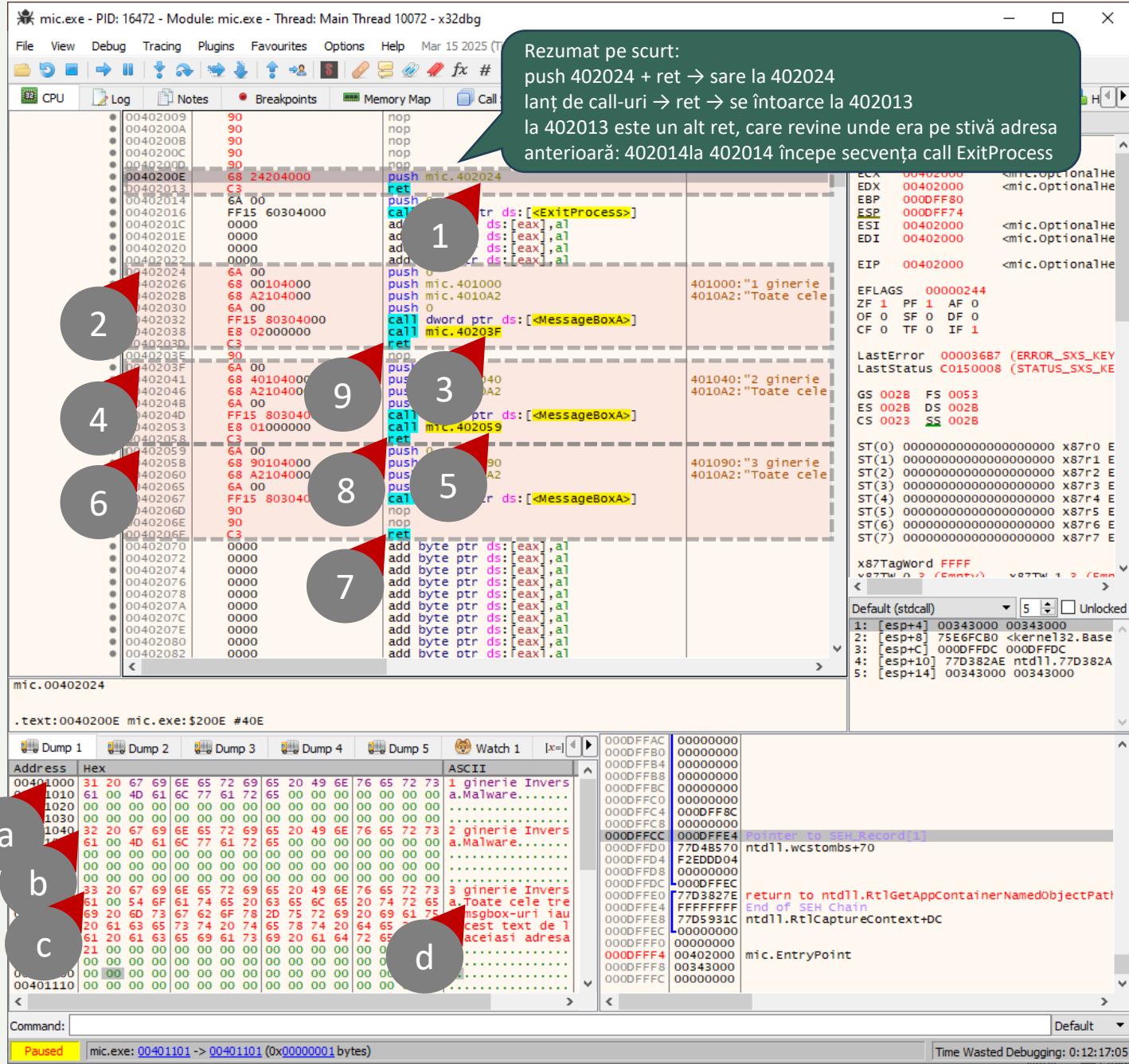


- a) 1. Execuția începe la codul injectat de la 00402024. Se execută prima secvență push + push + call MessageBoxA → ret.
- b) 2. După primul ret, execuția cade direct în următoarea secvență, la 00402013. Se execută din nou MessageBoxA, urmat de ret.
- c) 3. Al doilea ret sare în a treia secvență, la 00402039. Se execută parțial codul, dar după ret, controlul ajunge într-o zonă invalidă (00 00) → excepție fatală.
- d) Execuția în lanț cu ret funcționează doar dacă fiecare bloc este complet valid. În acest caz, doar primele două secvențe sunt executate cu succes. A treia se termină într-o zonă coruptă → EIP ajunge la 00402053 → crash (EXCEPTION_ACCESS_VIOLATION).



- a) Execuție în cascadă prin ret 1 La adresa 00402024, codul injectat este apelat prima dată (prin call din Entry Point). Acesta afișează primul MessageBox. 2 Instrucțiunea ret duce execuția la 00402029, unde se află a doua copie a codului injectat. Se afișează al doilea MessageBox. 3 După al doilea ret, execuția ajunge la 00402039, care conține al treilea bloc identic. Se afișează și al treilea MessageBox. Codul este valid și se încheie controlat.

 - b) Prin alinierea directă a celor trei blocuri și utilizarea ret, execuția curge automat dintr-un bloc în altul, fără salturi explicite. Aceasta este o tehnică eficientă de execuție în lanț în contexte precum shellcode, injectii, sau patching.



Pentru exercițiu, toate mesajele afișate în cele trei MessageBox-uri sunt luate de la aceeași adresă 4010A2 din segmentul .data. În schimb, titlurile acestor MessageBox-uri sunt luate de la adrese diferite, pentru a demonstra studenților că este posibil să reutilizăm același mesaj, dar să personalizăm titlul fiecărui dialog.

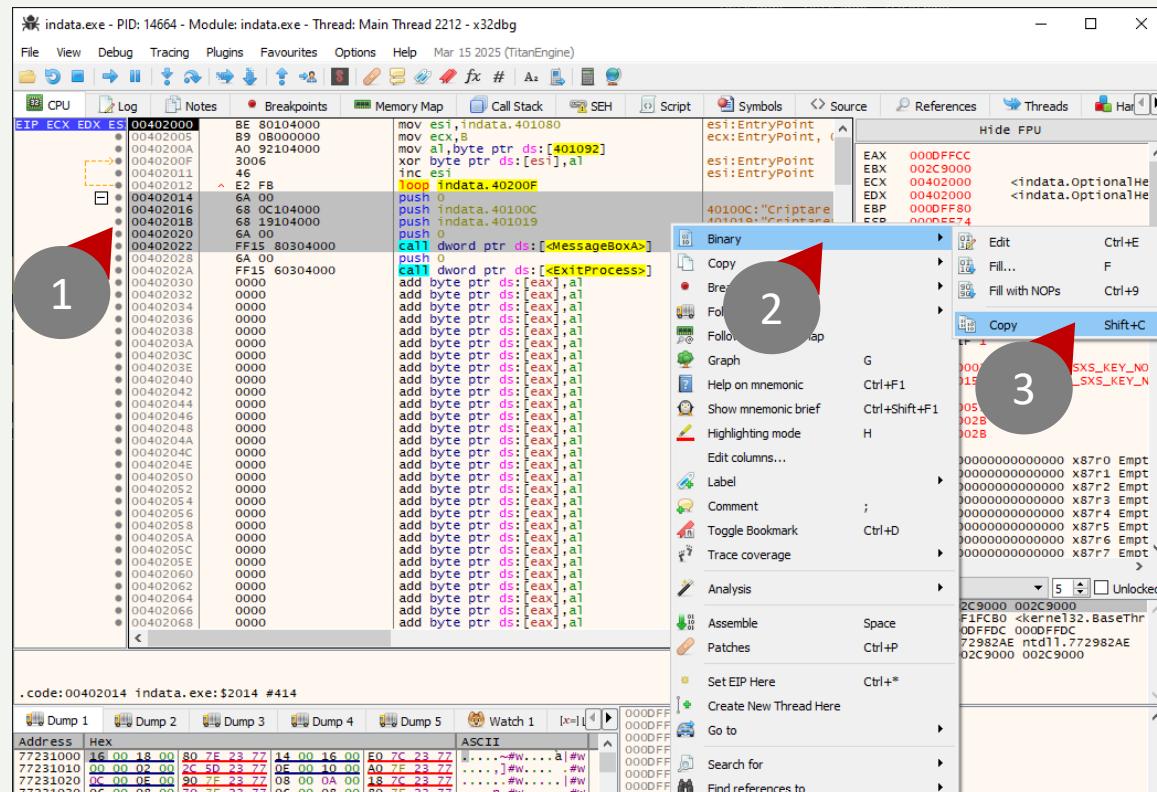
9.2

EXECUȚIA DE COD ÎN SECȚIUNEA .DATA



CONTEXT INTRODUCTIV

- În acest exemplu demonstrativ, relocăm o bucată de cod executabil din secțiunea .code (.text) într-o zonă liberă din secțiunea .data, care în mod normal nu este executabilă. Duplicăm codul, schimbăm permisiunile de memorie pentru a permite execuția și redirecționăm fluxul printr-un jmp. Această tehnică evidențiază posibilitatea rulării de cod din zone non-standard, un mecanism utilizat frecvent în malware sau tehnici avansate de ofuscare.



```

1. Se selectează codul dorit din secțiunea .code (indata.exe), inclusiv apeluri către MessageBoxA și ExitProcess.
2. Click dreapta pe cod → Binary.
3. Se alege Copy → Selection pentru a copia codul executabil în clipboard.
4. În Memory Map, se identifică zona corespunzătoare secțiunii .data (cu protecții RW-).
5. Se face click dreapta pe .data și se alege Follow in Dump pentru a deschide secțiunea în Dump View (pentru inserare ulterioară).

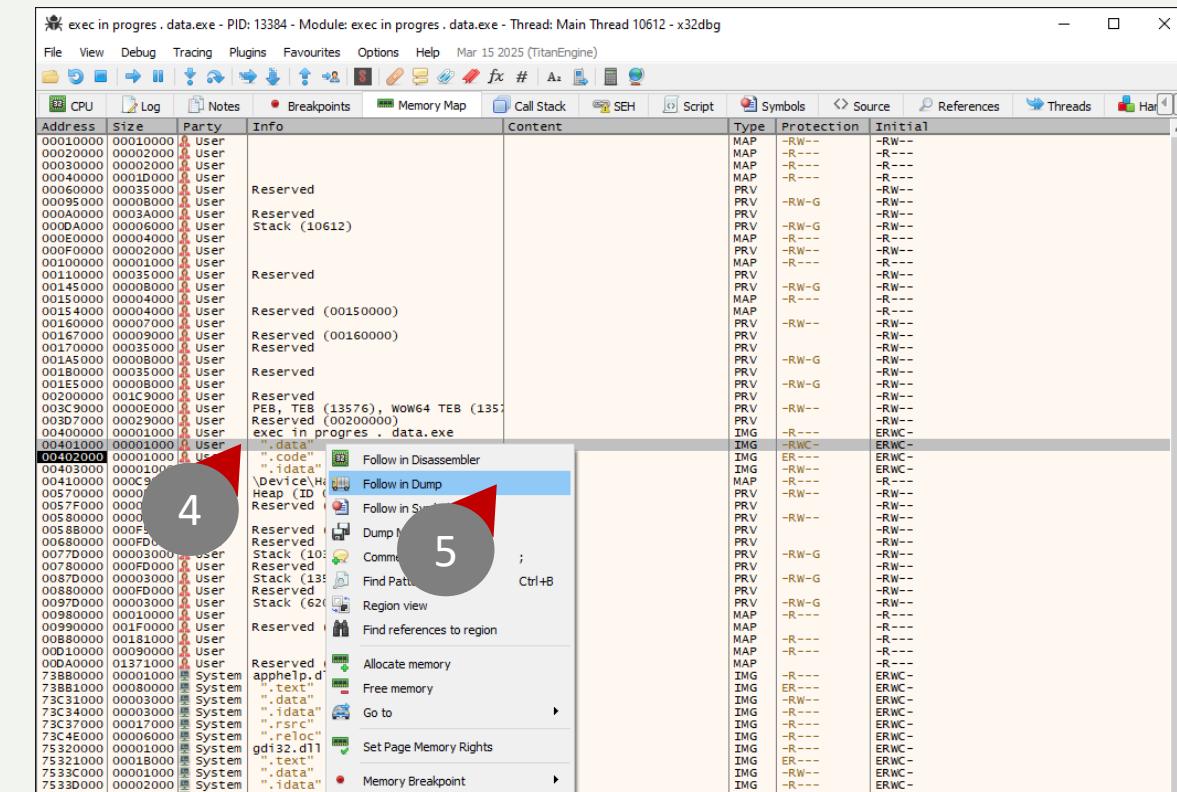
```

Code snippet from indata.exe assembly:

```

    BE 80104000 mov esi,1ndata.401080
    B8 0B000000 mov ecx,B
    A9 92104000 mov al byte ptr ds:[401092]
    33C0 xor byte ptr ds:[esi],al
    46 inc esi
    E2 FB loop indata.40200F
    40402014 push 0
    40402016 push indata.40100C
    40402018 push indata.401019
    40402020 push 0
    40402022 FF15 60304000 call dword ptr ds:[<MessageBoxA>]
    40402024 push 0
    40402026 call dword_ptr ds:[<ExitProcess>]
    40402028 add byte ptr ds:[eax],a1
    4040202A 00000000 add byte ptr ds:[eax],a1
    4040202C 00000000 add byte ptr ds:[eax],a1
    4040202E 00000000 add byte ptr ds:[eax],a1
    40402030 00000000 add byte ptr ds:[eax],a1
    40402032 00000000 add byte ptr ds:[eax],a1
    40402034 00000000 add byte ptr ds:[eax],a1
    40402036 00000000 add byte ptr ds:[eax],a1
    40402038 00000000 add byte ptr ds:[eax],a1
    4040203A 00000000 add byte ptr ds:[eax],a1
    4040203C 00000000 add byte ptr ds:[eax],a1
    4040203E 00000000 add byte ptr ds:[eax],a1
    40402040 00000000 add byte ptr ds:[eax],a1
    40402042 00000000 add byte ptr ds:[eax],a1
    40402044 00000000 add byte ptr ds:[eax],a1
    40402046 00000000 add byte ptr ds:[eax],a1
    40402048 00000000 add byte ptr ds:[eax],a1
    4040204A 00000000 add byte ptr ds:[eax],a1
    4040204C 00000000 add byte ptr ds:[eax],a1
    40402050 00000000 add byte ptr ds:[eax],a1
    40402052 00000000 add byte ptr ds:[eax],a1
    40402054 00000000 add byte ptr ds:[eax],a1
    40402056 00000000 add byte ptr ds:[eax],a1
    40402058 00000000 add byte ptr ds:[eax],a1
    4040205A 00000000 add byte ptr ds:[eax],a1
    4040205C 00000000 add byte ptr ds:[eax],a1
    4040205E 00000000 add byte ptr ds:[eax],a1
    40402060 00000000 add byte ptr ds:[eax],a1
    40402062 00000000 add byte ptr ds:[eax],a1
    40402064 00000000 add byte ptr ds:[eax],a1
    40402066 00000000 add byte ptr ds:[eax],a1
    40402068 00000000 add byte ptr ds:[eax],a1

```



Memory Map for exec in progress .data.exe:

Address	Size	Party	Info	Content	Type	Protection	Initial
00010000	00010000	User			MAP	-RW-	
00020000	00020000	User			MAP	-R---	
00030000	00030000	User			MAP	-R---	
00040000	00040000	User			MAP	-R---	
00050000	00050000	User			PRV	-RW-	
00095000	00080000	User			PRV	-RW-	
00040000	0003A000	User			PRV	-RW-G	
00050000	00060000	User			PRV	-RW-	
00060000	00040000	User			PRV	-RW-	
00070000	00020000	User			PRV	-RW-	
00080000	00010000	User			PRV	-RW-	
00145000	00080000	User			PRV	-RW-G	
00150000	00040000	User			MAP	-R---	
00154000	00040000	User			PRV	-RW-	
00160000	00070000	User			PRV	-RW-	
00170000	00035000	User			PRV	-RW-	
00180000	00035000	User			PRV	-RW-G	
001E5000	00080000	User			PRV	-RW-G	
00200000	001C9000	User	PEB, TEB (13576), wow64 TEB (13576)		PRV	-RW-	
003C0000	000E0000	User			PRV	-RW-	
003D7000	00029000	User			PRV	-RW-	
00400000	00010000	User	exec in progress .data.exe		IMG	-R---	ERWC
00403000	00010000	User	"...code"		IMG	-R---	ERWC
00410000	000C9000	User	".idata"		IMG	ER-	ERWC
00577000	00010000	User	".DeviceH"		MAP	-R---	
0057F000	00010000	User	Heap (ID: 10)		PRV	-RW-	
00580000	00010000	User	Reserved (10)		PRV	-RW-	
00600000	00010000	User	Reserved (10)		PRV	-RW-	
0077D000	00030000	User	Stack (10)		PRV	-RW-	
00780000	000D0000	User	Reserved (10)		PRV	-RW-	
0087D000	00030000	User	Stack (13)		PRV	-RW-	
00880000	000D0000	User	Reserved (13)		PRV	-RW-	
0097D000	00030000	User	Stack (62)		PRV	-RW-G	
00990000	001F0000	User	Stack (62)		PRV	-RW-	
00B80000	00181000	User	Reserved (62)		MAP	-R---	
00D10000	00090000	User	Reserved (62)		MAP	-R---	
00A00000	01371000	User	Reserved (62)		MAP	-R---	
73880000	00010000	System	appelp.d		IMG	-R---	ERWC
73881000	00080000	System	".text"		IMG	ER---	ERWC
73C31000	00030000	System	".data"		IMG	-RW-	ERWC
73C34000	00020000	System	".rdata"		IMG	-RW-	ERWC
73C37000	00017000	System	".reloc"		IMG	-R---	ERWC
73C4E000	00060000	System	gdi32.dll		IMG	-R---	ERWC
75320000	00010000	System	gdi32.dll		IMG	-R---	ERWC
75321000	0001B000	System	gdi32.dll		IMG	ER---	ERWC
7533C000	00010000	System	".text"		IMG	-RW-	ERWC
7533D000	00020000	System	".data"		IMG	-RW-	ERWC
7533F000	00010000	System	".rdata"		IMG	-RW-	ERWC
75340000	00010000	System	".rsrc"		IMG	-R---	ERWC

6. Se localizează zona liberă din secțiunea .data (vizibilă în Dump – adrese libere, valori zero).

7. Click dreapta pe începutul zonei libere → Binary.

8. Se alege opțiunea Paste (Ignore Size) pentru a insera codul copiat anterior, fără restricții de dimensiune.

9. Codul apare acum în .data, cu instrucțiunile vizibile în hex și ASCII – gata pentru executare (dar încă nepermisă de sistem).

6

7

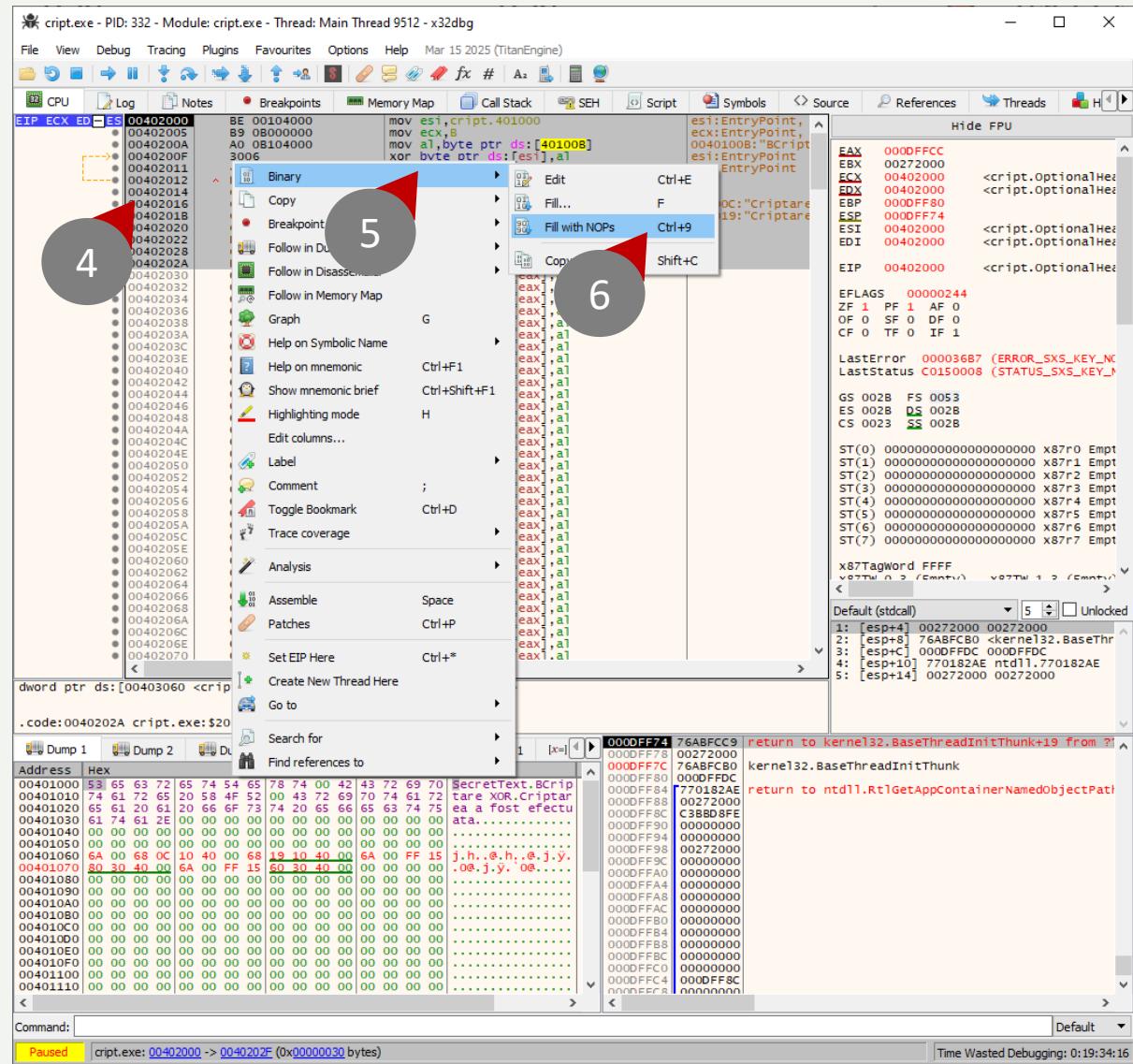
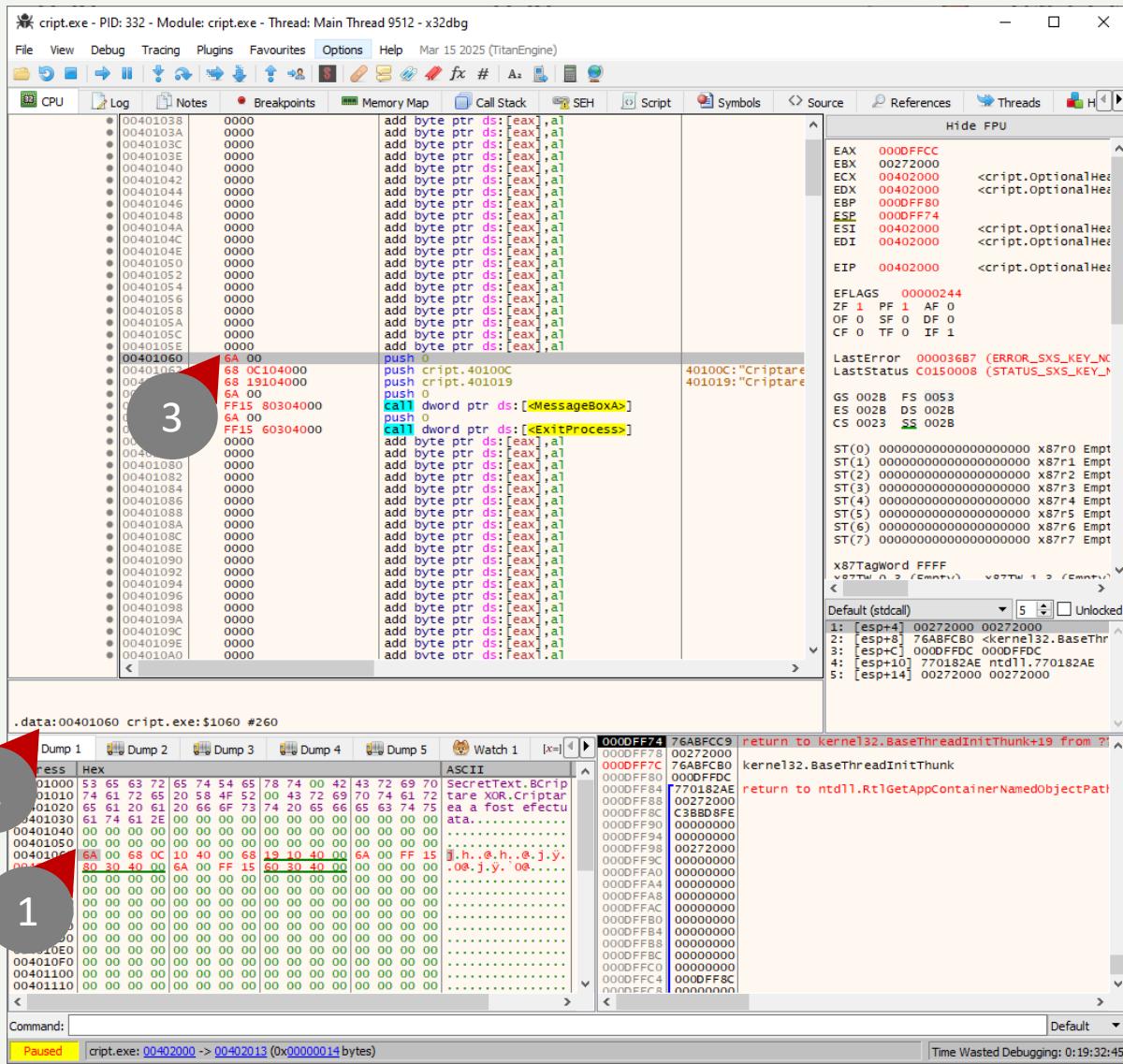
8

9

1. Observăm că în această zonă există deja cod — anterior injectat în secțiunea .data.
 2. Confirmăm că ne aflăm în secțiunea .data — lucru vizibil în Dump (adresă + nume secțiune).
 3. Comutăm în tab-ul CPU pentru a vizualiza codul din .data dezasamblat (disassembler view).
 4. Selectăm blocul de cod din .code care trebuie eliminat.

5. Click dreapta → alegem Binary.

6. Selectăm Fill with NOPs pentru a neutraliza codul — aici vom introduce un jmp spre .data, deci acest cod nu mai este necesar.



cript.exe - PID: 332 - Module: cript.exe - Thread: Main Thread 9512 - x32dbg

File View Debug Tracing Plugins Favourites Options Help Mar 15 2025 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Help

1. Click pe adresa din .code: 00401060 (004010E0) și apăsați dreapta pentru a deschide meniuul contextual.

2. Selectați opțiunea **Copy**.

3. Selectați opțiunea **Address** din meniu.

EIP ECX EDX ES 00402000 90 nop
00402001 90 nop
00402002 90 nop
00402003 90 nop
00402004 90 nop
00402005 90 nop
00402006 90 nop
00402007 90 nop
00402008 90 nop
00402009 90 nop
0040200A 90 nop
0040200B 90 nop
0040200C 90 nop
0040200D 90 nop
0040200E 90 nop
0040200F 90 nop
00402010 90 nop
00402011 90 nop
00402012 90 nop
00402013 90 nop
00402014 90 nop

OptionalHeader.A ^ Hide FPU
EAX 000DFFCC
EBX 00272000
ECX 00402000 <cript.OptionalHea
EDX 00402000 <cript.OptionalHea
EBP 000DFF80
ESP 000DFF74
ESI 00402000 <cript.OptionalHea
EDI 00402000 <cript.OptionalHea
EIP 00402000 <cript.OptionalHea
EFLAGS 000000244
ZF 1 PF 1 AF 0
OF 0 SF 0 DF 0
CF 0 TF 0 IF 1
LastError 000036B7 (ERROR_SXS_KEY_NC
LastStatus C0150008 (STATUS_SXS_KEY_N
GS 002B FS 0053
ES 002B DS 002B
CS 0023 SS 002B
ST(0) 00000000000000000000000000000000 x87r0 Empt
ST(1) 00000000000000000000000000000000 x87r1 Empt
ST(2) 00000000000000000000000000000000 x87r2 Empt
ST(3) 00000000000000000000000000000000 x87r3 Empt
ST(4) 00000000000000000000000000000000 x87r4 Empt
ST(5) 00000000000000000000000000000000 x87r5 Empt
ST(6) 00000000000000000000000000000000 x87r6 Empt
ST(7) 00000000000000000000000000000000 x87r7 Empt
x87TagWord FFFF
x87TW 0 2 (Emptv) x87TW 1 2 (Emptv)
Default (stdcall) 5 Unlocked
1: [esp+4] 00272000 00272000
2: [esp+8] 76ABFCB0 <kernel32.BaseThr
3: [esp+C] 000DFFDC 000DFFDC
4: [esp+10] 770182AE ntdl!770182AE
5: [esp+14] 00272000 00272000

Dump 5 ASCII 000DFF74 76ABFC99 return to kernel32.BaseThreadInitThunk+19 from ??
000DFF78 00272000
000DFF7C 76ABFCB0 kernel32.BaseThreadInitTh
000DFF80 000DFFDC
000DFF84 770182AE return to ntdl.RtlGetAppContainerNamedObjectPath
000DFF88 00272000
000DFF8C C3BB08FE
000DFF90 00000000
000DFF94 00000000
000DFF98 00272000
000DFF9C 00000000
000DFFA0 00000000
000DFFA4 00000000
000DFFA8 00000000
000DFFB0 00000000
000DFFB4 00000000
000DFFB8 00000000
000DFFBC 00000000
000DFFC0 00000000
000DFFC4 000DFF8C
000DFFC8 00000000

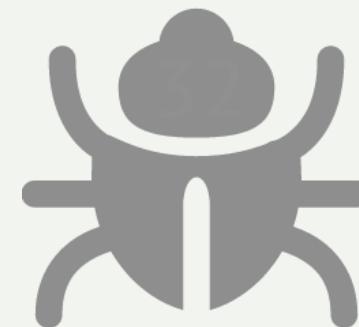
Command: Default
Paused cript.exe: 00401060 -> 00401060 (0x00000001 bytes) Time Wasted Debugging: 0:19:36:01

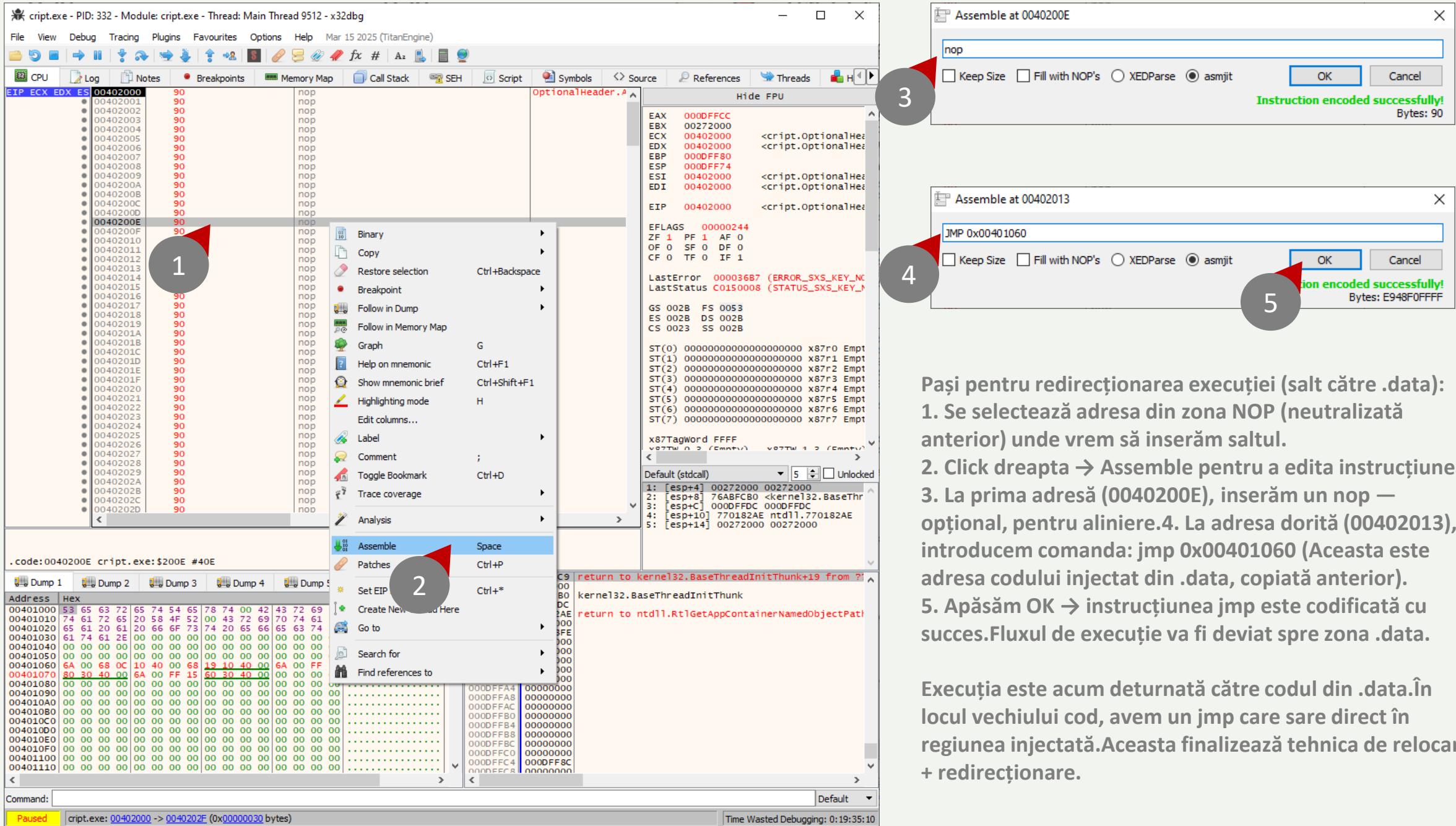
1. Se identifică începutul codului injectat în .data — vizibil atât în Dump cât și în tab-ul CPU (instructiuni NOP urmate de cod).

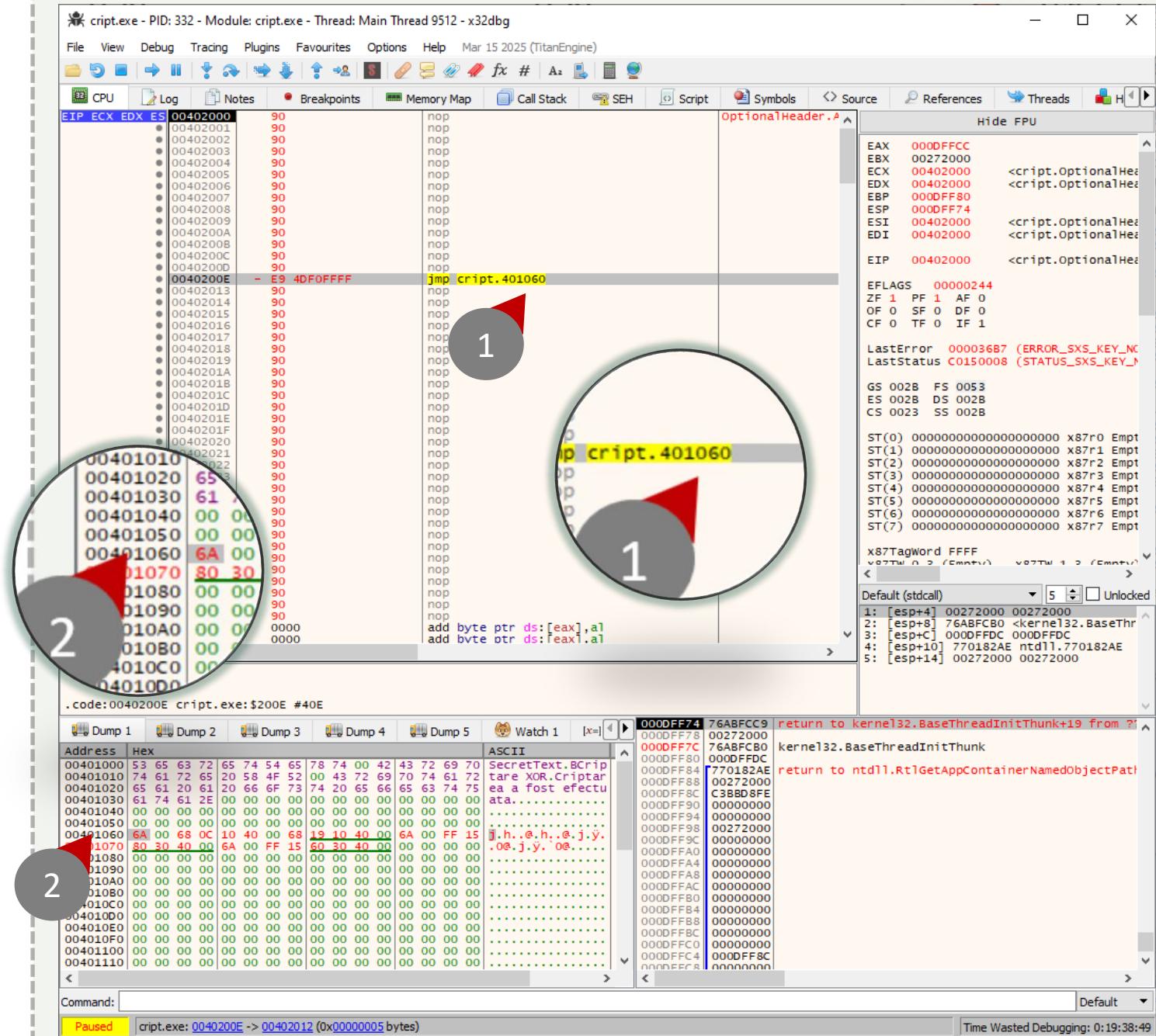
2. Click dreapta → Copy.

3. Alegem Address pentru a copia adresa exactă a începutului codului — va fi folosită într-o instrucțiune jmp sau call.

Se copiază adresa codului injectat din .data. Vom folosi această adresă pentru a crea un jmp din secțiunea .code, direcționând execuția către noua locație.

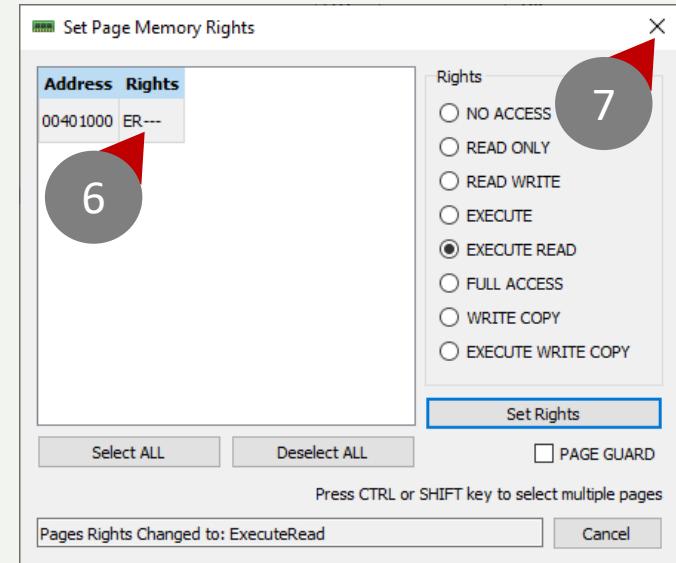
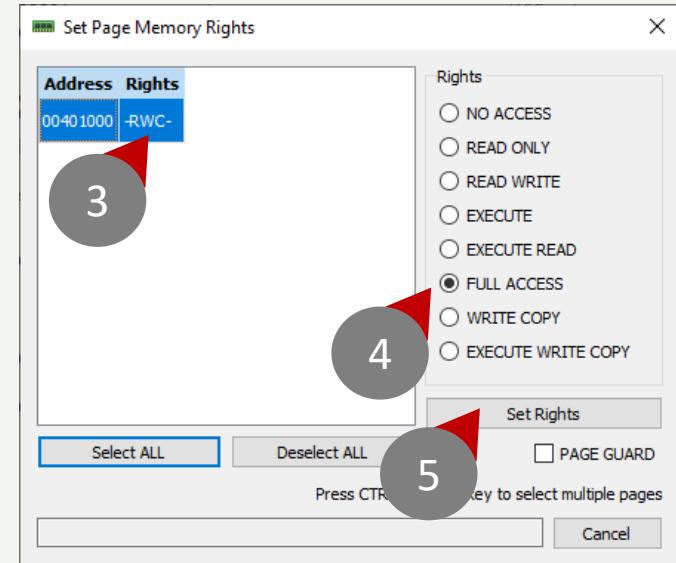
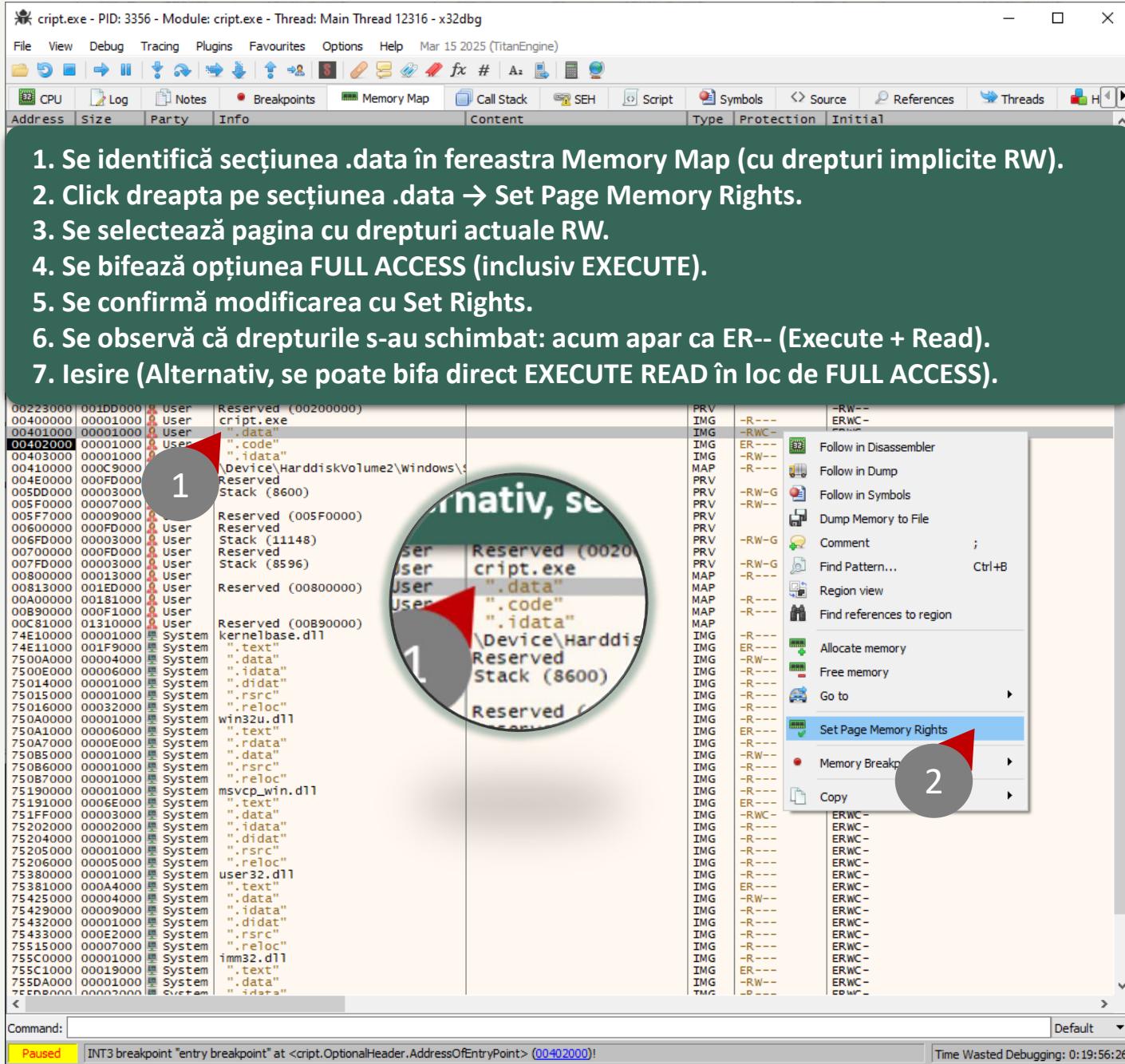






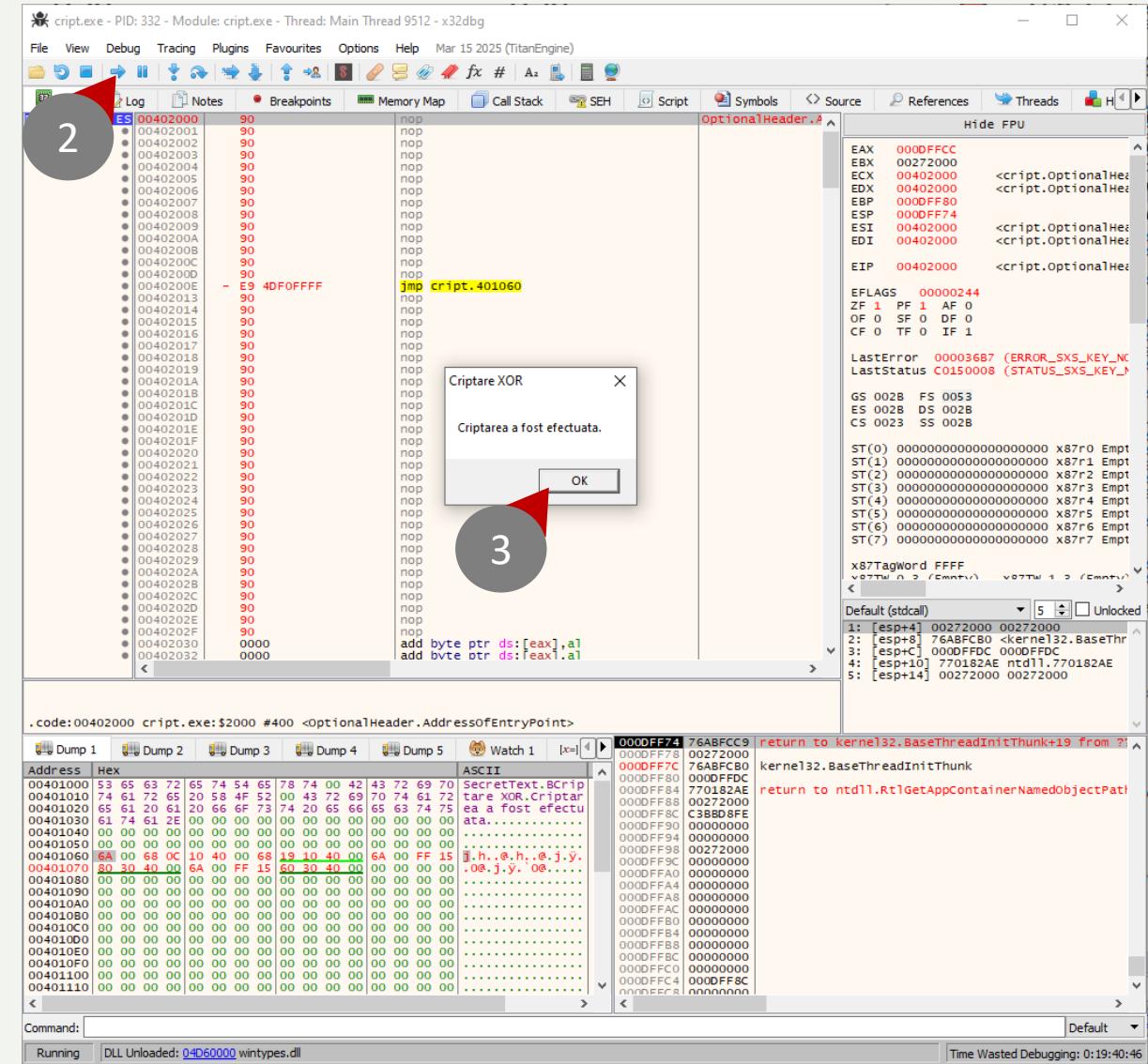
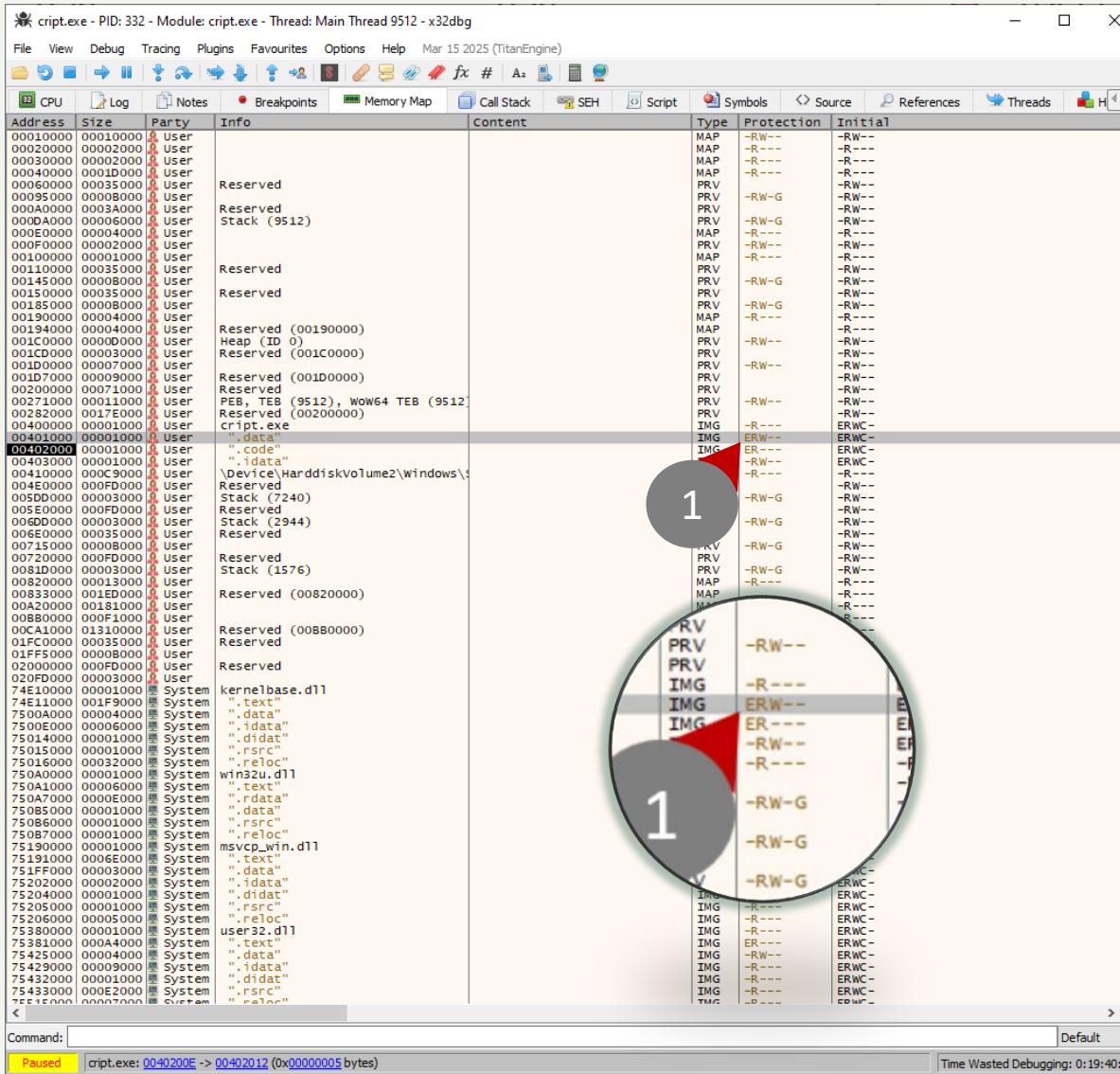
1. Instrucțiunea jmp 0x00401060 a fost inserată corect în secțiunea .code. Aceasta redirecționează execuția către codul din .data.2. În Dump, vedem clar că adresa 00401060 conține cod real, funcțional, nu doar date statice. Aceasta confirmă că saltul va ajunge într-o zonă validă de execuție.
 2. Execuția este pregătită pentru a porni din zona .data. Instrucțiunea jmp a fost inserată cu succes, iar codul injectat în .data va fi executat ca parte a fluxului programului.

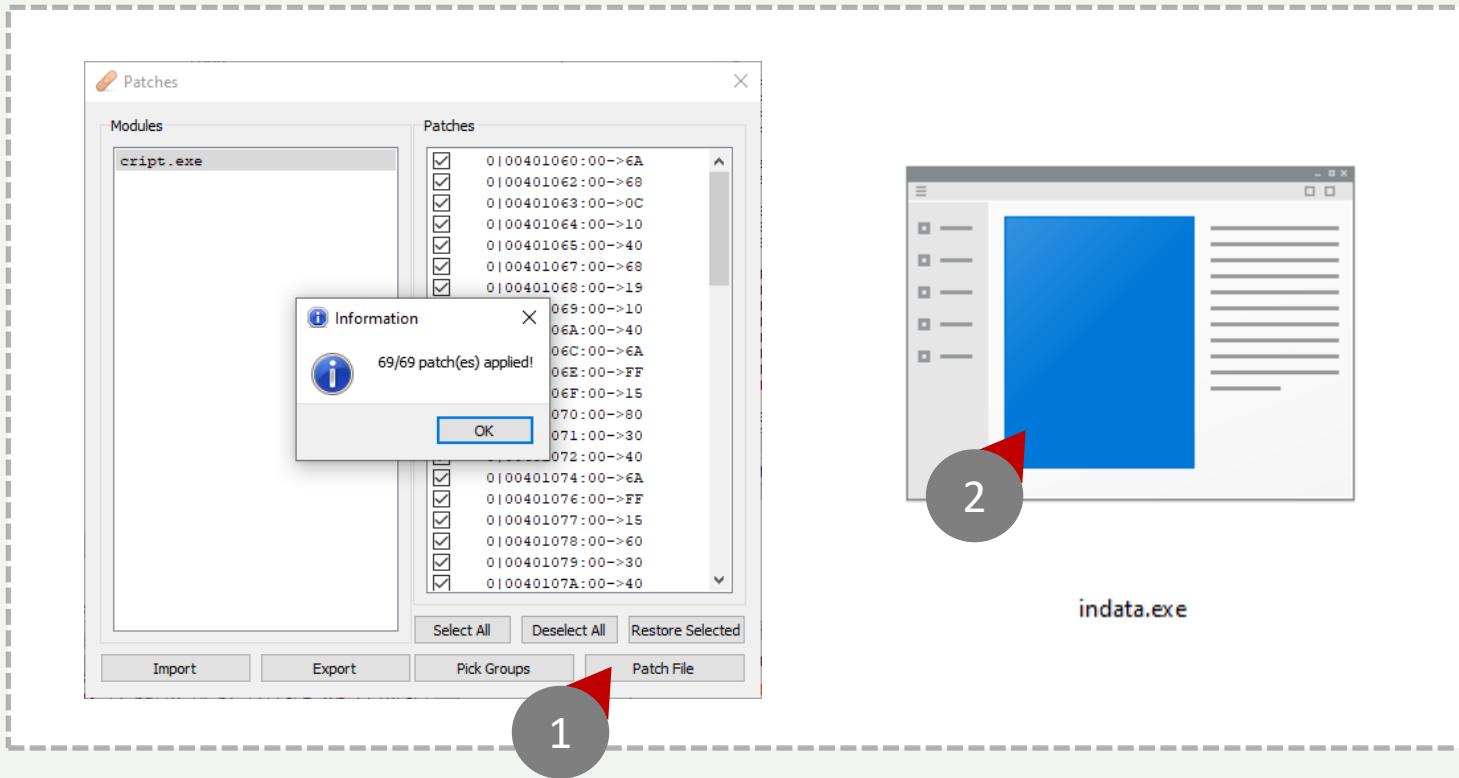
Execuția este pregătită pentru a porni din zona .data. Instrucțiunea jmp a fost inserată cu succes, iar codul injectat în .data va fi executat ca parte a fluxului programului.



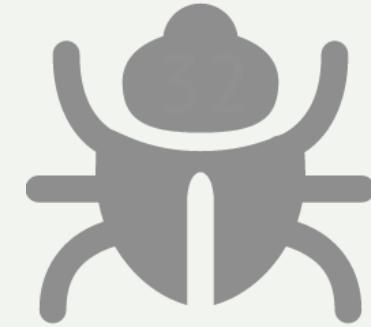
Permișunile secțiunii .data au fost extinse. Activarea EXECUTE permite rularea codului injectat, completând cu succes procesul de relocare și execuție.

1. Confirmăm că permisiunile din Memory Map pentru secțiunea .data sunt acum ER-- — permite EXECUTE, deci codul poate rula.
2. Execuția ajunge la instrucțiunea jmp către adresa injectată (codul relocat în .data).
3. MessageBox-ul apare cu textul „Criptarea a fost efectuată”, ceea ce înseamnă că fluxul de execuție a fost redirecționat și codul relocat rulează fără erori. Execuție confirmată în secțiunea .data. Datorită permisiunilor modificate și a saltului corect configurat, codul injectat se execută și generează mesajul final din aplicație.





1. Fereastra Patches confirmă aplicarea cu succes a tuturor modificărilor (69/69 patch-uri aplicate). Aici am injectat codul în .data, modificat permisiunile, redirectionat execuția cu jmp, și am validat funcționarea prin MessageBox. 2. După apăsarea Patch file, se generează un executabil nou (indata.exe) care conține deja toate patch-urile. Acesta poate fi rulat direct din Windows, fără debugger, iar comportamentul va fi identic: cod relocat și executat din .data.
2. Modificările au fost salvate. Executabilul final (indata.exe) conține toate patch-urile aplicate în sesiunea de analiză și poate fi rulat independent.



9.3

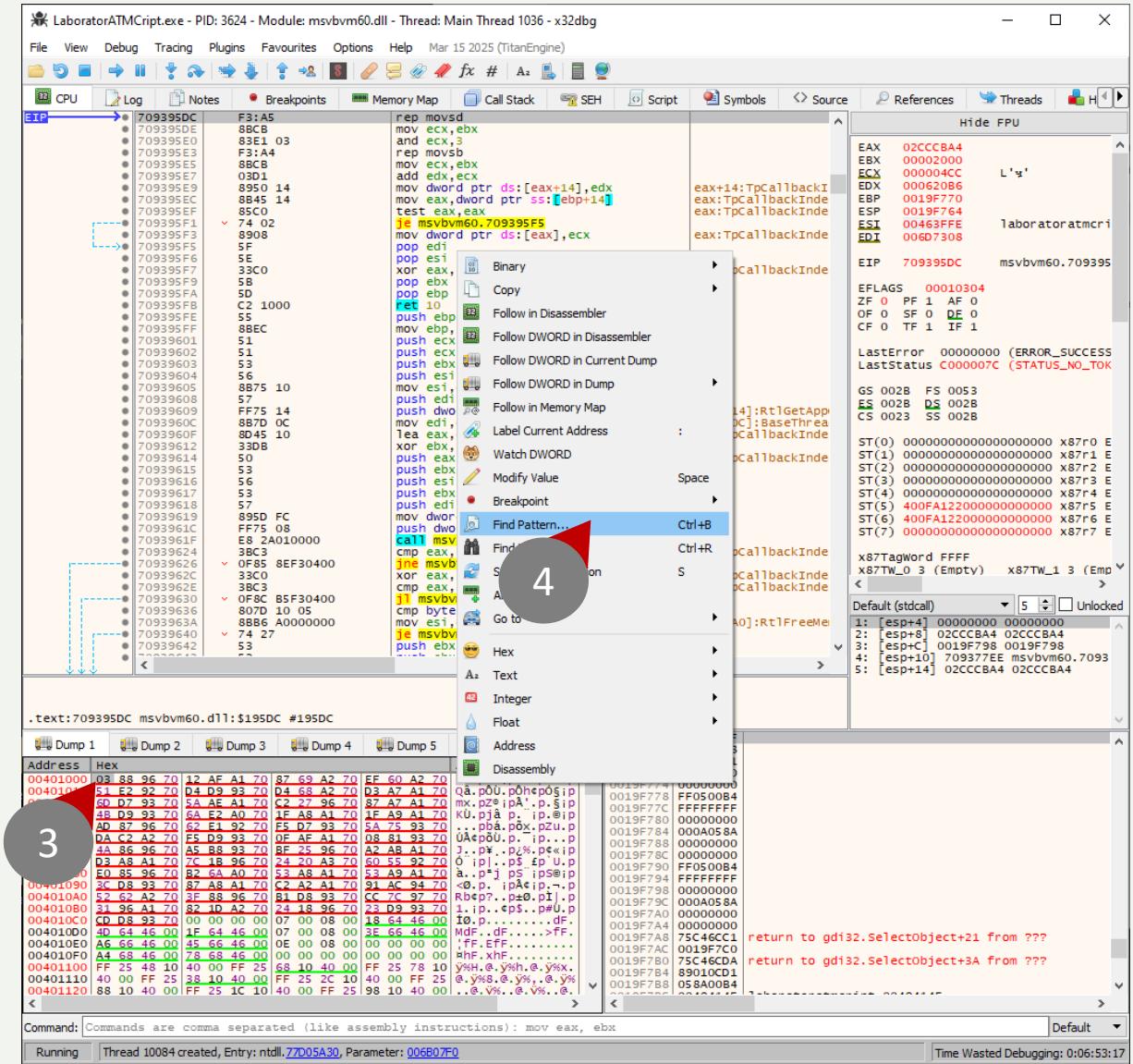
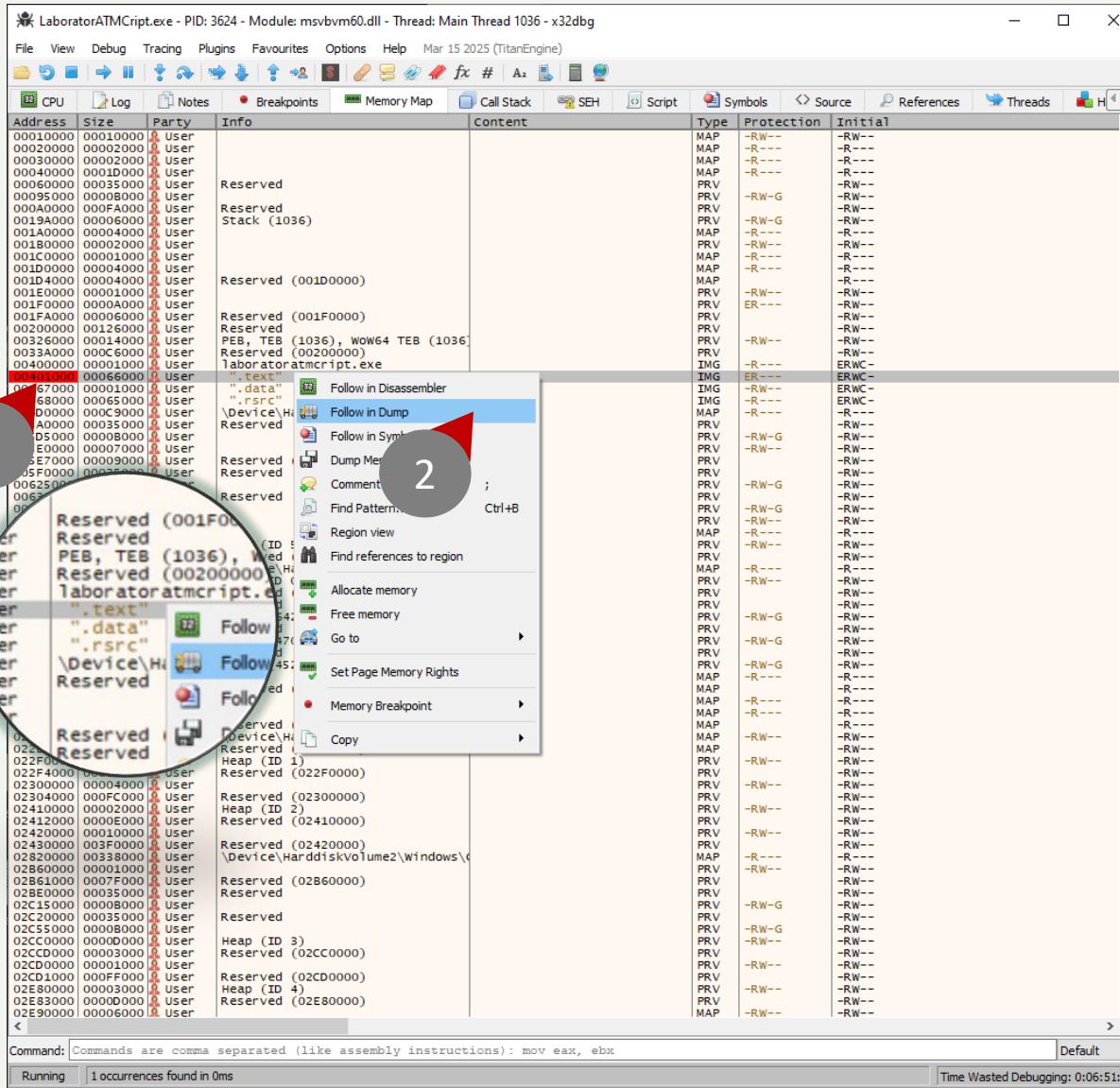
MODIFICAREA STRINGURILOR IN OBIECTE



CONTEXT INTRODUCTIV

- În acest experiment demonstrativ, explorăm cum pot fi localizate și modificate manual stringurile dintr-un executabil. Deși, în mod obișnuit, datele statice precum textele interfeței grafice sunt stocate în secțiunile .data sau .rdata, în unele cazuri acestea pot fi ascunse sau dispersate în alte regiuni ale memoriei.
- Scopul este să identificăm stringul unui buton din interfață (ex: „Verifică”), să-l localizăm în memorie, să-l modificăm (ex: în „Academia”), iar apoi să aplicăm patch-ul permanent în fișierul executabil.

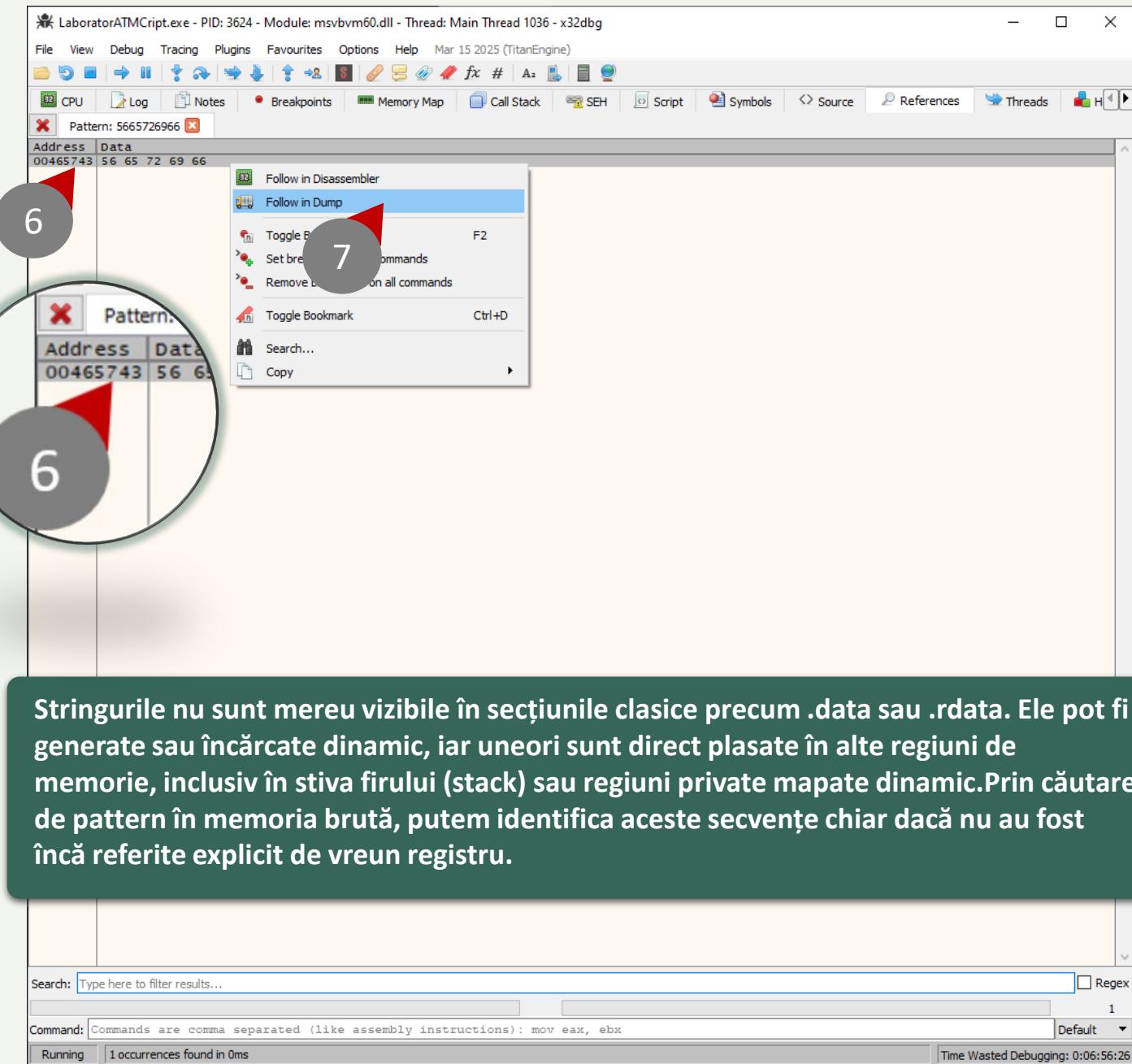
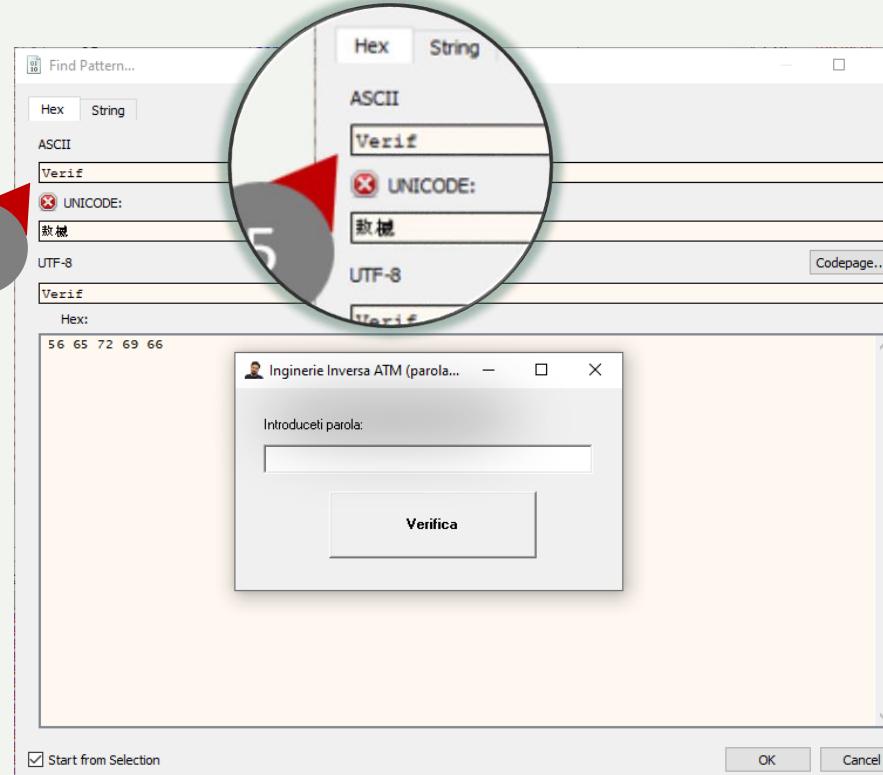
1. Selectează o zonă de memorie din fereastra „Memory Map”, unde ar putea fi stringuri — nu neapărat în .data sau .rdata.
 2. Dă click dreapta și alege „Follow in Dump” pentru a deschide acea zonă în Dump.
 3. În Dump se văd efectiv octetii și textul ASCII, unde putem recunoaște stringuri manual.
 4. În fereastra CPU, se face click dreapta și se selectează „Find Pattern” pentru a căuta o secvență de octeți în memorie (ex: un string sau o semnătură).



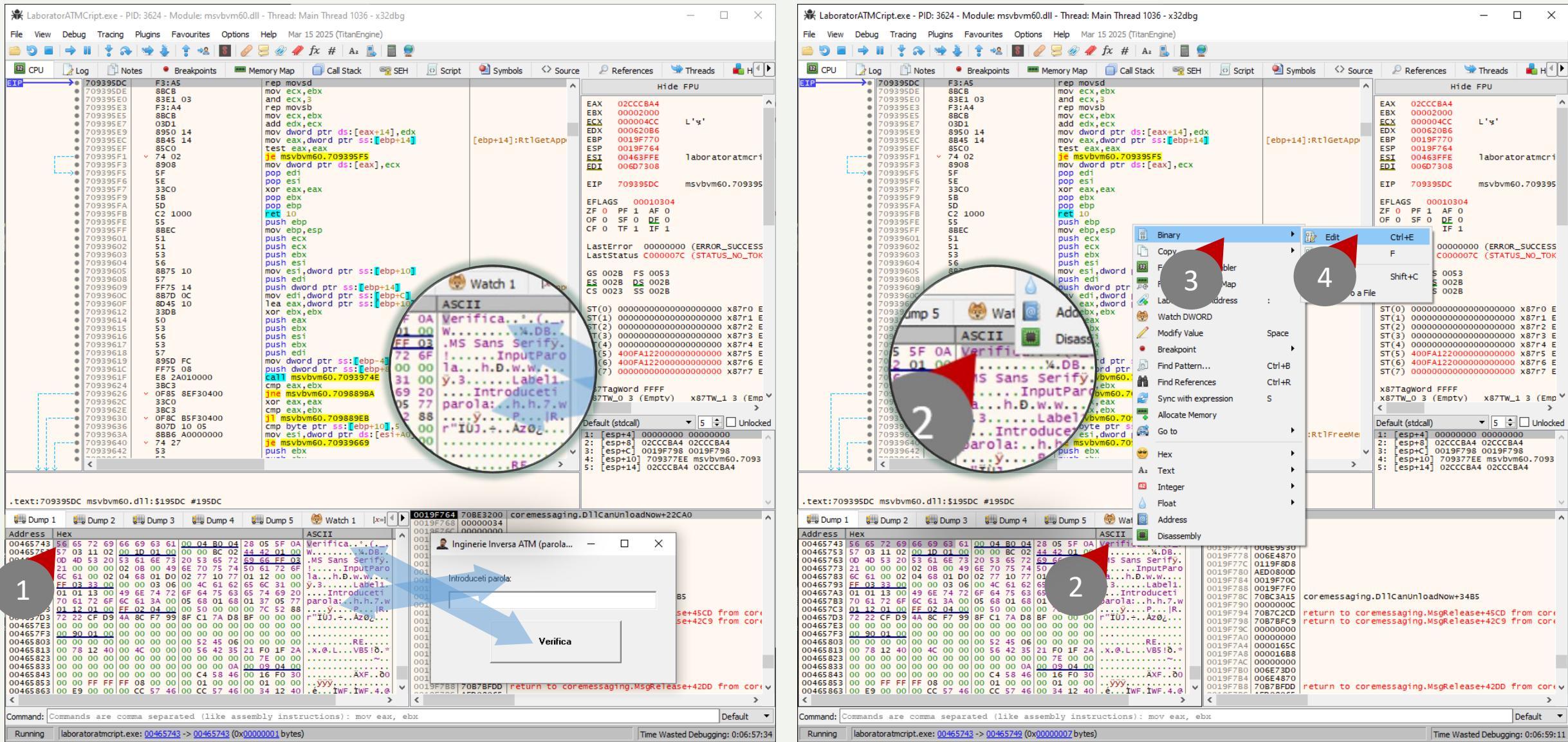
5. Se introduce textul „Verifică” în căsuța de căutare, în format UNICODE — căutăm stringul butonului.

6. Rezultatul căutării arată adresa unde este localizat stringul „Verifică” în memorie.

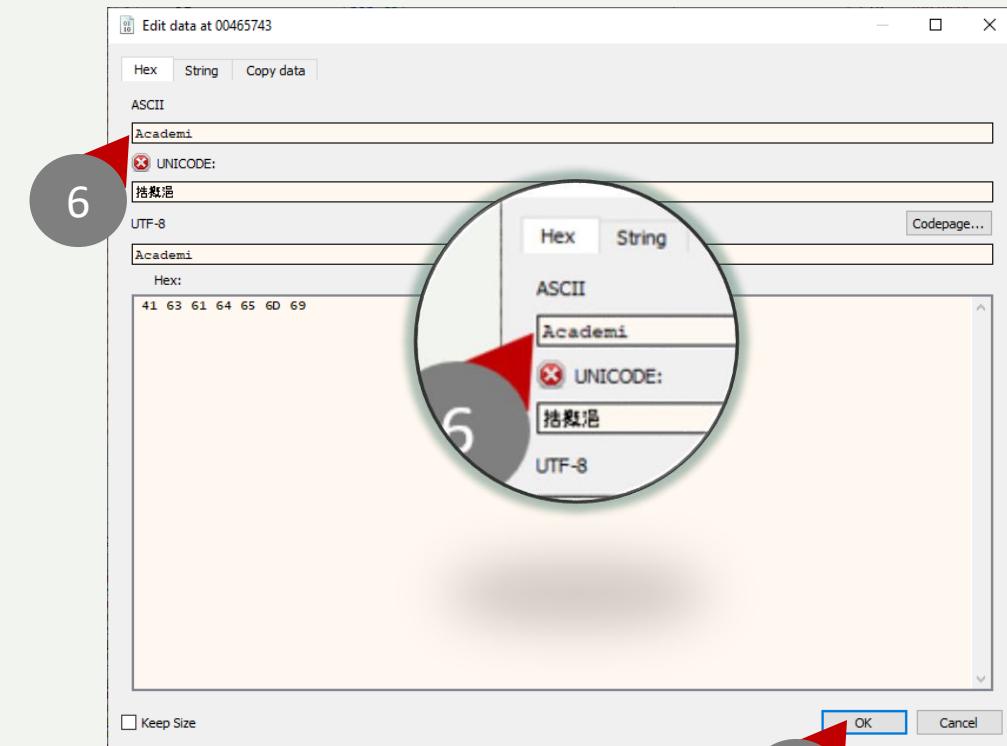
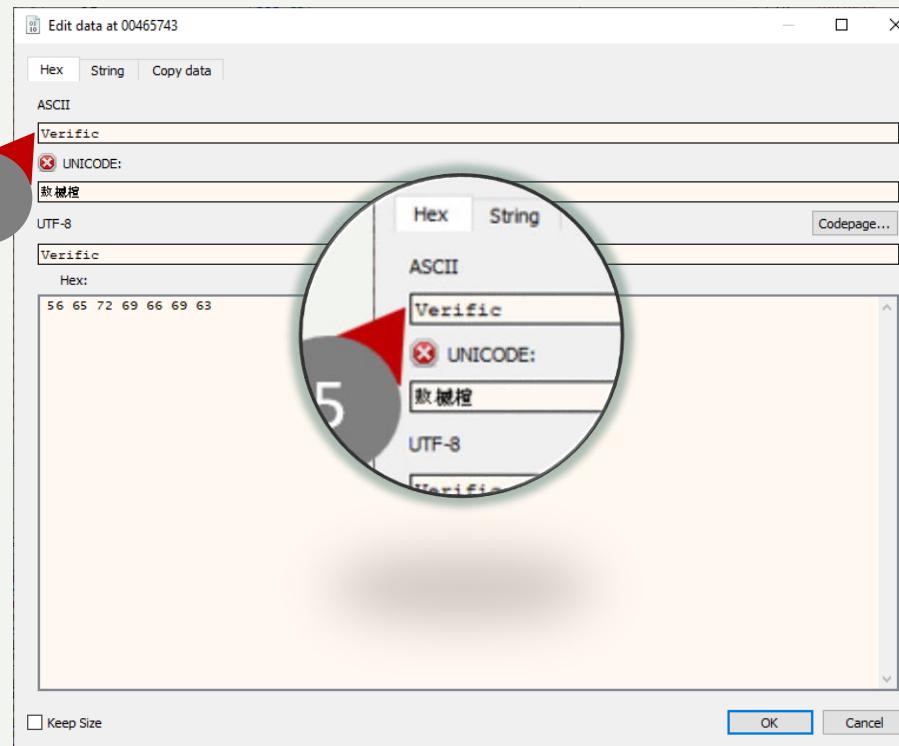
7. Click dreapta pe adresă → „Follow in Dump” pentru a vedea zona completă din memorie unde e stocat textul.



1. Am găsit stringul „Verifică” și îl vedem în zona Dump, sub formă de UNICODE.
 2. Confirmăm că stringul este mapat într-o zonă scrisă (de ex. RW-), deci poate fi modificat.
 3. Click dreapta pe instrucțiunea care referă acest string → Binary.
 4. Apoi alegem „Edit” pentru a modifica stringul (conținutul lui).



1. Selectăm stringul original „Verific” (format UNICODE), pe care vrem să-l înlocuim.
2. Introducem noul text în câmpul ASCII — aici „Academi”.
4. E important să nu depășim spațiul alocat în memorie. Apăsăm OK pentru a salva modificarea și a actualiza memoria procesului.



7

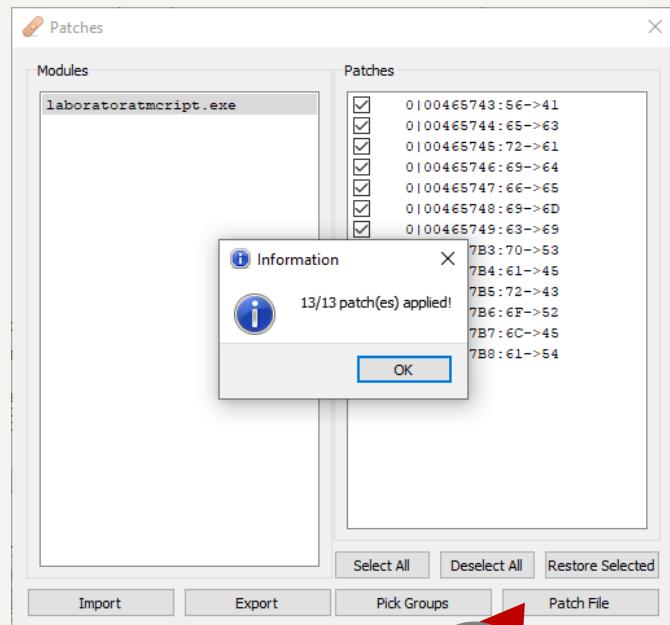
LaboratorATMScript.exe - PID: 3624 - Module: msvbvm60.dll - Thread: Main Thread 1036 - x32dbg

File View Debug Tracing Plugins Favourites Options Help Mar 15 2025 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols References Threads Help

```

709395DC F3:A5 rep movsd
709395DE 8BC8 mov ecx,ebx
709395E0 03E1 03 and ecx,3
709395E3 F3:A4 rep movsd
709395E5 8BCB mov ecx,ebx
709395E7 03D1 add edx,ecx
709395E9 8950 14 mov dword ptr ds:[eax+14],edx
709395EC 8B45 14 mov eax,dword ptr ss:[ebp+14]
709395EF 85C0 test eax,eax
709395F1 74 02 je msvbvm60.709395F5
709395F3 8908 mov dword ptr ds:[eax],ecx
709395F6 5F pop edi
709395F7 33C0 pop esi
709395F9 5B xor eax,eax
709395FA 5D pop ebx
709395FB C2 1000 ret 10
709395FF 55 push ebp
70939601 8BEC mov ebp,esp
70939602 51 push ecx
70939603 53 push ebx
70939604 56 push esi
70939605 8B75 10 mov esi,dword ptr ss:[ebp+10]
70939608 57 push edi
7093960C FF75 14 push dword ptr ss:[ebp+14]
7093960F 8B7D 0C mov edi,dword ptr ss:[ebp+10]
70939610 8D45 10 lea eax,dword ptr ss:[ebp+10]
70939612 33DB xor ebx,ebx
70939614 50 push eax
70939615 53 push ebx
70939616 56 push esi
70939617 53 push ebx
70939618 57 push edi
70939619 895D FC mov dword ptr ss:[ebp-4],ebx
7093961C FF75 08 push dword ptr ss:[ebp+8]
    call msvbvm60.709395F5
    EIP 77D44F3C ntdll.77D44F3C
    EFLAGS 00000206
    ZF 0 PF 1 AF 0
    OF 0 SF 0 DF 0
    CF 0 TF 0 IF 1
    LastError 00000000 (ERROR_SUCCESS)
    LastStatus C000000D (STATUS_INVALID)
    GS 0028 FS 0053
    ES 0028 DS 0028
    CS 0023 SS 0028
    ST(0) 00000000000000000000000000000000 x87r0 E
    ST(1) 00000000000000000000000000000000 x87r1 E
    ST(2) 00000000000000000000000000000000 x87r2 E
    ST(3) 00000000000000000000000000000000 x87r3 E
    ST(4) 00000000000000000000000000000000 x87r4 E
    ST(5) 00000000000000000000000000000000 x87r5 E
    ST(6) 00000000000000000000000000000000 x87r6 E
    ST(7) 00000000000000000000000000000000 x87r7 E
  
```



8. Confirmăm în Dump că stringul ASCII a fost schimbat corect — „Academia” apare în zona de memorie.

9. Aplicăm patch-ul final asupra fișierului executabil cu butonul Patch File.

- Interfața modificată afișează noul text pe buton: „Academia”.
- Comparatie: versiunea veche conținea textul original „Verifica”.

text:709395DC msvbvm60.dll#195DC #195DC

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1

Address	Hex	ASCII
0019F764	00000000	&"ThunderRT6Main"
0019F765	00000000	MBTOWCSE+210
0019F766	00000000	
0019F767	00000000	
0019F768	00000000	
0019F769	00000000	
0019F76A	00000000	
0019F76B	00000000	
0019F76C	00000000	
0019F76D	00000000	
0019F76E	00000000	
0019F76F	00000000	
0019F770	00000000	
0019F771	00000000	
0019F772	00000000	
0019F773	00000000	
0019F774	00000000	
0019F775	00000000	
0019F776	00000000	
0019F777	00000000	
0019F778	00000000	
0019F779	00000000	
0019F77A	00000000	
0019F77B	00000000	
0019F77C	00000000	
0019F77D	00000000	
0019F77E	00000000	
0019F77F	00000000	
0019F780	00000000	
0019F781	00000000	
0019F782	00000000	
0019F783	00000000	
0019F784	00000000	
0019F785	00000000	
0019F786	00000000	
0019F787	00000000	
0019F788	00000000	
0019F789	00000000	
0019F78A	00000000	
0019F78B	00000000	
0019F78C	00000000	
0019F78D	00000000	
0019F78E	00000000	
0019F78F	00000000	
0019F790	00000000	
0019F791	00000000	
0019F792	00000000	
0019F793	00000000	
0019F794	00000000	
0019F795	00000000	
0019F796	00000000	
0019F797	00000000	
0019F798	00000000	
0019F799	00000000	
0019F79A	00000000	
0019F79B	00000000	
0019F79C	00000000	
0019F79D	00000000	
0019F79E	00000000	
0019F79F	00000000	
0019F7A0	00000000	
0019F7A1	00000000	
0019F7A2	00000000	
0019F7A3	00000000	
0019F7A4	00000000	
0019F7A5	00000000	
0019F7A6	00000000	
0019F7A7	00000000	
0019F7A8	00000000	
0019F7A9	00000000	
0019F7AA	00000000	
0019F7AB	00000000	
0019F7AC	00000000	
0019F7AD	00000000	
0019F7AE	00000000	
0019F7AF	00000000	
0019F7B0	00000000	
0019F7B1	00000000	
0019F7B2	00000000	
0019F7B3	00000000	
0019F7B4	00000000	
0019F7B5	00000000	
0019F7B6	00000000	
0019F7B7	00000000	
0019F7B8	00000000	
0019F7B9	00000000	
0019F7BA	00000000	
0019F7BB	00000000	
0019F7BC	00000000	
0019F7BD	00000000	
0019F7BE	00000000	
0019F7BF	00000000	
0019F7C0	00000000	
0019F7C1	00000000	
0019F7C2	00000000	
0019F7C3	00000000	
0019F7C4	00000000	
0019F7C5	00000000	
0019F7C6	00000000	
0019F7C7	00000000	
0019F7C8	00000000	
0019F7C9	00000000	
0019F7CA	00000000	
0019F7CB	00000000	
0019F7CC	00000000	
0019F7CD	00000000	
0019F7CE	00000000	
0019F7CF	00000000	
0019F7D0	00000000	
0019F7D1	00000000	
0019F7D2	00000000	
0019F7D3	00000000	
0019F7D4	00000000	
0019F7D5	00000000	
0019F7D6	00000000	
0019F7D7	00000000	
0019F7D8	00000000	
0019F7D9	00000000	
0019F7DA	00000000	
0019F7DB	00000000	
0019F7DC	00000000	
0019F7DD	00000000	
0019F7DE	00000000	
0019F7DF	00000000	
0019F7E0	00000000	
0019F7E1	00000000	
0019F7E2	00000000	
0019F7E3	00000000	
0019F7E4	00000000	
0019F7E5	00000000	
0019F7E6	00000000	
0019F7E7	00000000	
0019F7E8	00000000	
0019F7E9	00000000	
0019F7EA	00000000	
0019F7EB	00000000	
0019F7EC	00000000	
0019F7ED	00000000	
0019F7EE	00000000	
0019F7EF	00000000	
0019F7F0	00000000	
0019F7F1	00000000	
0019F7F2	00000000	
0019F7F3	00000000	
0019F7F4	00000000	
0019F7F5	00000000	
0019F7F6	00000000	
0019F7F7	00000000	
0019F7F8	00000000	
0019F7F9	00000000	
0019F7F0	00000000	
0019F7F1	00000000	
0019F7F2	00000000	
0019F7F3	00000000	
0019F7F4	00000000	
0019F7F5	00000000	
0019F7F6	00000000	
0019F7F7	00000000	
0019F7F8	00000000	
0019F7F9	00000000	
0019F7F0	00000000	
0019F7F1	00000000	
0019F7F2	00000000	
0019F7F3	00000000	
0019F7F4	00000000	
0019F7F5	00000000	
0019F7F6	00000000	
0019F7F7	00000000	
0019F7F8	00000000	
0019F7F9	00000000	
0019F7F0	00000000	
0019F7F1	00000000	
0019F7F2	00000000	
0019F7F3	00000000	
0019F7F4	00000000	
0019F7F5	00000000	
0019F7F6	00000000	
0019F7F7	00000000	
0019F7F8	00000000	
0019F7F9	00000000	
0019F7F0	00000000	
0019F7F1	00000000	
0019F7F2	00000000	
0019F7F3	00000000	
0019F7F4	00000000	
0019F7F5	00000000	
0019F7F6	00000000	
0019F7F7	00000000	
0019F7F8	00000000	
0019F7F9	00000000	
0019F7F0	00000000	
0019F7F1	00000000	
0019F7F2	00000000	
0019F7F3	00000000	
0019F7F4	00000000	
0019F7F5	00000000	
0019F7F6	00000000	
0019F7F7	00000000	
0019F7F8	00000000	
0019F7F9	00000000	
0019F7F0	00000000	
0019F7F1	00000000	
0019F7F2	00000000	
0019F7F3	00000000	
0019F7F4	00000000	
0019F7F5	00000000	
0019F7F6	00000000	
0019F7F7	00000000	
0019F7F8	00000000	
0019F7F9	00000000	
0019F7F0	00000000	
0019F7F1	00000000	
0019F7F2	00000000	
0019F7F3	00000000	
0019F7F4	00000000	
0019F7F5	00000000	
0019F7F6	00000000	
0019F7F7	00000000	
0019F7F8	00000000	
0019F7F9	00000000	
0019F7F0	00000000	
0019F7F1	00000000	
0019F7F2	00000000	
0019F7F3	00000000	
0019F7F4	00000000	
0019F7F5	00000000	
0019F7F6	00000000	
0019F7F7	00000000	
0019F7F8	00000000	
0019F7F9	00000000	
0019F7F0	00000000	
0019F7F1	00000000	
0019F7F2	00000000	
0019F7F3	00000000	
0019F7F4	00000000	
0019F7F5	00000000	
0019F7F6	00000000	
0019F7F7	00000000	
0019F7F8	00000000	
0019F7F9	00000000	
0019F7F0	00000000	
0019F7F1	00000000	
0019F7F2	00000000	
0019F7F3	00000000	
0019F7F4	00000000	
0019F7F5	00000000	
0019F7F6	00000000	
0019F7F7	00000000	
0019F7F8	00000000	
0019F7F9	00000000	
0019F7F0	00000000	
0019F7F1	00000000	
0019F7F2	00000000	
0019F7F3	00000000	
0019F7F4	00000000	
0019F7F5	00000000	
0019F7F6	00000000	
0019F7F7	00000000	
0019F7F8</		

9.4

MODIFICAREA STRINGURIILOR IN MESAJE





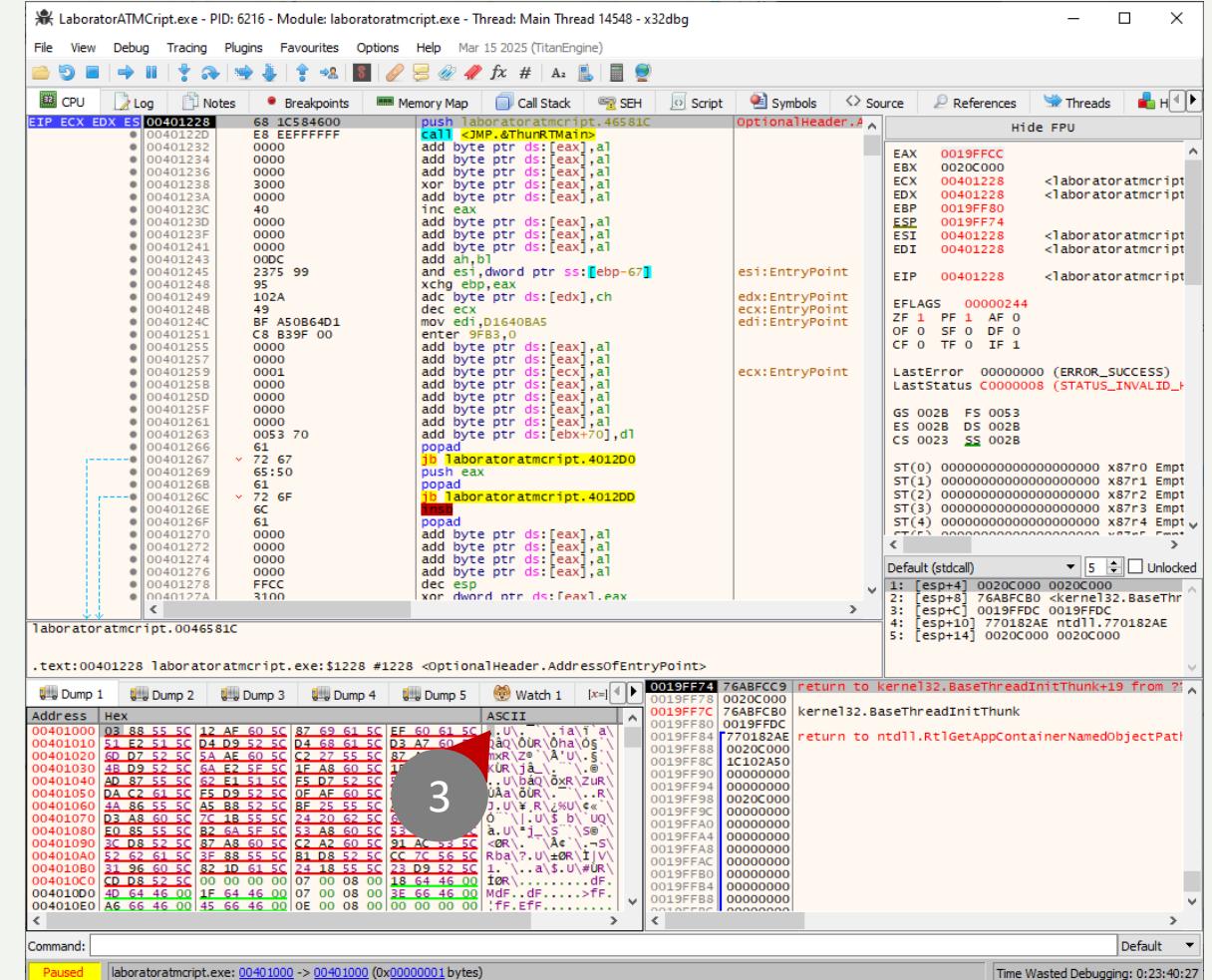
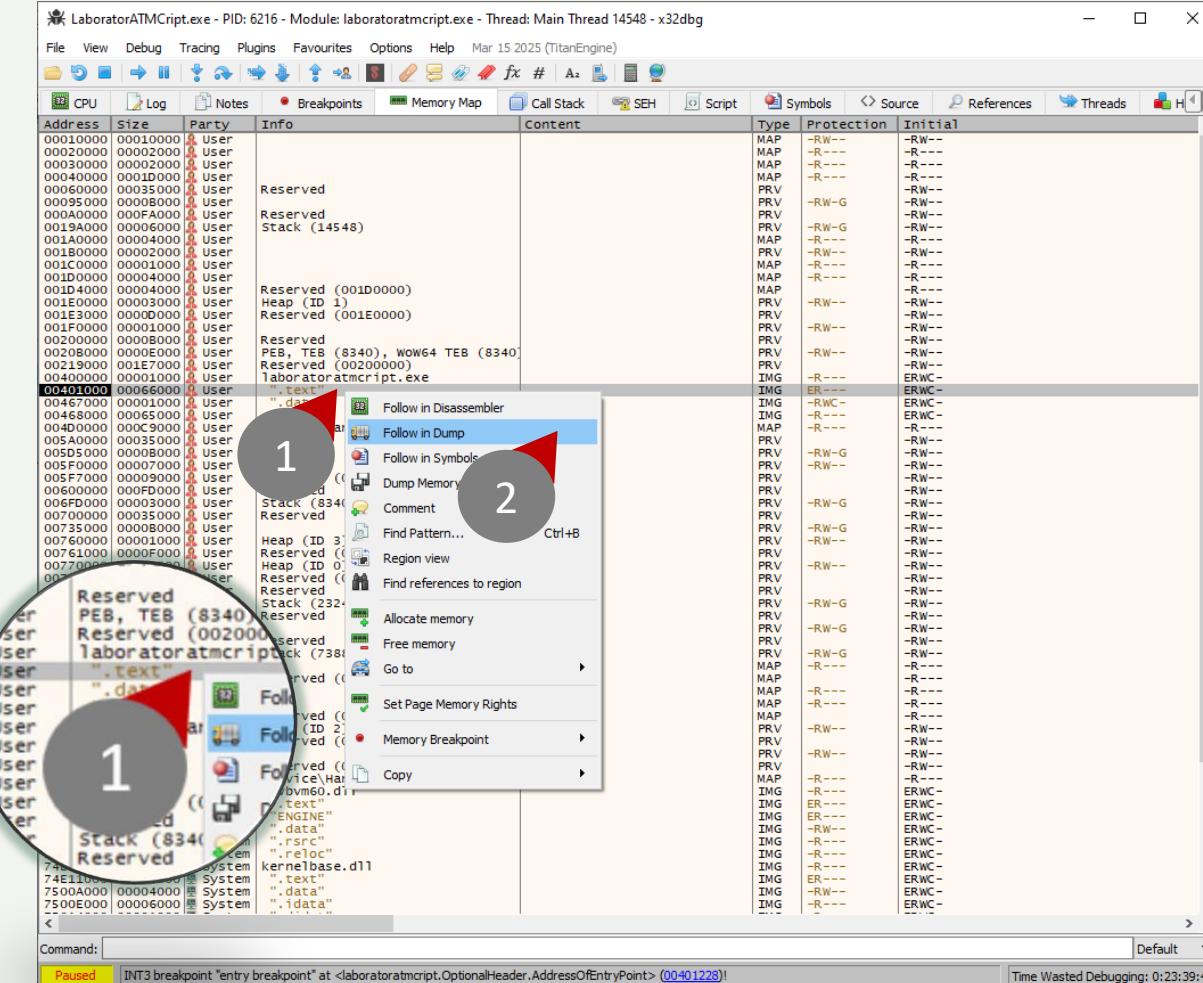
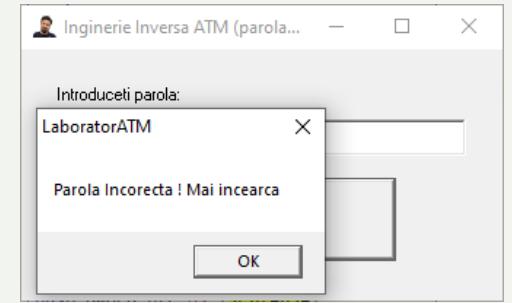
VERSIUNEA (I) CONTINUAM EXEMPLUL ANTERIOR ...

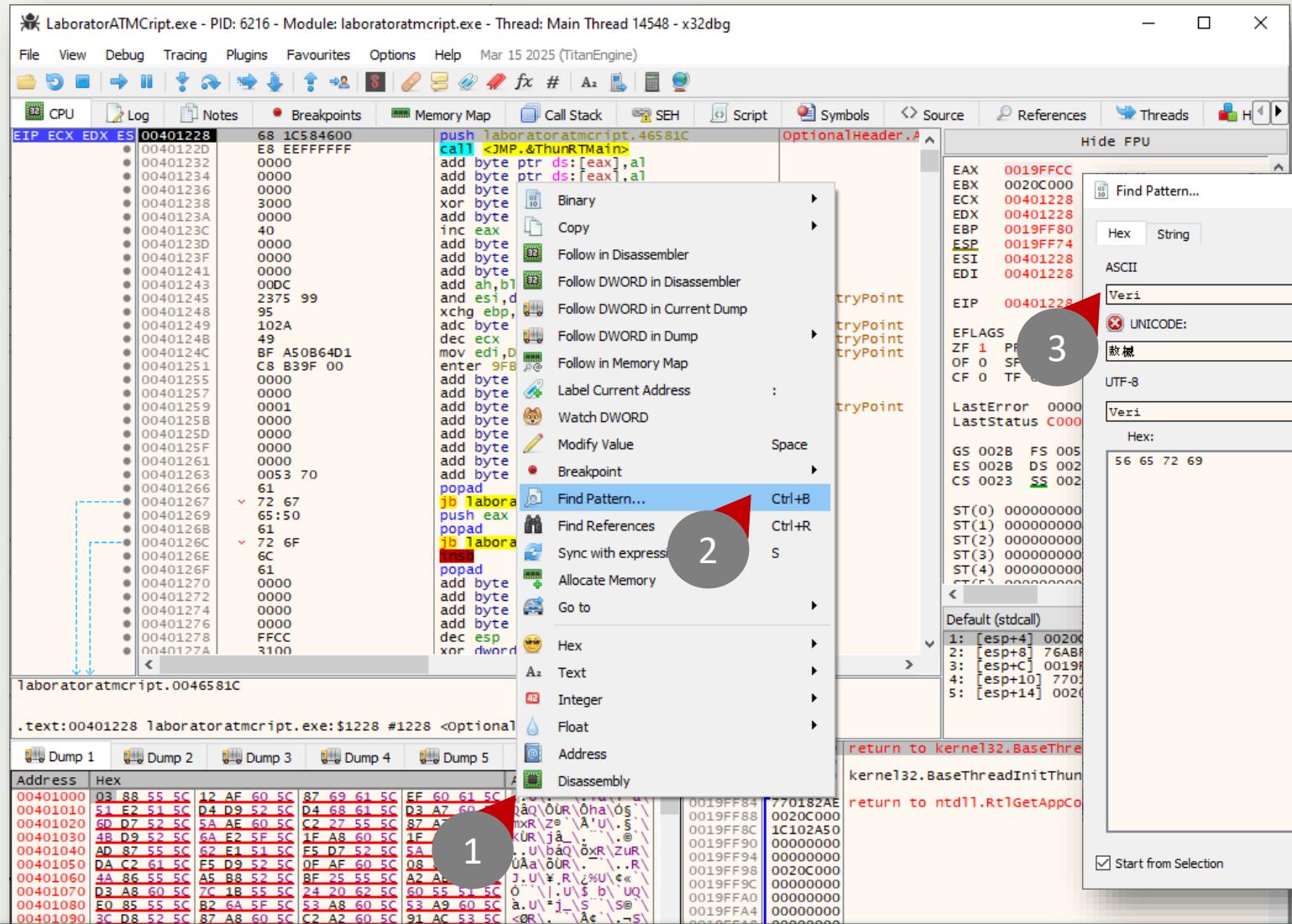
Metoda exploratorului !

În situațiile în care structura internă a unui program este necunoscută, iar indiciile sunt puține, intervine Metoda exploratorului. Această abordare nu urmează pași prestabiliți, ci presupune o navigare activă și atentă prin memorie, disasembler și dump, căutând vizual orice secvență recognoscibilă: un text, o semnătură, un fragment de cod. Scopul nu este identificarea imediată a ţintei, ci declanșarea unui „declick” mental — acel moment în care o bucată de informație aparent banală face legătura cu o logică ascunsă. Această metodă dezvoltă instinctul analitic și oferă libertatea de a explora, exact ca un cercetător care descoperă sensuri noi într-un peisaj necunoscut.

Nota: Stringurile pot fi stocate și în alte regiuni de memorie, nu doar în cele clasice.

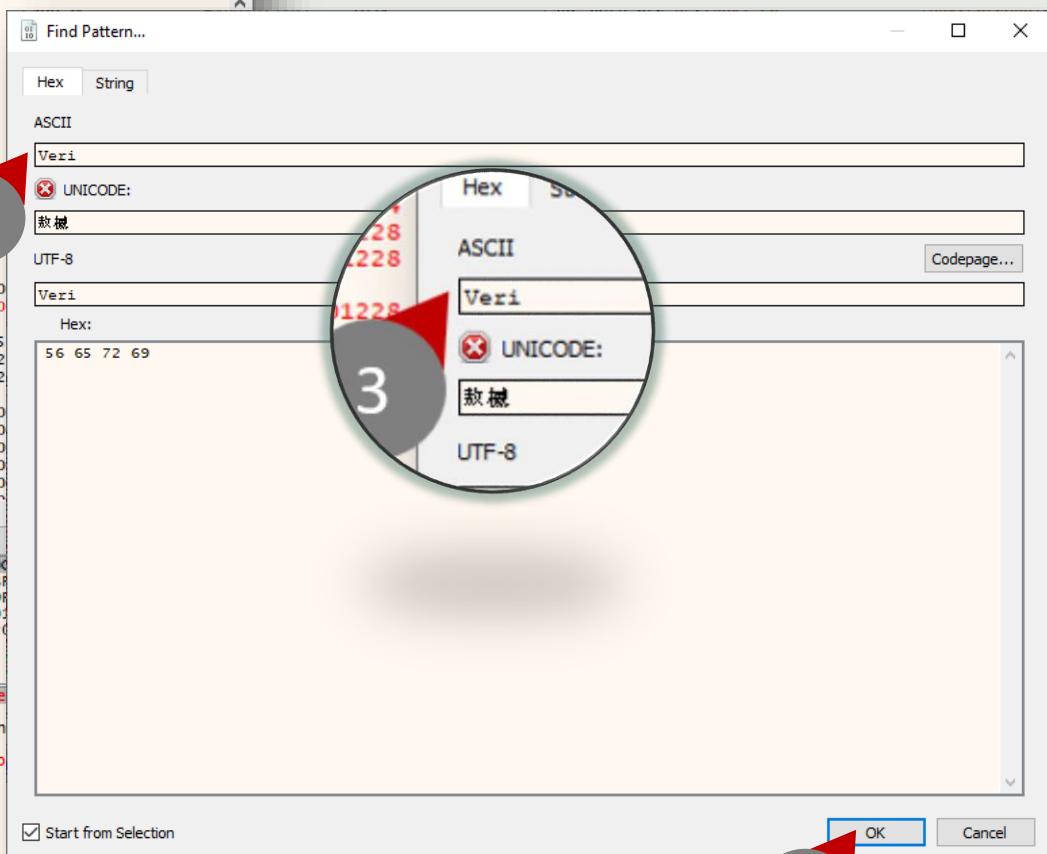
1. Se selectează secțiunea .text din Memory Map.
2. Click dreapta → Follow in Dump.
3. Dump-ul se deschide exact la începutul secțiunii .text, loc în care putem începe să căutăm pattern-uri.





1. Dăm click dreapta pe primul octet din zona Dump, corespunzător începutului secțiunii .text.
2. Se alege „Find Pattern...” pentru a căuta un string cunoscut.
3. Se scrie „Veri” în tabul String, iar hex-ul este generat automat.
5. Se apasă OK pentru a începe căutarea în .text.

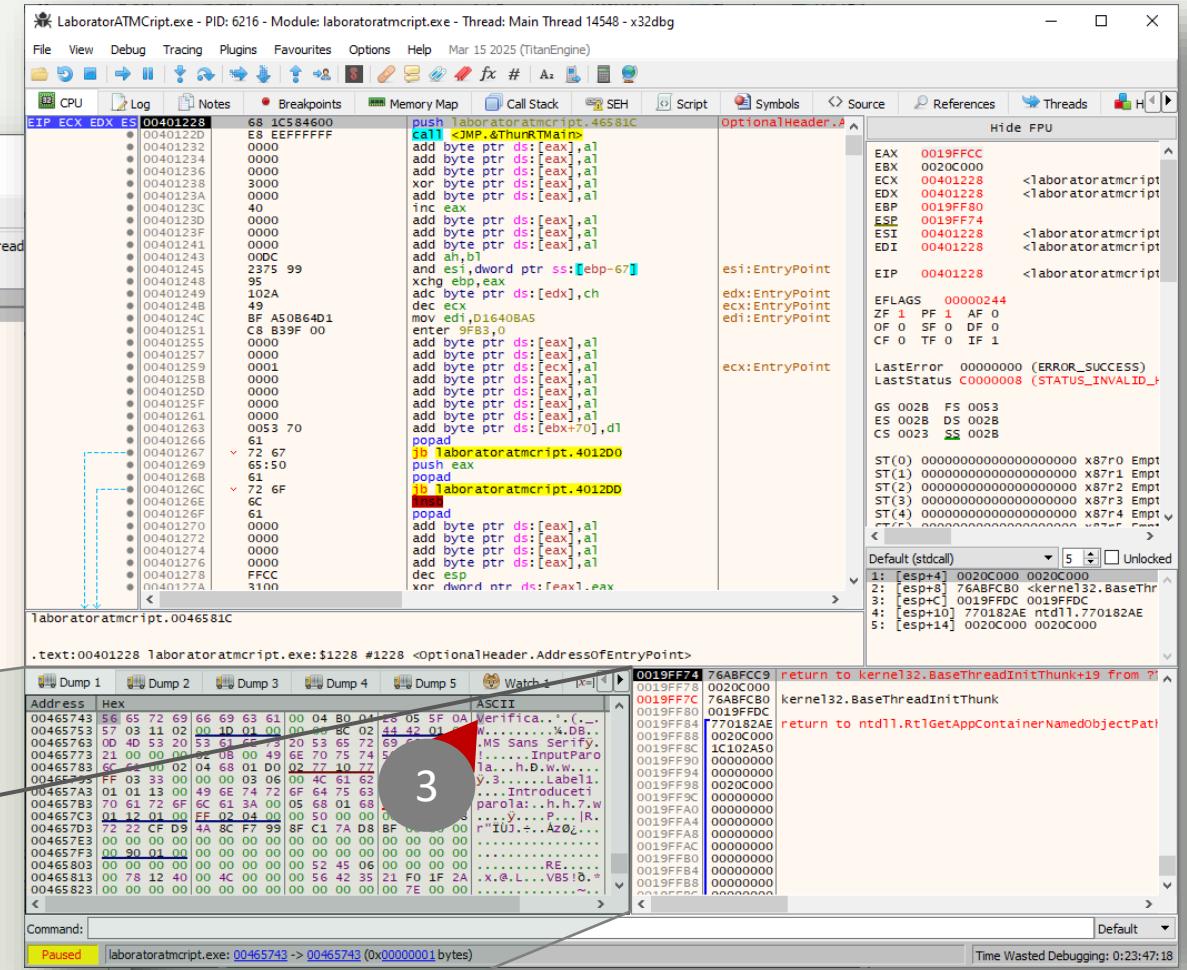
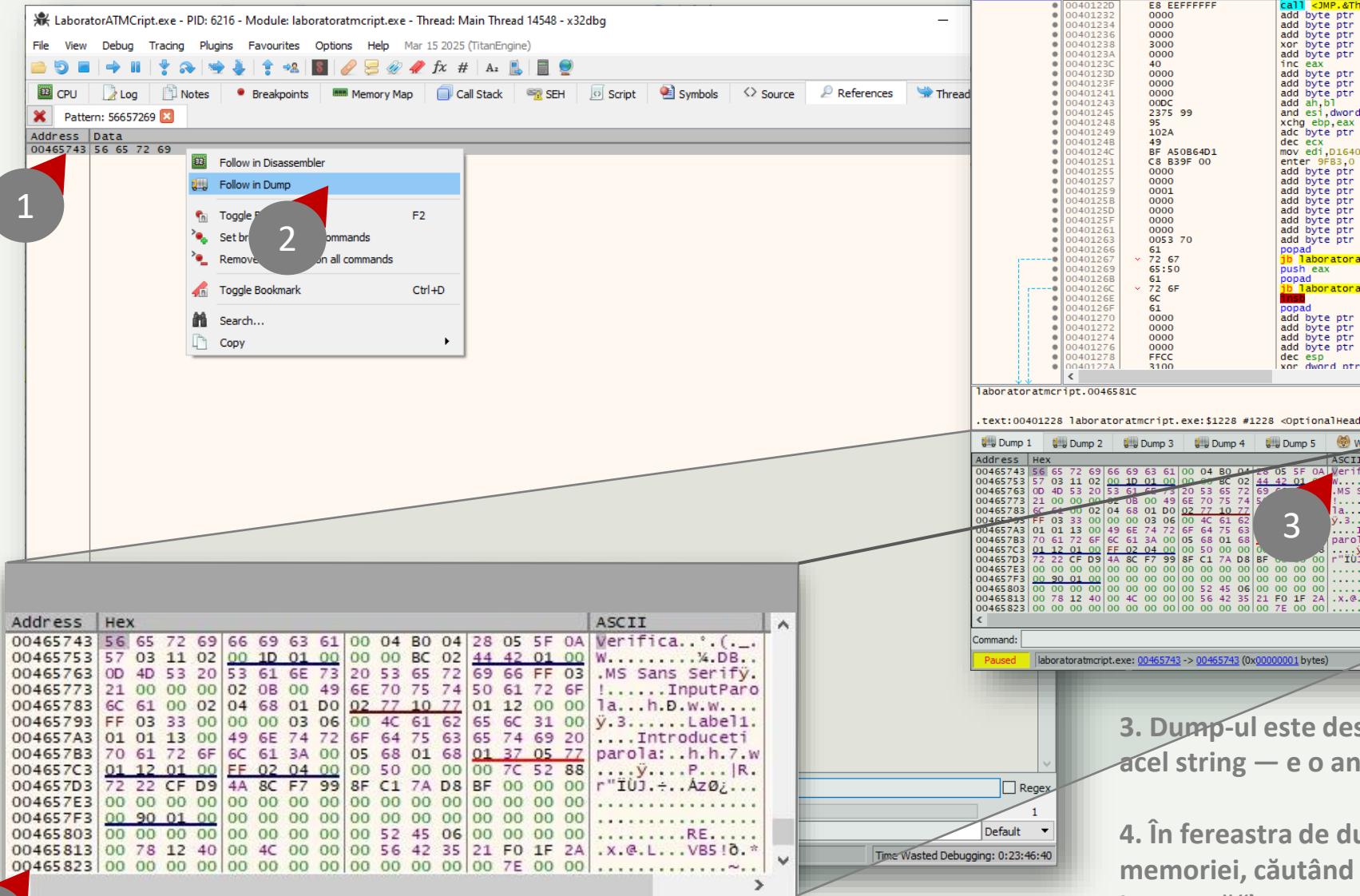
Căutăm stringul "Veri" de pe buton în format ASCII, în speranță că în memoria executabilului — mai precis în zona secțiunii .text — acesta este stocat împreună cu alte stringuri hardcodede, precum textul afișat de MessageBoxA.



Localizarea exactă a offsetului unde este hardcodat stringul verificat, chiar dacă nu se află în .data/.rdata.

4

1. Adresa stringului „Veri” a fost localizată în memorie (pattern găsit).
2. Se face Follow in Dump pentru a analiza manual zona.



3. Dump-ul este deschis în paralel cu zona de cod care folosește acel string — e o ancoră bună pentru identificare contextuală.

4. În fereastra de dump (video), se parcurge manual vecinătatea memoriei, căutând stringul din fereastra MessageBoxA ("Parola incorrectă") — pentru a fi modificat ulterior.

- Se identifică în dump stringul complet „Parola incorectă: Mai încearcă”, confirmând prezența lui în memorie.
- Se selectează în dump zona relevantă pentru editare.
- Se face click dreapta pe selecție → Binary.
- Se alege opțiunea Edit pentru a modifica mesajul afișat de MessageBoxA.

LaboratorATMCript.exe - PID: 3624 - Module: laboratoratmcrypt.exe - Thread: 5000 (switched from Main Thread) - x32dbg

File View Debug Tracing Plugins Favourites Options Help Mar 15 2025 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Stack SEH Script Symbols References Threads

```

xchgb ebx, eax
and byte ptr ds:[eax], al
add byte ptr ds:[eax], al
push eax
EAX 00000000
EBX 00000000
ECX 00000000
EDX 00000000
EBP 059EFF70
ESP 059EFD84
ESI 00327000
EDI 006B07F0

```

EIP 77D44F3C ntdll.77D44F3C

EFALGS 00000206
ZF 0 PF 1 AF 0
OF 0 SF 0 DF 0
CF 0 TF 0 IF 1

Last Error 00000000 (ERROR_SUCCESS)
Last Status C000000D (STATUS_INVALID)

GS 0028 FS 0053
ES 0028 DS 0028
CS 0023 SS 0028

eax=0

.text:0046601C laboratoratmcrypt.exe:\$6601C #6601C

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1

Address	Hex	ASCII
00465F83	6F 6E 00 00 DA 4E AD 33 99 66 CF 11 B7 0C 00	on...UN.3.f... .Label1. an...
00465F83	AA 00 60 D3 93 4C 61 62 65 6C 31 00	0.0.Inp
00465F83	33 99 66 CF 11 B7 0C 00 AA 00 D3 93 49 66 70 3.f... .Inp	utParola...P...
00465F83	75 74 50 61 72 6F 61 00 10 00 50 00 00 01	a...4...a.b
00465F83	00 00 00 00 34 01 00 00 00 00 00 00 00 01	0019F778 00EBBS30
00466003	00 63 00 00 E1 4E AD 33 99 66 CF 11 B7 0C 00	c...an.3.f... .Bar.o
00466013	AA 00 60 D3 93 20 00 60 00 00 00 00 00 00 00	0019F780 00000011
00466023	00 60 00 61 00 20 00 63 00 00 00 00 00 00 00	1.a...c.or.e.c
00466033	00 74 00 61 00 20 00 63 00 00 00 00 00 00 00	0019F790 00000004
00466043	00 50 00 61 00 72 4F 6C 00 00 00 00 00 00 00	P.ar.o.la.I .i
00466053	00 00 00 21 00 00 00 00 00 00 00 00 00 00 00	0019F798 00000000
00466073	00 66 00 63 00 00 00 00 00 00 00 00 00 00 00	0019F79A 00000004
00466083	00 00 00 00 5F 76 61 45 72 62 63 74 00 00 00	...VBA6.DL
00466093	4C 00 00 00 5F 76 61 45 72 62 63 74 00 00 00	0019F7A0 00000003
004660E3	76 65 72 66 6C 77 00 00 5F 76 62 63 74 00 00 00	verflow...vbas
00466083	72 65 61 72 4D 76 65 00 5F 76 62 63 4C 65 00	rvarMove...vba
00466083	6E 42 73 72 00 00 00 5F 76 62 63 4F 62 nbsr...vba	0019F7A1 00000000
004660D3	6A 53 65 74 00 5F 76 62 64 32 49 34 00 00	jset...vbaZI2
004660E3	58 57 76 61 53 74 00 4B 40 60 00 00 00 00 00	4...a.b
00466103	00 5F 5F 76 61 53 74 00 4B 40 60 00 00 00 00 00	vbaMove...vba
00466113	62 61 56 61 72 43 61 74 00 5F 76 62 63 74 00	vbaStro...vba
00466123	62 61 46 72 65 65 53 74 00 00 00 00 00 00 00 00	1.a...c.or.e.c
00466133	62 61 46 72 65 65 56 61 72 4C 69 73 00 00 00 00	baFreeStr...vba
00466143	00 5F 57 76 62 61 56 61 72 44 75 70 00 5F 76 62 63 74 00	baVarBu...vba
00466153	62 61 46 72 65 65 56 61 72 00 00 00 00 00 00 00	baFreeVar...vba
00466163	62 61 46 72 65 65 4F 62 00 00 00 00 00 00 00 00	baFreeObj...vba
00466173	62 61 48 72 65 73 4F 62 63 68 4F 62 baResult...vba	0019F7A2 00000000
00466183	62 61 46 72 65 73 4F 62 63 68 4F 62 baMove...vba	69 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00466193	71 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	Eq...4af
004661A3	65 00 00 00 BC 61 46 72 65 65 56 61 72 44 68 65 00	VBA6.DL
004661B3	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	baFree...vba
004661C3	00 FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00	overflow...vbast

Command: Commands are comma separated (like assembly instructions): mov eax, ebx

Running laboratoratmcrypt.exe: 0046601C -> 0046601C (0x0000001 bytes)

Time Wasted Debugging: 0:07:23:14

LaboratorATMCript.exe - PID: 3624 - Module: laboratoratmcrypt.exe - Thread: 5000 (switched from Main Thread) - x32dbg

File View Debug Tracing Plugins Favourites Options Help Mar 15 2025 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Stack SEH Script Symbols References Threads

```

xchgb ebx, eax
and byte ptr ds:[eax], al
add byte ptr ds:[eax], al
push eax
EAX 00000000
EBX 00000000
ECX 00000000
EDX 00000000
EBP 059EFF70
ESP 059EFD84
ESI 00327000
EDI 006B07F0

```

EIP 77D44F3C ntdll.77D44F3C

EFALGS 00000206
ZF 0 PF 1 AF 0
OF 0 SF 0 DF 0
CF 0 TF 0 IF 1

Last Error 00000000 (ERROR_SUCCESS)
Last Status C000000D (STATUS_INVALID)

GS 0028 FS 0053
ES 0028 DS 0028
CS 0023 SS 0028

eax=0

.text:0046601C laboratoratmcrypt.exe:\$6601C #6601C

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1

Address	Hex	ASCII
00465F83	00 09 00 00 3C 5F 46 00 74 5F 46 00 D0 72 46	...<_F_t...pr... .Label1. an...
00465F83	00 09 00 00 3C 5F 46 00 74 5F 46 00 C0 A5 60 00	3.f... .Inp
00465F83	6F 6E 00 00 DA 4E AD 33 99 66 CF 11 B7 0C 00	utParola...P...
00465F83	33 99 66 CF 11 B7 0C 00 AA 00 D3 93 49 66 70 3.f... .Inp	a...4...a.b
00465F83	6F 6E 00 00 DA 4E AD 33 99 66 CF 11 B7 0C 00	0019F778 00EBBS30
00465F83	33 99 66 CF 11 B7 0C 00 AA 00 D3 93 49 66 70 3.f... .Inp	c...an.3.f... .Bar.o
00465F83	6F 6E 00 00 DA 4E AD 33 99 66 CF 11 B7 0C 00	0019F780 00000011
00465F83	6F 6E 00 00 DA 4E AD 33 99 66 CF 11 B7 0C 00	1.a...c.or.e.c
00465F83	6F 6E 00 00 DA 4E AD 33 99 66 CF 11 B7 0C 00	0019F780 00000004
00466003	00 63 00 00 E1 4E AD 33 99 66 CF 11 B7 0C 00	P.ar.o.la.I .i
00466013	AA 00 60 D3 93 20 00 60 00 00 00 00 00 00 00	0019F798 00000000
00466023	00 60 00 61 00 20 00 63 00 00 00 00 00 00 00	0019F79A 00000004
00466033	00 74 00 61 00 20 00 63 00 00 00 00 00 00 00	...VBA6.DL
00466043	00 50 00 61 00 72 4F 6C 00 00 00 00 00 00 00	0019F7A0 00000003
00466053	00 00 00 21 00 00 00 00 00 00 00 00 00 00 00	0019F7A1 00000000
00466073	00 66 00 63 00 00 00 00 00 00 00 00 00 00 00	0019F7A2 00000000
00466083	00 00 00 00 5F 76 61 45 72 62 63 74 00 00 00	verflow...vbas
00466083	00 00 00 00 5F 76 61 45 72 62 63 74 00 00 00	rvarMove...vba
00466083	00 00 00 00 5F 76 61 45 72 62 63 74 00 00 00	nbsr...vba
004660D3	00 53 65 74 00 5F 76 62 64 32 49 34 00 00	jset...vbaZI2
004660E3	58 57 76 61 53 74 00 4B 40 60 00 00 00 00 00	4...a.b
00466103	00 5F 5F 76 61 53 74 00 4B 40 60 00 00 00 00	vbaMove...vba
00466113	62 61 56 61 72 43 61 74 00 5F 76 62 63 74 00	vbaStro...vba
00466123	62 61 46 72 65 65 53 74 00 00 00 00 00 00 00	1.a...c.or.e.c
00466133	62 61 46 72 65 65 56 61 72 4C 69 73 00 00 00	baFreeStr...vba
00466143	00 5F 57 76 62 61 56 61 72 44 75 70 00 5F 76 62 63 74 00	baVarBu...vba
00466153	62 61 46 72 65 65 56 61 72 00 00 00 00 00 00	baFreeVar...vba
00466163	62 61 46 72 65 65 4F 62 00 00 00 00 00 00 00	baFreeObj...vba
00466173	62 61 48 72 65 73 4F 62 63 68 4F 62 baResult...vba	0019F7A2 00000000
00466183	62 61 46 72 65 73 4F 62 63 68 4F 62 baMove...vba	69 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00466193	71 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	Eq...4af
004661A3	65 00 00 00 BC 61 46 72 65 65 56 61 72 44 68 65 00	VBA6.DL
004661B3	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	baFree...vba
004661C3	00 FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00	overflow...vbast

Command: Commands are comma separated (like assembly instructions): mov eax, ebx

Running laboratoratmcrypt.exe: 0046601C -> 0046603A (0x0000001 bytes)

Time Wasted Debugging: 0:07:25:13

1

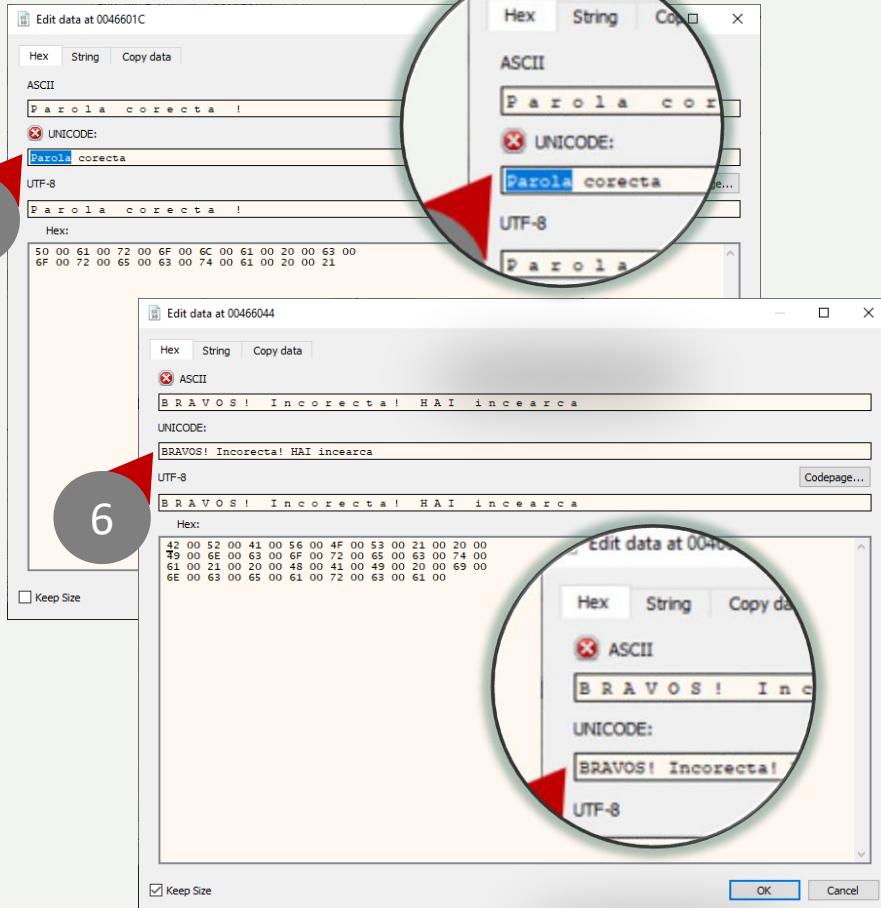
2

2

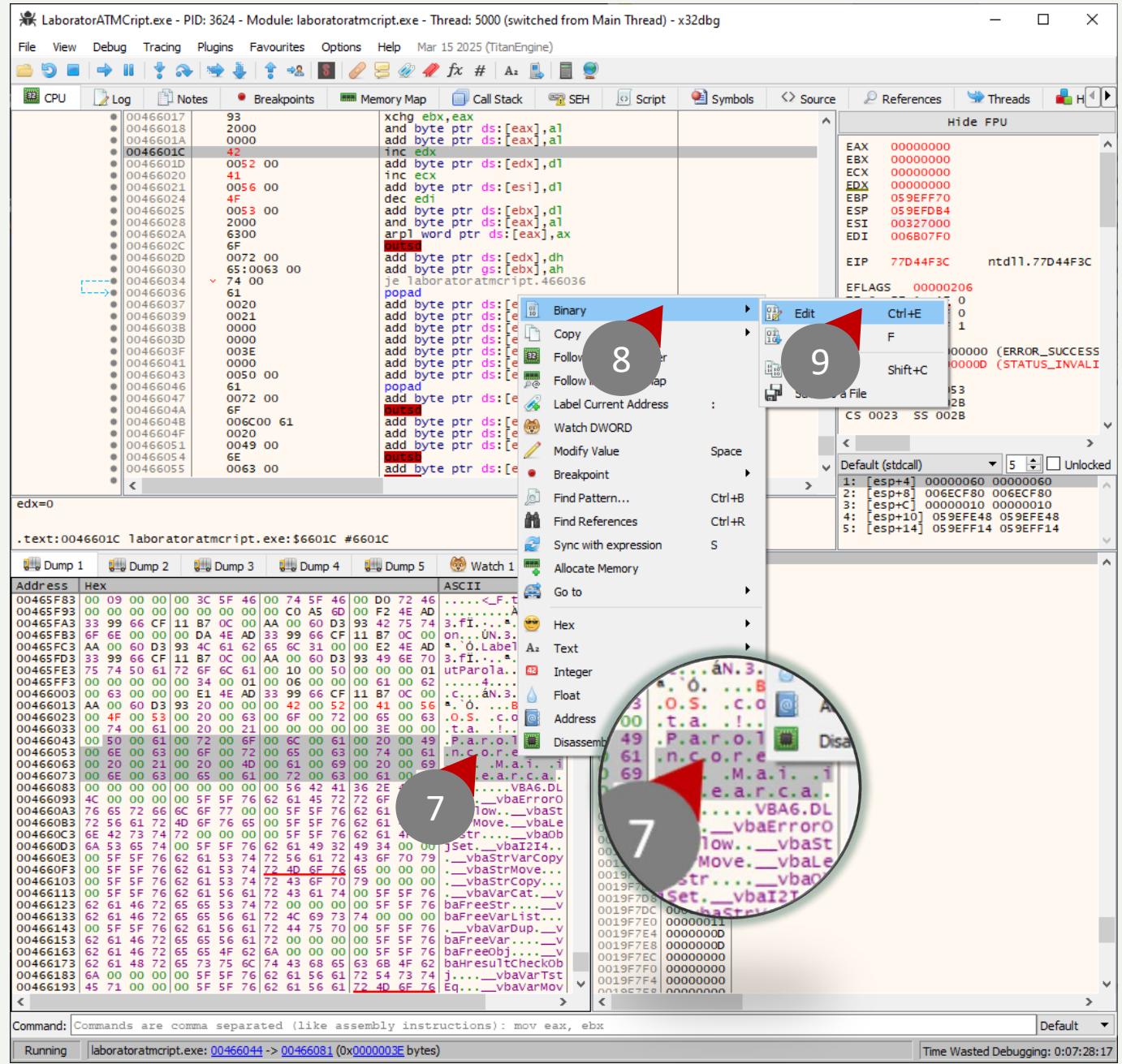
2

3

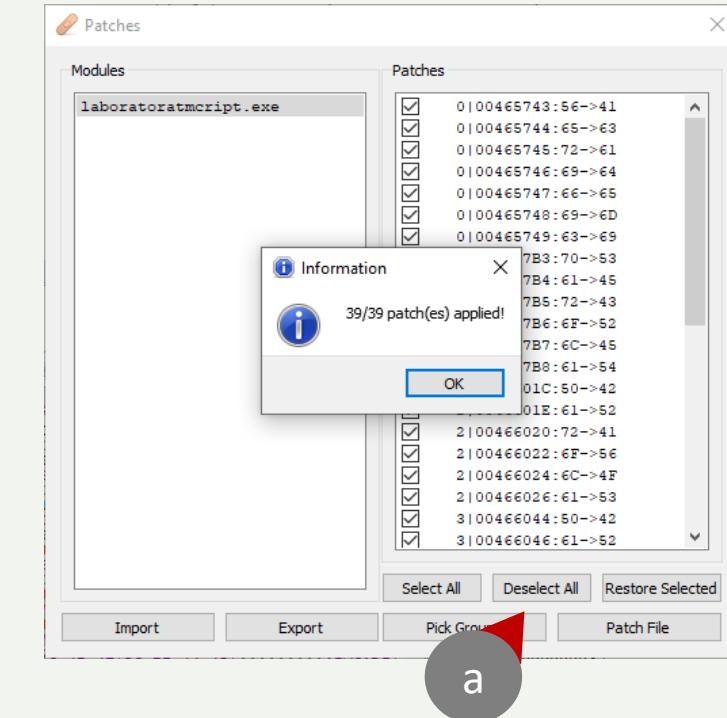
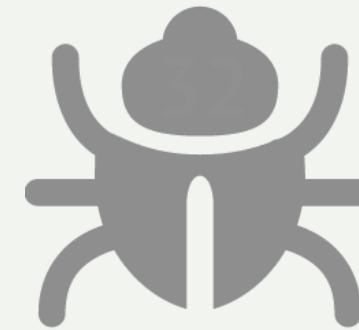
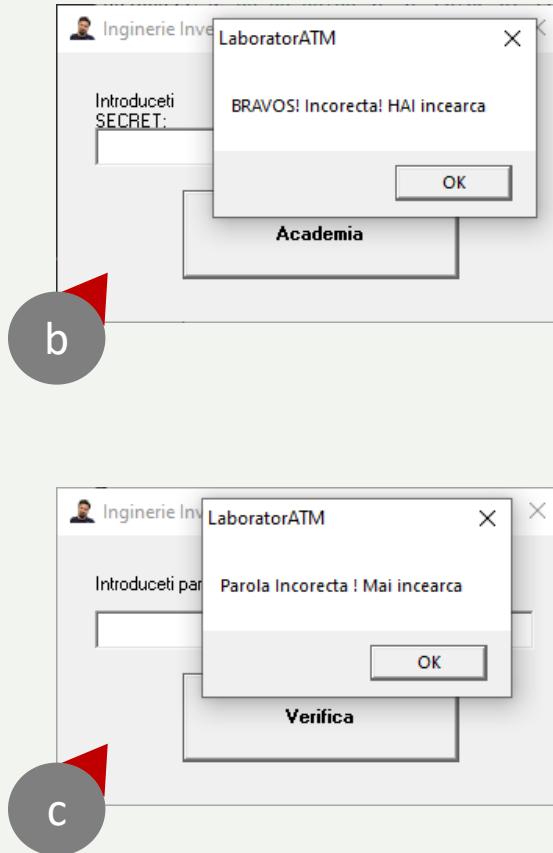
4



5. Se editează primul string: înlocuim „Parola incorrectă” cu „Parola corectă!” pentru a schimba mesajul pozitiv.
6. Se modifică al doilea string: transformăm mesajul original într-un text ironico-glorios – „BRAVOS! Incorrectă! HAI încearcă”.
7. Se selectează zona din dump în care se află stringul editat.
8. Click dreapta → Binary.
9. Alegem Edit pentru a confirma modificarea în memorie.



◆ În concluzie: prin editarea directă a stringurilor în dump și aplicarea patch-urilor din x32dbg, am reușit să personalizăm complet mesajele din interfața grafică a programului — fără a recompila codul sursă.



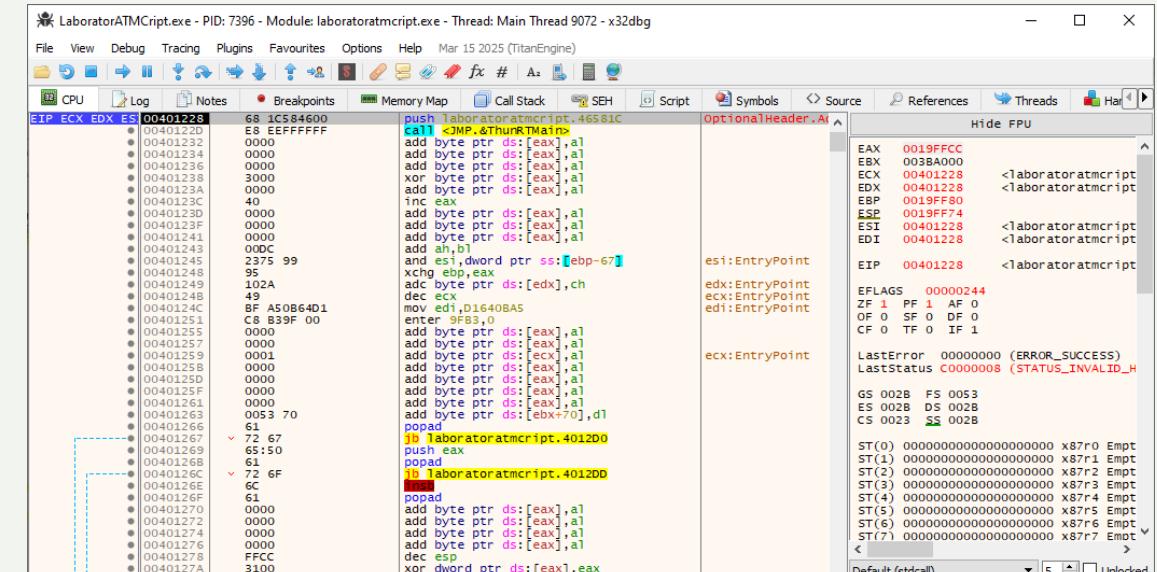
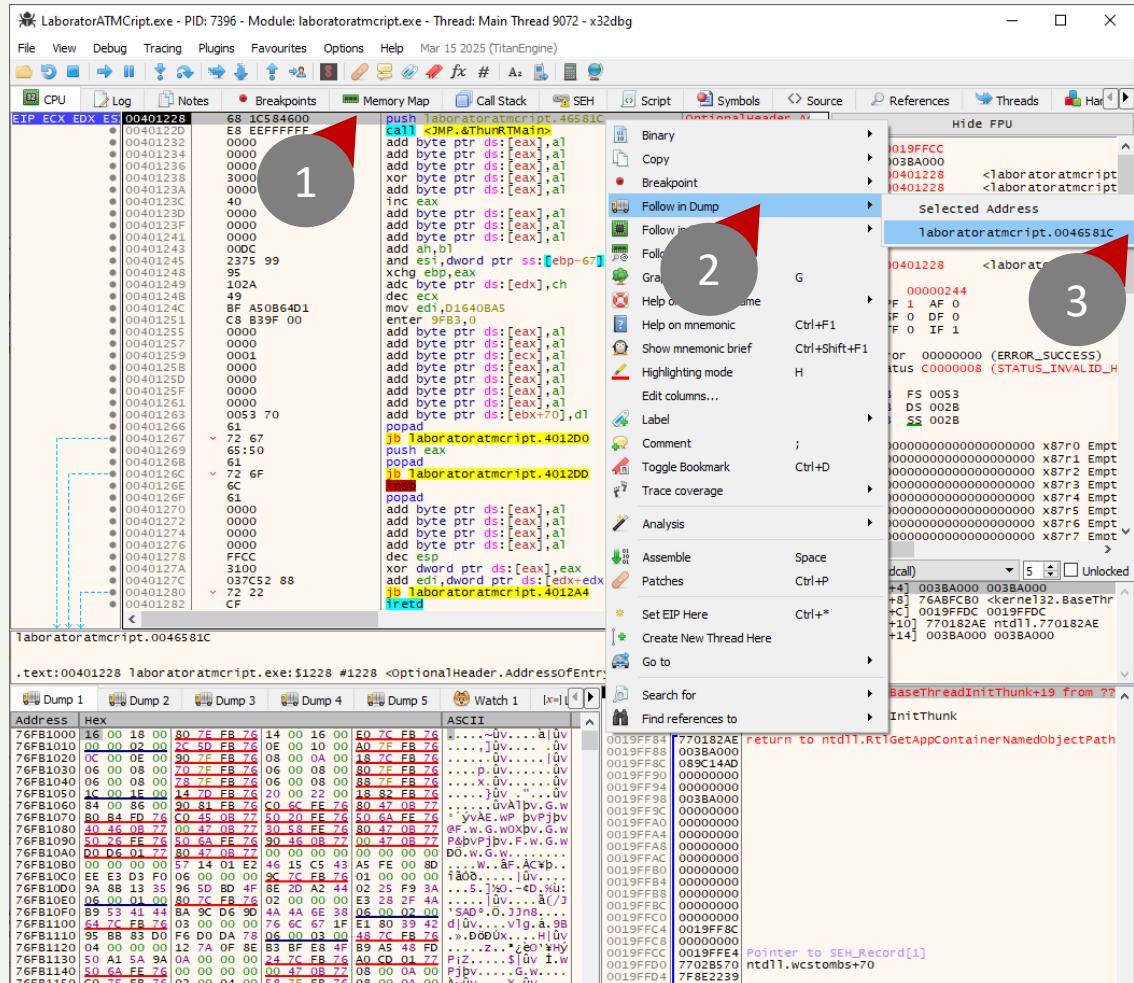
- Fereastra de patching confirmă aplicarea celor 39 de modificări în fișierul executabil `laboratoratmcript.exe`.
- Aplicația rulează cu stringul personalizat injectat: mesajul de eroare a fost înlocuit cu „BRAVOS! Incorrectă! HAI incearcă”, afișat în MessageBox.
- Alternativ, într-o versiune diferită a binarului, apare „Parola Incorrectă ! Mai incearca”, semn că patch-ul corespunde unei alte ediții/ramuri de test.



VERSIUNEA (II) CAUTAM SPECIFIC CE NE INTERESEAZA ...

Metoda ob^e_u-zului !

Atunci când fișierele sunt de dimensiuni mari, iar timpul de analiză este limitat, abordările exploratorii devin ineficiente (nu mai avem loc de explorări romantice prin dump). În aceste cazuri, se impune o strategie punctuală, directă, axată pe obiectivul imediat: identificarea rapidă a unui element cunoscut (de exemplu, un sir de caractere sau o instrucțiune specifică). Această abordare este denumită informal „metoda obezului” – o metaforă pentru situațiile în care nu este fezabil să se parcurgă întregul spațiu de memorie, ci este necesar să se ajungă imediat la esență. Se renunță astfel la parcurgerea vizuală, în favoarea unei căutări precise și automatizate. Este o metodă eficientă în contexte urgente, unde analiza trebuie să fie concentrată și rezultatele rapide.



1. Selectăm manual un octet din zona de cod executabil (.text).
2. Dăm click dreapta pe acel octet.
3. Alegem opțiunea Follow in Dump → Selected Address pentru a vizualiza zona din memorie.
4. Observăm dumpul asociat, unde dorim să căutăm cuvântul "corecta".
5. Facem click dreapta în dump.
6. Selectăm Find pattern pentru a căuta un string cunoscut.

1. Observăm în aplicație mesajul afișat într-un MessageBox: „Parola Incorecta ! Mai incerca” – ne interesează să găsim acest text în fișierul executabil.

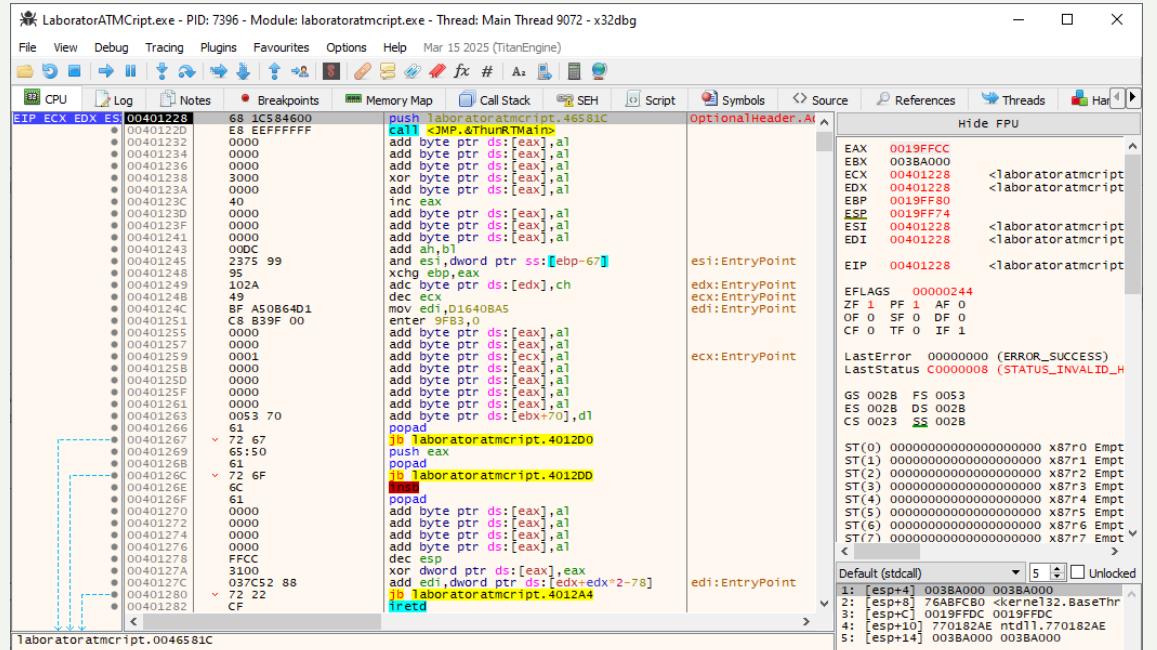
2. În fereastra „Find Pattern”, tastăm un cuvânt reprezentativ, de exemplu „Parola”, și bifăm opțiunea Start from selection (dacă e cazul).

3. Apăsăm OK pentru a începe căutarea în memorie.x32dbg afișează rezultatul căutării: adresa unde apare stringul este evidențiată.

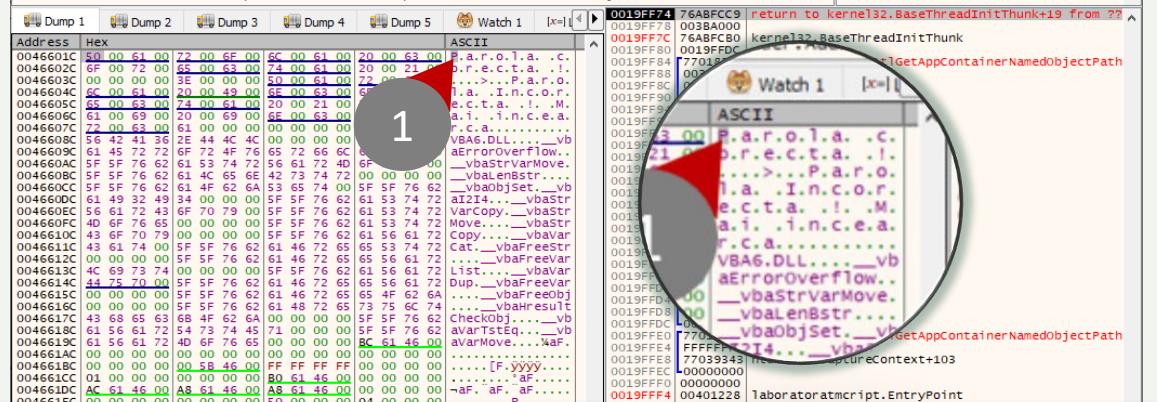
4. Dăm click dreapta pe adresa respectivă și selectăm Follow in Dump → Follow in Dump pentru a inspecta și edita mai ușor zona cu stringul.

5. Nu este necesar să navigăm vizual pentru a căuta un mesaj – este mult mai eficient să folosim funcția de căutare directă (pattern search) cu un fragment relevant de text.

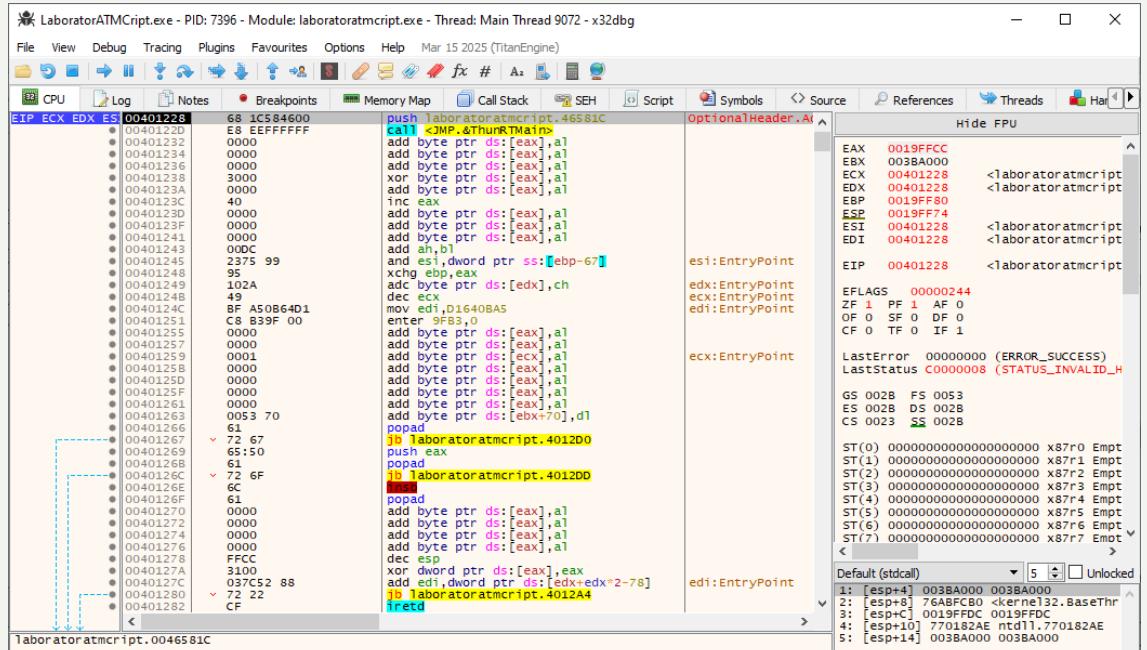
Dar partea pozitivă, în cazul căutării vizuale, este elementul neprevăzut — bucătăca de informație care duce la un declic mental.



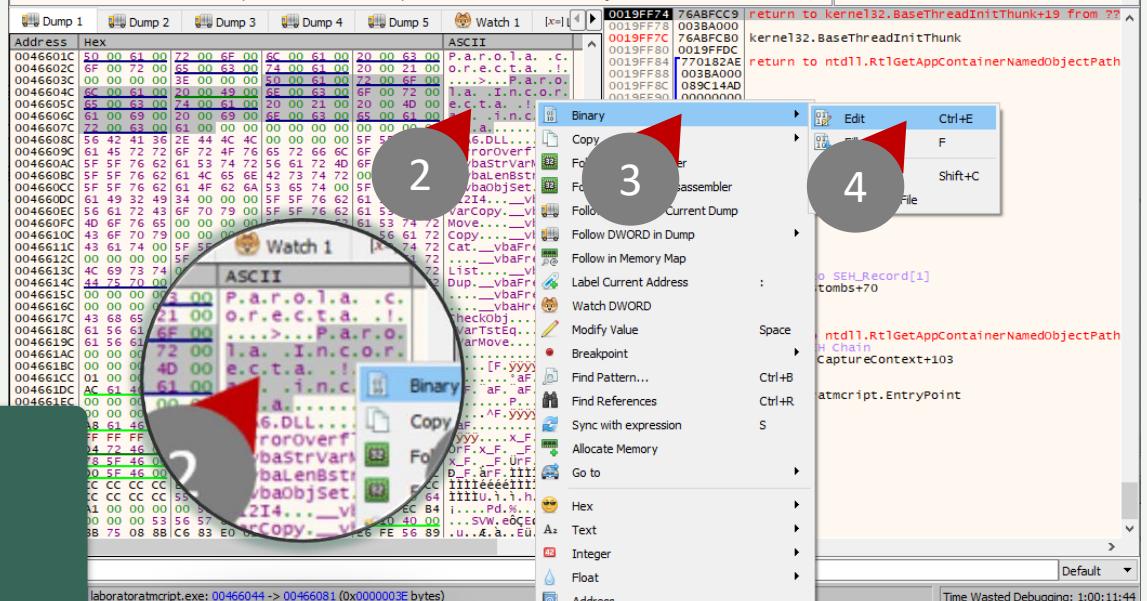
.text:00401228 laboratoratmcript.exe:\$1228 #1228 <OptionalHeader.AddressOfEntryPoint>

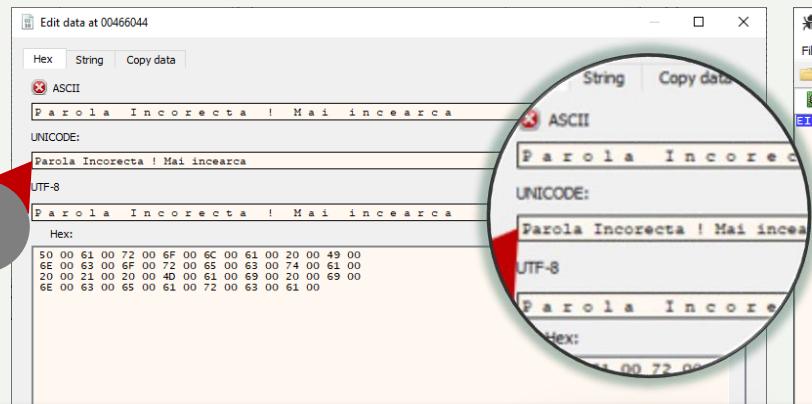


1. Ne-am poziționat în zona corectă din Dump (.text) și am găsit stringul dorit.
2. Facem click dreapta pe string → apare meniul contextual.
3. Navigăm la opțiunea Binary.
4. Selectăm Edit (Ctrl+E) pentru a deschide fereastra de editare.

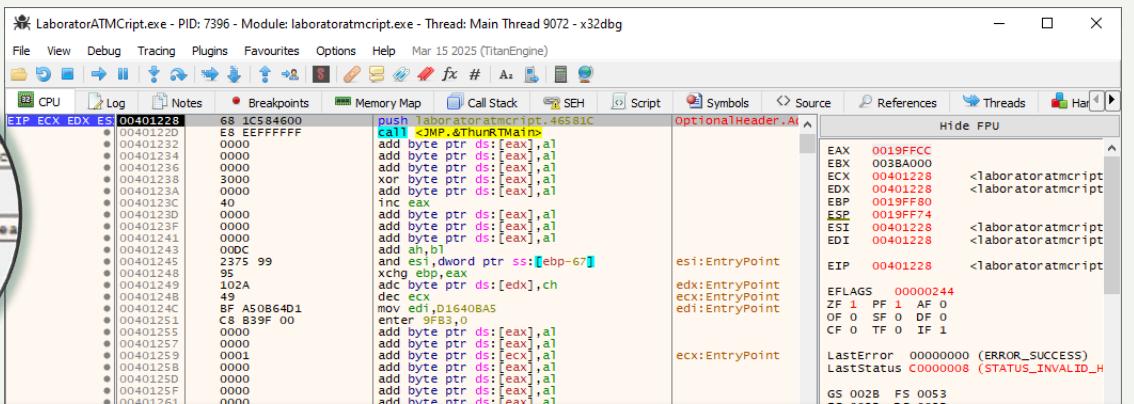


.text:00401228 laboratoratmcript.exe:\$1228 #1228 <OptionalHeader.AddressOfEntryPoint>





1



laboratoratmcrscript.exe

Patches

Modules

Information

20/20 patch(es) applied!

OK

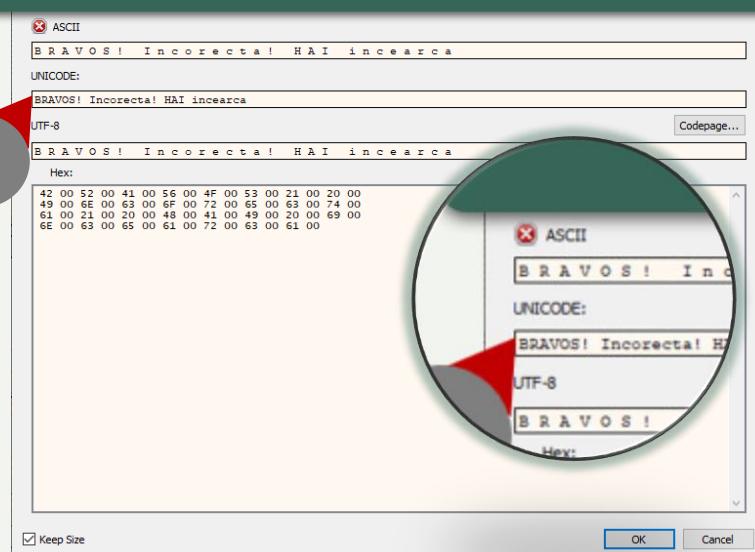
0100466044:50->42
0100466046:61->52
0100466048:72->41
010046604A:6F->56
010046604C:6C->4F
010046604E:61->53
0100466050:20->21
002:49->20
054:6E->49
056:63->6E
058:62->63
05A:72->6F
05C:65->72
05E:63->65
060:74->63
0100466062:61->74
0100466064:20->61
010046606A:4D->4B
010046606C:61->41
010046606E:69->49

Select All Deselect All Restore Selected

Import Export Patch File

a

1. Se deschide fereastra de editare a stringului (1), unde avem mesajul original „Parola Incorrectă ! Mai încearcă”.
 2. Se editează textul, spre exemplu în „BRAVOS! Incorrectă! HAI încearcă” fără a depăși lungimea (2).
 3. Modificarea este imediat vizibilă în dump (3).
 4. Se generează patch-ul cu noile date și se aplică toate modificările (a).
 5. La rulare, aplicația arată noul mesaj modificat în MessageBox (b), înlocuind textul original (c).

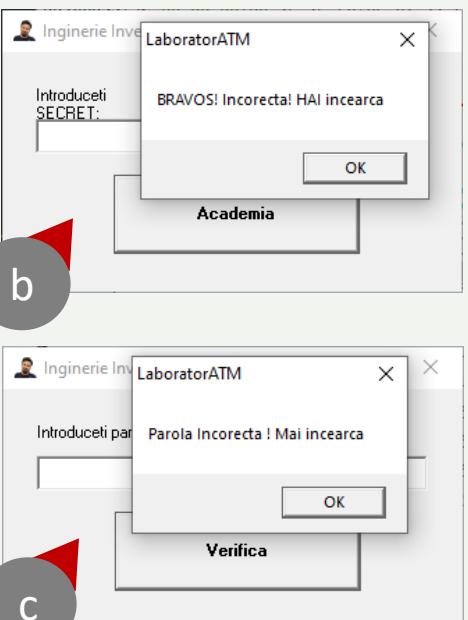


2



3

3



b

C

9.5

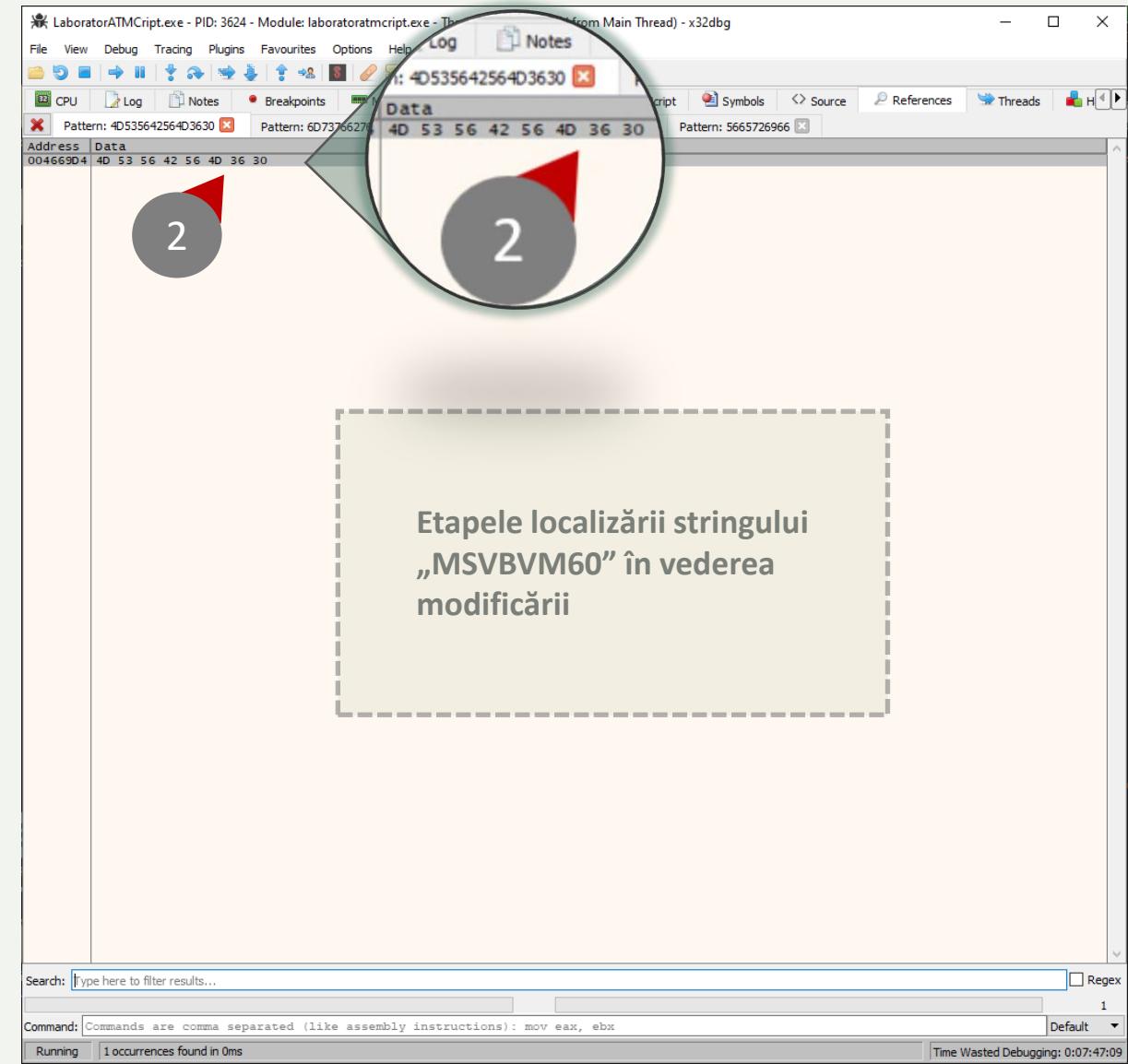
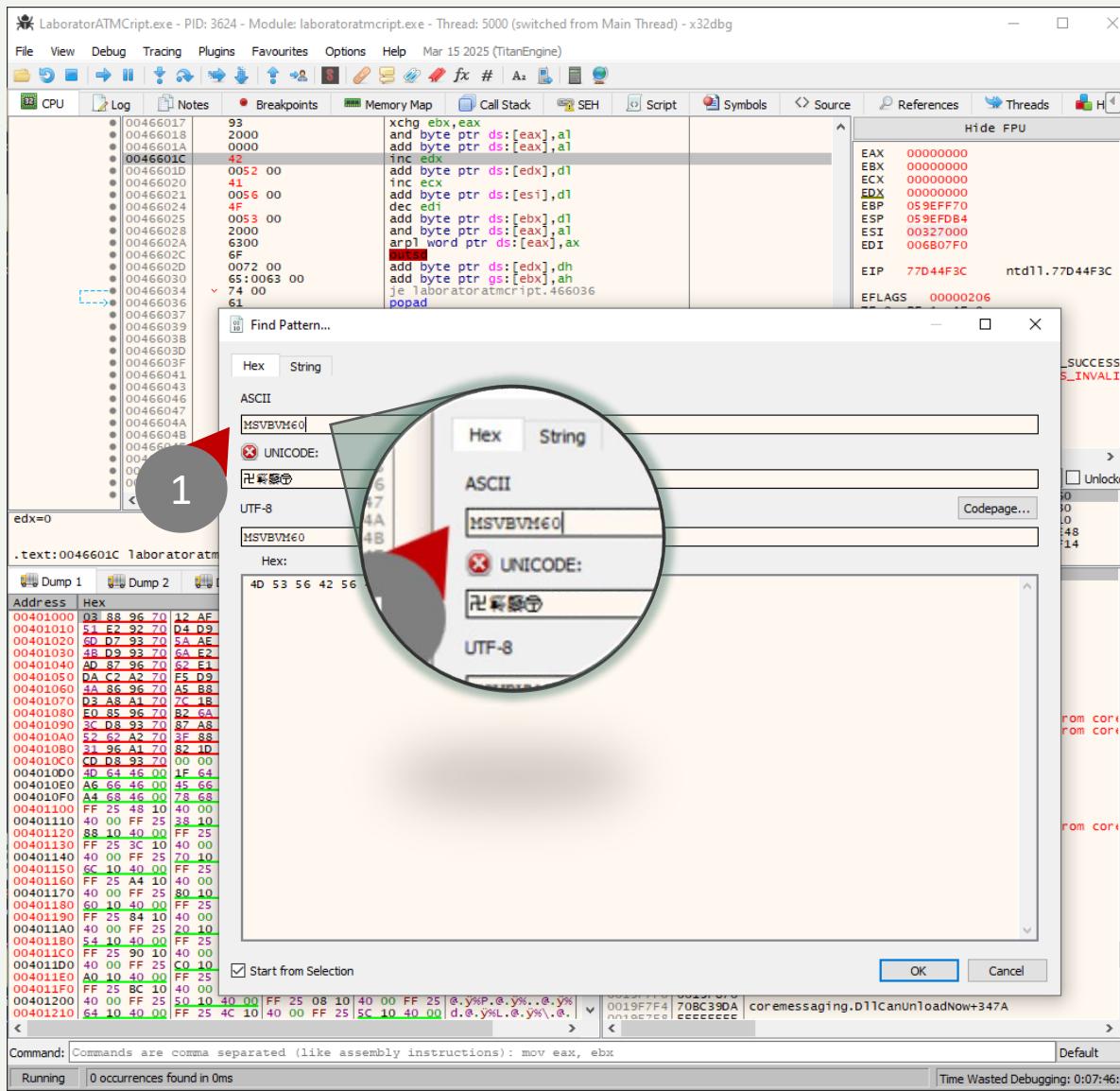
SCHIMBAREA MASINII VIRTUALE

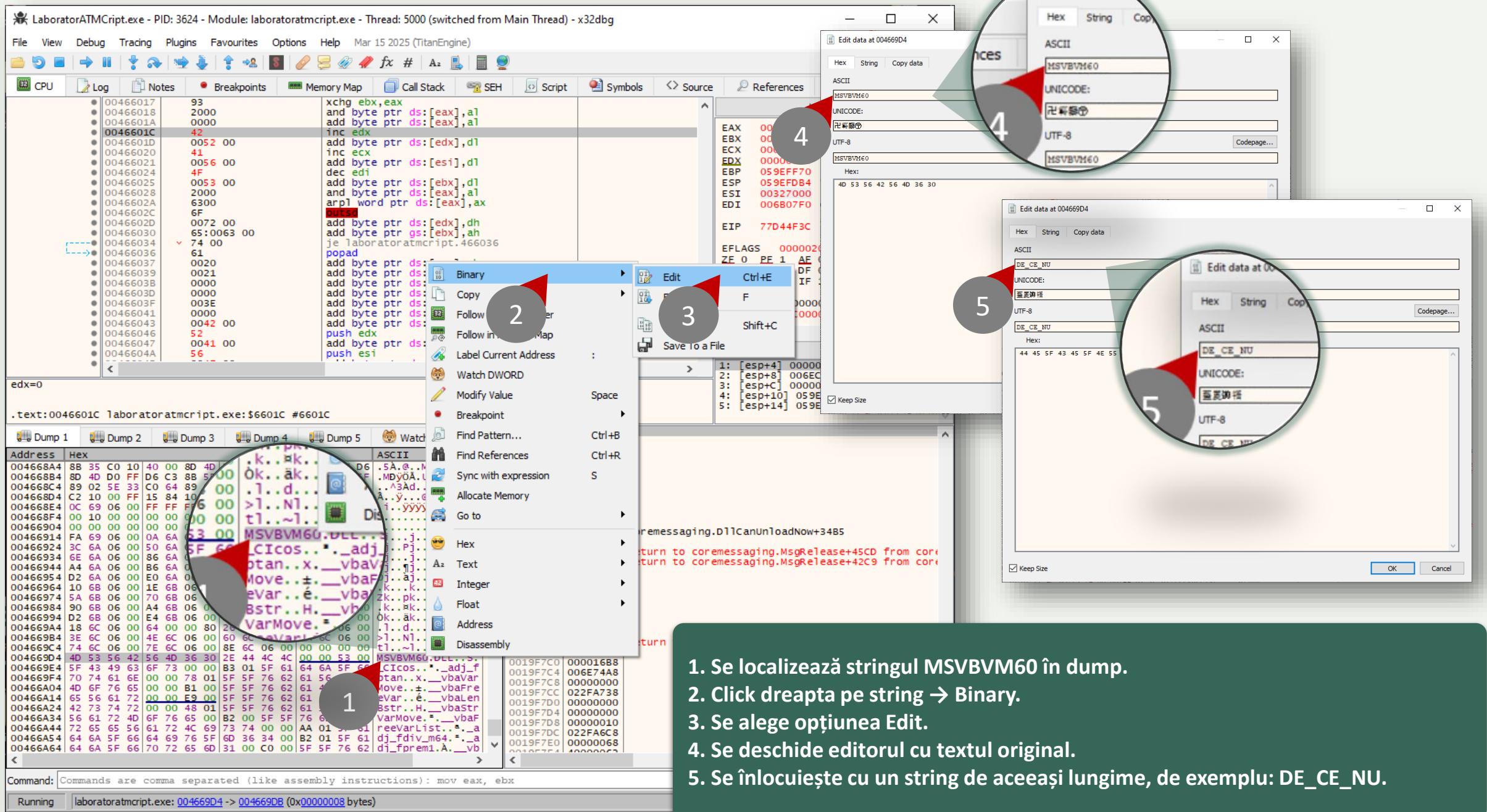


CONTEXT INTRODUCTIV

- În anumite scenarii de analiză sau manipulare a executabilelor, poate deveni necesară modificarea numelui unui DLL dinamic legat static în binar, cum este cazul bine-cunoscutului msvbvm60.dll. Această tehnică poate avea diverse scopuri, printre care: Redirecționarea încărcării către o versiune personalizată a DLL-ului; Evitarea detectării de către un antivirus; Crearea unui mediu de test controlat; Pregătirea pentru injectarea unei biblioteci proprii (ex: fakevbvm60.dll). Modificarea nu afectează direct comportamentul codului executabil, ci modul în care acesta își caută și încarcă bibliotecile externe.

1. Se deschide fereastra Find Pattern din meniul contextual, iar în câmpul ASCII se introduce sirul MSVBVM60 (vezi 1).
2. Acesta este numele bibliotecii dinamice (DLL) standard folosit de aplicațiile Visual Basic 6. După confirmarea căutării, sirul este găsit în memoria executabilului, afișat în zona de dump (2). Reprezentarea hexazecimală este: 4D 53 56 42 56 4D 36 30.





LaboratorATMcript.exe - PID: 3624 - Module: laboratoratmcript.exe - Thread: 5000 (switched from Main Thread) - x32dbg

File View Debug Tracing Plugins Favourites Options Help Mar 15 2025 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols References Threads Help

00466017 93 xchng ebx, eax
 00466018 2000 and byte ptr ds:[eax],al
 0046601A 0000 add byte ptr ds:[eax],al
0046601C 42 inc edx
 0046601D 0052 00 add byte ptr ds:[edx],dl
 00466020 41 inc ecx
 00466021 0056 00 add byte ptr ds:[esi],dl
 00466024 4F dec edi
 00466025 0053 00 add byte ptr ds:[ebx],dl

EAX: 00000000
 EBX: 00000000
 ECX: 00000000
EDX: 00000000
 EBP: 059EFF70
 ESP: 059EFDB4

1. Se localizează stringul în .text („MSVBVM60” în dump) → Find Pattern + vizualizare directă.
 2. Se modifică în alt nume de aceeași lungime (ex: DE_CE_NU) → se aplică patch-ul.

edx=0

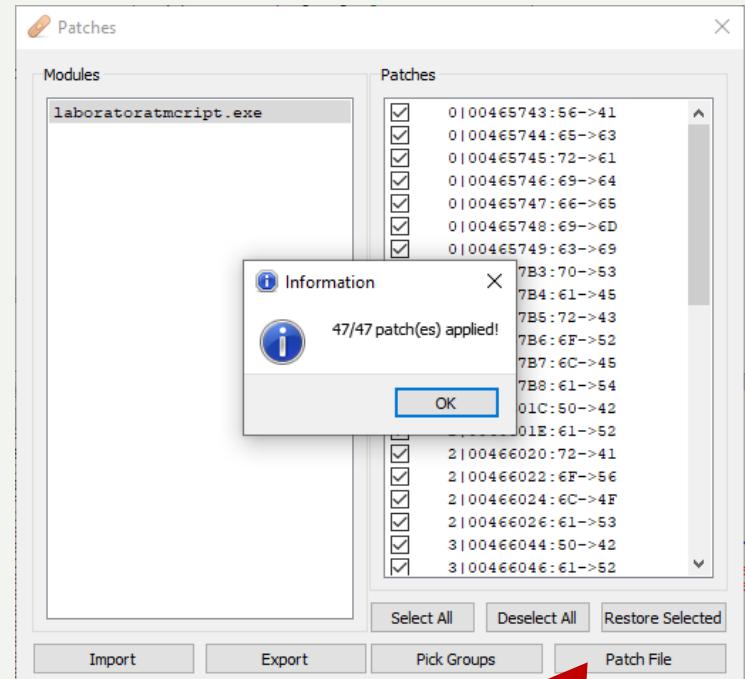
.text:0046601C Laboratoratmcript.exe:\$6601C #6601C

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1

Address	Hex	ASCII
004668A4	8B 35 C0 10 40 00 8D 4D E0 FF D6 8D 4D DC FF D6	.SA., MÄÖ.MÜÖ
004668B4	BD 4D D0 FF D6 C3 8B 55 14 8B 45 E4 8B 4D EC 5F	.MDYÖÄ.U..EÄ.M
004668C4	89 02 5E 33 C0 64 89 00 00 00 00 5B 8B E5 5D	.^Äd..
004668D4	C2 10 00 FF 15 84 10 40 00 90 90 90 9E 9E 9E	Ä..y..@.6 00
004668E4	0C 69 06 00 FF FF FF FF FF FF D4 69 06 00	.i..ÿÿÿÿÿÿ 06 00
004668F4	00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00	>T. N1. 1
00466904	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	t1. ~1. 1
00466914	FA 69 06 00 0A 6A 06 00 1A 6A 06 00 2A 6A 06 00	úi..j.
00466924	3C 6A 06 00 50 6A 06 00 04 02 00 80 60 6A 06 00	53 00
00466934	GE 6A 06 00 86 6A 06 00 96 6A 06 00 53 02 00 80	DE_CE_NU.DLL
00466944	A4 6A 06 00 B6 6A 06 00 C8 6A 06 00 77 02 00 80	CICos. ad
00466954	D2 6A 06 00 EO 6A 06 00 F4 6A 06 00 04 6B 06 00	return to coremessaging.MsgRelease+45CD from core
00466964	10 6B 06 00 1E 6B 06 00 34 6B 06 00 3E 6B 06 00	Öj..aj..
00466974	5A 6B 06 00 70 6B 06 00 7E 6B 06 00 60 02 00 80	k..k..
00466984	90 6B 06 00 A4 6B 06 00 B2 6B 06 00 B6 6B 06 00	zK..pk..~
00466994	D2 6B 06 00 E4 6B 06 00 F6 6B 06 00 06 06 06 00	Bstr..H.._vba
004669A4	18 6C 06 00 64 00 00 80 26 6C 06 00 34 6C 06 00	Move..±..vba
004669B4	3E 6C 06 00 4E 6C 06 00 60 6C 06 00 6A 6C 06 00	VarMove..±..vba
004669C4	74 6C 06 00 7E 6C 06 00 8E 6C 06 00 00 00 00 00	reeVar!0B7BFDD
004669D4	44 45 5F 43 2E 44 4C 4C 00 00 53 00	>T. N1. 1
004669E4	5F 43 49 63 6F 73 00 00 B3 01 5F 61 64 6A 5F F5	DE_CE_NU.DLL.S.
004669F4	70 74 61 6E 00 00 78 01 5F 5F 76 62 51 56 6A 5F	CICos. adj..f
00466A04	4D 6F 76 65 00 00 B1 00 5F 5F 76 62 61 44 6A 5F	btan..x..vbavar
00466A14	65 56 61 72 00 00 E9 00 5F 5F 76 62 61 44 6A 5F	Move..±..vbaRe
00466A24	42 73 74 72 00 00 48 01 5F 5F 76 62 61 44 6A 5F	eVar..é..vbalen
00466A34	56 61 72 4D 6F 76 65 00 B2 00 5F 5F 76 62 61 44	Bstr..H..vbaStr
00466A44	72 65 65 56 61 72 4C 69 73 74 00 00 AA 01 5F 61	VarMove..±..vbaF
00466A54	64 6A 5F 66 64 69 76 5F GD 36 34 00 B2 01 5F 61 dj_fdiv_m64..a	reeVarList..±..a
00466A64	64 6A 5F 66 70 72 65 6D 31 00 CO 00 5F 5F 76 62 di_fpremain.A._vb	0019F7DC 022FA6C8

Command: Commands are comma separated (like assembly instructions): mov eax, ebx Default

Running laboratoratmcript.exe: 004669D4 -> 004669DB (0x00000008 bytes) Time Wasted Debugging: 0:07:50:27



2



Fișierul executabil modificat (test3.exe) va căuta acum biblioteca „DE_CE_NU.dll” în loc de „msvbvm60.dll”.

- În primul executabil (test2.exe), vedem că s-a încărcat vechiul msvbvm60.dll.
- În al doilea executabil (test3.exe), vedem că s-a încărcat noul de_ce_nu.dll.

test2.exe - PID: 3824 - Module: test2.exe - Thread: Main Thread 7308 - x32dbg

File View Debug Tracing Plugins Favourites Options Help Mar 15 2025 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads

Address	Size	Party	Info	Content	Type	Protection	Initial
001D0000	00001000	User	Reserved (001E0000)		MAP	-R---	-R--
001E0000	00004000	User	Reserved (001E0000)		MAP	-R---	-R--
001F0000	00004000	User	Reserved (001E0000)		MAP	-R---	-R--
00200000	00001000	User	PEB, TEB (7308), WOW64 TEB (7308)		PRV	-RW--	-RW--
00288000	0000E000	User	Reserved (00200000)		PRV	-RW--	-RW--
00299000	00016700	User	test2.exe	".text"	IMG	-R---	ERWC
00401000	00066000	User		".data"	IMG	ER---	ERWC
00467000	00001000	User		".rsrc"	IMG	-RWC-	ERWC
00488000	00006500	User			IMG	-R---	ERWC
00490000	00009000	User	\Device\HarddiskVolume2\Windows		MAP	-R-	-R--
005A0000	00035000	User	Reserved		PRV	-RW-	-RW--
005D5000	00008000	User	Reserved		PRV	-RW-G	-RW--
005E0000	00035000	User	Reserved		PRV	-RW-	-RW--
00615000	00008000	User	Heap (ID 0)		PRV	-RW-G	-RW--
00620000	00020000	User	Reserved (00620000)		PRV	-RW-	-RW--
00640000	0000E000	User	Reserved		PRV	-RW-	-RW--
00720000	000FD000	User	Stack (13740)		PRV	-RW-	-RW--
0081D000	00003000	User	Reserved		PRV	-RW-G	-RW--
00820000	00003000	User	Stack (7032)		PRV	-RW-G	-RW--
00920000	00035000	User	Reserved		PRV	-RW-	-RW--
00955000	00008000	User	Reserved		PRV	-RW-G	-RW--
00960000	000FD000	User	Reserved		PRV	-RW-	-RW--
00A5D000	00003000	User	Stack (9908)		PRV	-RW-G	-RW--
00A60000	00013000	User	Reserved (00A60000)		PRV	-RW-	-RW--
00A73000	001ED000	User	Reserved (00A60000)		MAP	-R---	-R--
00C60000	00181000	User	Reserved (00A60000)		MAP	-R---	-R--
00DF0100	0000F100	User	Reserved (00DF0000)		MAP	-R---	-R--
00EE1000	00131000	User	Heap (ID 3)		PRV	-RW-	-RW--
02260000	0000F000	User	Reserved (02260000)		PRV	-RW-	-RW--
02280000	00002000	User	Heap (ID 2)		PRV	-RW-	-RW--
02282000	0000E000	User	Reserved (02280000)		PRV	-RW-	-RW--
02360000	00003000	User	Heap (ID 1)		PRV	-RW-	-RW--
02363000	0000D000	User	Reserved (02360000)		PRV	-RW-	-RW--
02370000	00010000	User	Reserved (02370000)		PRV	-RW-	-RW--
02380000	003F0000	User	\Device\HarddiskVolume2\Windows		MAP	-R-	-R--
02770000	00033800	User	Reserved (02370000)		PRV	-RW-	-RW--
6FBE0000	00001000	System	msvbvm60.dll	".text"	IMG	-R---	ERWC
6FBE1000	00105000	System	"ENGINE"	".data"	IMG	ER---	ERWC
6FC50000	00009000	System	"ENGINE"	".rsrc"	IMG	-R---	ERWC
6FCFC000	00003100	System	"ENGINE"	".reloc"	IMG	-R-	ERWC
6FD20000	00001000	System			IMG	-R-	ERWC
74E10000	00001000	System	kernelbase.dll	".text"	IMG	-R---	ERWC
74E11000	001F90	System		".data"	IMG	ER---	ERWC
7500A000	00004000	System	kernelbase.dll	".idata"	IMG	-R-	ERWC
7500E000	00006000	System	kernelbase.dll	".rsrc"	IMG	-R-	ERWC
75014000	00001000	System	kernelbase.dll	".didat"	IMG	-R-	ERWC
75016000	00032000	System	kernelbase.dll	".reloc"	IMG	-R-	ERWC
75030000	00001000	System	kernelbase.dll	".text"	IMG	-R---	ERWC
7504A000	00006000	System	kernelbase.dll	".data"	IMG	-R-	ERWC
7505A7000	0000E000	System	kernelbase.dll	".rdata"	IMG	-R-	ERWC
750B0000	00001000	System	kernelbase.dll	".idata"	IMG	-R-	ERWC
750B0000	00001000	System	kernelbase.dll	".rsrc"	IMG	-R-	ERWC
750B0000	00001000	System	kernelbase.dll	".reloc"	IMG	-R-	ERWC
750C0000	00001000	System	rpctr4.dll	".text"	IMG	ER---	ERWC
750C1000	0000A000	System	rpctr4.dll	".data"	IMG	-R-	ERWC
75168000	00003000	System	rpctr4.dll	".idata"	IMG	-R-	ERWC
7516C000	00003000	System	rpctr4.dll	".rsrc"	IMG	-R-	ERWC
7516F000	00001000	System	rpctr4.dll	".didat"	IMG	-R-	ERWC
75170000	00005000	System	rpctr4.dll	".reloc"	IMG	-R-	ERWC
75175000	00007000	System	rpctr4.dll	".text"	IMG	-R---	ERWC
75190000	00001000	System	msvcp_win.dll	".data"	IMG	ER---	ERWC
75191000	0006E000	System	msvcp_win.dll	".idata"	IMG	-R-	ERWC
751FF000	00003000	System	msvcp_win.dll	".rsrc"	IMG	-R-	ERWC
75202000	00002000	System	msvcp_win.dll	".didat"	IMG	-R-	ERWC
75204000	00001000	System	msvcp_win.dll	".reloc"	IMG	-R-	ERWC
75205000	00001000	System	msvcp_win.dll	".text"	IMG	-R---	ERWC

1

test3.exe - PID: 11196 - Module: test3.exe - Thread: Main Thread 10628 - x32dbg

File View Debug Tracing Plugins Favourites Options Help Mar 15 2025 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads

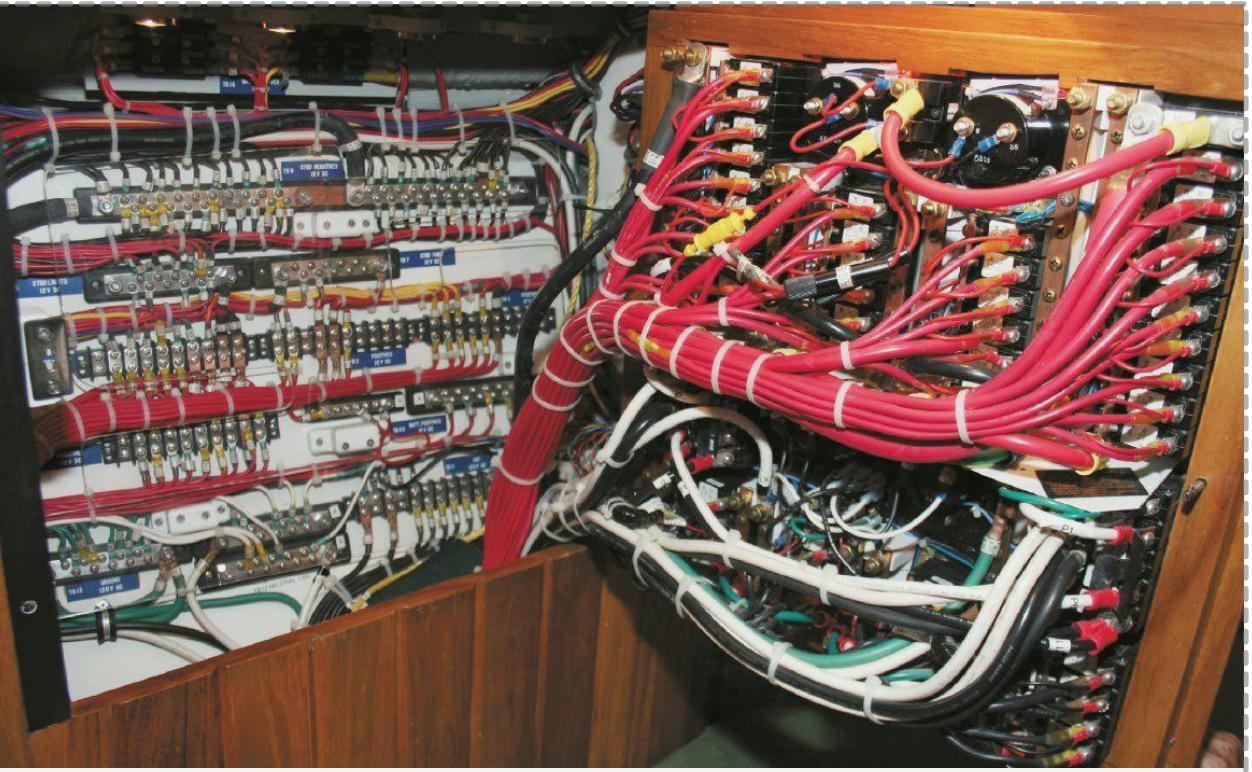
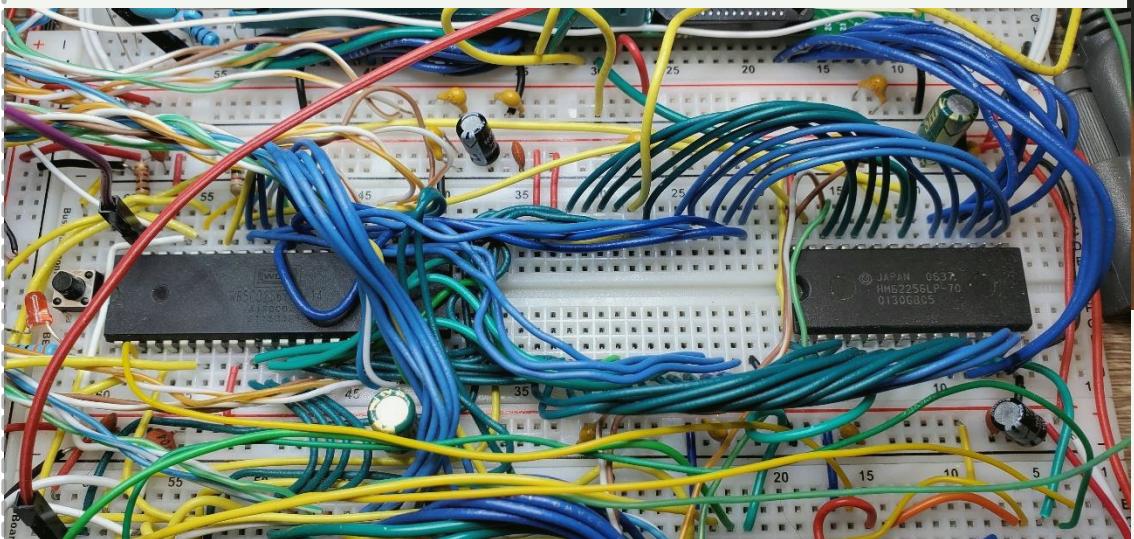
Address	Size	Party	Info	Content	Type	Protection	Initial
001D7000	00009000	User	Reserved (001E0000)		PRV	-RW-	-RW--
001E0000	00004000	User	Reserved (001E0000)		MAP	-R---	-R--
001E4000	00004000	User	Reserved (001E0000)		MAP	-R---	-R--
00200000	00018000	User	Reserved (001E0000)		PRV	-RW--	-RW--
00380000	00014000	User	PEB, TEB (10628), WOW64 TEB (10628)		PRV	-RW--	-RW--
00394000	00005C00	User	Reserved (00200000)		PRV	-RW--	-RW--
00400000	00001000	User	test3.exe	".text"	IMG	-R---	ERWC
00401000	00001000	User	test3.exe	".data"	IMG	ER---	ERWC
00467000	00001000	User	test3.exe	".rsrc"	IMG	-RWC-	ERWC
00490000	00009000	User	\Device\HarddiskVolume2\Windows		MAP	-R-	-R--
005A0000	00035000	User	Reserved		PRV	-RW-	-RW--
005D5000	00008000	User	Reserved		PRV	-RW-G	-RW--
005E0000	00035000	User	Reserved		PRV	-RW-	-RW--
00615000	00008000	User	Reserved		PRV	-RW-G	-RW--
00620000	00020000	User	Reserved		PRV	-RW-	-RW--
00655000	0000B000	User	Reserved		PRV	-RW-G	-RW--
00660000	00004000	User	Reserved		PRV	ER---	-RW--
00670000	00003C000	User	Reserved (00660000)		PRV	-RW-	-RW--
006AC000	00004000	User	Reserved (00670000)		PRV	-RW--	-RW--
00770000	0000FC000	User	Reserved		PRV	-RW-	-RW--
00879000	00004000	User	Stack (5792)		PRV	-RW-G	-RW--
00970000	0000FC000	User	Reserved (10736)		PRV	-RW-	-RW--
00A4C000	00004000	User	Reserved (6248)		PRV	-RW-G	-RW--
00A70000	0013D000	User	Reserved (00A70000)		PRV	-RW-	-RW--
00A83000	001ED000	User	Reserved (00A70000)		PRV	-RW-	-RW--
00E00000	00005000	User	Reserved		PRV	-R---	-R--
00ED5000	00132C000	User	Reserved (00E00000)		PRV	-RW-	-RW--
02210000	00010000	User	\Device\HarddiskVolume2\Users\Pal		MAP	-R-	-R--
02220000	00068000	User	Reserved (02220000)		MAP	-R-	-R--
022A0000	00004000	User	Reserved		MAP	-R-	-R--
022B8000	00005000	User	\Device\HarddiskVolume2\Windows		MAP	-R-	-R--
022C0000	00002000	User	Heap (ID 2)		PRV	-RW-	-RW--
022C2000	0000E000	User	Reserved (022C0000)		PRV	-RW-	-RW--
022E0000	00005000	User	Heap (ID 1)		PRV	-RW-	-RW--
022F0000	00010000	User	Reserved (022F0000)		PRV	-RW-	-RW--
02300000	0003F000	User	Reserved (022F0000)		MAP	-R-	-R--
026F0000	00338000	User	\Device\HarddiskVolume2\Windows		MAP	-R-	-R--
02A30000	000101000	User	Reserved (02A30000)		PRV	-RW-	-RW--
02A40000	00004000	User	Reserved		PRV	-RW-	-RW--
02A50000	0000B000	User	Reserved (02D50000)		PRV	-RW-G	-RW--
02B20000	00003000	User	Heap (ID 4)		PRV	-RW-	-RW--
02B23000	000035000	User	Reserved (02B20000)		PRV	-RW-	-RW--
02B25000	00003000	User	Reserved		PRV	-RW-	-RW--
02B2D000	0000D000	User	Heap (ID 3)		PRV	-RW-G	-RW--
02BD0000	00003000	User	Reserved (02BD0000)		PRV	-RW-	-RW--
02BE4000	00004000	User	Reserved (02BE0000)		PRV	-RW-	-RW--
02D51000	0000F000	User	Reserved (02D50000)		PRV	-RW-	-RW--
02D60000	00006000	User	Reserved (02D60000)		PRV	-RW-	-RW--
02D66000	003FA000	User	Reserved (02D60000)		PRV	-RW-	-RW--
03160000	000E2000	User	Reserved (03160000)		PRV	-RW-	-RW--
03B60000	00001000	User	Reserved (03B60000)		PRV	-RW-	-RW--
03B61000	000FF000	User	Reserved (03B61000)		PRV	-RW-	-RW--
03C30000	001260000	User	\Device\HarddiskVolume2		MAP	-R-	-R--
03CD0000	00001000	User	Reserved (03CD0000)		PRV	-RW-	-RW--
04D40000	0000F000	User	Reserved (04D40000)		PRV	-RW-	-RW--
046CD000	00003000	User	Stack (6820)		PRV	-RW-	-RW--
046DD000	0000FD000	User	Reserved		PRV	-RW-	-RW--
0466CD00	00003000	User	Stack (10048)		PRV	-RW-	-RW--
70180000	000074000	System	sxs.dll	".text"	IMG	-R---	ERWC
701F8000	00001000	System	sxs.dll	".data"	IMG	-R-	ERWC
701FE0000	00001000	System	sxs.dll	".idata"	IMG	-R-	ERWC
701F0000	00001000	System	sxs.dll	".rsrc"	IMG	-R-	ERWC
70088000	00001000	System	sxs.dll	".reloc"	IMG	-R-	ERWC
70881000	00009000	System	sxs.dll	".text"	IMG	ER---	ERWC
70911000	00001000	System	sxs.dll	".data"	IMG	-R-	ERWC
70912000	00001000	System	sxs.dll	".idata"	IMG	-R-	ERWC
70921000	00001000	System	sxs.dll	".rsrc"	IMG	-R-	ERWC
70426000	0000E000	User	de_ce_nu.dll	".text"	IMG	ER---	ERWC
70434000	00008000	User	de_ce_nu.dll	".data"	IMG	ER---	ERWC
7043C000	000031000	User	de_ce_nu.dll	".idata"	IMG	-R-	ERWC
70480000	00001000	System	ntypes.dll	".text"	IMG	ER---	ERWC
70848000	00001000	System	ntypes.dll	".data"	IMG	ER---	ERWC
70840000	00002000	System	ntypes.dll	".idata"	IMG	-R-	ERWC
70842000	00002000	System	ntypes.dll	".rsrc"	IMG	-R-	ERWC
70845000	00001000	System	ntypes.dll	".reloc"	IMG	-R-	ERWC
70846000	000017000	System	coremessaging.dll	".text"	IMG	ER---	ERWC
70860000	00001000	System	coremessaging.dll	".data"	IMG	ER---	ERWC

2

PROGRAMARE SOFTWARE DIRECT IN DEBUGGER? SIGUR CA DA !



- În acest exemplu demonstrativ, relocăm o bucată de cod executabil din secțiunea .code (.text) într-o zonă liberă din secțiunea .data, care în mod normal nu este executabilă. Duplicăm codul, schimbăm permisiunile de memorie pentru a permite execuția și redirectionăm fluxul printr-un jmp. Această tehnică evidențiază posibilitatea rulării de cod din zone non-standard, un mecanism utilizat frecvent în malware sau tehnici avansate de ofuscare.



- În acest exemplu demonstrativ, relocăm o bucată de cod executabil din secțiunea .code (.text) într-o zonă liberă din secțiunea .data, care în mod normal nu este executabilă. Duplicăm codul, schimbăm permisiunile de memorie pentru

BIBLIOGRAFIE / RESURSE

- Paul A. Gagniuc. *Antivirus Engines: From Methods to Innovations, Design, and Applications*. Cambridge, MA: Elsevier Syngress, 2024. pp. 1-656.
- Paul A. Gagniuc. *An Introduction to Programming Languages: Simultaneous Learning in Multiple Coding Environments*. Synthesis Lectures on Computer Science. Springer International Publishing, 2023, pp. 1-280.
- Paul A. Gagniuc. *Coding Examples from Simple to Complex - Applications in MATLAB*, Springer, 2024, pp. 1-255.
- Paul A. Gagniuc. *Coding Examples from Simple to Complex - Applications in Python*, Springer, 2024, pp. 1-245.
- Paul A. Gagniuc. *Coding Examples from Simple to Complex - Applications in Javascript*, Springer, 2024, pp. 1-240.
- Paul A. Gagniuc. *Markov chains: from theory to implementation and experimentation*. Hoboken, NJ, John Wiley & Sons, USA, 2017, ISBN: 978-1-119-38755-8.

<https://github.com/gagniuc>

