
HOBBYTE

MIA ΙΣΤΟΡΙΑ ΜΕ ΤΟΝ BILBO DEBUGGINS

Πρόλογος

- Πολλές φορές, η στατική ανάλυση ενός προγράμματος δεν είναι αρκετή για να κατανοήσουμε τη λειτουργία του.
- Σε αυτήν την περίπτωση μπορούμε να χρησιμοποιήσουμε εργαλεία που μας επιτρέπουν να παρακολουθούμε την εκτέλεση του προγράμματος όσο αυτή συμβαίνει, τους **debuggers**.
- Υπάρχουν πολλοί debuggers εκεί έξω, ανάλογα με:
 1. Την αρχιτεκτονική του επεξεργαστή.
 2. Το λειτουργικό σύστημα για το οποίο προορίζεται το πρόγραμμα.
 3. Τη γλώσσα στην οποία είναι γραμμένο.

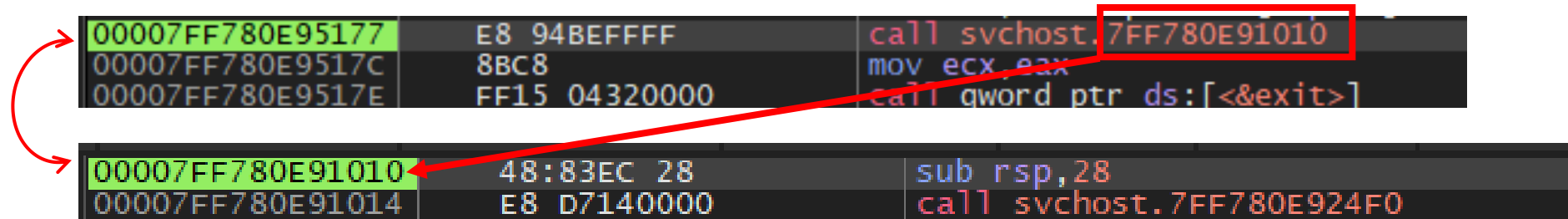
Κεφάλαιο 1 – Τρόποι debugging προγράμματος

- Ανεξάρτητα από τις μεταξύ τους διαφορές, όλοι οι debuggers παρέχουν συνήθως κάποιες κοινές δυνατότητες.
- Μία από αυτές είναι οι τρόποι εκτέλεσης ενός προγράμματος.
- Αυτοί είναι οι:
 - Step Into
 - Step Over
 - Step Out

Κεφάλαιο 1 – Τρόποι debugging προγράμματος

- **Step Into**

Όταν κάνουμε Step Into, το πρόγραμμα διακόπτεται εντολή, εντολή, χωρίς να παραλείπεται καμμία εντολή → Πολύ χρήσιμο όταν βρισκόμαστε σε κομμάτι κώδικα που θέλουμε να καταλάβουμε αναλυτικά τι κάνει.



00007FF780E95177	E8 94BEFFFF	call svchost.7FF780E91010
00007FF780E9517C	8BC8	mov ecx, eax
00007FF780E9517E	FF15 04320000	call qword ptr ds:[<exit>]


00007FF780E91010	48:83EC 28	sub rsp,28
00007FF780E91014	E8 D7140000	call svchost.7FF780E924F0

Step Into: Ο debugger θα εκτελέσει διακόπτοντας βηματικά εντολή εντολή το παραπάνω πρόγραμμα, μπαίνοντας μέσα στη συνάρτηση svchost.7FF780E91010

Κεφάλαιο 1 – Τρόποι debugging προγράμματος

- **Step over**

Μοιάζει με τη step into, μόνο που εδώ ο debugger θα σταματήσει το πρόγραμμα στην επόμενη γραμμή, χωρίς να <<μπει>> έτσι σε κλήσεις συναρτήσεων / system calls



00007FF780E95190	48:83EC 28	sub rsp,28
00007FF780E95194	E8 87000000	call svchost.7FF780E95220
00007FF780E95199	48:83C4 28	add rsp,28

00007FF780E95190	48:83EC 28	sub rsp,28
00007FF780E95194	E8 87000000	call svchost.7FF780E95220
00007FF780E95199	48:83C4 28	add rsp,28

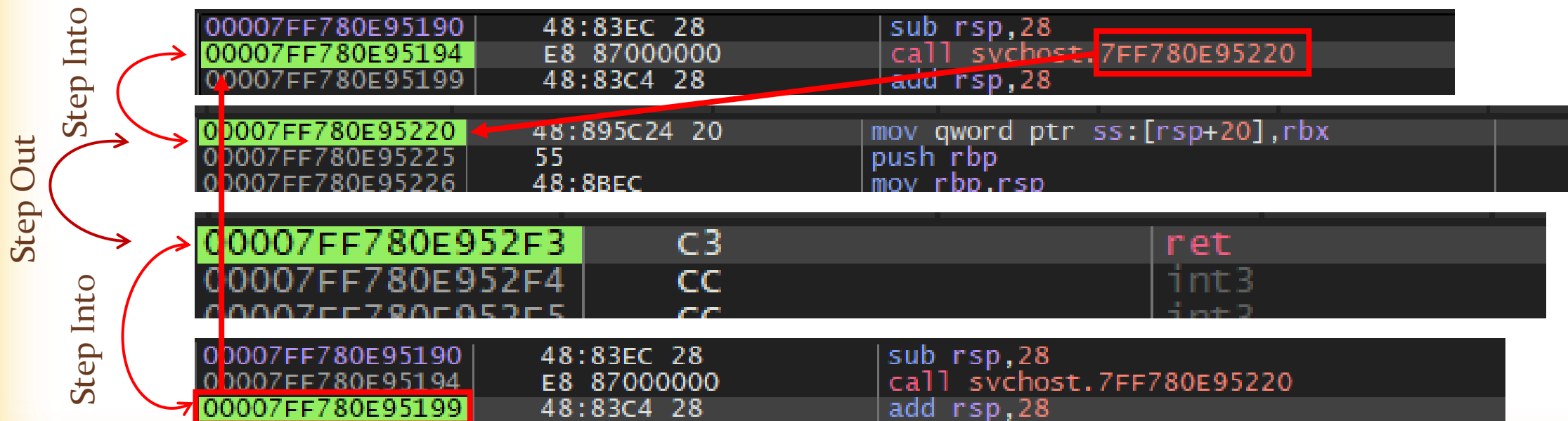
Step Over: Αντί το πρόγραμμα να διακοπεί από τον debugger στη διεύθυνση 7ff780e95220, όπως ορίζει η εντολή call, εδώ η εκτέλεση γίνεται γραμμή γραμμή και το πρόγραμμα σταματάει στην αμέσως επόμενη γραμμή.

Προσοχή: Αυτό δε σημαίνει ότι η συνάρτηση δεν εκτελείται. Εκτελείται και γίνεται monitor από τον debugger, όμως το πρόγραμμα δε θα διακοπεί παρά μόνο στην επόμενη γραμμή (ή αν έχουμε τοποθετήσει κάποιο breakpoint).

Κεφάλαιο 1 – Τρόποι debugging προγράμματος

- **Step Out**

Με το step until, το πρόγραμμα θα εκτελεστεί χωρίς διακοπή μέχρι να τελειώσει η τρέχουσα function (π.χ. ret). Χρήσιμο όταν π.χ. βρισκόμαστε σε μια συνάρτηση που ξέρουμε πώς λειτουργεί (π.χ. read/write) και θέλουμε να επιστρέψουμε στη συνάρτηση που την κάλεσε.



Κεφάλαιο 2 – BreakPoints

- Όπως καταλαβαίνετε, η βήμα βήμα εκτέλεση του κώδικα μπορεί να αποβεί painful ☹
- Πολλές φορές θέλουμε να σταματήσουμε σε σημεία όπου πιστεύουμε ότι συμβαίνει κάτι ενδιαφέρον. Οι περισσότεροι debuggers μας δίνουν τη δυνατότητα αυτή μέσω breakpoints.
- Για να εξηγήσουμε καλύτερα τι είναι τα breakpoints θα ξεκινήσουμε με έναν απλό debugger, τον debugger της python στο vscode!

Κεφάλαιο 2 - BreakPoints

The screenshot displays the Visual Studio Code Python debugger interface. On the left sidebar, the **VARIABLES** panel is expanded, showing the **Locals** section with the variable `res` having the value `36776077867527`. A red box highlights this section, with an arrow pointing to the text **Variable View**. Below it, the **WATCH** and **CALL STACK** panels are visible. The **CALL STACK** panel shows the current frame `calcVal` at line 6 of `example.py`. The main editor shows the code for `calcVal()` and `main()`. A red box highlights the breakpoint icon (a yellow square with a 'D') on line 6, with an arrow pointing to the text **Breakpoint**. The **TERMINAL** panel at the bottom shows the command prompt running the script, with the input `129182` entered.

```
C: > Users > mixlh > Downloads > example.py > calcVal

1  def calcVal():
2      res = 218312132 + 93832923
3      res = res ^ 12318923
4      res = res * 120912
5      res = res - 57
6      return res
7
8  def main():
9      number = int(input("Enter a number: "))
10     if (number == calcVal()):
11         print("You got it!")
12     else:
13         print("Nope")
14
15 if __name__ == "__main__":
16     main()
```

Variable View

Breakpoint

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\mixlh\Downloads> ^C
PS C:\Users\mixlh\Downloads>
PS C:\Users\mixlh\Downloads> c;; cd 'c:\Users\mixlh\Downloads'; & 'c:\Users\mixlh\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\mixlh\.vscode\extensions\ms-python.debugpy-2024.4.0-win32-x64\bundled\libs\debugpy\adapter\..\..\debugpy\launcher' '12045' '--' 'C:\Users\mixlh\Downloads\example.py'
Enter a number: 129182
□

Κεφάλαιο 2 – BreakPoints

- Στο προηγούμενο πρόγραμμα, αν θέλαμε να βρούμε την τιμή `number` που πρέπει να δώσουμε στατικά, θα έπρεπε να κάνουμε `manually` όλες τις πράξεις της συνάρτησης `calcVal`.
- Αντ' αυτού, θέτοντας ένα `breakpoint` κατά το `return`, μπορούμε να γλιτώσουμε τον κόπο εκτέλεσης όλων των πράξεων και να μάθουμε απ'ευθείας την τιμή του `res` (= 36776077867527).
- Το παραπάνω αποτελεί παράδειγμα ενός `Software BreakPoint`, καθώς τίθεται από τον ίδιο τον `debugger` σε εντολές του προγράμματος.

Κεφάλαιο 2 – BreakPoints

- Υπάρχουν όμως και τα hardware breakpoints, τα οποία τίθενται από το υλικό και μας επιτρέπουν να κάνουμε monitor αλλαγές στη μνήμη του προγράμματος. Συγκεκριμένα τα βασικά που μπορούμε να κάνουμε είναι:
 - Break on read
 - Break on write
 - Break on execute

Ή κάποιον συνδυασμό από τα παραπάνω.

Ακολουθεί λίγο πιο advanced παράδειγμα:

Κεφάλαιο 2 – BreakPoints

- Έστω το disassembly:

00007FF75BB210B0	8030 76	xor byte ptr ds:[rax],76	main.cpp:6
00007FF75BB210B3	48:8D40 01	lea rax,qword ptr ds:[rax+1]	
00007FF75BB210B7	FFC1	inc ecx	
00007FF75BB210B9	83F9 0D	cmp ecx,D	D: '\r'
00007FF75BB210BC	72 F2	jb example.7FF75BB210B0	
00007FF75BB210BE	48:8D5424 20	lea rdx,qword ptr ss:[rsp+20]	main.cpp:8

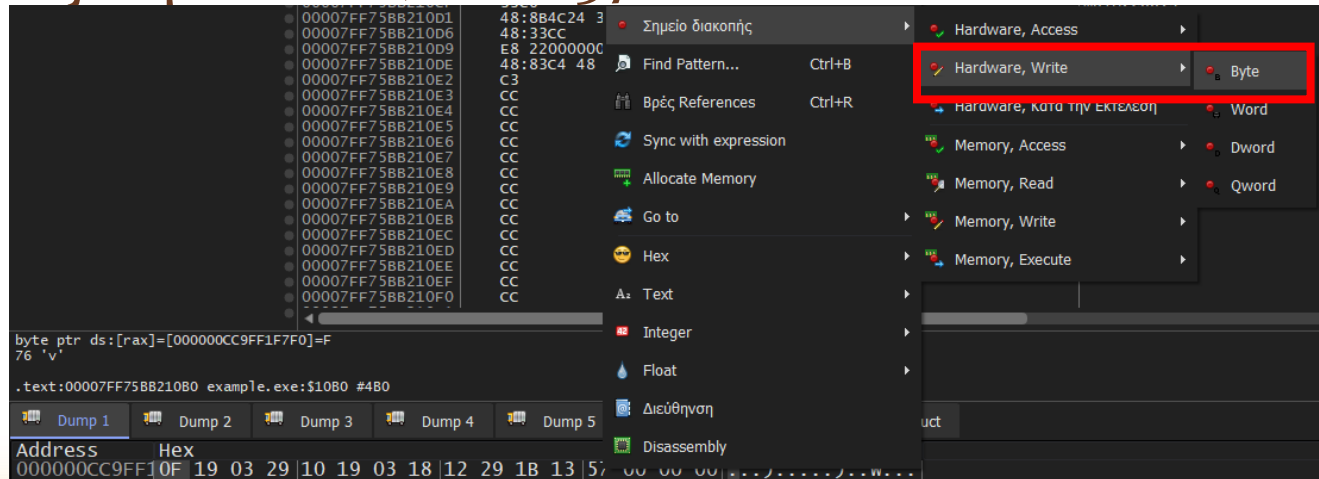
- Παρατηρούμε ότι ο παραπάνω κώδικας είναι ένα loop που θα εκτελεστεί μέχρι ο ecx να γίνει ίσος με 0x0d (ας πούμε ότι αρχικά ήταν μηδέν). Σε αυτό το loop γίνονται διαδοχικά XOR οι θέσεις μνήμης που δείχνει ο rax με το 0x76.
- Ένας τρόπος σύμφωνα με τα προηγούμενα να δούμε το τελικό αποτέλεσμα στη μνήμη θα ήταν βάζοντας breakpoint μετά το jump (διεύθυνση 0x07ff75bb210be) και βλέποντας την κατάσταση της μνήμης τότε

Κεφάλαιο 2 – BreakPoints

- Ωστόσο μπορούμε να το κάνουμε και με hardware breakpoint.

```
Address      Hex      ASCII
000000CC9FF10F 19 03 29 10 19 03 18 12 29 1B 13 57 00 00 00 ...). ....). .W...
```

- Αν μεταβούμε στη μνήμη που προσπελάζει ο `rax` θα δούμε ποια είναι τα 13 bytes που θα προσπελαστούν(`0x0d + 1` αφού ο `ecx` ήταν αρχικά μηδέν). Μπορούμε λοιπόν να θέσουμε breakpoint όταν γραφτεί και το τελευταίο byte μέσω `xor`, το `0x57`.



Κεφάλαιο 2 – BreakPoints

- Συνεχίζουμε την εκτέλεση...

Address	Hex	ASCTT
000000CC9FF1	79 6F 75 5F 66 6F 75 6E 64 5F 6D 65 21 00 00 00	you_found_me!...
000000CC9FF1	3E 34 9B BF 14 5A 00 00 00 00 00 00 00 00 00	>4.ζ.Ζ.....
000000CC9FF1	00 00 00 00 00 00 00 00 10 13 B2 5B F7 7F 00 00 ² [÷....
000000CC9FF1	00 00 00 00 00 00 00 00 89 13 B2 5B F7 7F 00 00 ² [÷....
000000CC9FF1	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000CC9FF1	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000CC9FF1	00 00 00 00 00 00 00 00 7D 25 CC 7A FE 7F 00 00}%!zþ...
000000CC9FF1	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000CC9FF1	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000CC9FF1	00 00 00 00 00 00 00 00 48 AA 6E 7C FE 7F 00 00H ^a n þ...
000000CC9FF1	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000CC9FF1	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Command:

Paused Hardware breakpoint (byte, read/write) at 000000CC9FF1F7FC!

- Και βλέπουμε ότι το πρόγραμμα σταματάει όταν γραφεί το ζητούμενο byte, δίνοντάς μας το “decrypted” string: “you_found_me!”.

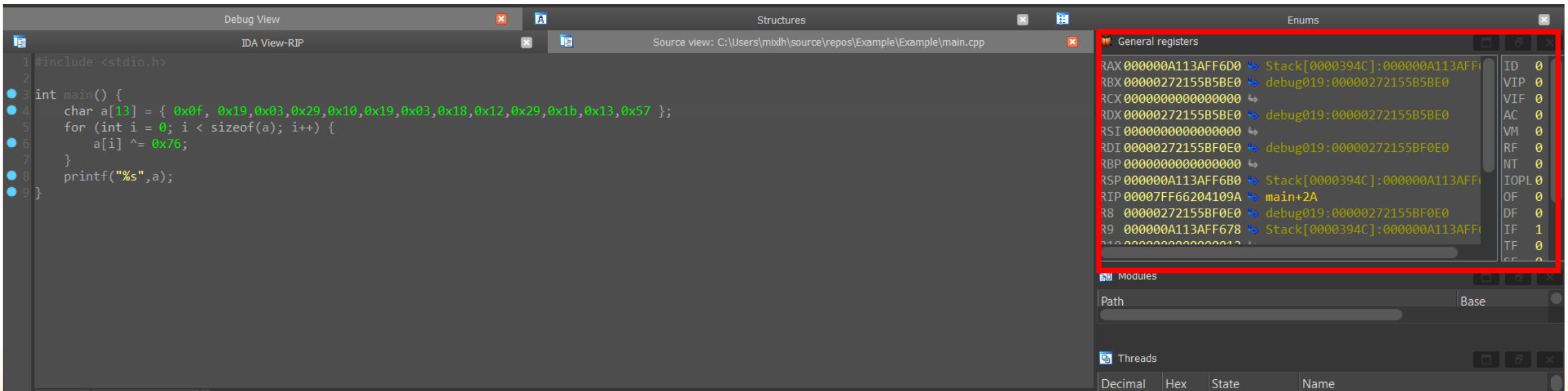
Κεφάλαιο 2 – BreakPoints

- Τα hardware breakpoints είναι πολύ χρήσιμα, ειδικά όταν γνωρίζουμε/υποθέτουμε ότι μια θέση μνήμης θα προσπελαστεί/μεταβληθεί, αλλά δεν γνωρίζουμε το πότε.
- Μπορούν να τεθούν σε επίπεδο byte, word, dword, qword αλλά ακόμα και σε επίπεδο σελίδας!
- Κατά την αναζήτηση για τη συγγραφή των διαφανειών, βρήκα ότι υπάρχει και ο τύπος των breakpoints για I/O operations (Kernel mode only), δηλαδή όταν εκτελεστεί κάποια εντολή εισόδου/εξόδου σε συγκεκριμένη θύρα.

Κεφάλαιο 3 - Views

- Ένας debugger προσφέρει διάφορα views κατά την εκτέλεση του προγράμματος.
- Έτσι, κανείς μπορεί να δει:

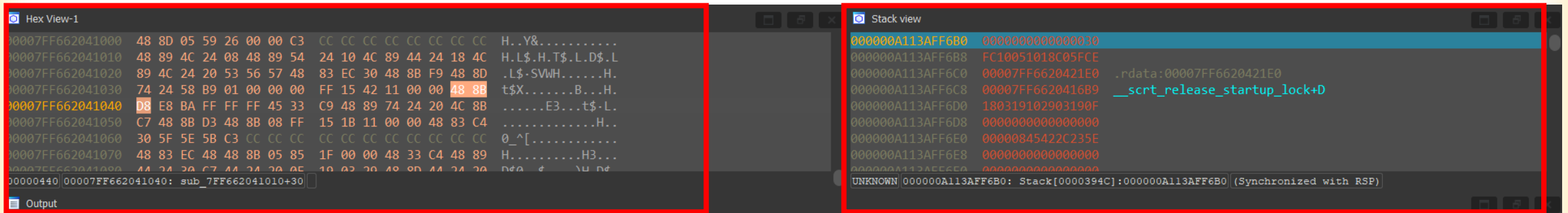
1. Τους καταχωρητές:



Κεφάλαιο 3 - Views

2. Τη στοίβα του προγράμματος

3. Τη μνήμη του προγράμματος



Memory View

Stack View

Κεφάλαιο 3 - Views

4. Το memory map του προγράμματος

Address	Size	Party	Info	Content	Type	Protection	Initial
000000007FFE0000	0000000000001000	User	KUSER_SHARED_DATA		PRV	-R---	-R---
000000007FFE8000	0000000000001000	User			PRV	-R---	-R---
0000000082EBC80000	000000000000FA000	User	Reserved		PRV	-RW--	-RW--
0000000082EBD7A000	00000000000006000	User	Stack (24860)		PRV	-RW-G	-RW--
0000000082EBE00000	00000000000198000	User	Reserved		PRV	-RW--	-RW--
0000000082EBF98000	00000000000009000	User	PEB, TEB (24860), TEB (4524), TEB (4524)		PRV	-RW--	-RW--
0000000082EBFA4000	0000000000005C000	User	Reserved (00000082EBE00000)		PRV	-RW--	-RW--
0000000082EC000000	000000000000F8000	User	Reserved		PRV	-RW--	-RW--
0000000082EC0F8000	00000000000005000	User	Stack (9156)		PRV	-RW-G	-RW--
0000000082EC100000	000000000000F8000	User	Reserved		PRV	-RW--	-RW--
0000000082EC1F8000	00000000000005000	User	Stack (30292)		PRV	-RW-G	-RW--
0000000082EC200000	000000000000FC000	User	Reserved		PRV	-RW--	-RW--
0000000082EC2FC000	00000000000004000	User	Stack (4524)		PRV	-RW-G	-RW--
0000027E70030000	00000000000010000	User			MAP	-RW--	-RW--
0000027E70040000	00000000000030000	User	\Device\HarddiskVolume3\Windows\		MAP	-R---	-R---
0000027E70050000	0000000000001F000	User			MAP	-R---	-R---
0000027E70070000	00000000000004000	User			MAP	-R---	-R---
0000027E70080000	00000000000001000	User			MAP	-R---	-R---
0000027E70090000	00000000000002000	User			PRV	-RW--	-RW--
0000027E700A0000	00000000000011000	User	\Device\HarddiskVolume3\Windows\		MAP	-R---	-R---
0000027E700C0000	00000000000011000	User	\Device\HarddiskVolume3\Windows\		MAP	-R---	-R---
0000027E700E0000	00000000000030000	User	\Device\HarddiskVolume3\Windows\		MAP	-R---	-R---
0000027E700F0000	000000000000CE000	User	\Device\HarddiskVolume3\Windows\		MAP	-R---	-R---
0000027E70100000	00000000000011000	User	\Device\HarddiskVolume3\Windows\		MAP	-R---	-R---
0000027E70110000	00000000000011000	User	\Device\HarddiskVolume3\Windows\		MAP	-R---	-R---
0000027E702C0000	0000000000000F000	User			PRV	-RW--	-RW--
0000027E702CF000	0000000000000F1000	User	Reserved (0000027E702C0000)		PRV	-RW--	-RW--
00007FF4E7480000	00000000000005000	User			MAP	-R---	-R---
00007FF4E7485000	000000000000F8000	User	Reserved (00007FF4E7480000)		MAP	-R---	-R---
00007FF4E7580000	0000000100020000	User	Reserved		PRV	-RW--	-RW--
00007FF5E75A0000	00000000000200000	User	Reserved		PRV	-RW--	-RW--
00007FF5E95A0000	00000000000001000	User			PRV	-RW--	-RW--
00007FF5E9580000	00000000000001000	User			MAP	-R---	-R---
00007FF662040000	00000000000001000	User	example.exe	Executable code	IMG	-R---	ERWC-
00007FF662041000	00000000000001000	User	".text"	Read-only initialized data	IMG	-R---	ERWC-
00007FF662042000	00000000000001000	User	".rdata"	Initialized data	IMG	-RW--	ERWC-
00007FF662043000	00000000000001000	User	".data"	Exception information	IMG	-R---	ERWC-
00007FF662044000	00000000000001000	User	".pdata"	Resources	IMG	-R---	ERWC-
00007FF662045000	00000000000001000	User	".rsrc"	Base relocations	IMG	-R---	ERWC-
00007FF662046000	00000000000001000	User	".reloc"		IMG	-R---	ERWC-
00007FFB8C98E0000	00000000000001000	System	vcruntime140.dll	Executable code	IMG	-R---	ERWC-
00007FFB8C98E1000	000000000000011000	System	".text"	Read-only initialized data	IMG	-R---	ERWC-
00007FFB8C98F2000	00000000000001000	System	".fothk"	Initialized data	IMG	-RW--	ERWC-
00007FFB8C98F3000	000000000000005000	System	".rdata"	Exception information	IMG	-R---	ERWC-
00007FFB8C98F8000	00000000000001000	System	".data"	Resources	IMG	-R---	ERWC-
00007FFB8C98F9000	00000000000001000	System	".pdata"	Base relocations	IMG	-R---	ERWC-
00007FFB8C98FA000	00000000000001000	System	"._RDATA"		IMG	-R---	ERWC-
00007FFB8C98FB000	00000000000001000	System	"._RSRC"		IMG	-R---	ERWC-
00007FFB8C98FC000	00000000000001000	System	"._RELOC"		IMG	-R---	ERWC-

5. Και γενικά ανάλογα τον debugger, μπορεί να συναντήσουμε thread view, handle view, καρτέλες για scripting κτλ.

Κεφάλαιο 4 – Modifying program flow

- Ένα πολύ σημαντικό στοιχείο των debuggers είναι η δυνατότητα που σου δίνουν να μεταβάλλεις την εκτέλεση του προγράμματος τόσο σε επίπεδο registers και μνήμης όσο και το να αλλάξεις δυναμικά τις εντολές του προγράμματος.
- Πολλές φορές, τα προγράμματα που καλούμαστε να κάνουμε reverse έχουν anti-debugging tricks. Μέσω patching/modifying με τις κατάλληλες τιμές, μπορούμε να κάνουμε bypass τα περισσότερα από αυτά τα τρικς.
- Ας δούμε ένα παράδειγμα τέτοιας τροποποίησης:

Κεφάλαιο 4 – Modifying program flow

```
1 using System;
2 using System.Runtime.CompilerServices;
3
4 // Token: 0x02000002 RID: 2
5 [CompilerGenerated]
6 internal class Program
7 {
8     // Token: 0x06000001 RID: 1 RVA: 0x00002050 File Offset: 0x00000250
9     private static void <Main>$(string[] args)
10     {
11         Console.WriteLine("Guess my number! - ");
12         string user_number = Console.ReadLine();
13         if (user_number != "" && int.Parse(user_number) == 56)
14         {
15             Console.WriteLine("Congrats!!");
16             return;
17         }
18         Console.WriteLine("Better Luck next time");
19     }
20 }
21
```

110 %

Locals

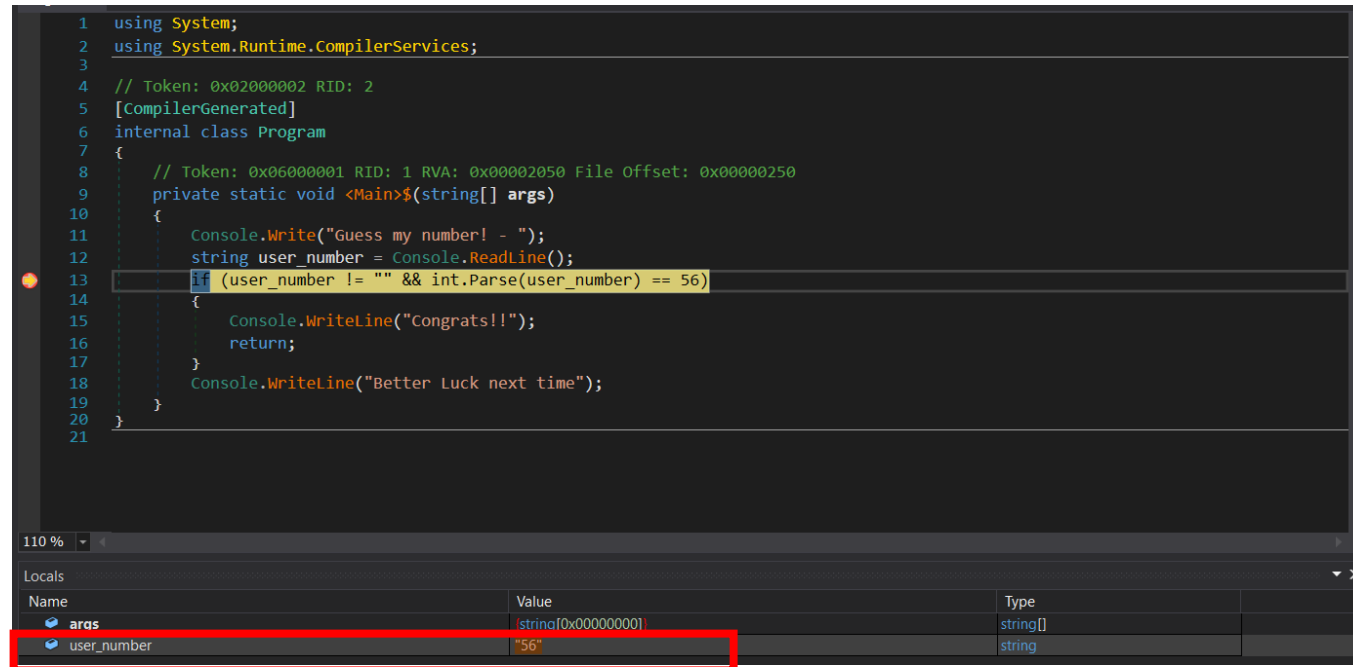
Name	Value	Type
args	(string[0x00000000])	string[]
user_number	"80"	string

Στο «χαζό» πρόγραμμα αριστερά, ο χρήστης καλείται να μαντέψει έναν αριθμό.

Παρατηρούμε ότι το input που δόθηκε είναι το 80 != 56, οπότε το guess ήταν λάθος.

Κεφάλαιο 4 – Modifying program flow

Ωστόσο, θέτοντας κατάλληλο breakpoint πριν την εκτέλεση του if statement, μπορούμε να τροποποιήσουμε το user value σε 56:



```
1 using System;
2 using System.Runtime.CompilerServices;
3
4 // Token: 0x02000002 RID: 2
5 [CompilerGenerated]
6 internal class Program
7 {
8     // Token: 0x06000001 RID: 1 RVA: 0x00002050 File Offset: 0x00000250
9     private static void <Main>$(string[] args)
10     {
11         Console.WriteLine("Guess my number! - ");
12         string user_number = Console.ReadLine();
13         if (user_number != "" && int.Parse(user_number) == 56)
14         {
15             Console.WriteLine("Congrats!!");
16             return;
17         }
18         Console.WriteLine("Better Luck next time");
19     }
20 }
21
```

110 %

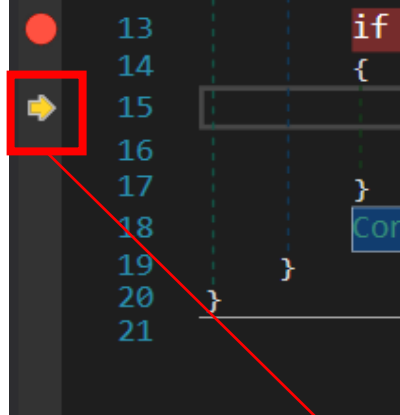
Locals

Name	Value	Type
args	string[0x00000000]	string[]
user_number	"56"	string

Με step over, διαπιστώνουμε ότι καταφέραμε να περάσουμε το check και μαντέψαμε σωστά τον αριθμό!

Κεφάλαιο 4 – Modifying program flow

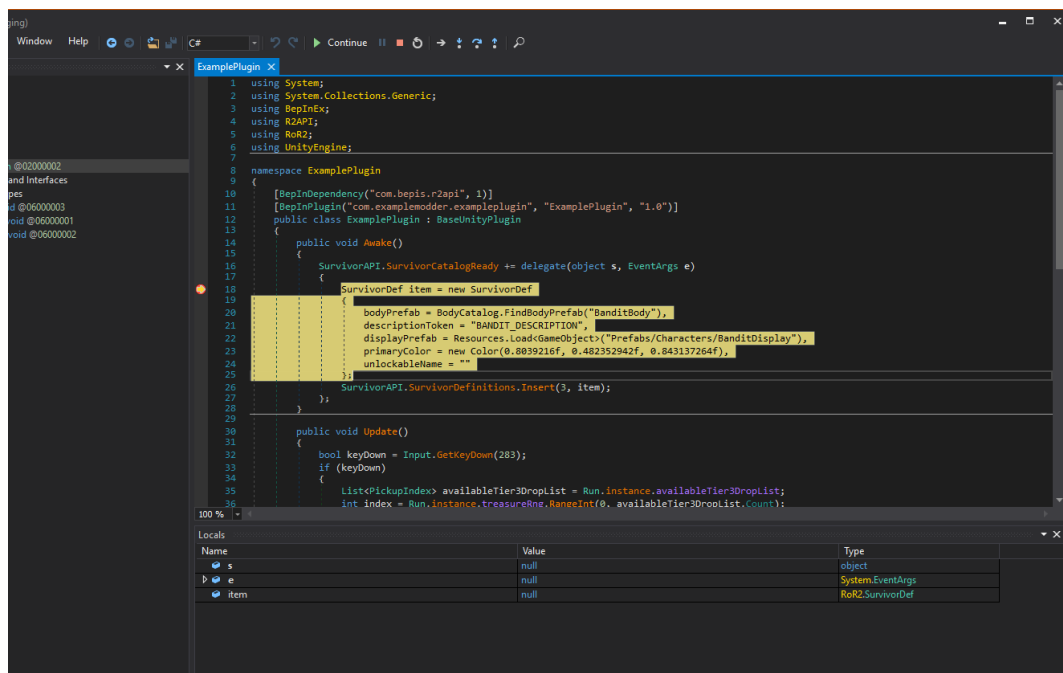
```
5 [CompilerGenerated]
6 internal class Program
7 {
8     // Token: 0x06000001 RID: 1 RVA: 0x00002050 File Offset: 0x00000250
9     private static void <Main>$(string[] args)
10    {
11        Console.Write("Guess my number! - ");
12        string user_number = Console.ReadLine();
13        if (user_number != "" && int.Parse(user_number) == 56)
14        {
15            Console.WriteLine("Congrats!!");
16            return;
17        }
18        Console.WriteLine("Better Luck next time");
19    }
20 }
21
```



Το πρόγραμμα μπήκε επιτυχώς μέσα στην if

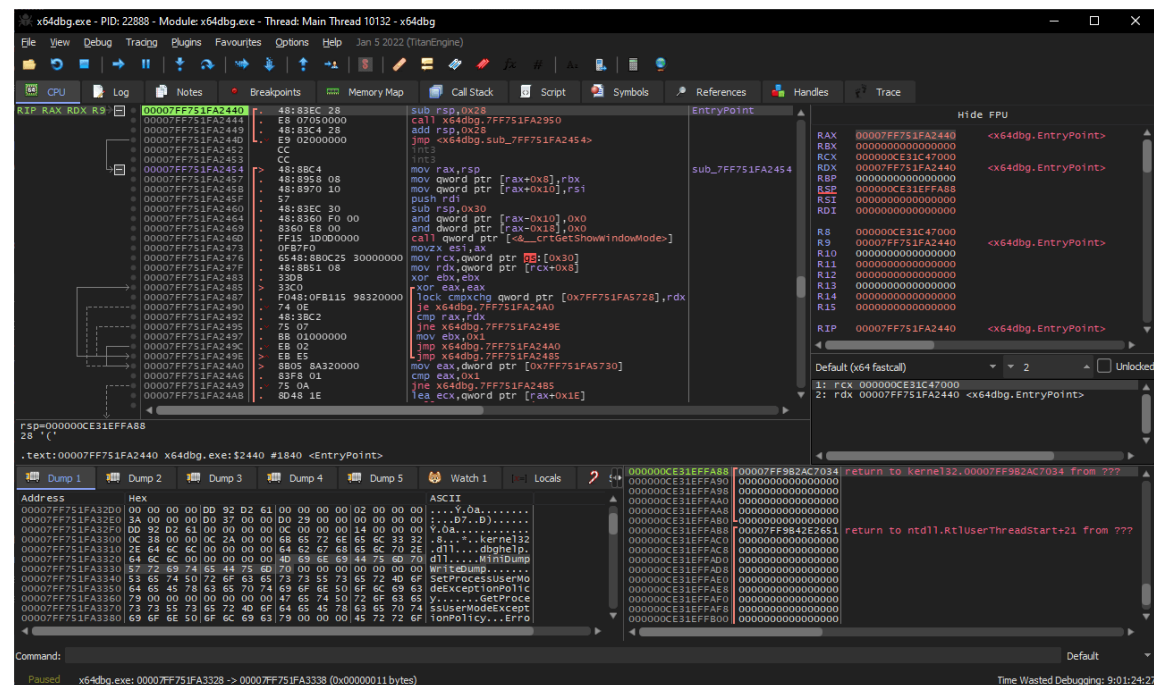
Επίλογος – Παραδείγματα debuggers

DnSpy



```
1 using System;
2 using System.Collections.Generic;
3 using UnityEngine;
4 using UnityEngine.UI;
5 using UnityEngine.SceneManagement;
6 using UnityEngine.Networking;
7
8 namespace ExamplePlugin
9 {
10     [Dependency(typeof(BaseUnityPlugin))]
11     [Dependency(typeof(BaseUnityPlugin))]
12     public class ExamplePlugin : BaseUnityPlugin
13     {
14         public void Awake()
15         {
16             SurvivorAPI.SurvivorCatalogReady += delegate(object s, EventArgs e)
17             {
18                 SurvivorDef item = new SurvivorDef
19                 {
20                     bodyPrefab = BodyCatalog.FindBodyPrefab("BanditBody"),
21                     descriptionToken = "BANDIT_DESCRIPTION",
22                     displayPrefab = Resources.Load<GameObject>("Prefabs/Characters/BanditDisplay"),
23                     primaryColor = new Color(0.809216f, 0.48235294f, 0.843137264f),
24                     unlockableName = ""
25                 };
26                 SurvivorAPI.SurvivorDefinitions.Insert(3, item);
27             };
28         }
29
30         public void Update()
31         {
32             bool keyDown = Input.GetKeyDown(283);
33             if (keyDown)
34             {
35                 List<PickupIndex> availableTier3DropList = Run.Instance.availableTier3DropList;
36                 int index = Run.Instance.treasureRng.RangeInt(0, availableTier3DropList.Count);
37             }
38         }
39     }
40 }
```

x86dbg



```
00007FF751FA2440 48:8BEC 28 sub esp,0x28
00007FF751FA2444 E8:07050000 call x64dbg.7FF751FA2950
00007FF751FA2449 48:83C4 28 add esp,0x28
00007FF751FA244D E9:02000000 jmp <x64dbg.sub_7FF751FA2454>
00007FF751FA2453 CC int3
00007FF751FA2454 CC int3
00007FF751FA2457 48:8B8C 08 mov rax,rsi
00007FF751FA245B 48:8958 08 mov qword ptr [rax+0x8],rbx
00007FF751FA245F 48:8970 10 mov qword ptr [rax+0x10],rsi
00007FF751FA2463 57 push rdi
00007FF751FA2466 48:83EC 30 sub rbp,0x30
00007FF751FA246A 48:8B80 F0 00 mov qword ptr [rax-0x10],0x0
00007FF751FA246E 8B80 E8 00 and dword ptr [rax-0x18],0x0
00007FF751FA2472 FF15 10000000 call qword ptr [cs:_crGetShowWindowMode]
00007FF751FA2473 0FB7F0 movzx esi,ax
00007FF751FA2476 6548:8B0C25 30000000 mov rcx,qword ptr [0x30]
00007FF751FA247E 48:8B51 08 mov rdx,qword ptr [rcx+0x8]
00007FF751FA2483 3308 xor ebx,ebx
00007FF751FA2485 33C0 xor eax,eax
00007FF751FA2487 F048:0F8115 98320000 lock cmovchg qword ptr [0x7FF751FA5728],rdx
00007FF751FA248D 74 0E je x64dbg.7FF751FA24A0
00007FF751FA248E 48:12BC 02 cmp rax,rdx
00007FF751FA2490 75 07 jne x64dbg.7FF751FA249E
00007FF751FA2492 8B 01 mov ebx,0x1
00007FF751FA2494 jmp x64dbg.7FF751FA24A0
00007FF751FA2496 E8 02 jmp x64dbg.7FF751FA2485
00007FF751FA2498 8B80 8A320000 mov eax,dword ptr [0x7FF751FA5730]
00007FF751FA249C 83F8 01 cmp eax,0x1
00007FF751FA249E 75 0A jne x64dbg.7FF751FA2485
00007FF751FA24A0 48:8B48 1E lea ecx,qword ptr [rax+0x1E]
```

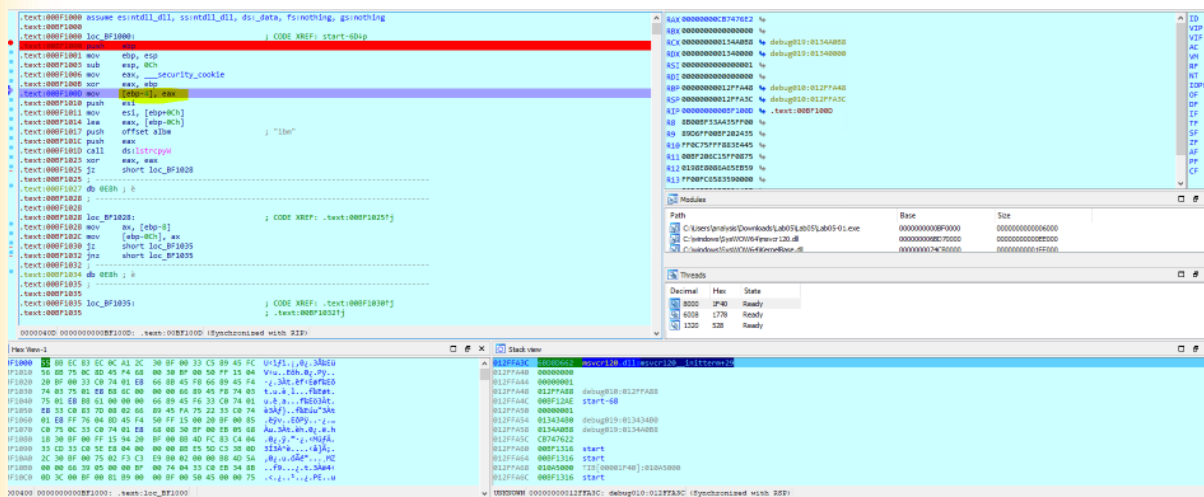
Ιδανικός για .NET executables.
Επίσης είναι disassembler/decompiler.

Μόνο για windows executables και ο
προσωπικά αγαπημένος μου debugger.

Επίλογος – Παραδείγματα debuggers

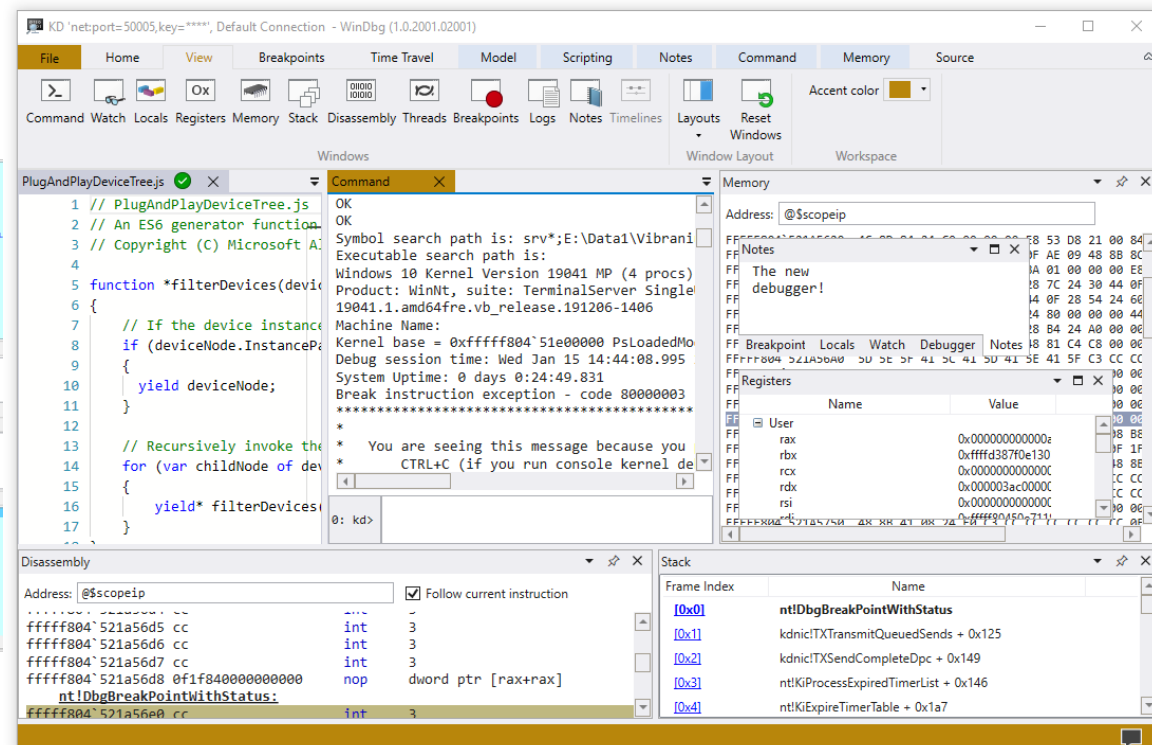
Windbg

IDA



Ναι, εκτός από disassembler/decompiler είναι και debugger, ακραίο?

Κάνει και για linux και για windows.



Ο Windbg υποστηρίζει τόσο user- όσο και kernel-mode debugging.

Επίλογος – Παραδείγματα debuggers

GDB

```
(gdb) info breakpoints
Num      Type           Disp Enb Address              What
1        breakpoint     keep y   0x000000000040053b in factorial at example.c:4
(gdb) run
Starting program: /home/akulkarni/projects/gdb-basics/factorial

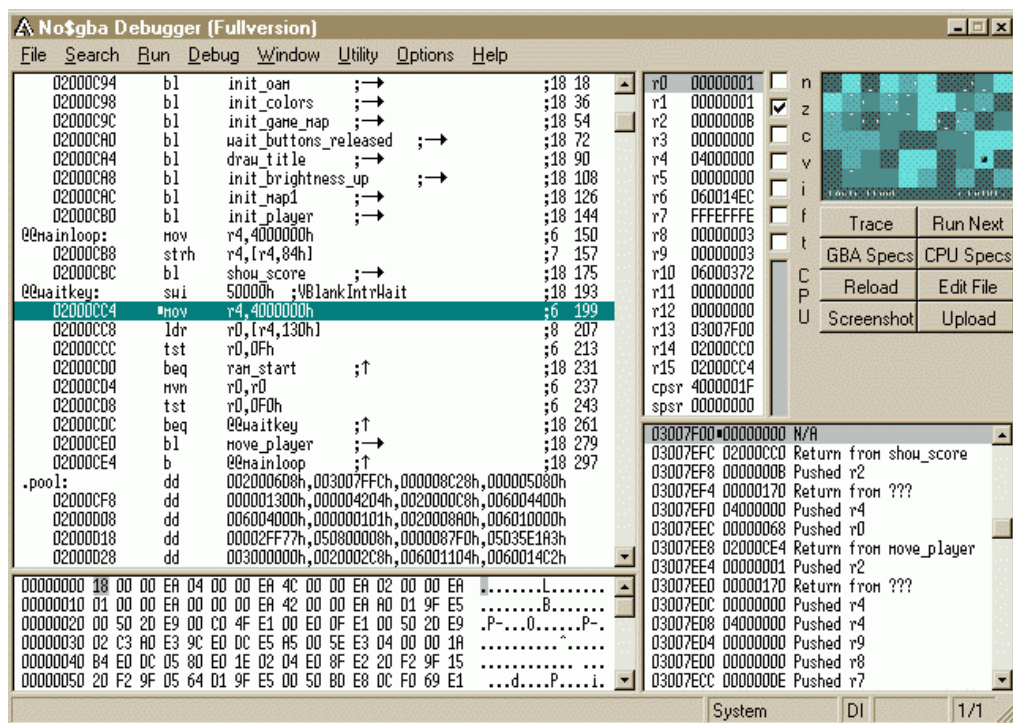
Breakpoint 1, factorial (n=5, a=1) at example.c:4
4          printf("Value of n is %d\n",n);
(gdb) condition 1 n==2
(gdb) info breakpoints
Num      Type           Disp Enb Address              What
1        breakpoint     keep y   0x000000000040053b in factorial at example.c:4
          stop only if n==2
          breakpoint already hit 1 time
(gdb) continue
Continuing.
Value of n is 5
Value of n is 4
Value of n is 3

Breakpoint 1, factorial (n=2, a=60) at example.c:4
4          printf("Value of n is %d\n",n);
```

O go to linux debugger. More on that later

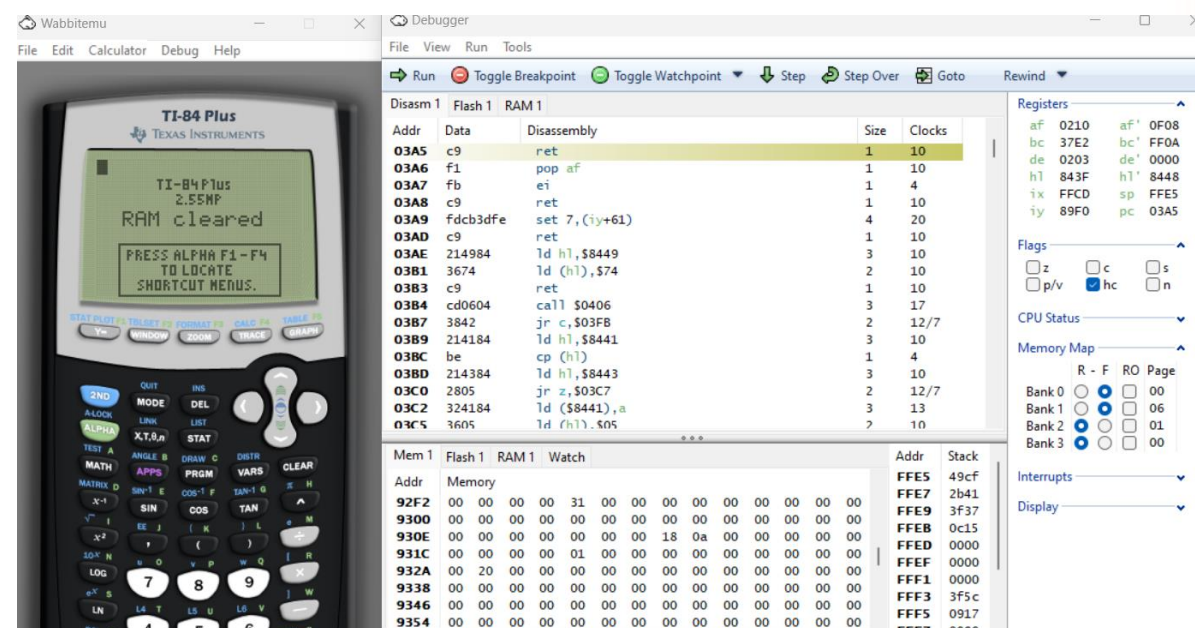
Επίλογος – (Ιδιαίτερα) Παραδείγματα debuggers

No\$gba



Πέρα από emulator για game boy παιχνίδια, διαθέτει και debugger.

Wabbitemu



Emulator + Debugger για κομπιουτεράκια της Texas Instruments.

Resources

- [Learn GDB](#)
- [Learn x64dbg](#)
- [HTB - Reverse: BombsLanded, DebugMe challenges](#)
- [PicoCTF - GDB baby step 1-4](#)
- [A great list of debuggers](#)

Παράρτημα - GDB



- GDB = GNU Debugger.
- Χρησιμοποιείται για debugging σε linux elfs.
- Υποστηρίζει scripting.
- Συνήθως κανείς δε χρησιμοποιεί τον vanilla gdb (AT&T syntax = μπλιαχ, και δε δείχνει σχεδόν καμμία πληροφορία στην οθόνη αν δεν τη ζητήσεις).
- Υπάρχουν λοιπόν διάφορα gdb flavors (pwndbg, gef, gdb-peda κτλ.).

Παράρτημα – Gef (Gdb Enhanced Features)

Όπως λέει και το όνομα, ο gef είναι ένας ενισχυμένος gdb που δίνει πολύ περισσότερες πληροφορίες για την εκτέλεση του προγράμματος σε σχέση με τον απλό gdb.

Έτσι, με κάθε step βλέπουμε στην οθόνη τη γραμμή του disassembled κώδικα που βρισκόμαστε, καθώς και την εικόνα των καταχωρητών και της στοίβας.

```
[ Legend: Modified register | Code | Heap | Stack | String ]
registers
$rax : 0x0
$rbx : 0x0
$rcx : 0x00007ffff7ffcca0 → 0x0004095d00000000
$rdx : 0x0
$rsp : 0x00007fffffe530 → 0x0000000000000000
$rbp : 0x00007fffffe560 → 0x000000000000007f → <__libc_csu_init+0> push r15
$rsi : 0x00007ffff7dd1b78 → 0x0000000000000000
$rdi : 0x20000
$rip : 0x0000000000000079 → <main+64> mov QWORD PTR [rbp-0x28], rax
$e8 : 0x00007ffff7fec700 → 0x00007ffff7fec700 → [loop detected]
$e9 : 0x1
$e10 : 0x0
$e11 : 0x246
$e12 : 0x0000000000000058 → <_start+0> xor ebp, ebp
$e13 : 0x00007fffffe640 → 0x0000000000000001
$e14 : 0x0
$e15 : 0x0
$eflags: [carry PARITY adjust ZERO sign trap INTERRUPT direction overflow resume virtualx86 identification]
$cs: 0x002b $ds: 0x0033 $es: 0x0000 $fs: 0x0000 $gs: 0x0000

stack
0x00007fffffe530 +0x0000: 0x0000000000000000 ← $rsp
0x00007fffffe538 +0x0008: 0x0000000000000000
0x00007fffffe540 +0x0010: "myfile.txt"
0x00007fffffe548 +0x0018: 0x000000000000007f ("xt"? )
0x00007fffffe550 +0x0020: 0x00007fffffe640 → 0x0000000000000001
0x00007fffffe558 +0x0028: 0xd7c3f14d3cddb000
0x00007fffffe560 +0x0030: 0x000000000000007f → <__libc_csu_init+0> push r15 ← $rbp
0x00007fffffe568 +0x0038: 0x00007ffff7a2d830 → <__libc_start_main+240> mov edi, eax

code:i386:x86-64
0x40078c <main+51> mov esi, 0x400874
0x400791 <main+56> mov rdi, rax
0x400794 <main+59> call 0x400550 <fopen@plt>
→ 0x400799 <main+64> mov QWORD PTR [rbp-0x28], rax
0x40079d <main+68> cmp QWORD PTR [rbp-0x28], 0x0
0x4007a2 <main+73> jne 0x4007bc <main+99>
0x4007a4 <main+75> lea rax, [rbp-0x20]
0x4007a8 <main+79> mov rsi, rax
0x4007ab <main+82> mov edi, 0x400876

source:vsprintf.c+20
15 int main ()
16 {
17     FILE * pFile;
18     char szFileName[]="myfile.txt";
19     // pFile=0x00007fffffe538 → 0x0000000000000000, szFileName=0x00007fffffe540 → "myfile.txt"
→ 20     pFile = fopen (szFileName,"r");
21     if (pFile == NULL)
22         PrintError ("Error opening '%s'",szFileName);
23     else
24     {
25         // file successfully open

threads
[#0] Id 1, Name: "vsprintf", stopped, reason: SINGLE STEP

trace
[#0] 0x400799 → Name: main()

gef>
```

Παράρτημα – Gdb: Expanding the arch

- Τέλος, υπάρχει και ο gdb-multiarch ο οποίος χρησιμοποιείται για debugging προγραμμάτων σε αρχιτεκτονική διαφορετική της x86.
- Συνδυάζεται με τα υπάρχοντα flavors του gdb, επιτρέποντας έτσι debugging προγραμμάτων με familiar τρόπο

```
0x7fa40824 +0x000c: 0x772c5c80 → lui gp, 0x2
0x7fa40828 +0x0010: 0x00000010
0x7fa4082c +0x0014: 0x006d4cb8 → 0x006d4d98 → 0x006d7350 → 0x006d5048 → 0x006d4db8 → 0x00000003
0x7fa40830 +0x0018: 0x7fa408e4 → 0x00000000
0x7fa40834 +0x001c: 0x00000010

code:mips:MIPS32
0x7700c018 sw v0, 16(sp)
0x7700c01c li v0, 4302
0x7700c020 syscall
→ 0x7700c024 addiu sp, sp, 32
0x7700c028 beqz a3, 0x7700c048
0x7700c02c nop
0x7700c030 0x7c03e83b
0x7700c034 move a0, v1
0x7700c038 lw v1, -29232(gp)

threads
[ #0 ] Id 1, Name: "Deuteron", stopped, reason: STOPPED

trace
[ #0 ] 0x7700c024 → addiu sp, sp, 32
[ #1 ] 0x772849cc → d_socket_poll()
[ #2 ] 0x40570c → worker_timer_expired()
[ #3 ] 0x77284fa4 → d_socket_poll_run()
[ #4 ] 0x402ce8 → _ftext()

0x7700c024 in ?? () from target:/lib/libc.so.0
gef> set follow-fork-mode child
gef> c
Continuing.
```

Gef + GDB-multiarch

THE END

10/10 → THE CREATOR
