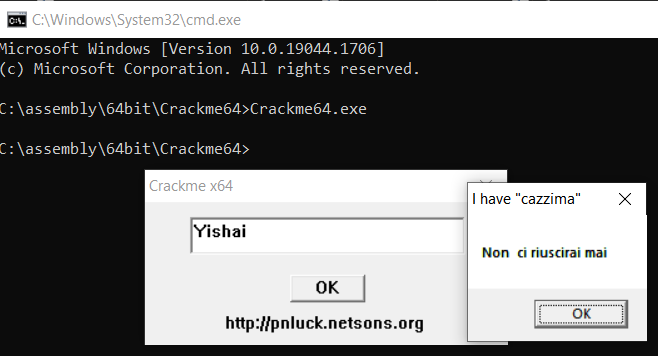
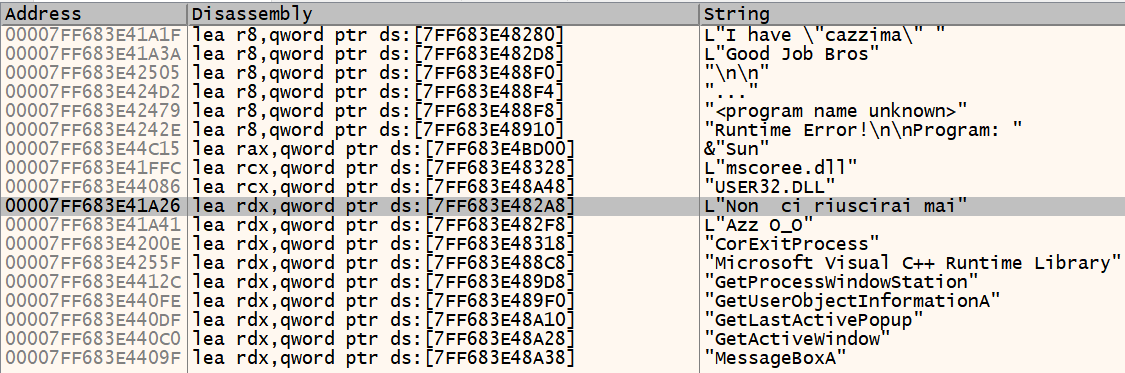
בס"ד

**Crackme64\_SolutionWay – ישי לוטווק**

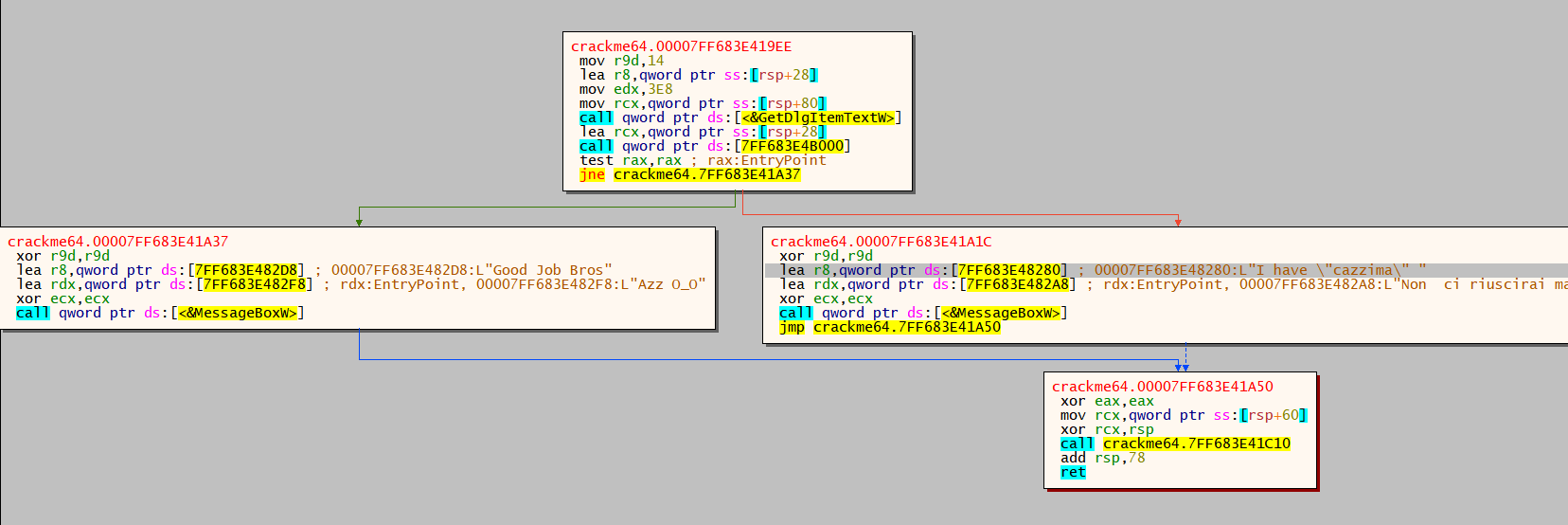
דבר ראשון נריץ בcmd ונראה מה קורה:



נעלה את הקובץ לx64dbg נריץ אותו לentryPoint, ונחפש את המחרוזת "Non ci riuscirai mai":



נלחץ על השורה המודגשת, ונסתכל על הקוד שמתקבל:



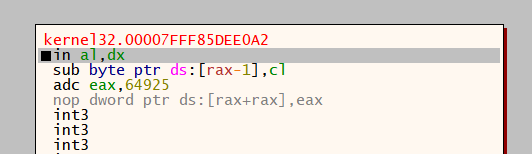
ניתן לראות שצד שמאל הוא האזור הטוב שבו מודפסת הודעת ההצלחה.

נריץ את הקוד בדיבאגר:

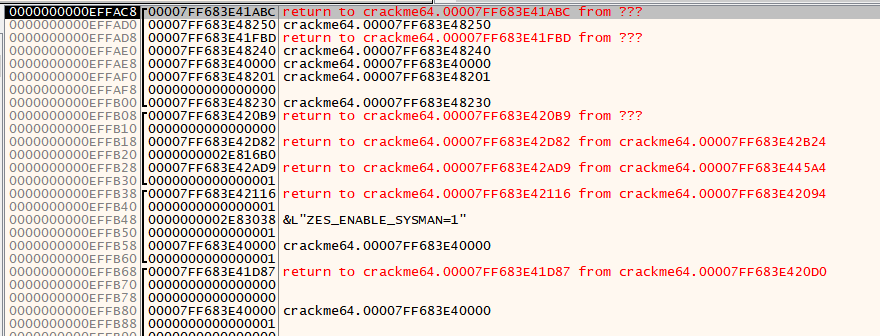


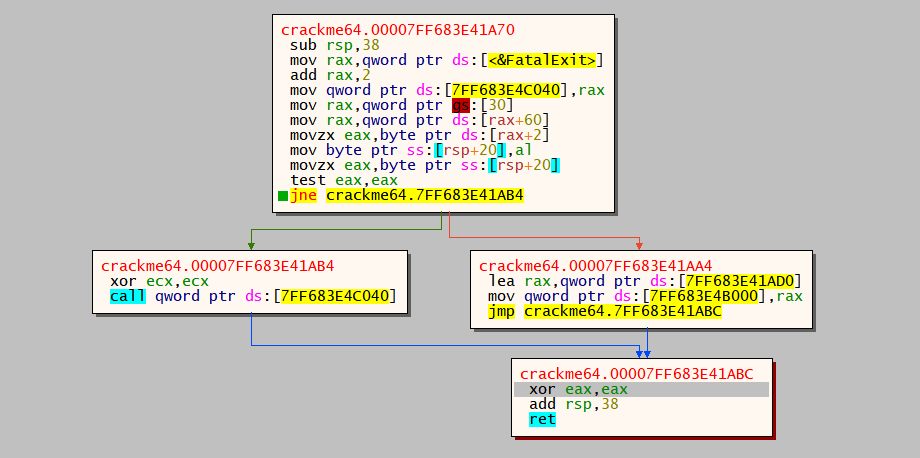
אוייש. כנראה יש מנגנון אנטידיבאג שגורם לקוד לקרוס לפני הקטע המעניין:

נלחץ על \* ונסתכל על השורה שבה הקוד קורס:



נלך אחורה דרך כתובות החזרה במחסנית הקריאות:





השורה של הxor (זאת שמודגשת) היא כתובת החזרה. כלומר הפקודה call בסוף הבלוק משמאל קראה לפונקציה שבתוכה היינו כרגע ובה אירעה הקריסה.

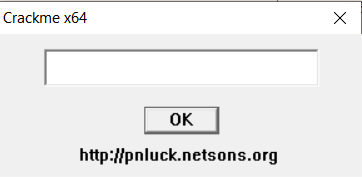
(נשים bp על השורה לפני ההתפצלות כדי שלא נשכח את הנקודה הזו אך נסמן את הbp כdisable כדי שלא יפריע לנו – הריבוע הירוק בתמונה למעלה)

ייתכן שמגנון האנטי דיבאג גורם לקוד להגיע לענף השמאלי ולקרוס במקום ללכת דרך הענף הימני שבו בכלל אין קריאה לפונקציה.

בוא ננסה רגע להעלות את הקובץ מחדש, ללכת לpeb+2 ולאפס אותו, ואז להמשיך את הריצה. אולי זה יעזור ואז נבין שכנראה יש איזו פונקציה שבודקת אם אנחנו מדבגים את הקוד באמצעות הערך שנמצא בבית השלישי של הpeb..



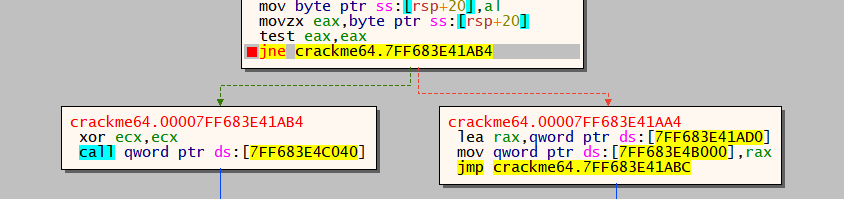
ונריץ:



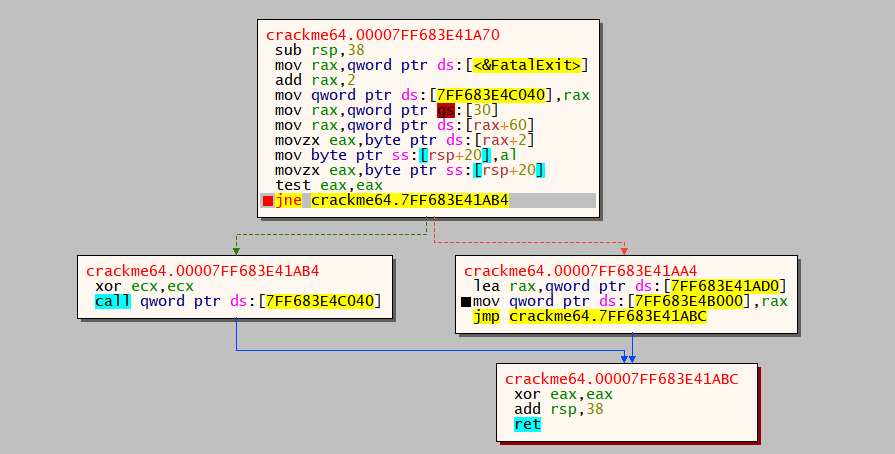
יש!! החלון נפתח!

נרצה לבדוק אם בהתפצלות בה פנינו קודם שמאלה וקרסנו הקוד עכשיו התנהג באופן שונה.

נחזור לנקודה הזו ונשים bp ואחר כך נחזור על מה שעשינו כעת (עם הpeb וכו') שוב.

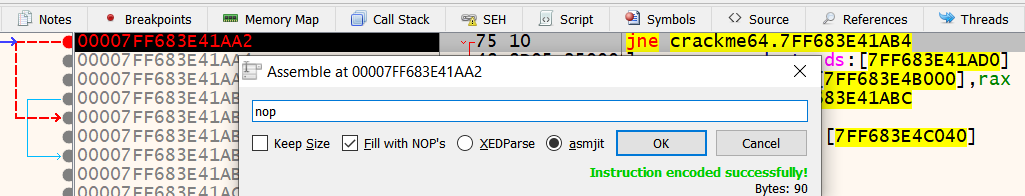


הקוד עצר בנקודה המיוחלת. עכשיו נצעד צעד צעד ונראה מה קורה:



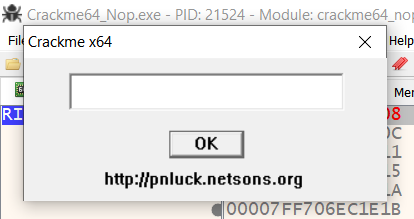
ניתן לראות את הריבוע השחור בענף הימני. כלומר זה ששינינו את הערך בבית השלישי בpeb ל0 גרם לקוד ללכת לענף הימני ולא כמו מקודם – לענף השמאלי (שגורם לקריסה).

נבצע פיצפוץ קטן כדי שלא נצטרך בכל הרצה לשנות את הבית השלישי בpeb:



|  |  |
| --- | --- |
| לפני | אחרי |
|  |  |

נשמור את הקובץ החדש המפוצפץ בשם Crackme64\_Nop.exe ונריץ אותו בדיבאגר לראות אם עכשיו זה יעבוד כמו שצריך:



עובד!

יכול להיות שסיימנו לנטרל את מנגנון האנטי דיבאג. ננסה להמשיך ואם נראה שיש עוד מנגנונים נעצור ונטפל בהם.

כרגע זה נראה שהייתה איפה שהוא פונקציית isDebuggerPresent שגרמה לקוד ללכת בענף השמאלי שהראינו בתמונות לעיל (אותו ענף שגורם לקוד לקרוס).

עכשיו נשים bp בנקודה של ההתפצלות בין האזור הטוב לאזור הרע,

נכניס סיסמא ונריץ את הקוד:



נשים לב שהתנאי בודק אם מה שיש בתוך הרגיסטר rax שווה ל0,

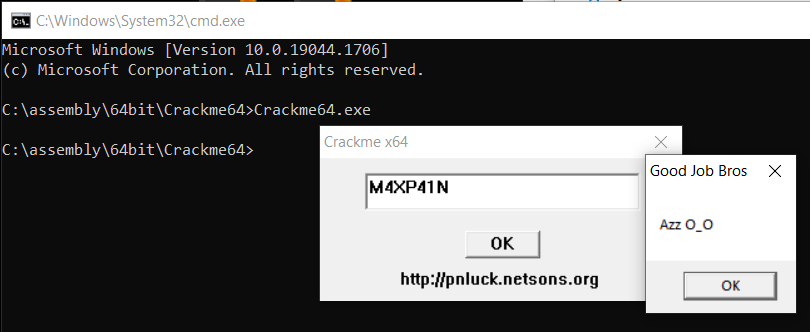
אם הוא 0 הולכים לצד ימין – הצד הרע, אחרת הולכים לצד הטוב.

ייתכן מאוד שהרגיסטר rax מכיל את הערך המוחזר מהפונקציה האחרונה שנקראת לפני ההתפצלות. ניכנס אליה רגע ונראה מה היא מכילה:

|  |
| --- |
|  |
|  |

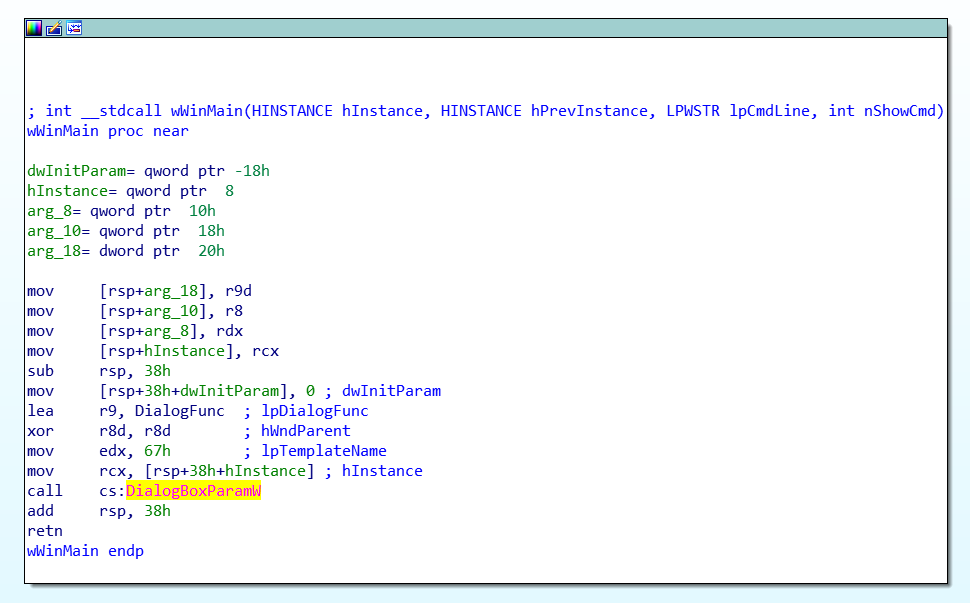
זה נראה שבודקים פה עם הסיסמא היא: **M4XP41N**.

ננסה את מזלנו - נפתח את הcmd ונריץ את הקובץ עם הסיסמה הזו:

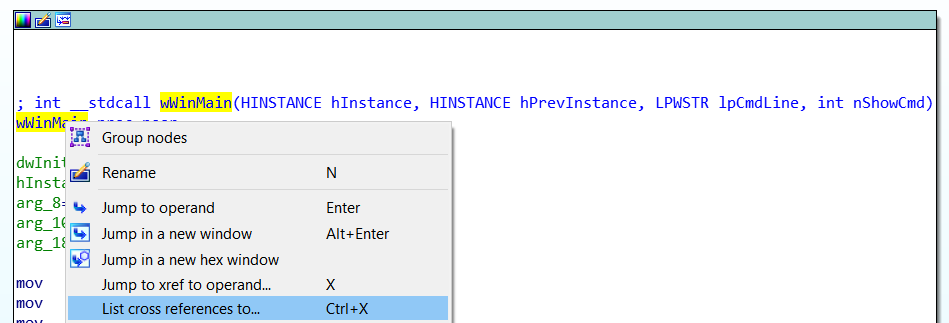


ב"ה!

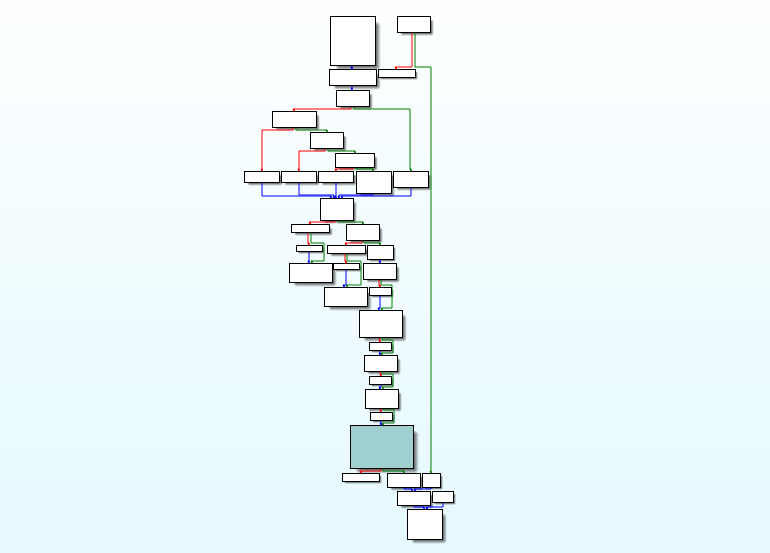
עכשיו נבדוק איפה המיין. בטח בida זה יהיה יותר פשוט. נפתח את הקובץ בida:



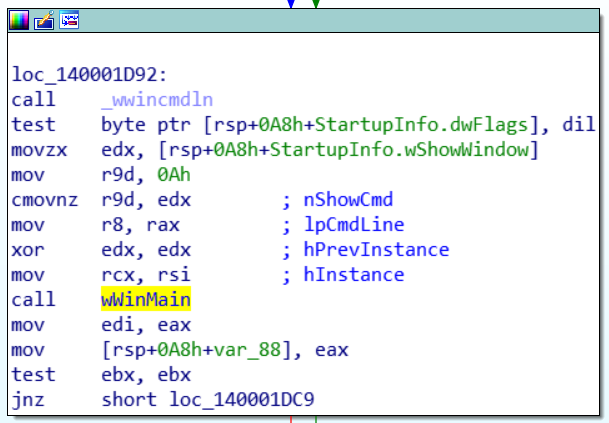
נלך אחורה באמצעות החלון הבא:



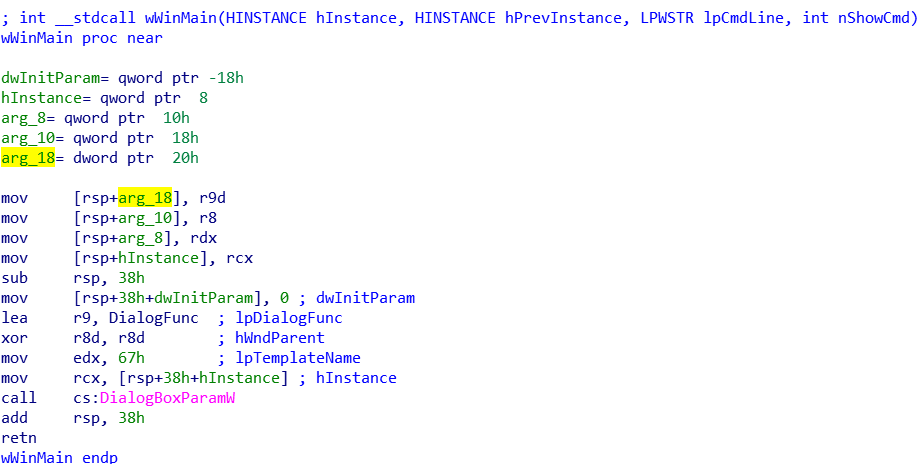
נגיע למבנה הבא כאשר אנו נמצאים במלבן המסומן בכחול:



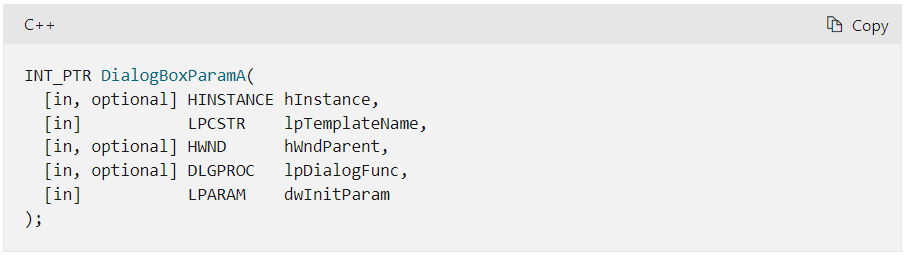
המלבן הכחול מקרוב, ניתן לראות את הקריאה לפונקציה main מסומנת בצהוב:

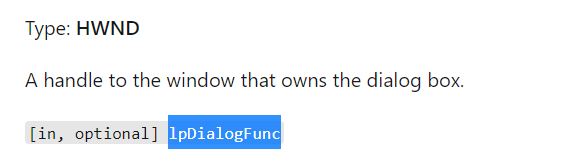


נחזור לחלון שנפתח לנו כשפתחנו את ida:



נחפש באינטרנט מה מקבלת הפונקציה DialogBoxParamW:

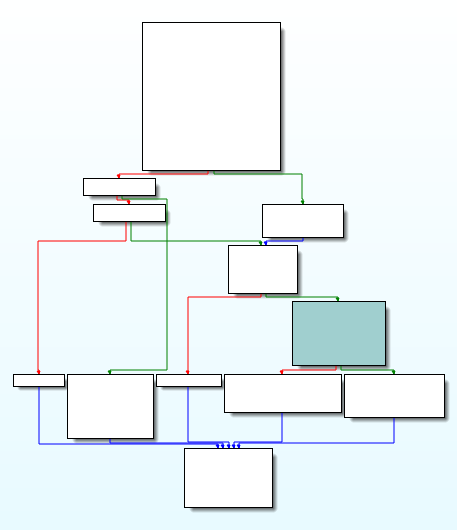




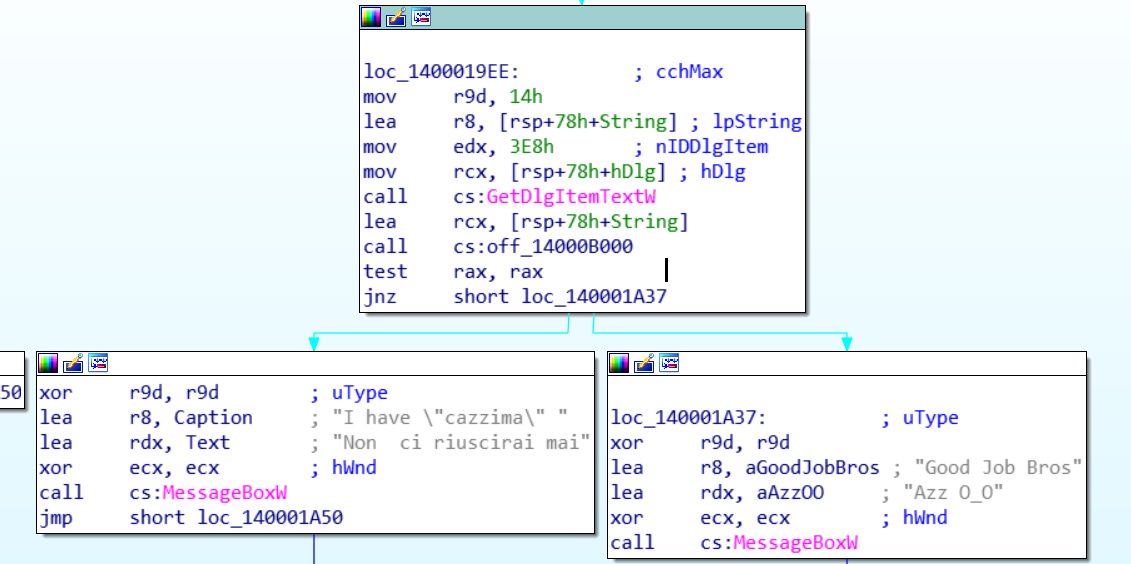
נראה שאנו שולחים לה בפרמטר lpDialogFunc את הפונקציה שמנהלת את העניינים..

ניכנס אליה וננסה למצוא את הקוד שראינו בx64dbg:

המבנה של הפונקציה הזו הוא המבנה הבא, כאשר המלבן הכחול מסתיים בתנאי שמפצל בין האזור הרע לאזור הטוב **שכבר ראינו בx64dbg:**



והנה המלבן הכחול מקרוב עם ההתפצלות לאזור הרע ולאזור הטוב:



לסיום רק נציין שכנראה שיש קוד שרץ לפני המיין – אותו קוד שאותו פצפצנו כדי לנטרל את מנגנון האנטי דיבאג.

מ.ש.ל.