מבוכים וסריאלים

מאת Zerith (אורי / Uri Farkas)

הקדמה

לפני שבוע הסתובבתי בפורום ספרדי מסוים שמתעסק ברוורסינג בין השאר, וראיתי בפוסטים הנעוצים פוסט אחד שמכיל כל מיני קראקמיים שפורסמו בפורום לאורך השנים, אם פתרו אותם או לא ומדריך לפיתרון אם ישנו.

עברתי על הרשימה הארוכה ומצאתי קראקמי שפורסם ב-2007, ולא נפתר עד 2008 ! (היה הפרש של בערך 10 חודשים). חשבתי לעצמי שאני חייב לראות למה לקח כל כך הרבה זמן לפתור את הקראקמי, אז הורדתי אותו, והתחלתי לחקור אותו.

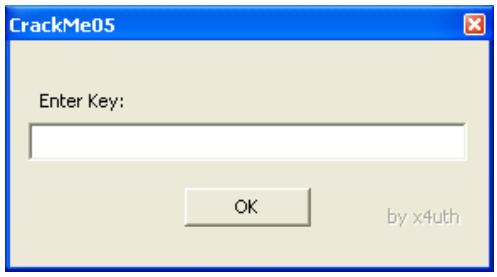
לאחר יום אחד הצלחתי לפתור את הקראקמי, בצורה יפה מאוד לדעתי, אז החלטתי לכתוב עליו. אני חושב שתוכלו ללמוד הרבה מהמאמר הזה על דרך החשיבה בחקירת קראקמיים.

ניתן להשיג את הקראקמי מהכתובת הבאה:

http://personal.telefonica.terra.es/web/carlos-ea/ie/CrackMe05.rar

נתחיל בחקירה

בפתיחה ראשונית של התכנית ללא דיבאגר, ניתן לראות חלון קטן שמבקש סריאל:

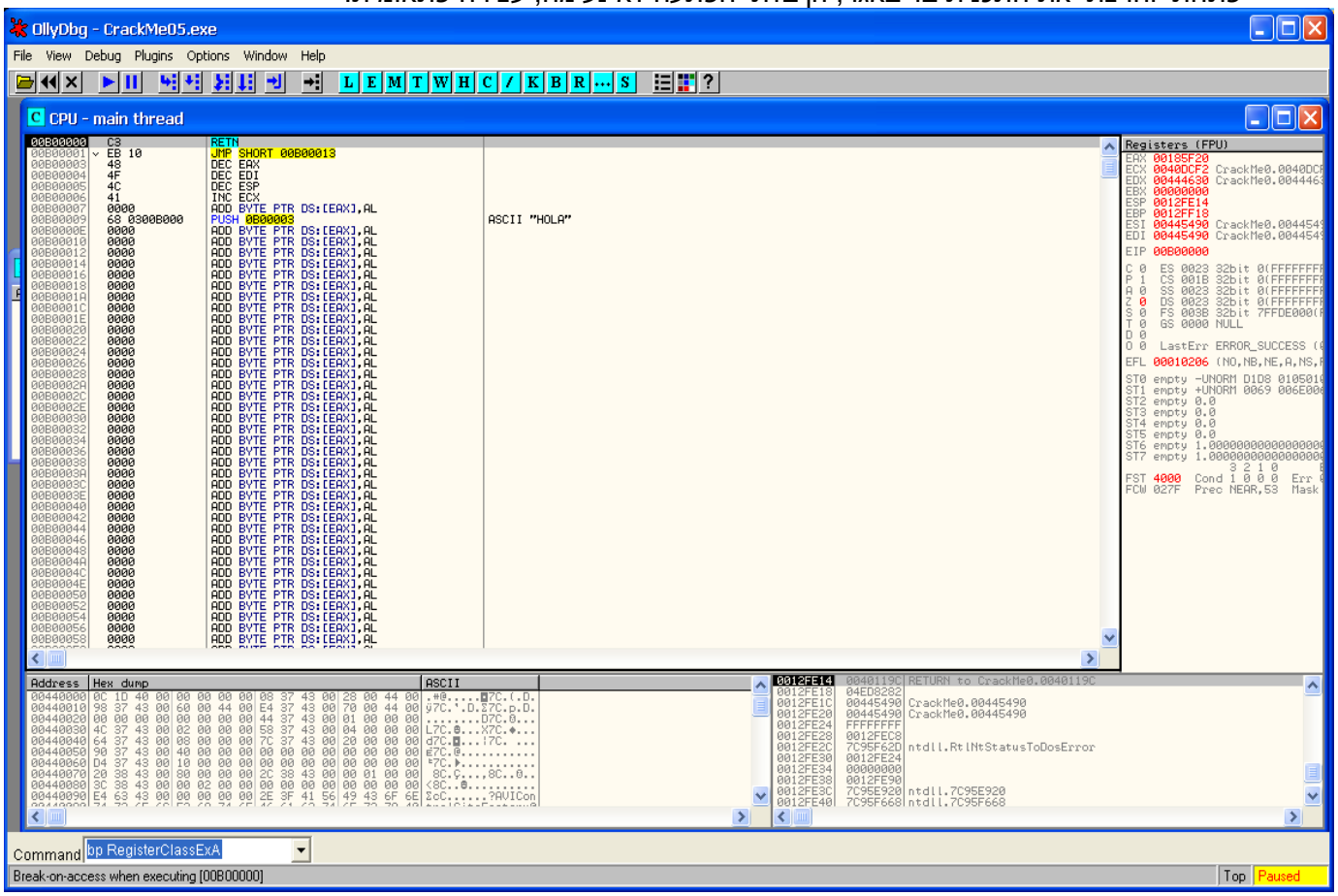


לאחר הכנסת סריאל אקראי נפתח MessageBox שמכיל את הטקסט 'Invalid Key', מכך אנחנו יודעים שיש לנו עם מה לעבוד.

ובכן, נפתח את המטרה שלנו בדיבאגר וננסה להריץ רגיל, רק כדי לראות אם אין שום הפתעות בפנים.

[יש לזכור כי אני בא מנקודת הנחה שהדיבאגר שלכם נקי לגמרי, ללא כל Plug-ins מופעלים או אנטי-אנטי דיבאגינג.]

פתחתי והרצתי את התכנית בדיבאגר, וקיבלתי הפתעה לא נעימה, עצירה פתאומית:



למטה ניתן לראות את הסיבה לעצירה –"Break-on-access when executing [00B00000]"

ובכתובת 0x00B00009 ניתן לראות ברכה מכותב הקראקמי, Hola גם לך.

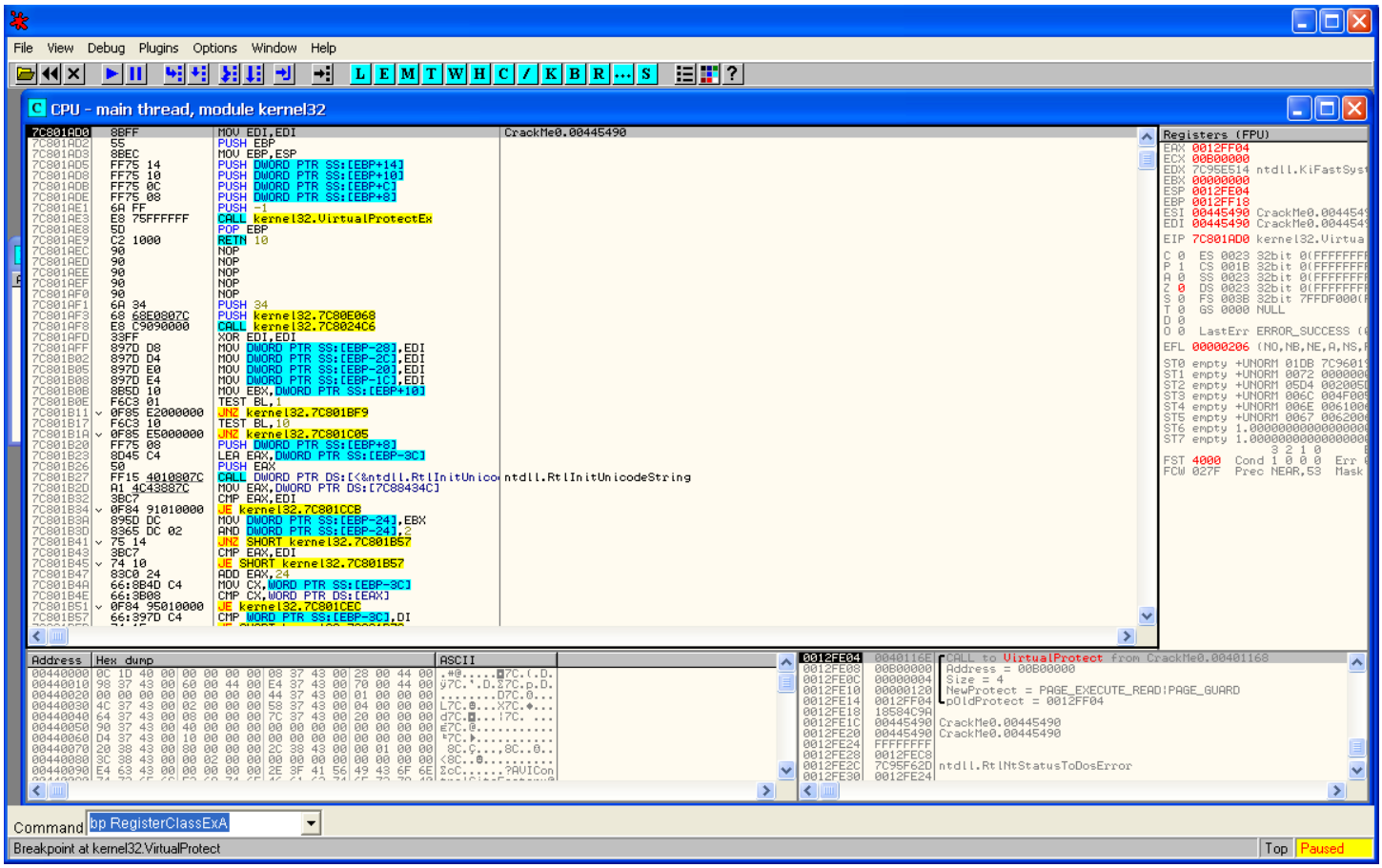
אם ננסה להמשיך ולרוץ – התכנית תסגר.

כאן אפנה אתכם למאמר נוסף שלי על Anti-Anti Debugging שפורסם בגליון הרביעי של המגזין, בו אני מסביר על שימוש באנטי דיבאגינג שמשתמש בפונקציית ה-Memory Breakpoints של הדיבאגר.

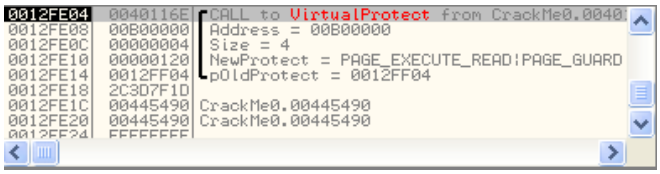
לפי העצירה Break-On-Access, ניתן לדעת כי הונח פה Memory Breakpoint, או יותר נכון במקרה הזה: Memory breakpoint מדומה.

כפי שהסברתי במאמר שלי על אנטי-אנטי דיבאגינג, Memory Breakpoint הוא פשוט שינוי הגנת דף הזיכרון ל-PAGE_GUARD. על מנת לשנות את הגנת הדף יש לקרוא לפונקציה VirtualProtect, אז בואו נחפש את הקריאה ☺.

נפתח מחדש את התכנית בדיבאגר ונשים Breakpoint (Software) על הפונקציה VirtualProtect, ונריץ. ו...עצרנו:



כפי שחשדנו ניתן לראות למטה שהרשאות הדף המיושמים על הכתובת 0x00B00000 בקריאה הזאת מכילות בין השאר את הרשאת ה-PAGE_GUARD.



עכשיו כשאנחנו בטוחים, נפעל כמו שהצעתי במאמר על Anti-Anti Debugging, נשנה את ההוראה שגרמה לעצירה, RETN, להוראה אחרת שתגרום לחריגה. במקרה שלי בחרתי בהוראה WRMSR - אבל זה שרירותי לגמרי.

[במקום לעשות זאת כל פעם ידנית, ניתן להשתמש בפלאגין Phant0m שמציע אנטי-אנטי דיבאגינג לטכניקה הנ"ל.]

נמשיך להריץ, ולמרבה ההפתעה, החלון נפתח - כנראה שעקפנו את כל האנטי-דיבאגינג ☺.

חקירת התכנית עצמה

לאחר שעקפנו את כל הבדיקות, נוכל להתרכז במטרה האמיתית שלנו בקראקמי הזה- למצוא את הסריאל.

לפי החקירה הראשונית של התכנית ללא דיבאגר, אנו יודעים כי בהכנסת סריאל שגוי, אנחנו מקבלים MessageBox, או הודעת "Bad Boy".

אנו מעוניינים לדעת איך הגענו ל-Bad Boy, מהו הגורם שהביא אותנו אליו. מכיוון שכך נשים Breakpoint בפונקציה MessageBoxA. לאחר מכן נכניס סריאל אקראי כלשהו ונלחץ OK.

למרבה הצער, הודעת ה-Bad Boy מופיעה, אך אין עצירה של הדיבאגר לפני! מה יש לעשות במקרה הזה? אסביר לכם טכניקה מאוד נפוצה בקרב תכניות כאלו עם הודעות Bad Boy:

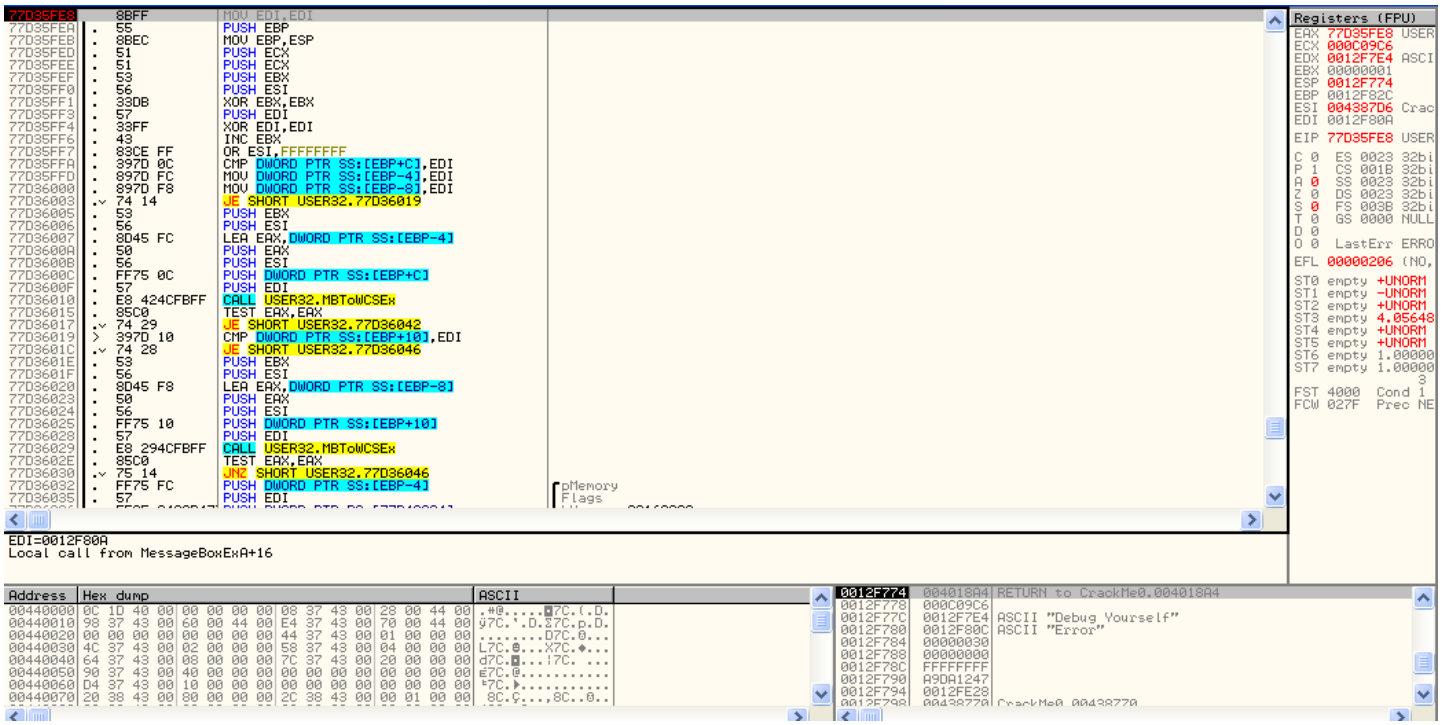
רבות מפונקציות ה-API של ווינדוס הן בעצם מעטפת של פונקציה המרחיבה אותם. למשל, הפונקציה CreateWindowA פשוט קוראת לפונקציה CreateWindowExA, "עוטפת אותה".

אותו הדבר בדיוק קורה בפונקציה MessageBoxA שהיא מעטפת של הפונקציה MessageBoxExA, שהיא גם מעטפת- מעטפת של הפונקציה MessageBoxTimeoutA, שהיא (גם!) מעטפת של הפונקציה MessageBoxTimeoutW (לא כל כך מבלבל אה? תוכלו לבדוק את זה בדיסאסמבלר).

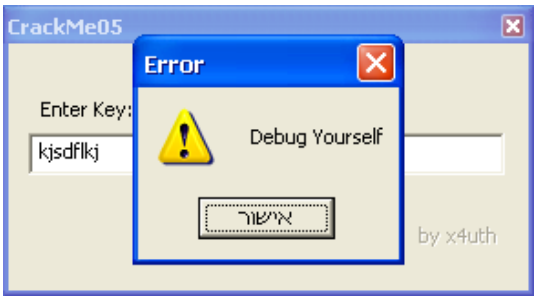
אז איך זה קשור למקרה שלנו? במקרה והתכנית הייתה משתמשת בקריאה ישירות ל-MessageBoxA היה לנו פשוט מאוד לגלות זאת, הפעולה האינסטינקטיבית שלנו בעת הופעת MessageBox כזו היא לשים Breakpoint על הפונקציה MessagBoxA.

מה שהתכנית עושה זה פשוט לקרוא לפונקציות פנימיות יותר בתוך שלבי המעטפות, כך שיהיה יותר קשה לריוורסר למצוא את הקריאה.

לאחר שניסיתי לשים Breakpoint על MessageBoxExA – ושוב לא עצר, חשבתי לרדת נמוך יותר בשלבים, ושמתי אותו על MessageBoxTimeoutA – זה מה שקיבלתי:



שהוביל להופעת ההודעה:



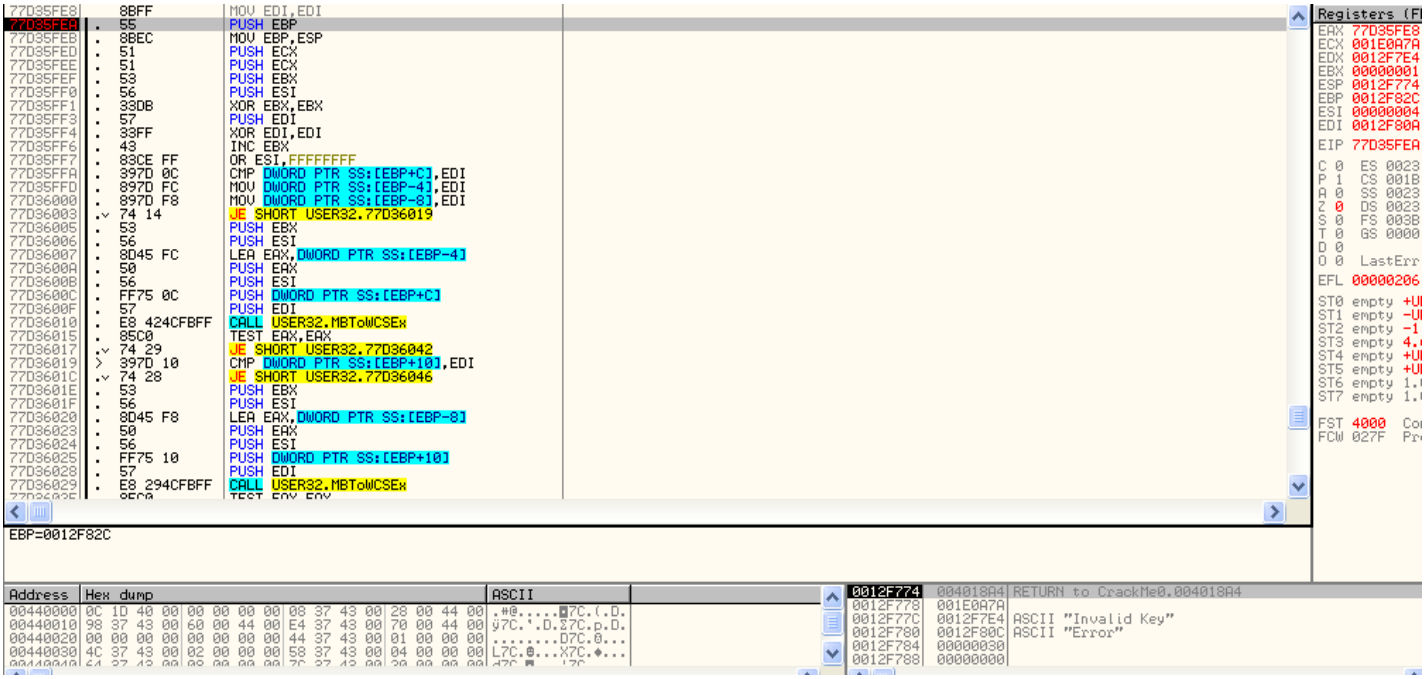
למרות הטקסט הלא מובן, אפשר להבין שהם זיהו את ה-Breakpoint שלנו..יש פה שימוש באנטי-דיבאגינג!

כעת אציג לכם עוד טכניקה נפוצה שמשתמשים בה:

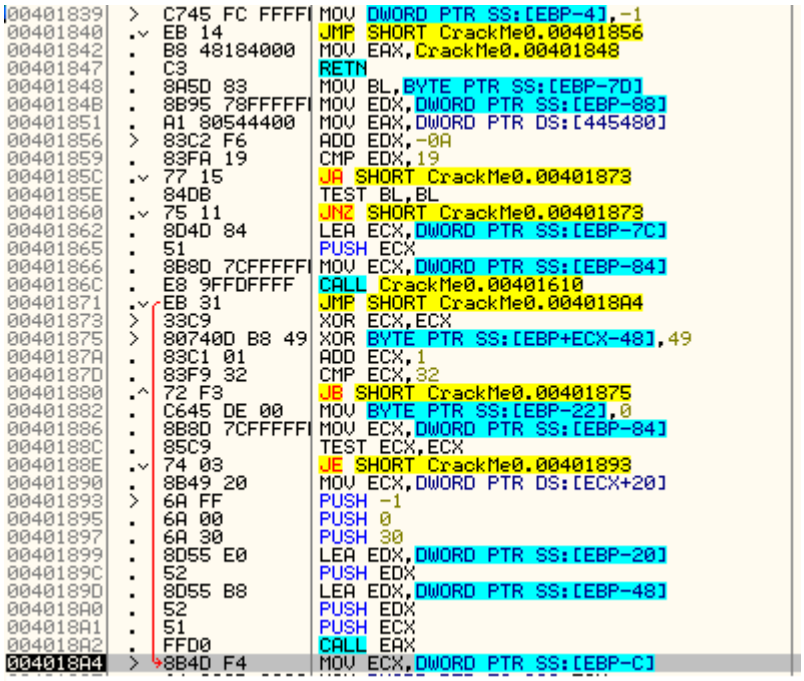
על מנת למנוע שימוש ב-Software Breakpoints, כפי שהסברתי במאמר על Anti-Anti Debugging, התכנית מחפשת את הבתים שמייצגים את האופקוד של ה-Software Breakpoint - **INT3**.

כנראה שהתכנית זיהתה את ה-Breakpoint ששמנו בתחילת הפונקציה.

לאחר ניסיון נוסף הבנתי שהתכנית בודקת רק את הבית הראשון של הפונקציה לנוכחות Breakpoint, לכן נוכל לשים את ה-Breakpoint שלנו הוראה אחת אחריו, ואנחנו מסודרים.



מפה נצעד ל-RETN כדי לראות מי קרא לפונקציה, ונגיע למקום הקריאה:



ניתן לראות בקוד שהקריאה ל-Bad Boy לא תתבצע אם הקפיצה בכתובת 0x00401860 לא תתבצע, **וגם** הקפיצה ב-0x0040185C לא תתבצע. ראשית נראה מהו הגורם לקפיצה בכתובת 0x0040185C, נשים Breakpoint כמה הוראות לפני הקפיצה (בחרתי בכתובת 0x00401856), ונכניס מחדש סריאל.

לאחר כמה ניסיונות הבנתי כי מדובר באורכו של הסריאל, לכן אם הסריאל גדול מ-35 (0x19+0x0A), הוא נחשב כלא תקין.

אך- גם אם הסריאל קטן מ-0x0A (10) הוא לא יהיה תקין (בשל גלישת גבולות, Overflow).

כעת נעבור לתנאי השני שתלוי בערך של האוגר BL, בכתובת 0x401860, ולשם כך נצטרך ללכת יותר אחורה.

עלינו לראות מה משפיע על ערכו של BL- סימנתי באדום וכחול את ההוראות המשפיעות.

בשורה שמסומנת באדום הערך של המשתנה אשר מאוחסן בכתובת EBP-7D מושם ב-BL, ערכו של משתנה זה תלוי לחלוטין בהוראה שמסומנת בכחול כפי שניתן לראות.

לאחר שהכנסתי סריאל מחדש ועצרתי בכתובת 0x004017E0, ראיתי כי מדובר באותיות של הסריאל, תירגמתי את הקוד הנ"ל לפסאודו-סי כדי להקל על ההבנה:

```
while (i < strlen(Serial))
{
      if (Serial[i] ^  Serial[i+1] == 0)
            BL = 1;
      Serial[i] -= 1;
      if (!(Serial[i] < 0x4F && Serial[i] > 0x40 )
            BL = 1;
      i++;
}
```

הפעולה הראשונה שמתבצעת היא פעולת XOR לוגי בין אות אחת לאות הבאה. כידוע, בפעולה לוגית XOR התוצאה תהיה אפס אך ורק כאשר שני המשתנים יהיו באותו הערך.

יש פה שני תנאים:

1. כל אות בסריאל צריכה להיות שונה מהאות הקודמת לה.
2. כל אות בסריאל צריכה להיות בין הערכים 0x50 – 0x41 (תחשבו למה הגדלתי את הערכים ב-אחד), שהם הייצוג המספרי של התווים 'P'-'A'.

מכך למדנו שהסריאל שלנו צריך לעמוד בשלושת התנאים:

1. אורך הסריאל חייב להיות בטווח 10-35 אותיות.
2. כל אות בסריאל שונה מהאות הקודמת לה.
3. כל אות בסריאל חייבת להיות בטווח התווים 'P'-'A'. (שימו לב כי אלו אותיות ב-Uppercase)

נקבע Breakpoint במקום ווידוי תקינות הסריאל, נכניס סריאל תקין – ונעצור.

```
00401856  >  83C2 F6      ADD EDX,-0A
00401859  .  83FA 19      CMP EDX,19
0040185C  .v 77 15        JA SHORT CrackMe0.00401873
0040185E  .  84DB         TEST BL,BL
00401860  .v 75 11        JNZ SHORT CrackMe0.00401873
00401862  .  8D4D 84      LEA ECX,DWORD PTR SS:[EBP-7C]
00401865  .  51           PUSH ECX
00401866  .  8B8D 7CFFFFFF MOV ECX,DWORD PTR SS:[EBP-84]
0040186C  .  E8 9FFDFFFF  CALL CrackMe0.00401610
```

לשמחתנו עברנו את הבדיקות! ☺

במידה ועברנו את הבדיקות, מתבצעת קריאה לפונקציה הנמצאת בכתובת 0x00401610 עם הסריאל כפרמטר – ולאחריה קפיצה לאותה הכתובת עם הקריאה ל-MessageBoxTimeoutA.

לכן, נוכל להסיק שהרוטינה לבדיקת נכונות הסריאל נמצאת בפונקציה שב-0x00401610.

עכשיו מתחילים את הכיף האמיתי

אם נצעד בקוד עם הסריאל הנוכחי שהכנסנו, נגיע ל-0x004016EE – שם ידחף הערך 0x00 למחסנית ותקרא הפונקציה שבאוגר EAX, ויופיע לנו "Invalid Key", ה-Bad Boy שלנו.

ניתן לראות בקוד קפיצות אחרות לכתובת 0x004016EA שקוראת לאותה הפונקציה ב-EAX אך דוחפת את הערך0x01, אם נגרום לזרימת הקוד להגיע לשם בצורה ידנית, נראה שבאמת במקרה הזה יופיע ה-Good Boy שלנו!

כעת, כשאנו יודעים שהמטרה שלנו היא להגיע לקפיצה ל-0x004016EA, נוכל להתחיל לעבוד.

בתחילת הפונקציה שני האוגרים EBX ו-EDI מאותחלים עם ערכים קבועים, דבר חשוב לזכור כשחוקרים תכניות הוא לשים לב לערכים קבועים, כי הם לא נמצאים שם סתם. אז:

EBX = 0x00442D1

EDI = 0x00442A5F

לאחר מכן ישנן שתי קריאות שלא עושות הרבה אז לא נתייחס אליהן. בקוד המתחיל בכתובת 0x00401672 מתחילה לולאה כלשהי, לפי הסימון של Olly. (לפני תחילת הלולאה, הסריאל שניתן כפרמטר, הושם ב-EBP).

כדי להקל עליכם, תרגמתי את הלולאה לפסאודו-סי:

```
while (true)
{
      if (EBX && EDI == 0x00442A23)
            goto GoodBoy;
      if (*SerialPointer == 0x00) //עברנו על כל הסריאל ולא צדקנו
            goto BadBoy;
      if ( (*EBX | 0x04 == 4) || (*EDI | 0x04 == 4) )
            goto BadBoy;

      switch ((*SerialPointer & 0x0F) & 0x03)
      {
            case 0:
                  EBX -= 1;
            case 1:
                  EBX -= 0x1A;
            case 3:
                  EBX += 1;
            case 2:
                  EBX += 0x1A;
      }
      switch ((*SerialPointer & 0x0F) >> 2)
      {
            case 0:
                  EDI -= 1;
            case 1:
                  EDI -= 0x1A;
            case 3:
                  EDI += 1;
            case 2:
                  EDI += 0x1A;
      }
      SerialPointer++; //ההוראה הזאת מתבצעת בכל לולאה
}
```

כעת כשהקוד **הרבה** יותר קריא, נוכל להתחיל לנתח אותו.

ניתן לראות בתחילת הלולאה כי ערכיהם של האוגרים EBX ו-EDI משווים לערך 0x00442A23- במקרה **ששניהם** שווים, יש לנו Good Boy.

עכשיו, נראה מה משפיע על שני האוגרים האלה.

בהמשך רואים שערכיהם של האוגרים מושפעים על ידי ה-switch שמושפע על ידי הסריאל שלנו, כל אות בסריאל משפיעה על ערכיהם של EBX ו-EDI בצורה שונה, היות ויש לנו שני switch שונים, למשל, ניתן לראות שאם נכניס ל-switch את האות 'B' (0x41 = 0x01 – 0x42, זוכרים אז שהתכנית מחסירה לכל אות בסריאל אחד?), EBX יקטן ב-0x1A, אך EDI יקטן ב-1.

אפשר להבין מכך, שנצטרך להתחשב בכל אות ביחס לערך של EDI **וגם** ביחס לערך של EBX – כי היא משפיע על שניהם בצורה שונה.

עכשיו כשאנחנו מבינים את זה, כל מה שנצטרך לעשות זה להגיע עם שני האוגרים לערך המיוחל, 0x00442A23, באמצעות אותיות הסריאל.

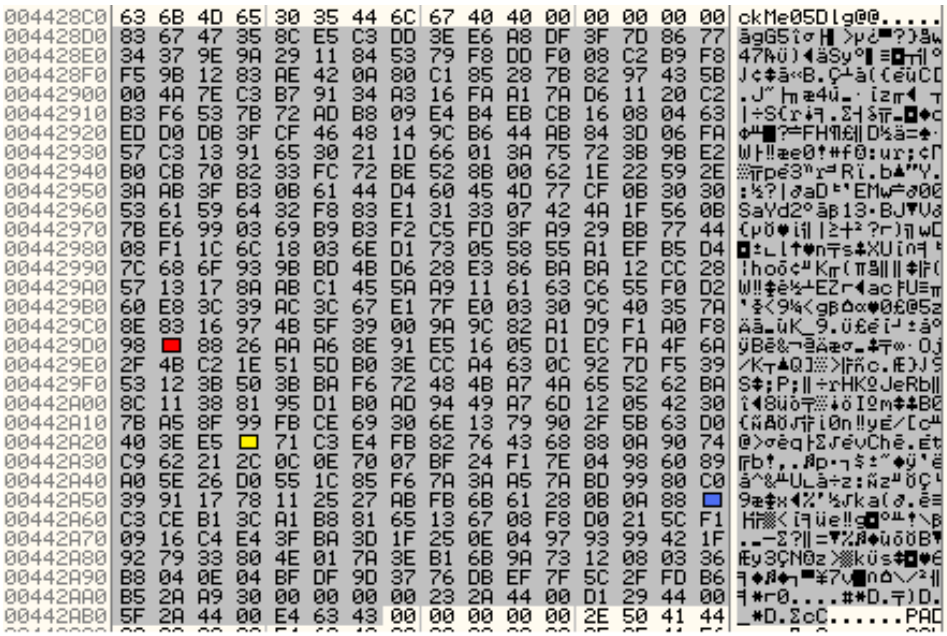
אך, זה לא הכל! למעלה בקוד ישנה פעולה לוגית לערך שנמצא בכתובת הנוכחית של EBX ו-EDI:

```
if ((*EBX | 0x04 == 4) || (*EDI | 0x04 == 4))
goto BadBoy;
```

זאת אומרת **שאם אפילו כתובת אחת שבאוגרים מכילה ערך שהביט ה-3 שלו דלוק** (מתחילים את הספירה מ-0) **יש לנו Bad Boy!**

ניתן להבין מכך, שיש כתובות מסוימות – שאסור לנו "לדרוך" עליהן, עובדה זו מגבילה אותנו מאוד. נצטרך תמונה של הזיכרון מסביב לכתובת שאנחנו צריכים להגיע אליה , 0x00442A23 ,ולסמן את הכתובות שאסור לנו לדרוך עליהן, לפי הערכים שהן מכילות.

איך נוכל בכלל לדעת מה גודל התמונה שעלינו לקחת? זה יכול להיות אינסופי!



- הכתובת שמסומנת ב**צהוב** היא הכתובת שעלינו להגיע אליה, 0x00442A23.
- הכתובת שמסומנת ב**אדום** היא כתובתה של EBX.
- הכתובת שמסומנת ב**כחול** היא כתובתה של EDI.

כיוון שאנחנו לא יכולים לדעת את גודל התמונה של הזיכרון שעלינו לקחת כדי לכסות את **כל** המקומות שאסור לנו לדרוך עליהם, נצטרך להסתכל ולחשוב קצת.

ניתן לראות בלוק גדול של בתים אקראיים, שמתחיל ומסתיים ב-5 בתי NULL, ואחריו יש כבר אותיות, שלא נראות כמו חלק ממפת הזיכרון.

כמובן, שאי אפשר להסיק מכך הרבה, אך נקרא לזה "ניחוש מושכל", כי זה בעצם כל מה שעושים ב-Reverse Engineering, ניחושים מושכלים.

עכשיו, אנחנו יודעים שעלינו לפלס את הדרך שלנו דרך מפת הזיכרון לכתובת 0x00442A23, משני נקודות מוצא שונות (הכתובות השונות שב-EBX ו-EDI), באמצעות אותיות הסריאל, ויש כתובות מסוימות שאסור לנו לדרוך עליהם.

זה לא במקרה..מזכיר לכם משהו?

דרך שעלינו ללכת בה על מנת להגיע למקום מסוים, אך יש מקומות שאסור לנו לדרוך עליהם. יש לנו אפשרות להזיז את עצמנו מקום אחד ימינה או מקום אחד שמאלה (הוספה והחסרה של האוגרים), או הזזה של עצמו ב-0x1A כתובות. אם ניקח את המפה שלנו ונחלק אותה לשורות של 0x1A (26) בתים בכל שורה, נוכל אפילו לקרוא להזזה של 0x1A כתובות "למעלה" ו-"למטה"!

לי זה מזכיר מאוד מבוך!

אם נחלק את המפה כפי שאמרתי, נוכל פשוט לכוון את עצמנו באמצעות הסריאל אל המטרה שלנו משתי נקודות המוצא, או "השחקנים", בדרך מתואמת, כשכל אות מהווה לגבי כל שחקן תנועה אחת – ימינה, שמאלה, למעלה או למטה!

בכדי שנוכל לקחת את הרעיון הזה של המבוך וליישם אותו למציאת הסריאל, נצטרך קודם לקחת את מפת הזיכרון שלנו ולחלק אותה לשורות של 26 (0x1A), ואז לסמן את כל הכתובות שאסור לנו לדרוך עליהן, נוכל לקרוא להן "קירות", בדיוק כמו במבוך!

על מנת לבצע את המהלך הזה, כתבתי תוכנה ב-Python:

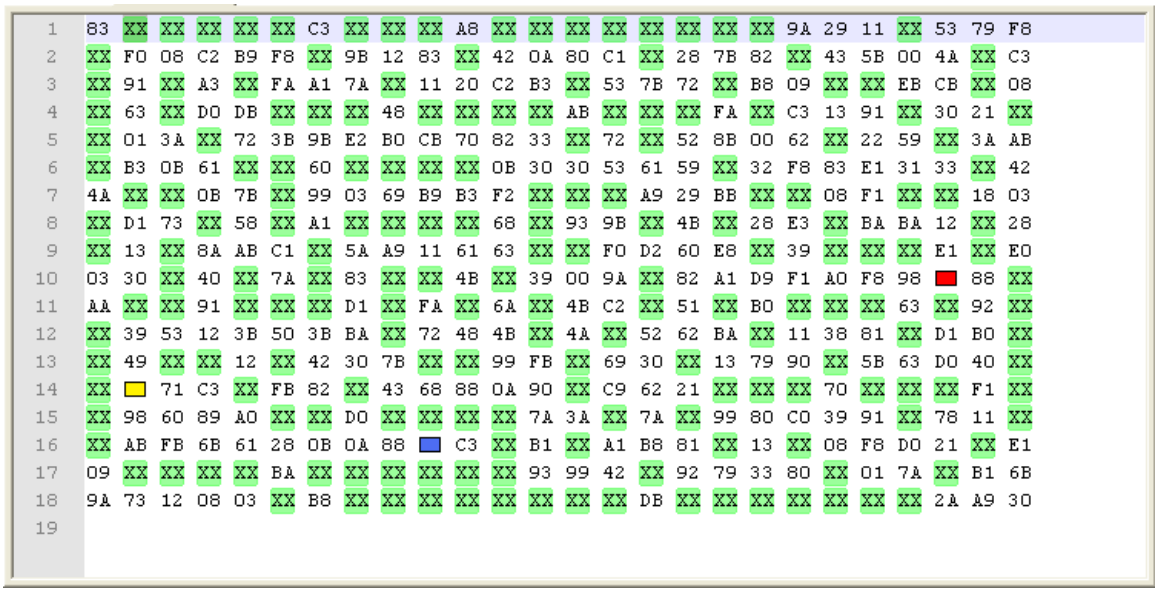
```
File = open("maze.mem",'r');
Output = open("Output.txt",'a');
Rows = File.read().split('\n');

for row in range(0,len(Rows)):
      Bytes = Rows[row].split('\x20');
      for Byte in range (0, len(Bytes)):
            if (int(Bytes[Byte], 16) & 4 == 4):
                  Bytes[Byte] = 'XX';

            Output.write(Bytes[Byte] + '\x20');
Output.write('\n');
```

[maze.mem היא מפת הזיכרון שלנו, שחילקתי לשורות של 26.]

והתוצאה:



```
1  83 XX XX XX XX XX C3 XX XX XX A8 XX XX XX XX XX XX XX XX 9A 29 11 XX 53 79 F8
2  XX F0 08 C2 B9 F8 XX 9B 12 83 XX 42 0A 80 C1 XX 28 7B 82 XX 43 5B 00 4A XX C3
3  XX 91 XX A3 XX FA A1 7A XX 11 20 C2 B3 XX 53 7B 72 XX B8 09 XX XX EB CB XX 08
4  XX 63 XX D0 DB XX XX XX 48 XX XX XX XX AB XX XX XX FA XX C3 13 91 XX 30 21 XX
5  XX 01 3A XX 72 3B 9B E2 B0 CB 70 82 33 XX 72 XX 52 8B 00 62 XX 22 59 XX 3A AB
6  XX B3 0B 61 XX XX 60 XX XX XX XX 0B 30 30 53 61 59 XX 32 F8 83 E1 31 33 XX 42
7  4A XX XX 0B 7B XX 99 03 69 B9 B3 F2 XX XX XX A9 29 BB XX XX 08 F1 XX XX 18 03
8  XX D1 73 XX 58 XX A1 XX XX XX XX 68 XX 93 9B XX 4B XX 28 E3 XX BA BA 12 XX 28
9  XX 13 XX 8A AB C1 XX 5A A9 11 61 63 XX XX F0 D2 60 E8 XX 39 XX XX XX E1 XX E0
10 03 30 XX 40 XX 7A XX 83 XX XX 4B XX 39 00 9A XX 82 A1 D9 F1 A0 F8 98 88 XX
11 AA XX XX 91 XX XX XX D1 XX FA XX 6A XX 4B C2 XX 51 XX B0 XX XX XX 63 XX 92 XX
12 XX 39 53 12 3B 50 3B BA XX 72 48 4B XX 4A XX 52 62 BA XX 11 38 81 XX D1 B0 XX
13 XX 49 XX XX 12 XX 42 30 7B XX XX 99 FB XX 69 30 XX 13 79 90 XX 5B 63 D0 40 XX
14 XX    71 C3 XX FB 82 XX 43 68 88 0A 90 XX C9 62 21 XX XX XX 70 XX XX XX F1 XX
15 XX 98 60 89 A0 XX XX D0 XX XX XX XX 7A 3A XX 7A XX 99 80 C0 39 91 XX 78 11 XX
16 XX AB FB 6B 61 28 0B 0A 88    C3 XX B1 XX A1 B8 81 XX 13 XX 08 F8 D0 21 XX E1
17 09 XX XX XX XX BA XX XX XX XX XX XX 93 99 42 XX 92 79 33 80 XX 01 7A XX B1 6B
18 9A 73 12 08 03 XX B8 XX XX XX XX XX XX XX XX DB XX XX XX XX XX XX XX 2A A9 30
19
```

[הדגשתי בירוק את הקירות בעזרת Notepad++.]

סימנתי את שני השחקנים, ה**כחול** וה**אדום**, ואת נקודת היציאה בצבע **צהוב**.

עכשיו, כשיש לנו את המבוך בצורה מאוד ויזואלית וברורה, נוכל לבדוק איזה אותיות משפיעות באיזו צורה – ואז נוכל פשוט לבנות לנו מסלול מתאים.

כפי שאתם זוכרים, כשבדקנו את תקינות הסריאל – ראינו כי האותיות האפשרויות לסריאל נעות בין 'A'-ל-'P', כל מה שאנו צריכים לעשות בשביל לדעת, מה המהלך שכל אות מהווה בשביל כל שחקן – זה לקחת את ערך ה-ASCII שלה ולהציב ב-switch של כל שחקן.

לאחר הצבה של כל אות הגעתי לאפשרויות הבאה:

שחקן אדום:
שמאלה - 'A','E','I','M'.
למעלה - 'B','F','J','N'.
למטה – 'C','G','K','O' .
ימינה - 'D','H','L','P'.

שחקן כחול:
שמאלה - 'A','B','C','D'.
למעלה - 'E','F','G','H'.
למטה - 'I','J','K','L'.
ימינה - 'M','N','O','P'.

אם תסתכלו על המהלכים האפשריים של כל שחקן והאותיות שמייצגות כל מהלך, תוכלו לראות תבנית מסוימת.

ניתן להשתמש ב-4 אותיות שונות כדי לייצג מהלך של כל שחקן, ואם תסתכלו טוב, אצל השחקן ה**אדום**- האותיות 'A','E','I','M', גורמות לשחקן ללכת שמאלה, אך, אותן האותיות בדיוק אצל השחקן ה**כחול**, יכולות להזיז אותו לכל כיוון. למשל, ניתן להשתמש ב-'E' בכדי לזוז שמאלה ב**אדום** אך למעלה ב**כחול**.

זאת אומרת, שכשאנחנו מבצעים את התנועה שמאלה עם השחקן ה**אדום**, בגלל שאותה האות משפיעה גם על השחקן השני, נוכל לבצע עם השחקן ה**כחול** כל תנועה שאנחנו רוצים בו זמנית.

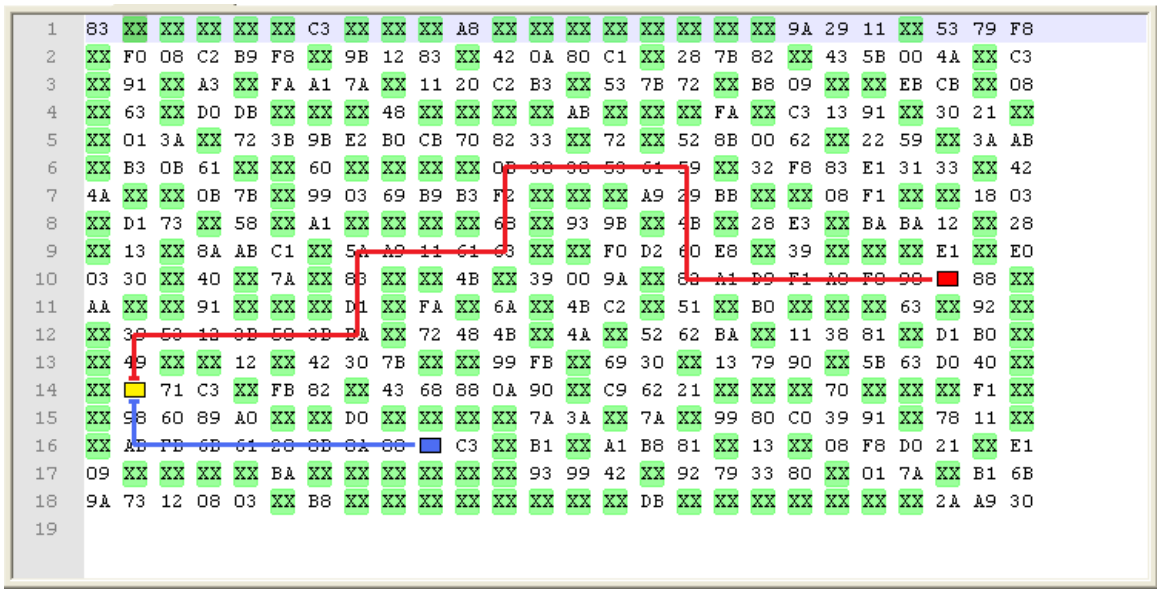
אותו הכלל תקף עם כל התנועות האחרות, כך ש-בעיית התיאום כבר לא נראית כל כך מסובכת.

נחזור לחוקים שלנו:

1. **אורך הסריאל חייב להיות בטווח 10-35 אותיות.**
2. **כל אות בסריאל שונה מהאות הקודמת לה.**
3. **כל אות בסריאל חייבת להיות בטווח התווים 'A'-'P'. (שימו לב שזה אותיות ב-Uppercase)**

אנחנו יכולים להתעלם מהחוק השלישי כי כבר טיפלנו בו, כך שנשארו לנו שני חוקים:

1. **אורך הסריאל חייב להיות בטווח 10-35 אותיות** - משמעות הדבר שעלינו להשלים את המבוך ב-35 צעדים לכל היותר.
2. **כל אות בסריאל שונה מהאות הקודמת לה**. אם תבדקו, תראו שזה יוצר לנו בעיה לא קטנה:



אם ניקח את הדרכים המובנות מאליהן, שסימנתי ב**אדום** וב**כחול** – נוכל לסכם את המהלכים ככה:

אדום:

שמאלה 7x, למעלה 4x, שמאלה 5x, למטה 3x, שמאלה 4x, למטה 3x, שמאלה 6x, למטה 2x.

וכחול:

שמאלה 8x, למעלה 2x.

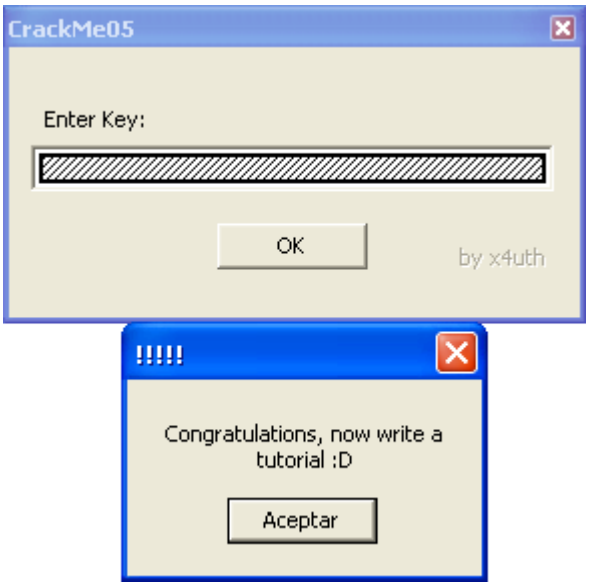
ננסה להמיר זאת לאותיות, תיווצר לנו התנגשות, כי בתחילה נרצה בו זמנית שמאלה בשחקן ה**אדום** וגם ב**כחול**, **ורק אות אחת** מהווה שמאלה **גם בשחקן ה**אדום **וגם בשחקן ה**כחול, **'A'**.

לא נוכל לבצע את הדרך הנ"ל מבלי לרשום אות שונה מהקודמת לה, דבר השובר לנו את חוקי המשחק, עלינו לחשוב על דרך חלופית שלא גורמת לנו לחזור על אותיות פעמיים ברצף.

<u>אז זאת המשימה שאני נותן לכם, חברים,</u>

האתגר שלי אליכם הוא למצוא סריאל שעובד בהתאם לכל החוקים שיוביל אותכם ל-Good Boy.

את הסריאל(ים) יש לפרסם כתגובה בפוסט של פרסום הגיליון.. **<u>בהצלחה!</u>** ☺


CrackMe05
Enter Key:
OK
by x4uth
!!!!!
Congratulations, now write a tutorial :D
Aceptar