

## 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.

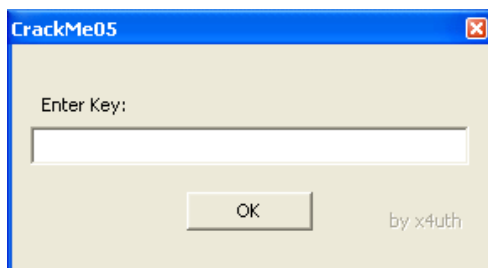
לאחר יום אחד הצלחתי לפתור את הקראקמי, בצורה יפה מאוד לדעתי, אז החלטתי לכתוב עליו. אני חושב שתוכלו ללמוד הרבה מהמאמר הזה על דרך החשיבה בחקירת קראקמיים.

ניתן להשיג את הקראקמי מהכתובת הבאה:

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

### נתחיל בחקירה

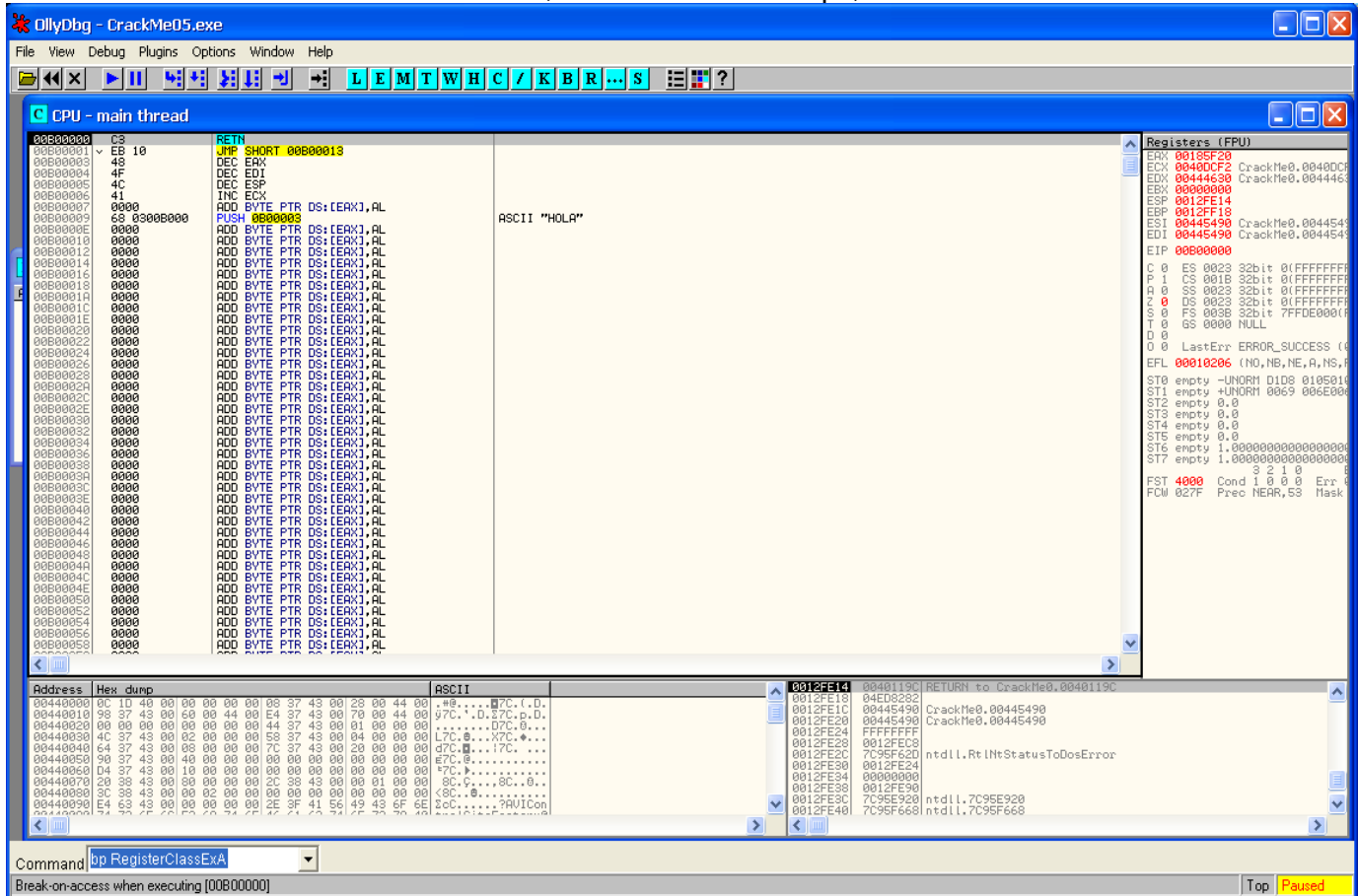
בפתיחה ראשונית של התכנית ללא דיבאגר, ניתן לראות חלון קטן שמבקש סריאל:



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 ונסה 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 דיבאגיינג.

פנתתי 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.

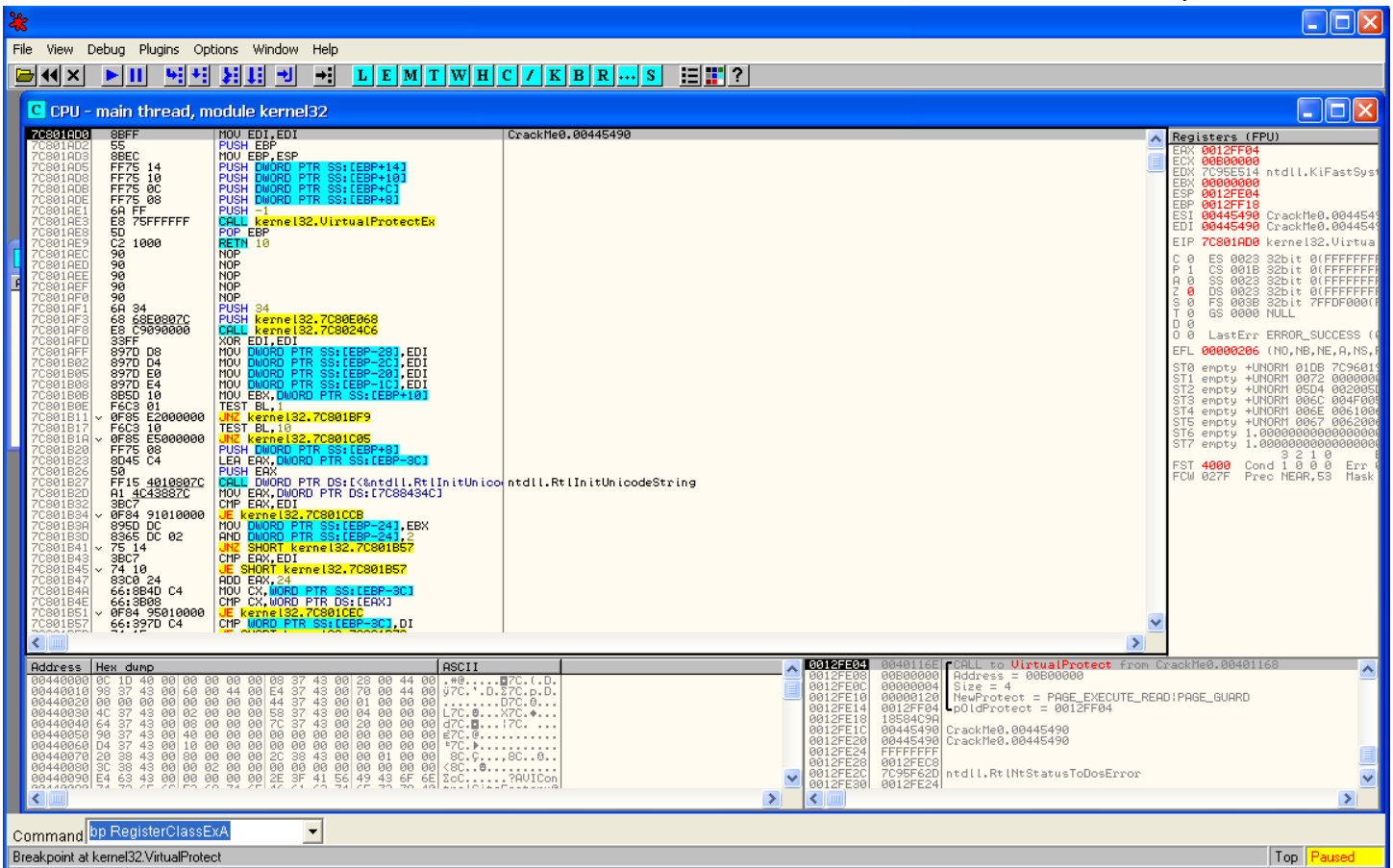


כאן אפנה אתכם למאמר נוסף שלי על **Anti-Anti Debugging** שפורסם בגליון הרביעי של המגזין, בו אני מסביר על שימוש באנטי דיבאגינג שמשמש בפונקציית ה-Memory Breakpoints של הדיבאגר.

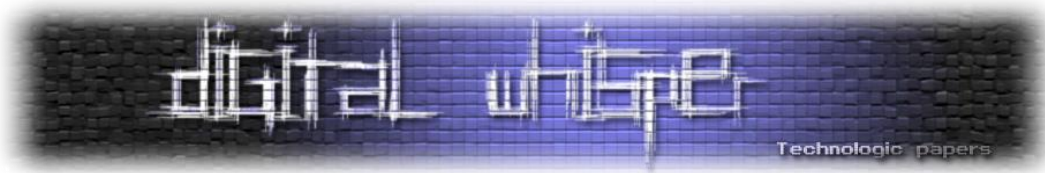
לפי העצירה Break-On-Access, ניתן לדעת כי הונח פה Memory Breakpoint, או יותר נכון במקרה הזה: Memory breakpoint מדומה.

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... ועצרונו:



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



```

0012FE04 0040116E CALL to VirtualProtect from CrackMe0.0040
0012FE08 00B00000 Address = 00B00000
0012FE0C 00000004 Size = 4
0012FE10 00000120 NewProtect = PAGE_EXECUTE_READ|PAGE_GUARD
0012FE14 0012FF04 OldProtect = 0012FF04
0012FE18 2C3D7F1D
0012FE1C 00445490 CrackMe0.00445490
0012FE20 00445490 CrackMe0.00445490
0012FF74 FFFFFFFF

```

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".

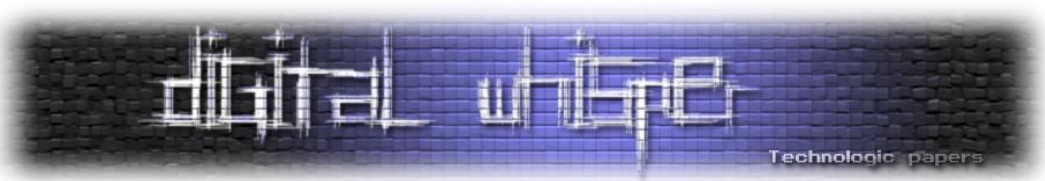
אנו מעוניינים לדעת איך הגענו ל-Bad Boy, מהו הגורם שהביא אותנו אליו. מכיוון שכך נשים Breakpoint בפונקציה MessageBoxA. לאחר מכן נכניס סריאל אקראי כלשהו ונלחץ OK.

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 MessageBoxA.

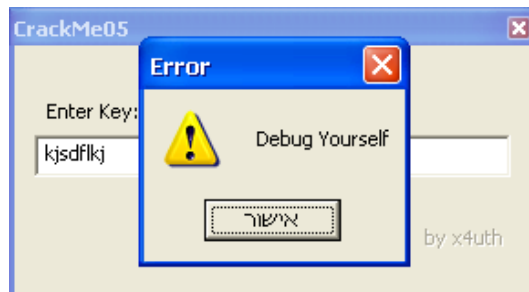


מה שהתכנית עושה זה פשוט לקרוא לפונקציות פנימיות יותר בתוך שלבי המעטפות, כך שיהיה יותר קשה לריוורסר למצוא את הקריאה.

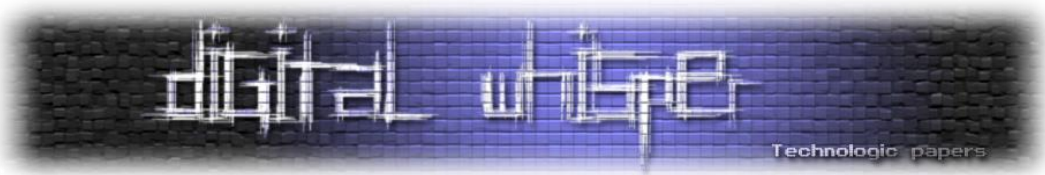
לאחר שניסיתי לשים Breakpoint על MessageBoxExA – ושוב לא עזר, חשבתי לרדת נמוך יותר בשלבים, ושמתו אותו על MessageBoxTimeoutA – זה מה שקיבלתי:

Address	Hex	dump	ASCII	
00440000	0C 1D 40 00	00 00 00 00 00 37 43 00	28 00 44 00	.#0...7C.L.D.
00440010	98 37 43 00	00 00 00 44 00 E4 37 43 00	78 00 44 00	97C.L.D.37C.p.D.
00440020	00 00 00 00	00 00 00 00 00 44 37 43 00	01 00 00 00	.....07C.B..
00440030	4C 37 43 00	02 00 00 00 00 58 37 43 00	04 00 00 00	L7C.B...X7C.♦...
00440040	64 37 43 00	08 00 00 00 00 7C 37 43 00	20 00 00 00	d7C.B...17C....
00440050	98 37 43 00	00 00 00 00 00 00 00 00 00	00 00 00 00	E7C.0.....
00440060	D4 37 43 00	10 00 00 00 00 00 00 00 00	00 00 00 00	17C.♦.....
00440070	20 38 43 00	00 00 00 00 2C 38 43 00	00 01 00 00	.8C.C...8C..0..

שהוביל להופעת הודעה:



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



כעת אציג לכם עוד טכניקה נפוצה שמתמשים בה:

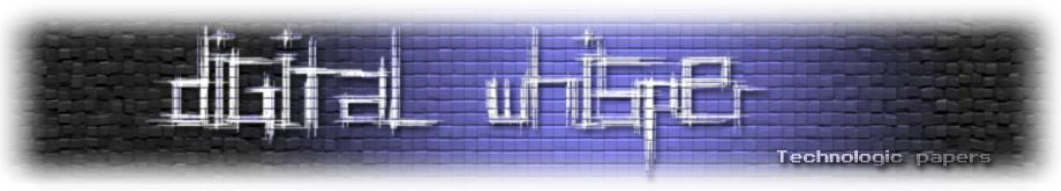
על מנת למנוע שימוש ב-Software Breakpoints, כפי שהסברתי במאמר על Anti-Anti Debugging, התכנית מחפשת את הבתים שמייצגים את האופקוד של ה-INT3-Software Breakpoint.

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.

The screenshot displays a debugger interface with three main sections:

- Assembly Window:** Shows a list of instructions starting from address 77D35FE9. Key instructions include:
  - 77D35FE9: MOV EDI, EDI
  - 77D35FEB: PUSH EBP
  - 77D35FED: MOV EBP, ESP
  - 77D35FEE: PUSH ECX
  - 77D35FEF: PUSH ECX
  - 77D35FF0: PUSH EBX
  - 77D35FF1: PUSH ESI
  - 77D35FF2: XOR EBX, EBX
  - 77D35FF3: PUSH EDI
  - 77D35FF4: XOR EDI, EDI
  - 77D35FF6: INC EBX
  - 77D35FF7: OR ESI, FFFFFFFF
  - 77D35FF8: CMP DWORD PTR SS:[EBP+C], EDI
  - 77D35FFD: MOV DWORD PTR SS:[EBP-4], EDI
  - 77D36000: MOV DWORD PTR SS:[EBP-8], EDI
  - 77D36003: JE SHORT USER32.77D36019
  - 77D36005: PUSH EBX
  - 77D36006: PUSH ESI
  - 77D36007: LEA EAX, DWORD PTR SS:[EBP-4]
  - 77D36008: PUSH EAX
  - 77D36009: PUSH ESI
  - 77D3600C: PUSH DWORD PTR SS:[EBP+C]
  - 77D3600F: PUSH EDI
  - 77D36010: CALL USER32.MBTtoMCSEX
  - 77D36015: TEST EAX, EAX
  - 77D36017: JE SHORT USER32.77D36042
  - 77D36019: CMP DWORD PTR SS:[EBP+10], EDI
  - 77D3601C: JE SHORT USER32.77D36046
  - 77D3601E: PUSH EBX
  - 77D3601F: PUSH ESI
  - 77D36020: LEA EAX, DWORD PTR SS:[EBP-8]
  - 77D36023: PUSH EAX
  - 77D36024: PUSH ESI
  - 77D36025: PUSH DWORD PTR SS:[EBP+10]
  - 77D36028: PUSH EDI
  - 77D36029: CALL USER32.MBTtoMCSEX
  - 77D3602C: TEST EAX, EAX
- Registers Window:** Shows the state of various registers. Notable values include:
  - EAX: 77D35FE8
  - ECX: 001E0A7A
  - EDX: 0012F7E4
  - EBX: 00000001
  - ESP: 0012F774
  - EBP: 0012F82C
  - ESI: 00000004
  - EDI: 0012F80A
  - EIP: 77D35FEA
- Hex Dump Window:** Shows memory data at address 00440000. The dump includes ASCII characters like ".#e....D7C.(.D." and "y7C.'.D.37C.p.D.".



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

```

00401839 > C745 FC FFFF MOV DWORD PTR SS:[EBP-4],-1
00401840 > EB 14 JMP SHORT CrackMe0.00401856
00401842 > B8 48184000 MOV EAX,CrackMe0.00401848
00401847 > C3 RETN
00401848 > 8A5D 83 MOV BL,BYTE PTR SS:[EBP-7D]
0040184B > 8B95 78FFFFFF MOV EDX,DWORD PTR SS:[EBP-83]
00401851 > A1 80544400 MOV EAX,DWORD PTR DS:[445480]
00401856 > 83C2 F6 ADD EDX,-0A
00401859 > 83FA 19 CMP EDX,19
0040185C > 77 15 JA SHORT CrackMe0.00401873
0040185E > 84DB TEST BL,BL
00401860 > 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
00401871 > EB 31 JMP SHORT CrackMe0.004018A4
00401873 > 33C9 XOR ECX,ECX
00401875 > 80740D B8 49 XOR BYTE PTR SS:[EBP+ECX-48],49
0040187A > 83C1 01 ADD ECX,1
0040187D > 83F9 32 CMP ECX,32
00401880 > 72 F3 JB SHORT CrackMe0.00401875
00401882 > C645 DE 00 MOV BYTE PTR SS:[EBP-22],0
00401886 > 8B8D 7CFFFFFF MOV ECX,DWORD PTR SS:[EBP-84]
0040188C > 85C9 TEST ECX,ECX
0040188E > 74 03 JE SHORT CrackMe0.00401893
00401890 > 8B49 20 MOV ECX,DWORD PTR DS:[ECX+20]
00401893 > 6A FF PUSH -1
00401895 > 6A 00 PUSH 0
00401897 > 6A 30 PUSH 30
00401899 > 8D55 E0 LEA EDX,DWORD PTR SS:[EBP-20]
0040189C > 52 PUSH EDX
0040189D > 8D55 B8 LEA EDX,DWORD PTR SS:[EBP-48]
004018A0 > 52 PUSH EDX
004018A1 > 51 PUSH ECX
004018A2 > FF00 CALL EAX
004018A4 > 8B4D F4 MOV ECX,DWORD PTR SS:[EBP-C]

```

ניתן לראות בקוד שהקריאה ל-Bad Boy לא תבצע אם הקפיצה בכתובת 0x00401860 לא תבצע, וגם הקפיצה ב-0x0040185C לא תבצע. 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).

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

```

004017CD . 3302 XOR EAX,EAX
004017CF . 8995 78FFFFFF MOV DWORD PTR SS:[EBP-88],EDX
004017D5 . 85F6 TEST ESI,ESI
004017D7 . 7E 3A JLE SHORT CrackMe0.00401813
004017D9 . 8DA424 000000 LEA ESP,DWORD PTR SS:[ESP]
004017E0 > 8A4415 84 MOV AL,BYTE PTR SS:[EBP+EDX-7C]
004017E4 . 8A4C15 85 MOV CL,BYTE PTR SS:[EBP+EDX-7B]
004017E8 . 32C8 XOR CL,AL
004017EA . 0F94C1 SETE CL
004017ED . 0AD9 OR BL,CL
004017EF . 2C 01 SUB AL,1
004017F1 . 884415 84 MOV BYTE PTR SS:[EBP+EDX-7C],AL
004017F5 . 3C 4F CMP AL,4F
004017F7 . 0F9FC1 SETG CL
004017FA . 3C 40 CMP AL,40
004017FC . 0F9CC0 SETL AL
004017FF . 0AC8 OR CL,AL
00401801 . 0AD9 OR BL,CL
00401803 . 83C2 01 ADD EDX,1
00401806 . 3BD6 CMP EDX,ESI
00401808 . ^ 7C D6 JL SHORT CrackMe0.004017E0
0040180A . 8995 78FFFFFF MOV DWORD PTR SS:[EBP-88],EDX
00401810 > 885D 83 MOV BYTE PTR SS:[EBP-7D],BL ←
0040181A > C745 FC 0000 MOV DWORD PTR SS:[EBP-4],0
0040181F . A1 80544400 MOV EAX,DWORD PTR DS:[445480]
00401821 . 85C0 TEST EAX,EAX
00401823 . 74 16 JE SHORT CrackMe0.00401839
00401826 . 8038 CC CMP BYTE PTR DS:[EAX],0CC
00401828 . 75 11 JNZ SHORT CrackMe0.00401839
0040182D . B9 09000000 MOV ECX,9
00401832 . BE B0874300 MOV ESI,CrackMe0.004387B0
00401835 . 8D7D B8 LEA EDI,DWORD PTR SS:[EBP-48]
00401837 . F3:A5 REP MOVS DWORD PTR ES:[EDI],DWORD PTR
00401839 > 66:A5 MOVS WORD PTR ES:[EDI],WORD PTR DS:[E
00401840 > C745 FC FFFF MOV DWORD PTR SS:[EBP-4],-1
00401842 . EB 14 JMP SHORT CrackMe0.00401856
00401844 . B8 48184000 MOV EAX,CrackMe0.00401848
00401847 . C3 RETN ←
00401848 . 8A5D 83 MOV BL,BYTE PTR SS:[EBP-7D]
0040184B . 8B95 78FFFFFF MOV EDX,DWORD PTR SS:[EBP-88]
00401851 . A1 80544400 MOV EAX,DWORD PTR DS:[445480]
00401856 > 83C2 F6 ADD EDX,-0A
    
```

עלינו לראות מה משפיע על ערכו של BL- סימנתי ב**אדום** ו**כחול** את ההוראות המשפיעות.

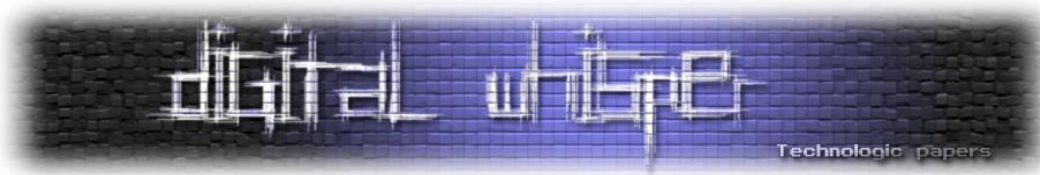
בשורה שמסומנת ב**אדום** הערך של המשתנה אשר מאוחסן בכתובת EBP-7D מושם ב-BL, ערכו של משתנה זה תלוי לחלוטין בהוראה שמסומנת ב**כחול** כפי שניתן לראות.

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 0x41 – 0x50 (תחשבו למה הגדלתי את הערכים ב-אחד), שהם הייצוג המספרי של התווים 'A'-'P'.

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 'A'-'P' (שימו לב כי אלו אותיות ב-Uppercase)

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

```

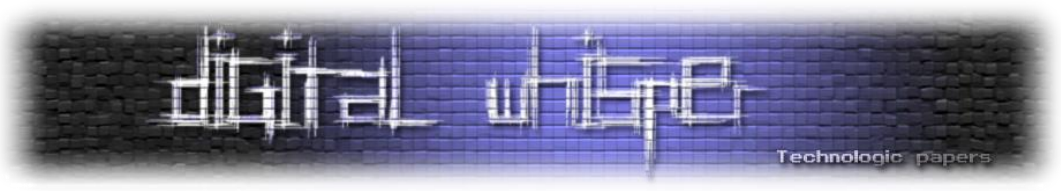
00401856 > 83C2 F6      ADD EDI,-0A
00401859 . 83FA 19      CMP EDI,19
0040185C . 77 15       JA SHORT CrackMe0.00401873
0040185E . 840B       TEST BL,BL
00401860 . 75 11       JNZ SHORT CrackMe0.00401873
00401862 . 8D4D 84     LEA ECX,DWORD PTR SS:[EBP-7C]
00401865 . 51         PUSH ECX
00401866 . 8B8D 7CFF  MOV ECX,DWORD PTR SS:[EBP-84]
0040186C . E8 9FFDFF  CALL CrackMe0.00401610

```

לשמחתנו עברנו את הבדיקות! ☺

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

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



עכשו מתחילים את הכיף האמיתי

```

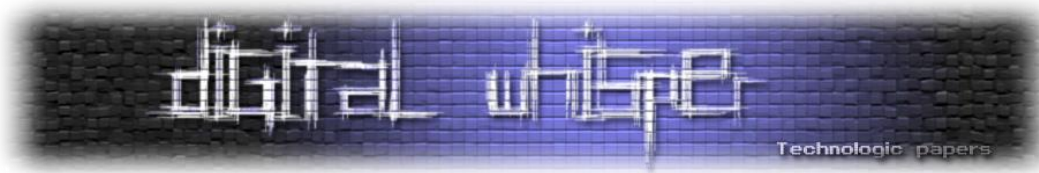
0040163A . BB D1294400 MOV EBX, CrackMe0.00442901
0040163F . BF 5F2A4400 MOV EDI, CrackMe0.00442A5F
00401644 . E8 DA060000 CALL CrackMe0.00401D23
00401649 . 83C4 04 ADD ESP, 4
0040164C . 894424 14 MOV DWORD PTR SS:[ESP+14], EAX
00401650 . C74424 20 00 MOV DWORD PTR SS:[ESP+20], 0
00401653 . 5CB8 TEST EAX, EAX
0040165A . 74 08 JE SHORT CrackMe0.00401664
0040165C . 50 PUSH EAX
0040165D . E8 EE020000 CALL CrackMe0.00401950
00401662 . EB 02 JMP SHORT CrackMe0.00401666
00401664 . 33C0 XOR EAX, EAX
00401666 . C74424 20 FF MOV DWORD PTR SS:[ESP+20], -1
0040166F . 8B6C24 20 MOV EBP, DWORD PTR SS:[ESP+20]
00401672 . 31FB 232A4400 CMP EBX, CrackMe0.00442A23
00401678 . 75 08 JNZ SHORT CrackMe0.00401682
0040167A . 81FF 232A4400 CMP EDI, CrackMe0.00442A23
00401680 . 74 68 JE SHORT CrackMe0.004016EA
00401682 . 8A55 00 MOV DL, BYTE PTR SS:[EBP]
00401685 . 84D2 TEST DL, DL
00401687 . 74 65 JE SHORT CrackMe0.004016EE
00401689 . 8A0F MOV CL, BYTE PTR DS:[EDI]
0040168B . 8A0B OR CL, BYTE PTR DS:[EBX]
0040168D . F6C1 04 TEST CL, 4
00401690 . 75 5C JNZ SHORT CrackMe0.004016EE
00401692 . 80E2 0F AND DL, 0F
00401695 . 0FB6F2 MOVZX ESI, DL
00401698 . 8BD6 MOV EDX, ESI
0040169A . 83E2 03 AND EDX, 3
0040169D . 83FA 03 CMP EDX, 3
004016A0 . 77 19 JN SHORT CrackMe0.004016BB
004016A2 . FF2495 141744 JMP DWORD PTR DS:[EDX*4+401714]
004016A9 . 33EB 01 SUB EBX, 1
004016AC . EB 00 JMP SHORT CrackMe0.004016BB
004016AE . 33EB 1A SUB EBX, 1A
004016B1 . EB 08 JMP SHORT CrackMe0.004016BB
004016B3 . 33C3 01 ADD EBX, 1
004016B6 . EB 03 JMP SHORT CrackMe0.004016BB
004016B8 . 83C3 1A ADD EBX, 1A
004016BB . C1EE 02 SHR ESI, 2
004016BE . 83FE 03 CMP ESI, 3
004016C1 . 77 22 JA SHORT CrackMe0.004016E5
004016C3 . FF24B5 241744 JMP DWORD PTR DS:[ESI*4+401724]
004016CA . 33EF 01 SUB EDI, 1
004016CD . 33C5 01 ADD EBP, 1
004016D0 . EB A0 JMP SHORT CrackMe0.00401672
004016D2 . 33EF 1A SUB EDI, 1A
004016D5 . 33C5 01 ADD EBP, 1
004016D8 . EB 98 JMP SHORT CrackMe0.00401672
004016DA . 33C7 01 ADD EDI, 1
004016DD . 33C5 01 ADD EBP, 1
004016E0 . EB 90 JMP SHORT CrackMe0.00401672
004016E2 . 33C7 1A ADD EDI, 1A
004016E5 . 33C5 01 ADD EBP, 1
004016E8 . EB 88 JMP SHORT CrackMe0.00401672
004016EA . 6A 01 PUSH 1
004016EC . EB 02 JMP SHORT CrackMe0.004016F0
004016EE . 6A 00 PUSH 0
004016F0 . 8B10 MOV EDX, DWORD PTR DS:[EAX]
004016F2 . 8BC8 MOV ECX, EAX
004016F4 . 8B82 5C010000 MOV EAX, DWORD PTR DS:[EDX+15C]
004016FA . FF06 CALL EAX
004016FC . 8B4C24 18 MOV ECX, DWORD PTR SS:[ESP+18]
00401700 . 64:8900 0000 MOV DWORD PTR FS:[0], ECX
00401707 . 59 POP ECX
00401708 . 5F POP EDI
00401709 . 5E POP ESI
0040170A . 5D POP EBP
0040170B . 5B POP EBX
0040170C . 83C4 10 ADD ESP, 10
0040170F . C2 0400 RETN 4

```

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 הערך 0x01, 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

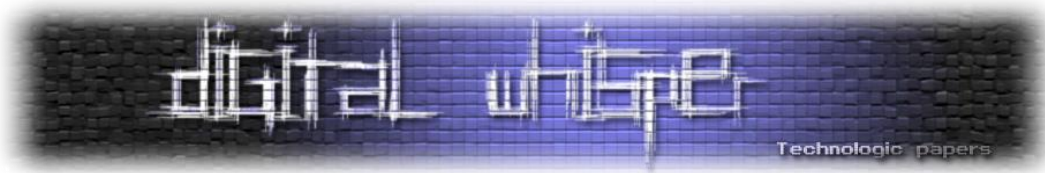
EDI = 0x00442A5F

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) // צדקנו ולא
        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 משפיעה על שניהם בצורה שונה.

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

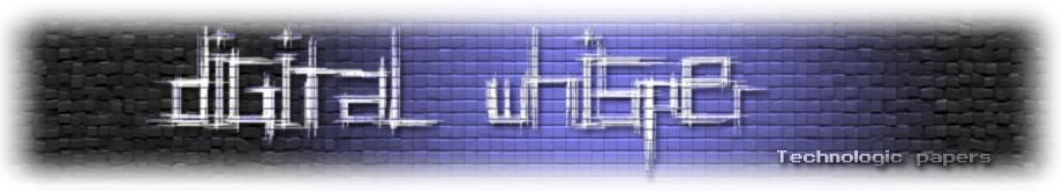
אך, זה לא הכל! למעלה בקוד ישנה פעולה לוגית לערך שנמצא בכתובת הנוכחית של EBX ו-EDI:

```
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!



004428C0	63	68	4D	65	30	35	44	6C	67	40	40	00	00	00	00	00	ckMe05D1g@0....
004428D0	83	67	47	35	8C	E5	C3	DD	3E	E6	A8	DF	3F	7D	86	77	äg55iσ H>ρd?38w
004428E0	34	37	9E	9A	29	11	84	53	79	F8	DD	F0	08	C2	B9	F8	47Nü)4äsyo E□-ñl°
004428F0	F5	98	12	83	AE	42	0A	80	C1	85	28	78	82	97	43	58	Jc#ä«B.C-ä(CeuCD
00442900	00	4A	7E	C3	B7	91	34	A3	16	FA	A1	7A	D6	11	20	C2	.J"  mæ4ü.  zr◄ ד
00442910	B3	F6	53	7B	72	AD	B8	09	E4	B4	EB	CB	16	08	04	63	÷Str+7.2+3π-□+c
00442920	ED	00	DB	3F	CF	46	48	14	9C	B6	44	AB	84	3D	06	FA	□■?FHññ D%ä=+
00442930	57	C3	13	91	65	30	21	1D	66	01	3A	75	72	38	98	E2	W !æ0!#f0:ur;cΓ
00442940	B0	CB	70	82	33	FC	72	BE	52	88	00	62	1E	22	59	2E	πpe3"r=Ri.b▲"Y.
00442950	3A	AB	3F	B3	0B	61	44	D4	60	45	4D	77	CF	0B	30	30	:%? σaD+EMw#σ00
00442960	53	61	59	64	32	F8	83	E1	31	33	07	42	4A	1F	56	0B	Sayd2°äp13·BUU0
00442970	7B	E6	99	03	69	B9	B3	F2	C5	FD	3F	A9	29	BB	77	44	(ρ0+iñ z+2°r)ñwD
00442980	08	F1	1C	6C	18	03	6E	D1	73	05	58	55	A1	EF	B5	D4	□±L †on†s#XUinñ°
00442990	7C	68	6F	93	9B	BD	48	D6	28	E3	86	BA	BA	12	CC	28	!ho0c#Kπ(π3   ††(
004429A0	57	13	17	8A	AB	C1	45	5A	A9	11	61	63	C6	55	F0	D2	W!#±%±E2r◄acFU=π
004429B0	60	E8	3C	39	AC	3C	67	E1	7F	E0	03	30	9C	40	35	7A	*±<9%<gb00#0005z
004429C0	8E	83	16	97	4B	5F	39	00	9A	9C	82	A1	D9	F1	A0	F8	ää_uK_9.ü6e i±:ä°
004429D0	98	88	26	AA	A6	8E	91	E5	16	05	D1	EC	FA	4F	6A	6A	ÛBε&~äæσ. #τ◦:Oj
004429E0	2F	4B	C2	1E	51	5D	B0	3E	CC	A4	63	0C	92	7D	F5	39	/Kτ&Q π>††c. #E)J9
004429F0	53	12	3B	50	3B	BA	F6	72	48	4B	A7	4A	65	52	62	BA	S#;P;  +rHK0JeRb
00442A00	8C	11	38	81	95	D1	B0	AD	94	49	A7	6D	12	05	42	30	i4Bü0†πi0I0m#B0
00442A10	7B	A5	8F	99	FB	CE	69	30	6E	13	79	90	2F	5B	63	D0	(ñA0Û†i0ñ!!yE/Co
00442A20	40	3E	E5	71	C3	E4	FB	82	76	43	68	88	0A	90	74	74	@>σeq Σ2evChē.ēt
00442A30	C9	62	21	2C	0C	0E	70	07	BF	24	F1	7E	04	98	60	89	††b†. #.p◦.γ±"◊"ü'ä
00442A40	A0	5E	26	D0	55	1C	85	F6	7A	3A	A5	7A	BD	99	80	C0	ä^%ULä±z:ñz"üC4
00442A50	39	91	17	78	11	25	27	AB	FB	6B	61	28	0B	0A	88	88	9æ#4%°%ka(σ.ē=
00442A60	C3	CE	B1	3C	A1	B8	81	65	13	67	08	F8	D0	21	5C	F1	H†< i7üelc◊°+†\p
00442A70	09	16	C4	E4	3F	BA	30	1F	25	0E	04	97	93	99	42	1F	.-2?  =7%#äü00B†
00442A80	92	79	33	80	4E	01	7A	3E	B1	6B	9A	73	12	08	03	36	äy3CN0z #küš#0#6
00442A90	B8	04	0E	04	BF	DF	9D	37	76	DB	EF	7F	5C	2F	FD	B6	†◊#◊◊#7◊◊◊◊◊◊◊◊
00442AA0	B5	2A	A9	30	00	00	00	00	23	2A	44	00	D1	29	44	00	†*r0...#*D.†D0.
00442AB0	5F	2A	44	00	E4	63	43	00	00	00	00	00	2E	50	41	44	-*D.ΣcC.....PAC

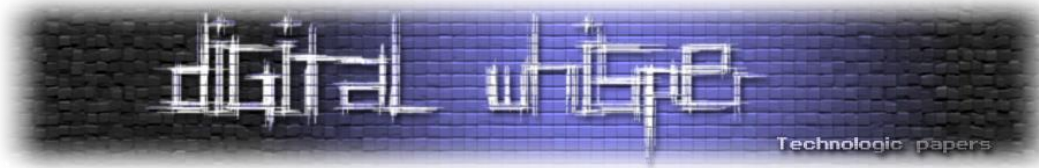
- הכתובת שמסומנת בצהוב היא הכתובת שעלינו להגיע אליה, 0x00442A23.
- הכתובת שמסומנת באדום היא כתובתה של EBX.
- הכתובת שמסומנת בכחול היא כתובתה של EDI.

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) בתיים 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!

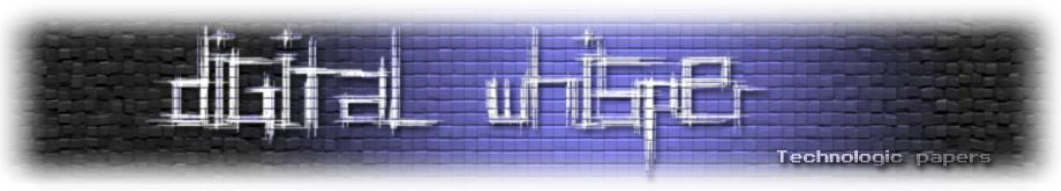
על מנת לבצע את המהלך הזה, כתבתי תוכנה ב-Python:

```
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.]



התוצאה:

1	83	XX	XX	XX	XX	XX	C3	XX	XX	XX	A8	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	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	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	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	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	XX	XX	XX	XX	XX	XX	XX	XX	2A	A9	30
19																										

[הדגשתי בירוק את הקירות בעזרת Notepad++]

סימנתי את שני השחקנים, הכחול והאדום, ואת נקודת היציאה בצבע צהוב.

עכשיו, כשיש לנו את המבוך בצורה מאוד וויזואלית 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 האפשריות 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 **האדום**- האותיות 'A','E','I','M', גורמות לשחקן ללכת שמאלה, אך, אותן האותיות בדיוק אצל השחקן **הכחול**, 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 **האדום**, בגלל שאותה האות משפיעה גם על השחקן השני, נוכל לבצע עם השחקן **הכחול** כל תנועה שאנחנו רוצים בו זמנית.

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

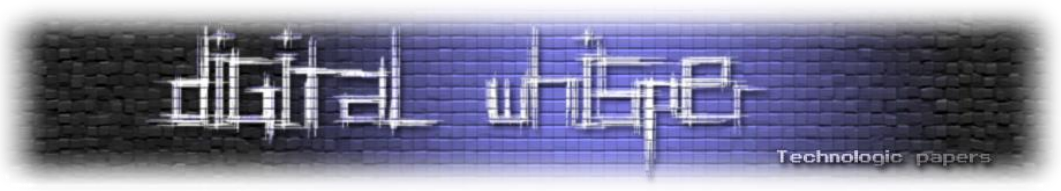
נחזור לחוקים שלנו:

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:





1	83	XX	XX	XX	XX	XX	C3	XX	XX	XX	A8	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	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
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	6B	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	83	A1	D9	F1	AC	F0	90	88	XX	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	30	53	13	3D	50	3D	5A	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	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	6D	61	20	0B	0A	00	C3	XX	B1	XX	A1	B8	81	XX	13	XX	08	F8	D0	21	XX	E1	XX	
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	XX	XX	XX	XX	XX	XX	XX	XX	XX	2A	A9	30
19																											

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

**אדום:**

שמאלה x7, למעלה x4, שמאלה x5, למטה x3, שמאלה x3, למטה x6, שמאלה x2.

**וכחול:**

שמאלה x8, למעלה x2.

נונסה להמיר זאת לאותיות, תיווצר לנו התנגשות, כי בתחילה נרצה בו זמנית שמאלה בשחקן ה**אדום** וגם **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.

אז זאת המשימה שאני נותן לכם, חברים.

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.. **בהצלחה!** 😊

