



---

# I am Very Good - Stage3 Pwned!

מאת רונן שוסטין (Antartic) ופולג הדר (P)

---

## הקדמה

בשנת 2009, במסגרת הכנס "IL-Hack 2009", פרסם השב"כ ארבעה אתגרים בתחום ה-Reverse Engineering מסוג CrackMe. מטרת פרסום אתגרים אלו הייתה מציאת אנשים בעלי כשרונות בתחום ה-RE. לאחר פתירת כל אתגר הופיעה על המסך הודעה המבשרת על הצלחה ואליה מצורפת כתובת דואר אלקטרוני לשליחת קו"ח. האתגרים ממוספרים מ-0 עד 3 על פי רמות הקושי. במאמר זה נתמקד באתגר האחרון מתוך הארבעה (stage3.exe).

במאמר זה נסקור טכניקות אנטי-דיבאגינג מסויימות. כרקע אנו ממליצים לקרוא את מאמרו המצויין של Zerith בנושא מתוך DigitalWhisper:

<http://www.digitalwhisper.co.il/files/Zines/0x04/DW4-3-Anti-Anti-Debugging.pdf>

בנוסף כדי להבין לעומק את הנושאים עליהם נדבר במאמר אנו ממליצים להיעזר במקורות אלו:

**Intel x86 Instruction Set:**

<http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.htm>

**MSDN:**

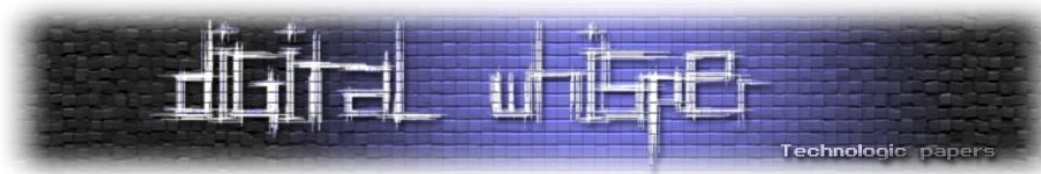
<http://msdn.microsoft.com/en-us/library/windows/desktop/ff818516%28v=vs.85%29.aspx>

**כלים נדרשים:**

**Ollydbg:**

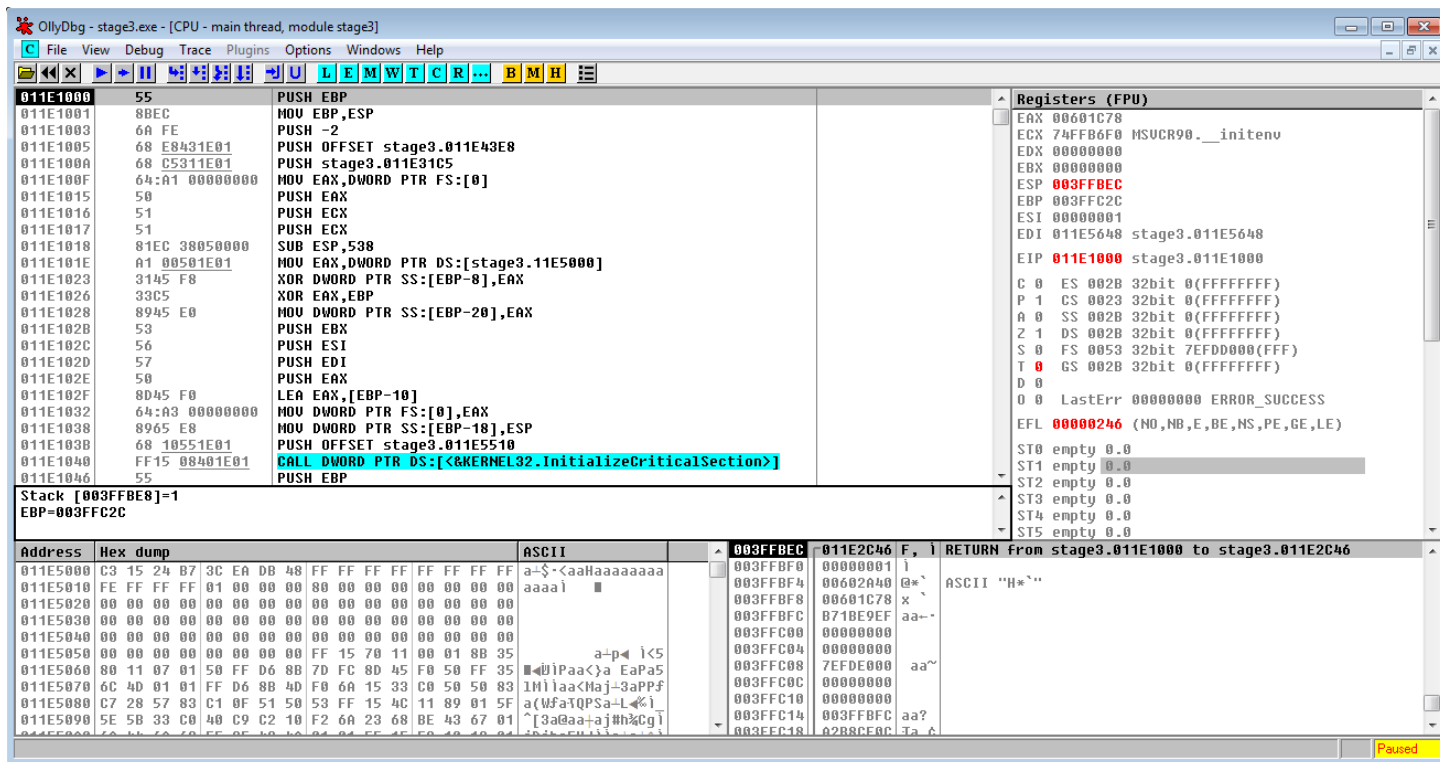
<http://www.ollydbg.de>

אין לייחס חשיבות לכתובות הזיכרון המופיעות במאמר, הן עלולות להשתנות מהרצה להרצה עקב מנגנון ה-ASLR הגורם לשינוי כתובת הבסיס בכל הרצה.



## ניתוח התוכנית

נפתח את Ollydbg ונטען את התוכנית stage3.exe. נתחיל לצעוד (Step) בקוד התוכנית ולהתחיל לנתח אותו.



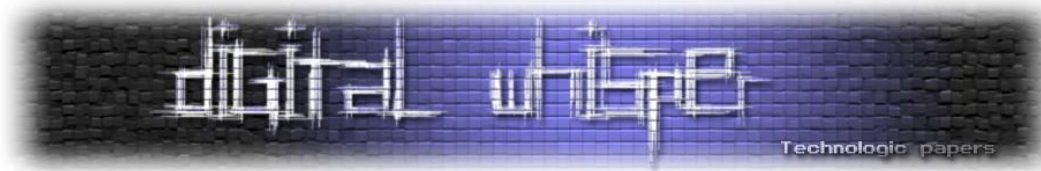
בתחילת התוכנית ניתן להבחין בהוראה INT 3 המשמשת לרוב דיבאגרים בתור נקודת עצירה:

5D	POP EBP
8365 FC 00	AND DWORD PTR SS:[EBP-4], 00000000
CC	<b>INT3</b>
C745 FC FFFFFFFF	MOV DWORD PTR SS:[EBP-4], -2
EB 26	<b>JMP SHORT 00401088</b>
33C0	XOR EAX, EAX
40	INC EAX
C3	<b>RETN</b>

כאשר ההוראה מתבצעת, השליטה מועברת אל ה-Exception Handler ומשם נקבעת זרימת התוכנית. מפתחים נוהגים להשתמש בהוראה זו כטכניקת אנטי-דיבאגינג, משום שהרצה של הוראה זו בסביבת דיבאגר תגרום לו לזהות כי זו אחת מנקודות העצירה שהמשתמש קבע לו, ולכן לא יעביר את השליטה אל ה-Exception Handler.

I am Very Good - Stage3 Pwned!

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)



המפתח יוכל לנצל עובדה זאת, ובעזרת משתנה שיתנהג כדגל (flag) שישתנה אך ורק אם השליטה הועברה אל ה-Exception Handler ניתן יהיה לזהות אם דיבאגר מצורף לתוכנית ולשלוט בזרימת התוכנית בהתאם.

בדיקה דומה תבוצע בכל פעם שהשליטה תעבור אל ה-Exception Handler, ונזכיר זאת עוד בהמשך. נצרך דוגמא הממחישה את הבדיקה המבוצעת:

```
int debugger_flag = 1; /* Assume a debugger is attached. */
__try {
    __asm {
        int 3; /* INT3 trap for the debugger. */
    }
}
__except (EXCEPTION_EXECUTE_HANDLER) {
    debugger_flag = 0; /* Debugger flag set if the control was passed to the exception
handler. */
}

if (1 == debugger_flag)
    MessageBoxA(NULL, "Debugger Detected", "Debugger Detected", MB_OK);
```

כעת, נמשיך לעקוב אחר קוד התוכנית ולנתחו:

59	POP ECX
C745 FC 0100	MOV DWORD PTR SS:[EBP-4],1
CD 01	INT 1
C745 FC FEFF	MOV DWORD PTR SS:[EBP-4],-2
EB 26	JMP SHORT 00041123
33C0	XOR EAX,EAX
40	INC EAX
C3	RET

ניתן לראות בקוד הוראות מסוג INT 1, תפקיד הוראה זו הינו לייצר חריגה שונה מזו ש-INT 3 מייצר, אך מימוש האנטי-דיבאגייג בתוכנית זו זהה לגבי שתי ההוראות. נמשיך בניתוח התוכנית:

68 0C552501	PUSH OFFSET 0125550C	ASCII "%x %x %x %x"
68 00552501	PUSH OFFSET 01255500	
68 28562501	PUSH OFFSET 01255528	
68 08552501	PUSH OFFSET 01255508	
68 A4412501	PUSH OFFSET 012541A4	
FF15 BC402501	CALL DWORD PTR DS:[&MSUCR90.scanf]	
83C4 14	ADD ESP,14	
83F8 04	CMP EAX,4	
7D 05	JGE SHORT 01251154	
E8 80110000	CALL 012522D4	

I am Very Good - Stage3 Pwned!

[www.DigitalWhispeh.co.il](http://www.DigitalWhispeh.co.il)



ניתן לראות כי ישנו קלט לארבעה משתנים בעזרת הפונקציה scanf. הפורמט המצופה על-ידי הפונקציה הינו ארבעה ערכים הקסדצימליים.

בתרגום חופשי לשפת C, הקריאה לפונקציה scanf נראית כך:

```
scanf("%x %x %x %x", &pass0, &pass1, &pass2, &pass3);
```

כאשר ניתן לראות את כתובות המשתנים שנדחפים בסדר הפוך למחסנית:

```
&pass0 = 0x01245508;
&pass1 = 0x01245628;
&pass2 = 0x01245500;
&pass3 = 0x0124550C;
```

נקבע נקודת עצירה מסוג חומרה (Hardware Breakpoint). כאשר תתבצע קריאה מכל אחת מכתובות המשתנים הדיבאגר יעצור, וכך נוכל לעקוב בקלות ולראות אילו פעולות התוכנית מבצעת על הקלט שהכנסנו.

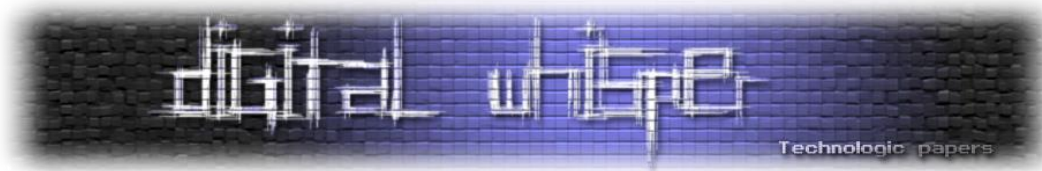
Slot	Type	Address	Module	Status	Disassembly	Comment
1	Access:1	011E5508	stage3	Active		
2	Access:1	011E5628	stage3	Active		
3	Access:1	011E5500	stage3	Active		
4	Access:1	011E550C	stage3	Active		

כאמור, הפונקציה scanf מחזירה את מספר הארגומנטים אליהם המידע נקלט בהצלחה. ואכן ניתן לוודא זאת בהמשך הקוד, כאשר ישנה השוואה עם המספר ארבע. במידה והתנאי לא התקיים, נקפוץ לפונקציה אשר תודיע לנו כי הסיסמא שגויה, ולבסוף תזמן את יציאת התהליך (ExitProcess).

אם הקלט שהוכנס תאם ל-Format String, כלומר נקלטו ארבעה ערכים הקסדצימליים, התוכנית תמשיך לפעול. מכאן ואילך, נתייחס לסדר הסיסמאות בהתאם לסדר הקליטה מן המשתמש, כפי שניתן לראות בקריאה לפונקציה scanf.

בהמשך ריצת התוכנית, ניתקל בהוראה הבאה:

E5 02	IN EAX,2	I/O command
-------	----------	-------------



ההוראה IN הינה הוראה מיוחסת (Privileged Instruction) מסוג I/O. במעבדי ה-x86 כל הוראות ה-I/O ניתנות להרצה ב-Ring 0. כאשר ננסה להריץ הוראה מסוג I/O תחת רמה מיוחסת פחות (Ring 3), פעולה זו תגרום לזריקת חריגה (Exception) מסוג General Protection או #GP, מכאן תעבור השליטה אל ה-Exception Handler ותבוצע בדיקה דומה לזו שהדגמנו בתחילת המאמר.

לקריאה נוספת על הנושא, אנו ממליצים לעיין בפרק I/O Privilege Level 15.5.1 בספר Intel Manuals. כאנשים המתעסקים בתחום ה-RE, השימוש בנקודות עצירה הינו חלק בלתי נפרד מתהליך הדיבאגינג, באתגר זה נתקלנו בקושי לעשות זאת ונסביר כעת ממה נבע קושי זה.

6A 00	PUSH 0
6A 00	PUSH 0
6A 00	PUSH 0
68 4A210C01	PUSH 010C214A
6A 00	PUSH 0
6A 00	PUSH 0
FF15 20400C01	CALL DWORD PTR DS:[&KERNEL32.CreateThread]

ניתן להבחין בקריאה לפונקציה CreateThread מתוך ה-API של Windows:

```
CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE) 0x010C214A, NULL, 0, NULL);
```

הקריאה לפונקציה תיצור Thread חדש אשר יריץ את הפונקציה הנמצאת בכתובת 0x010C214A במרחב הכתובות הוירטואלי של התהליך שקרא לפונקציה (Calling process):

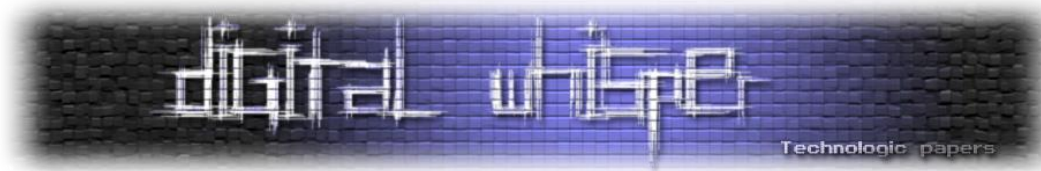
C74424 08 0123456	MOV DWORD PTR SS:[ESP+8],67452301
C74424 0C 89ABCDE	MOV DWORD PTR SS:[ESP+0C],EFCDA889
C74424 10 FEDCBA9	MOV DWORD PTR SS:[ESP+10],98BADCFE
C74424 14 7654321	MOV DWORD PTR SS:[ESP+14],10325476

אם נתעמק קצת בקוד הפונקציה, נגלה ארבעה קבועים בגודל ארבעה בתים כל אחד. חיפוש קצר בגוגל יביא אותנו למסקנה כי ישנו מימוש של אלגוריתם ה-MD5 בפונקציה. בהמשך הקוד מתבצעת השוואה בין שני Hash-ים (Checksum), במידה והם אכן שווים, התוכנית תמשיך כהלכה, אחרת, נקבל את ההודעה המפורסמת:

"Debugger Detected :")

ולאחר מכן תזומן הפונקציה ExitProcess.

אז מה בעצם עומד מאחורי האלגוריתם הזה ומדוע אנחנו לא יכולים להשתמש בנקודות עצירה?



ההשוואה מתבצעת על Checksum של הקוד המקורי (כלומר, ללא שינויים) אל מול ה-Checksum של הקוד במצבו הנוכחי (ה-Code Section). כאשר נשתמש בנקודת עצירה על הוראה מסוימת, הבית הראשון של האופקוד יתחלף ל-0xCC (האופקוד של ההוראה INT 3). בעקבות זאת, כאשר יתבצע ה-Checksum הבא, הוא יהיה שונה מה-Checksum המקורי, דבר שיוביל לזיהוי הדיבאגר.

6A 04	PUSH 4
59	POP ECX
BF 5C510C01	MOV EDI,OFFSET 010C515C
8D75 D4	LEA ESI,[EBP-2C]
33C0	XOR EAX,EAX
F3:A7	REPE CMPS DWORD PTR DS:[ESI],DWORD PTR ES:[EDI]
75 1A	JNE SHORT 010C2223

## הסיסמאות

קודם לכן קבענו נקודות עצירה מסוג Hardware BP על כל אחת מכתובות המשתנים אליהם התבצע קלט הסיסמאות. העצירה הראשונה התבצעה בעת ניסיון קריאה מכתובת המשתנה אליו נקלטה הסיסמא השלישית.

E8 190E0000	CALL 002E20C7	ASCII "(:!!!%s!!!:)"
FF35 00552E00	PUSH DWORD PTR DS:[2E5500]	
68 BC412E00	PUSH OFFSET 002E41BC	
8D85 C0FCFFFF	LEA EAX,[EBP-340]	
50	PUSH EAX	
FF15 C0402E00	CALL DWORD PTR DS:[<&MSVC90.sprintf>]	
83C4 0C	ADD ESP,0C	

נראה כי ישנה קריאה לפונקציה sprintf עם המשתנה אליו קלטנו את הסיסמא השלישית:

```
sprintf(buffer, "(:!!!%s!!!:)", pass2);
```

בהינתן סיסמא "ABCDEFFE", המחרוזת הסופית תיראה כך:

```
(:!!!ABCDEFFE!!!:)
```

ניתקל בקריאה הבאה לפונקציה:

E8 150D0000	CALL stage3.0095204C
-------------	----------------------

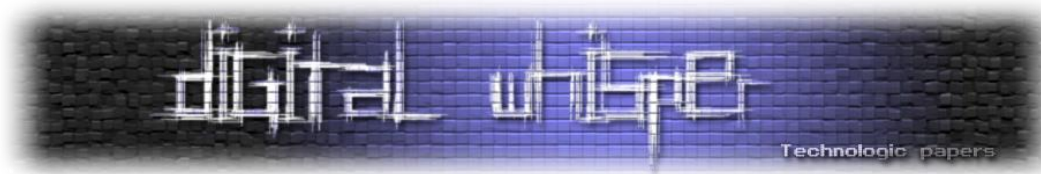
הפונקצייה הבאה הינה המימוש לאלגוריתם ה-MD5 עליו דיברנו קודם, היא מקבלת את הסיסמא השלישית ה'מקודדת' על-ידי sprintf והופכת אותו ל-Hash (פונקציית גיבוב קריפטוגרפית). עקב השימוש ב-MD5 ניתן להסיק כי ה-Format String בפונקציית ה-sprintf הינו Salt שמטרתו היא למנוע את גילוי הסיסמא על-ידי שימוש ב-Rainbow Tables ו-Dictionary Attack. קריאה לפונקציה נוספת תחזיר מחרוזת המייצגת את ה-Hash שנוצר בפורמט הקסדצימלי:

B8 28554000	MOV EAX,OFFSET 00405528	ASCII "acc0a69629dd1831c5980cb95ea0d181"
E8 9A0D0000	CALL 004020E4	

I am Very Good - Stage3 Pwned!

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)





בהמשך הקוד ניתן למצוא תבנית קוד מוכרת, היוצרת מחרזות המייצגת Hash בפורמט הקסדצימלי.

68 2CA729F8	PUSH F829A72C	ASCII "aa472df4"
68 DC414000	PUSH OFFSET 004041DC	
68 DB638E76	PUSH 768E63DB	
6A 39	PUSH 39	
6A 65	PUSH 65	
6A 33	PUSH 33	
6A 63	PUSH 63	
6A 66	PUSH 66	
6A 36	PUSH 36	
6A 38	PUSH 38	
6A 65	PUSH 65	ASCII "%c%c%c%c%c%c%c%c%x%s%x"
68 E8414000	PUSH OFFSET 004041E8	
8D85 E0FEFFFF	LEA EAX,[EBP-120]	
50	PUSH EAX	
FF15 C0404000	CALL DWORD PTR DS:[<&MSUCR90.sprintf>]	

כך נראית המחרוזת שמחזירה הפונקציה (ה-Hash):

0012F8E4	Σ↑	ASCII "e86fc3e9768e63dbaa472df4f829a72c"
004041E8	ΣAQ	ASCII "%c%c%c%c%c%c%c%c%x%s%x"

נראה כי לאחר מכן מתבצעת השוואה בין ה-Hash הנוכחי שהוחרזר ע"י הפונקציה לבין ה-Hash שנוצר מקלט הסיסמא השלישית:

5D	POP EBP	ASCII "Not Good !!!!!"
8B85 CCFEFFFF	MOV EAX,DWORD PTR SS:[EBP-134]	
0FBE8405 E0FEFFFF	MOVSX EAX,BYTE PTR SS:[EAX+EBP-120]	
8B8D CCFEFFFF	MOV ECX,DWORD PTR SS:[EBP-134]	
0FBE89 28554000	MOVSX ECX,BYTE PTR DS:[ECX+405528]	
3BC1	CMP EAX,ECX	
✓ 74 13	JE SHORT 00401C4A	
BE 00424000	MOV ESI,OFFSET 00404200	
8B7D 08	MOV EDI,DWORD PTR SS:[EBP+8]	
A5	MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]	
A5	MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]	
A5	MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]	
66:A5	MOVS WORD PTR ES:[EDI],WORD PTR DS:[ESI]	
A4	MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]	
✓ E9 95020000	JMP 00401EDF	

ניתן לראות כי מתבצעת השוואה בין תוכן הבתים של כל אחת מן המחרוזות (המייצגות את ה-Hash-ים) כדי לבדוק האם הסיסמא שהקליד המשתמש נכונה. במידה והן אינן שוות, תופיע המחרוזת "Not Good"!!!! על המסך עם Dump של הזיכרון, אחרת, התוכנית תמשיך לרוץ כמתוכנן.

בכדי למצוא את הסיסמא, נצטרך להשתמש ב-Brute Force מפני שהיא מיוצגת כ-Hash.



## מה אנחנו יודעים על הסיסמא?

- יש בידינו את ה-Hash של הסיסמא המקורית.
  - הסיסמא היא מספר הקסדצימלי בגודל ארבעה בתים, כלומר, הערך המירבי יהיה 0xFFFFFFFF.
  - ה-Salt בו השתמשו בכדי להצפין את הסיסמא הינו: "(:!!!%x!!!:)"
- נוכל לכתוב כלי שיבצע Brute Force בהתחשב בנתונים שרשמנו לעיל. אנו בחרנו ב-Python למטרה זו, ולפניכם הקוד:

```
"""
Written by Ronen Shustin and Peleg Hadar
For Digital Whisper
"""

import hashlib

i = 0 # Loop iterator. Also the hex number used for hashing.
enc = "e86fc3e9768e63dbaa472df4f829a72c" # Hashed password to BF.
buf = "" # Buffer.

while (0xFFFFFFFF > i) : # The password is hexadecimal number, 4 bytes sized.
    pass4 = hex(i).rstrip("L").lstrip("0x") # Strip the '0x' from the hex number (iterator) and the 'L' if
    long.
    buf = hashlib.md5("(:!!!" + pass4 + "!!!:)").hexdigest() # Generates the MD5 hash with the
    salt.

    if (0 == (i % 15000000)) : # Indicator - print loop iterator every 15 million iterations.
        print hex(i)

    if (enc == buf) : # Checks if the generated hash is equal to the hashed password.
        print pass4
        break

    i += 1 # Loop iterator increment.
```

כאשר הרצנו את התוכנית על מחשב עם מעבד Core i5 בעל ארבע ליבות, התוכנית פיצחה את ה-Hash תוך 02:19:21 שעות ועצרה ב-0xABCDDBCBA, וכך מצאנו את הסיסמא השלישית.





נמשיך לעקוב אחרי התוכנית ונראה כי הדיבאגר יעצור בעת ניסיון קריאה מכתובת המשתנה אליו נקלטה הסיסמא הרביעית:

A1 0C551A00	MOV EAX,DWORD PTR DS:[stage3.1A550C]
35 6F6F6F6F	XOR EAX,6F6F6F6F
A3 0C551A00	MOV DWORD PTR DS:[stage3.1A550C],EAX
A1 0C551A00	MOV EAX,DWORD PTR DS:[stage3.1A550C]
33D2	XOR EDX,EDX
83C9 FF	OR ECX,FFFFFFFF
F7F1	DIV ECX
69C0 A32F6760	IMUL EAX,EAX,60672FA3
B9 33C0F7F0	MOV ECX,F0F7C033
2BC8	SUB ECX,EAX
894D E4	MOV DWORD PTR SS:[EBP-1C],ECX

נתרגם את האלגוריתם הבא לשפת C:

```
int pass = *(0x01a550c); //User 4th password.  
int final = 0xffffffff; //Final result.  
  
pass ^= 0x6f6f6f6f;  
final /= pass;  
pass *= 0x60672fa3;  
final = (0xf0f7c033 - pass);
```

במידה ומנת החילוק שווה לאפס, פעולת הכפל באלגוריתם לא תשפיע על התוצאה הסופית, והיא תהיה שווה ל-0xf0f7c033.

במידה ומנת החילוק שווה לאחד, התוצאה תוכפל ב-0x60672fa3 ותושם ב-pass. לאחר מכן ערכו של pass יחוסר מערכו של final, כלומר:

```
final = (0xf0f7c033 - pass) = 0x90909090
```

איך נוכל להגיע למנת חילוק ששווה לאחד?

כפי שראינו קודם, במקרה שלנו מנת החילוק תהיה שונה רק כאשר המחלק שווה למחולק, זאת מפני ש-0xffffffff הינו המספר הגדול ביותר בטווח הנתון לנו (מספר הקסדצימלי בגודל ארבעה בתים), אך איך נוכל להגיע למצב בו המחלק שווה למחולק? נסתכל על ההוראה הבאה מתוך האלגוריתם:

```
pass ^= 0x6f6f6f6f
```

נצטרך קלט שלאחר הוראת ה-XOR שלעיל יהיה שווה ל-0xffffffff.

I am Very Good - Stage3 Pwned!

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)



כיוון שהשימוש ב-XOR הינו סימטרי, נוכל למצוא את הקלט הנ"ל:

$0xffffffff \wedge 0x6f6f6f6f = 0x90909090$
---

בסוף האלגוריתם, נדחפת תוצאת החישוב אל המחסנית והשימוש בה נעשה בהמשך הקוד.

68 84514000 C3	PUSH OFFSET 00405184 RETN
-------------------	------------------------------

ניתן לראות כי כתובת הזיכרון של תוצאת החישוב נדחפת למחסנית, ומיד אחריה ישנה קריאה להוראת RETN, כלומר, זוהי הכתובת אותה המעבד יריץ (EIP) לאחר שיחזור מהפונקציה. הקלט של המשתמש לתוך כתובת החזרה, מאפשרת לו לשלוט על קוד התוכנית משום שהערכים שיקלטו יתורגמו ל-Opcodes.

במקרה שלנו הערך 0xf0f7c033 יתורגם ל:

33C0 F7F0 C606 00 C3	XOR EAX,EAX DIV EAX MOV BYTE PTR DS:[ESI],0 RETN
-------------------------------	---

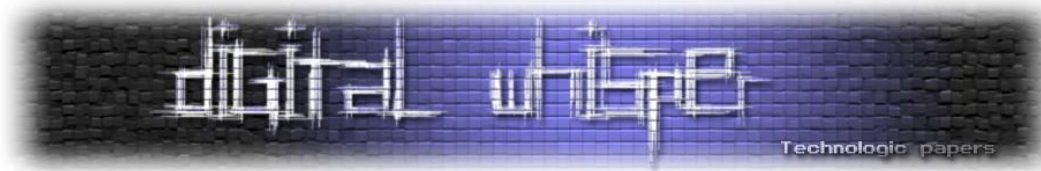
ערכו של האוגר EAX שווה לאפס כתוצאה מ-XOR עם עצמו. לאחר מכן מתבצעת חלוקה באפס אשר תגרום לחריגה (Exception) מסוג Divide Error או #DE. הערך 0x90909090 יתורגם כמובן ל:

90 90 90 90	NOP NOP NOP NOP
----------------------	--------------------------

דבר שימשיך את פעילות התוכנית בצורה תקינה, מכאן ניתן להסיק שהסיסמא היא 90909090.

813D 28564000 AFB 74 05 E8 D00D0000 C745 FC 07000000	CMP DWORD PTR DS:[405628],DEADBEAF JE SHORT 00401504 CALL 004022D4 MOV DWORD PTR SS:[EBP-4],7
---	--

בהמשך התוכנית הדיבאגר יעצור בעת ניסיון קריאה מכתובת המשתנה אליו קלטנו את הסיסמא השנייה. ניתן להבחין בהשוואה ברורה של ערך המשתנה אליו קלטנו את הסיסמא השנייה עם הערך 0xDEADBEAF. במידה וההשוואה נכשלת, ישנה קריאה לפונקציה שבסופה תופיע ההודעה Wrong "Password", אחרת התוכנית תמשיך לפעול כמתוכנן. מכאן ניתן להסיק שהסיסמא השנייה הינה DEADBEAF.



5D	POP EBP
FF15 1C404000	CALL DWORD PTR DS:[<&KERNEL32.GetCurrentProcessId>]
3905 08554000	CMP DWORD PTR DS:[405508],EAX
74 05	JE SHORT 00401613
E8 C10C0000	CALL 004022D4

נמשיך לעקוב אחר קוד התוכנית, ונראה כי הדיבאגר יעצור בעת ניסיון קריאה מכתובת המשתנה אליו קלטנו את הסיסמא הראשונה. ניתן לראות כי ישנה קריאה לפונקציית GetCurrentProcessID מתוך ה-API של Windows, המחזירה את ה-PID של התהליך הקורא לפונקציית (Calling process) בפורמט הקסדצימלי. נבחין מיד לאחר מכן בהשוואה של ערך המשתנה אליו קלטנו את הסיסמא הראשונה עם הערך שהחזירה הפונקציה, כלומר הסיסמא הראשונה אינה קבועה ומשתנה מהרצה להרצה. בפשטות, עלינו לבדוק מהו ה-PID של stage3.exe, להמיר אותו למספר הקסדצימלי, וכך מצאנו את הסיסמא הראשונה.

לאחר הרצת התוכנית והרצת הקלטים הדרושים, נקבל את התוצאה הבאה:

```

C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Administrator\My Documents\Downloads\IAmVeryGood>stage3.exe
Enter Password:
CDC DEADBEAF ABCDDCBA 90909090
Wait...
Mail Address:
47 6F 6F 64 20 4A 6F 62 21 0D 0A 53 65 6E 64 20 - Good Job!..Send
65 6D 61 69 6C 20 74 6F 20 - email to
40 67 6D 61 69 6C 2E - with subject
63 6F 6D 20 77 69 74 68 20 73 75 62 6A 65 63 74 - : stage3_4578506
3A 20 73 74 61 67 65 33 5F 34 35 37 38 35 30 36 - f411de0d443dfc3c
66 34 31 31 64 65 30 64 34 33 64 66 63 33 63 - 4fde67ede...Atta
34 66 64 65 36 37 65 64 65 2E 0D 0A 41 74 74 61 - ch your resume.
63 68 20 79 6F 75 72 20 72 65 73 75 6D 65 2E 20 - Good luck!
47 6F 6F 64 20 6C 75 63 6B 21
C:\Documents and Settings\Administrator\My Documents\Downloads\IAmVeryGood>_

```

מזל טוב! ☺

## סיכום

במאמר ניתחנו את האתגר הרביעי תוך כדי סקירה של שיטות ה-Anti-Debugging אשר מומשו בתוכנית. עקבנו אחר זרימת הקוד, קבענו נקודות עצירה על כתובות משתני הקלט של הסיסמאות, שהובילו אותנו לאלגוריתמים אשר ביצעו בדיקות או מניפולציות על קלט הסיסמא, חקרנו את האלגוריתמים וכך לבסוף הגענו לארבע הסיסמאות.

I am Very Good - Stage3 Pwned!

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)



## קישורים לקריאה נוספת

לקוראים המעוניינים להרחיב בנושאים עליהם כתבנו במאמר, מצורפת רשימה של קישורים שימושיים:

### Intel x86 Instruction Set:

<http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.htm>

### MSDN:

- <http://msdn.microsoft.com/en-us/library/windows/desktop/ff818516%28v=vs.85%29.aspx>

המאמר של זריף "Anti Anti-Debugging" מתוך Digital-Whisper:

- <http://www.digitalwhisper.co.il/files/Zines/0x04/DW4-3-Anti-Anti-Debugging.pdf>

סדרת המדריכים של lena151 למעוניינים להתחיל בReverse Engineering:

- <http://www.tuts4you.com/download.php?list.17>

הבלוג של R4ndom, כולל המון מדריכים:

- <http://www.thelegendofrandom.com/blog>

מאגר קראקמי (Crackme) בכל מיני רמות:

- <http://www.crackmes.de>