

# פרק 7

## נולקות

### הסוואה ואנ'צת נ'תוח



# נולקות: וירוסים וחיות אחרות



כיום מקובל לקרא לכולם וירוסים  
או נוזקות



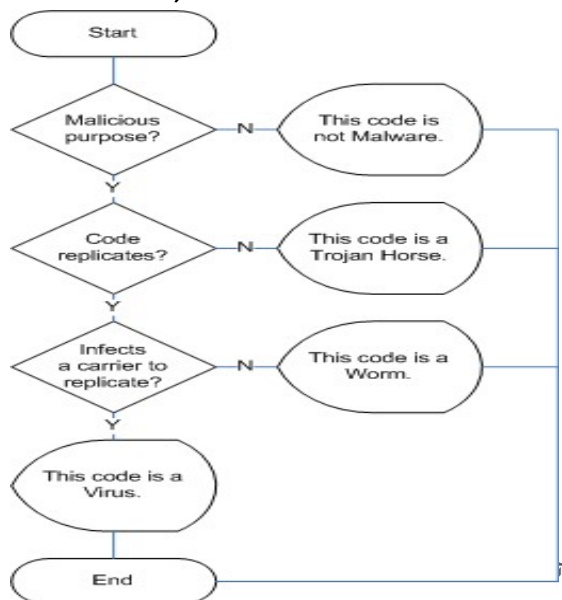
## נזקה

- נזקה הינה מונח כוללני המתייחס למגוון שונים של **תוכנה עוינת או מזיקה**, ובכלל זאת וירוסים, תולעים, סוסים טרויאניים, סוגי פרסומות, נזקות כופר, נזקות שמטרתן להפחיד את המשתמש ועוד.

- נזקה יכולה להיות קוד בינארי, סקריפט, active content ועוד. **הנזקה מוגדרת על ידי כוונתה העוינת,**

**והתנהגותה כנגד המשתמש.** לפיכך, המונח נזקה אינו כולל תוכנה אשר גורמת לנזק ללא כוונה.

- נזקה יכולה להיות מכוונת לתפוצה רחבה, או להיות ממוקדת ומוכוונת למטרה ספציפית...



## כמה האזרות

- חדירה

- הפעלה ראשונית של קוד זדוני במחשב
- נעשה באמצעות ווקטורי תקיפה

- התקבעות

- הבטחת המשך ההפעלה העתידית של הנוזקה
  - גם אחרי אתחול
- הפיכה לשווה

- הדבקה

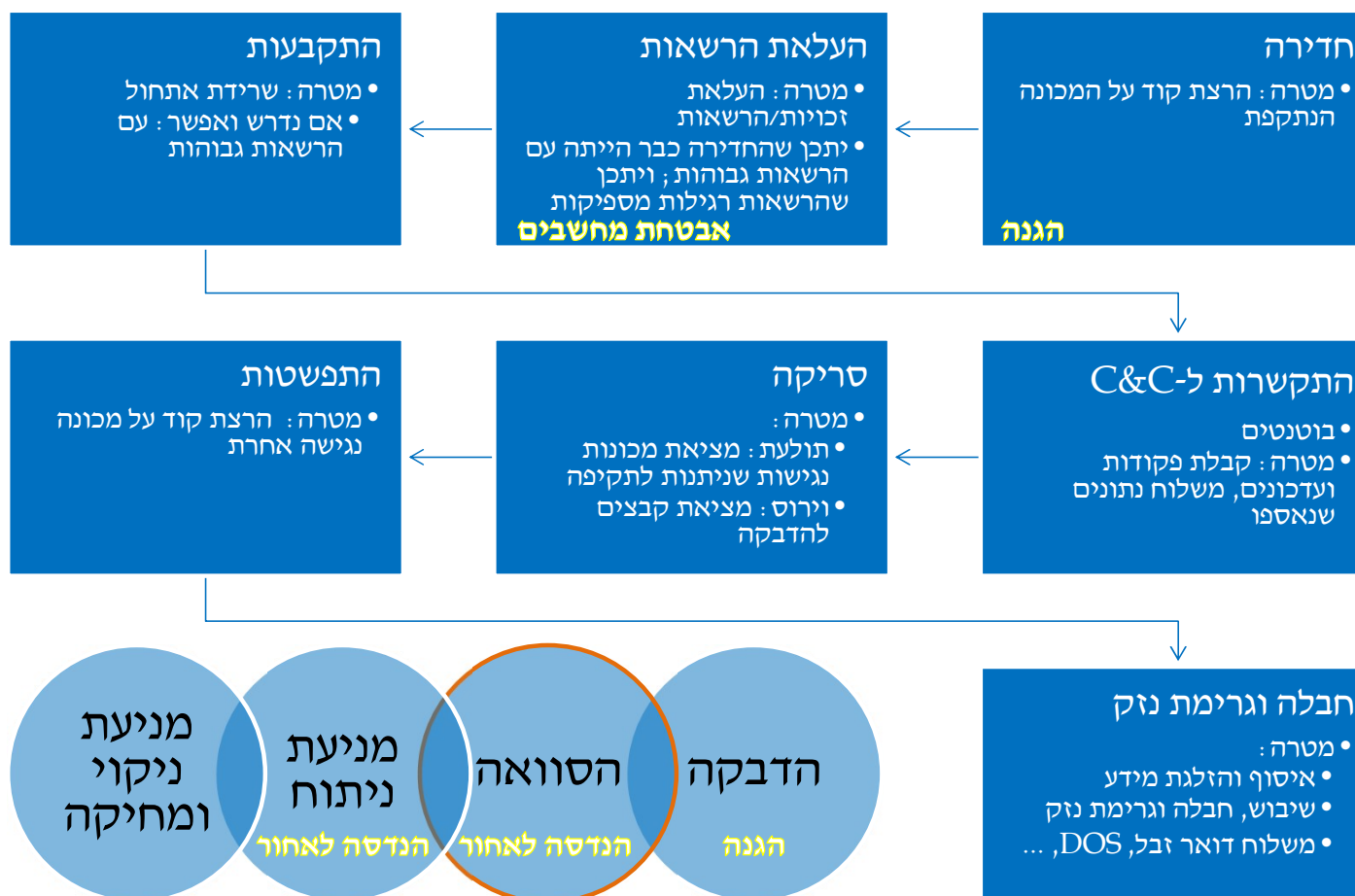
- שינוי קבצים או מידע אחר
  - שמאפשר בעקיפין להדביק מחשבים נוספים
  - או שמאפשר לנוזקה להתקבע
  - או להיות מורצת בהמשך
- המונח בשימוש גם במובן של חדירה

- הסוואה

- הסתתרות באופן שלא מאפשר לזהות את קיום הנוזקה



# מבנה כללי של פצילות נולקה



# הקשרים בלאן ריזה

- נזקה יכולה לרוץ במחשב במגוון הקשרים
- כתהליך
- כדרייבר
  - DLL או
  - או ברכיב של מ"ה
- כקובץ הרצה
  - או עי"י הדבקת קובץ הרצה קיים
  - בתקוה שמישהו יריץ אותו
- ב-DOS : TSR (terminate and stay resident)
  - תוך כדי השתלטות על וקטור הפסיקות
- כ-plugin של תוכנה אחרת
  - או סקריפט שמורץ על ידי תוכנה אחרת
- בשלד של תהליך אחר
  - אם פרץ אליו, למשל עי"י חריגה מחוצץ
- ואפילו כ-hypervisor
  - ולהריץ את מ"ה תחתיה
- וגם שילובים שלהם



# הדבקת קבצי הרצה

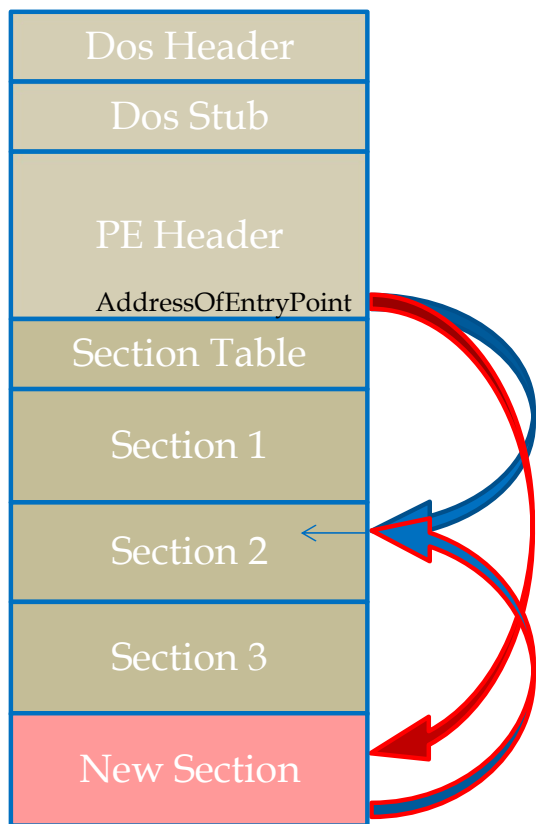
וביצוע הוקינג

## • הדבקת קובץ הרצה כוללת כמה חלקים :

- הוספת קוד לקובץ ההרצה.
  - ע"י הוספת section נוסף.
  - או ע"י הגדלת section קיים.
  - או שימוש במקום פנוי ב-section קיים.
  - חלק מהוירוסים דוחסים את תוכן הקובץ.
    - כדי לשמור על גודלו ללא שינוי.
    - הוירוס צריך לפתוח מחדש את הקוד בזמן ריצה.
  - כולל קפיצה לקוד המקורי בסיום עליית הוירוס.
  - חלק מהוירוסים מוחקים את הקוד המקורי,
    - ✓ או משנים אותו, או סתם לא קופצים אליו.
- הפניית החישוב לקוד שנוסף.
  - על ידי שינוי כתובת תחילת הריצה בקובץ ההרצה.
  - או על ידי ביצוע הוקינג מתחילת התכנית המקורית.
- הבטחת המשך פעולת הוירוס בזמן ריצת התכנית.
  - אם הוירוס מעוניין בכך.
  - על ידי השתלטות על וקטור הפסיקות.
  - או הוקינג לקריאות מ"ה.
  - או הוקינג לקטעי קוד מרכזיים בתכנית.



# הדבקת קבצי הרצה



• קובץ הרצה מקורי.

• הוספת Section חדש.

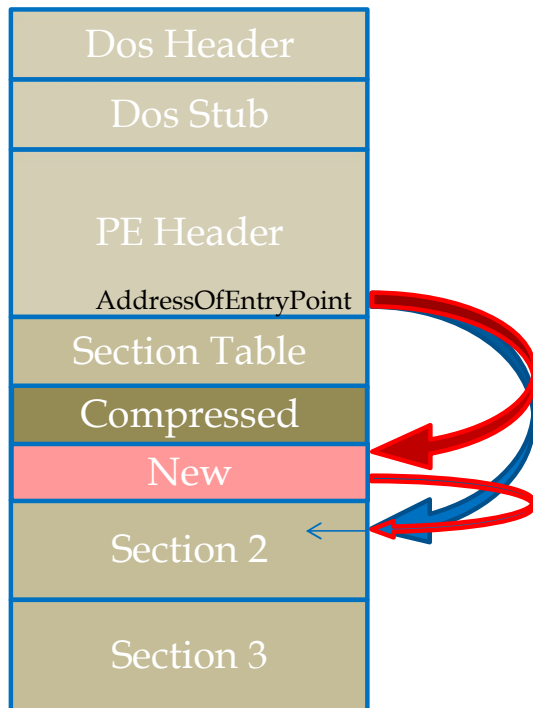
- שכולל את הקוד המזיק.
- וקוד שמבטיח המשך פעילות.
- כאשר עוברים לתכנית המקורית.

• הפניית כתובת תחילת הריצה.

• וקפיצה לקוד המקורי.



# הדבקת קבצי הרצה עם דחיסה



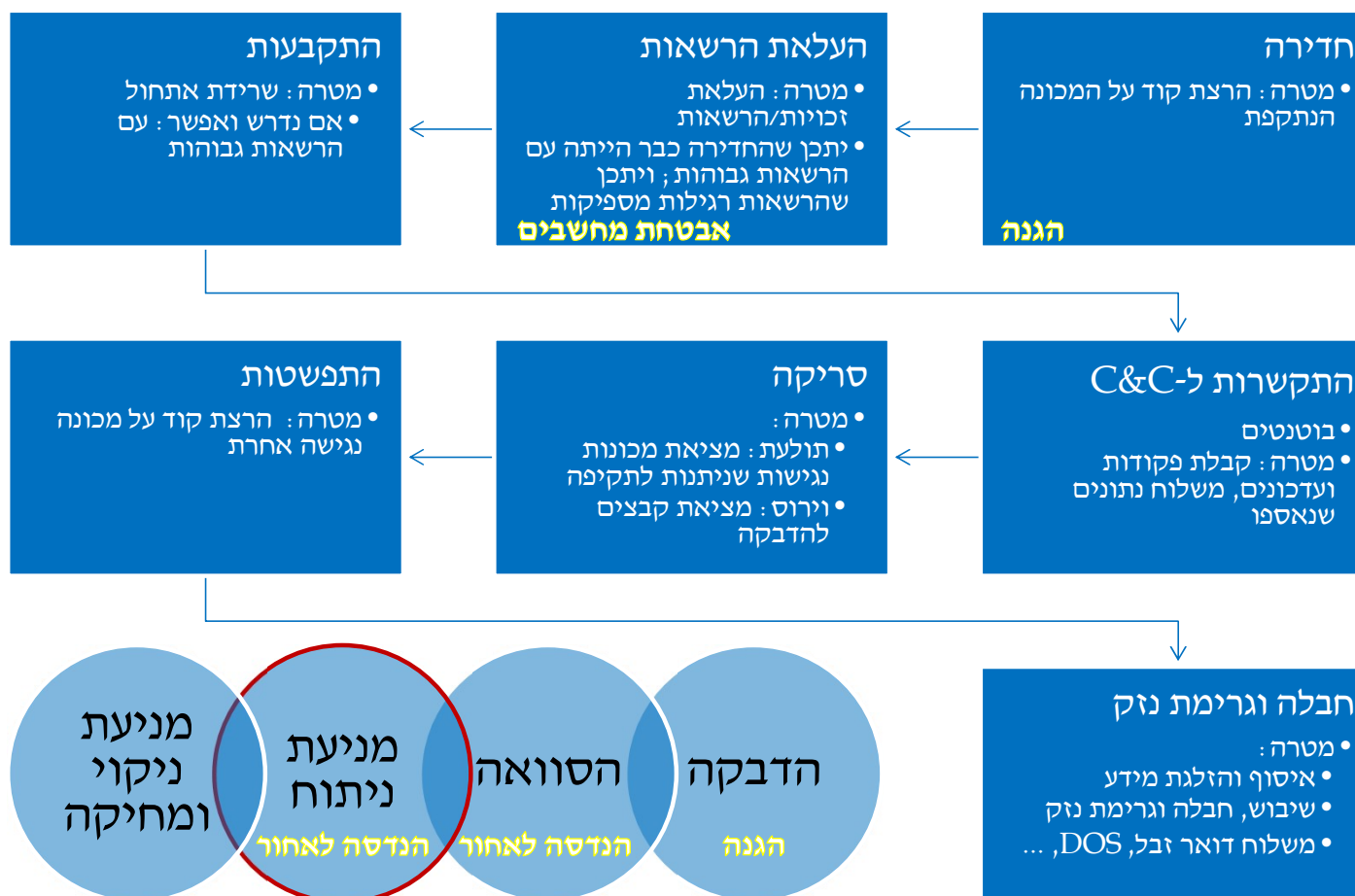
- דחיסת Section של נתונים.
  - עדיף לא Section של קוד.
    - כדי לא להפריע לביצוע relocation.
    - לעיתים גם באזורי נתונים נדרש relocation.
- הוספת קוד חדש באותו Section.
  - משלים לאותו גודל שהיה.
  - כולל קוד פתיחת הדחיסה.
    - כולל טיפול ב-relocation, אם צריך.
  - ואת הקוד המזיק.
  - וקוד שמבטיח המשך פעילות.
    - כאשר עוברים לתכנית המקורית.
- הפניית כתובת תחילת הריצה.
- וקפיצה לקוד המקורי.
- בזמן ריצה:
  - העתקת קוד הוירוס למקום פנוי, ופתיחת הדחיסה.

# RootKits

- שימוש בהוקינג על מנת להסתיר את עצם קיום הנוזקה
  - Rootkit רץ בדרך כלל בצורה של דרייבר – מנצל את ההפרדה בין kernel וה-user
  - מבצע ניטור וטיפול ב-winApi על מנת להסתיר את קיומו. משמש כגשר בין התוכנה שביקשה את המידע (כדוגמת רשימת קבצים) לבין מערכת ההפעלה.
- איך ניתן לגלות Rootkit?
  - לפני ההדבקה?
  - אחרי?



# מבנה כללי של פצילות נולקה



# Anti-Reversing

• לתוכנות מסוימות, כגון נוזקות, יש צורך בהסתרת הפעילות מפני גורמים מסוימים, ביניהם

- כלי סריקה אוטומטים, כדוגמת אנטי-וירוסים, מסוגלים לגלות קוד עוין ע"י חתימות
  - או ע"י זיהוי דפוס התנהגות מסוים שנחשב כלא טריוויאלי
- חוקרים שתפקידם למנוע את פעילות התוכנה
  - בדרך כלל מדובר בכאלו שעוסקים בפיתוח אמצעי נגד
- חברות מתחרות שמנסות לגנוב קוד

• Anti-Reversing הוא אוסף של טכניקות שמטרתן להקשות על גילוי התוכנה ועל ביצוע RE

- אין אפשרות מוחלטת למנוע RE
- אך אפשר לבלבל את החוקר, לבזבז את זמנו, ולהפריע לו
- שיטות הגנה מתוחכמות דורשות חוקרים יותר מנוסים והמון זמן מחקר

• אנו נדון בטכניקות אלה, ונתייחס גם להתמודדות מולן



# טכניקות Anti-Reversing

- ניתן לחלק את השיטות לכמה סוגים

- Obfuscation

- טכניקות להקשות על הקורא להבין את הקוד, או לעכב אותו

- Anti-Disassembly

- טכניקות למנוע את הצגת הקוד האמיתי לחוקר או סורק

- דחיסת והצפנת הקוד בקובץ ההרצה

- Anti-Debugging

- טכניקות לזיהוי סביבת ריצה עוינת

- ולשינוי (או שיבוש או עצירת) פעילות התוכנית כתוצאה מכך

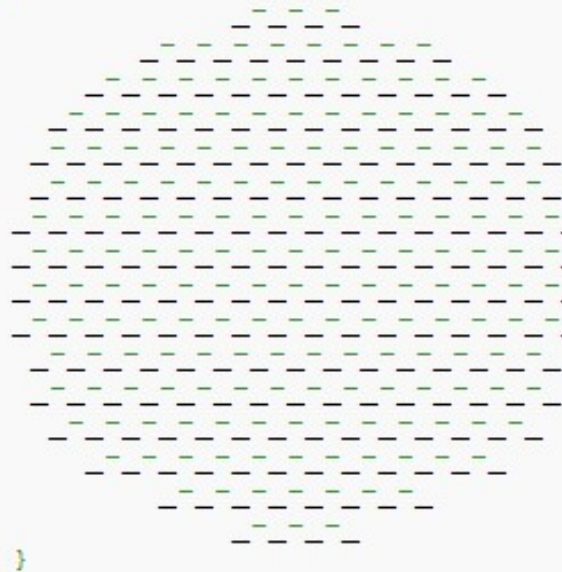
- נציג כמה רעיונות בסיסיים על מנת להדגים את הנושא

- כל רעיון בסיסי ניתן להרחבה לשיטה מתקדמת יותר



# מה מחשב הקוד הזה?

```
#define _ -F<00||--F-OO--;  
int F=00,OO=00;main(){F_OO();printf("%1.3f\n",4.*-F/OO/OO);}F_OO()  
{
```



\_ ⇒ -F<00||--F-OO--;

-- ⇒ --F<00||--F-OO--;

F=00; OO=00;

-F<00||--F-OO--; -F<00||--F-OO--; -F<00||--F-OO--; -F<00||--F-OO--;

...

printf("...", -4.0\*F/OO/OO);

→ F=-201, OO=-16 (F counts the '\_'s, OO counts the rows)

תשובה: 3.141



# והקוד הזה?

```
+++++++ [ >++++++>+++++++>+++>+<<<<- ]  
>++.>+.+++++. .+++.>+.<<+++++++.  
> .+++ .----- .----- .>+ .> .
```

**Brainfuck** is the most famous [esoteric programming language](#), and has inspired the creation of a host of other languages. Due to the fact that the last half of its name is often considered one of the most offensive words in the English language, it is sometimes referred to as brainf\*\*\*, brainf\*ck, brainfsck, b\*\*\*\*fuck, brainf\*\*k, branflakes, or BF. This can make it a bit difficult to search for information regarding brainfuck on the web, as the proper name might not be used at all in some articles.



# Obfuscation

- Obfuscation זה סט של כלים להקשות על הקורא להבין את הקוד
  - על ידי כתיבת הקוד בצורה קשה להבנה
- אפשר לבצע obfuscation גם בשפה עילית וגם באסמבלי
- דוגמאות
  - כתיבה מסובכת של קוד פשוט
    - כל מי שבדק תרגילים נתקל בקוד כזה ☺
  - משפטי תנאי ולולאות בהן לא ברור לקורא איזה קוד יבוצע
  - הוספת "garbage code" – קוד שאינו חלק רלוונטי מפעילות התוכנה, אך עלול לבזבז שעות לחוקר
  - הצפנה/דחיסה של הקוד





# Obfuscation – פשוטה ופשוטה

```
#include <Windows.h>

void Test()
{
    MessageBoxA(0, "Hi", "Hi", 0);
}

void main()
{
    _asm
    {
        pushad

        mov eax, 1
        cmp eax, 1
        jne BadStuff // This conditional jump will never happen!
        jmp Stuff

BadStuff:
    _emit 0x0F // Emitting some invalid opcode.

Stuff:
    call Test // My real "malicious" instructions.

    popad
    }
}
```

```
013D1020 53      PUSH EBX
013D1021 56      PUSH ESI
013D1022 57      PUSH EDI
013D1023 60      PUSHAD
013D1024 B8 01000000 MOV EAX, 1
013D1025 C8      CMP EAX, 1
013D1026 75 02    JNZ SHORT RE_Test1.013D1030
013D1027 EB 01    JMP SHORT RE_Test1.013D1031
013D1028 0F8CA    PSUBSB MM1, MM2
013D1029 FFFF    CALL
013D102A FF61 5F  JMP NEAR DWORD PTR DS:[ECX+5F]
013D102B 5E      POP ESI
013D102C 5B      POP EBX
013D102D C3      RETN
013D102E 90      DB 00
```

דוגמה פשוטה ב-C:

```
static int two=1;
```

```
main() {
    If (two != 1) {
        // Do some bad stuff
    } else {
        Test(); // The real code
    }
}
```

From "leetMatrix" blog

הנדסה לאחור – חורף תשפ"א

© פרופי אלי ביהם, אביעד כרמל, עמר קדמיאל

29.06.2023



# תזכורת: דיס-אסמבלי

- כזכור, דיס-אסמבלר בסיסי מפרש פקודות זו אחר זו
  - על פי סדרן בזיכרון
  - ע"י תרגום פקודה אחר פקודה (Linear Sweep)
- דוגמא לאלגוריתם בסיסי

```
while (position < BUF_SIZE) {  
    x86_insn_t insn;  
    int size = x86_disasm(buf, BUF_SIZE, 0, position, &insn);  
    if (size != 0) {  
        char disassembly_line[1024];  
        x86_format_insn(&insn, disassembly_line, 1024, intel_syntax);  
        printf("%s\n", disassembly_line);  
        position += size;  
    } else {  
        printf("db  0%2Xh\n", buf[0]);  
        position++;  
    }  
}
```



# אנטי-דיס-אסמבלי

- אנטי-דיס-אסמבלי היא סדרת טכניקות לבלבול דיס-אסמבלר
  - כך שיפרש את הקוד לא נכון
  - או שמקשות עליו לפרש את הקוד



# אנטי-דזיס-אסמבלי - ריבוי משמעותיות

- מה עושה הקוד הבא?

00401328	E8 C7042424	CALL 246417F4
0040132D	3040 00	XOR BYTE PTR DS:[EAX],AL
00401330	E8 A7060000	CALL <JMP.&msvrt.printf>
00401335	8D7424 1E	LEA ESI,DWORD PTR SS:[ESP+1E]

- הוא נראה לא הגיוני – קריאה ל-printf ללא פרמטרים

- והקוד הזה?

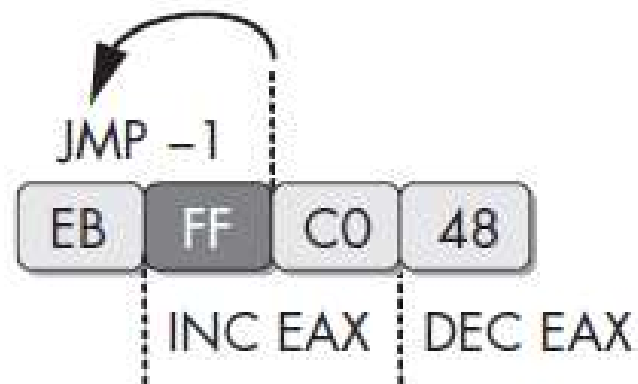
00401329	C70424 24304000	MOV DWORD PTR SS:[ESP],crackV2.00403024	ASCII "Enter the password:"
00401330	E8 A7060000	CALL <JMP.&msvrt.printf>	
00401335	8D7424 1E	LEA ESI,DWORD PTR SS:[ESP+1E]	

- מדובר באותו הקוד. אבל הפעם ל-printf יש פרמטרים



# אנטי-דזיס-אסמבלי - חפיפת קוד

• מה עושה הקוד הבא?



• זה למעשה (כמעט) פקודת nop בת 4 בטים

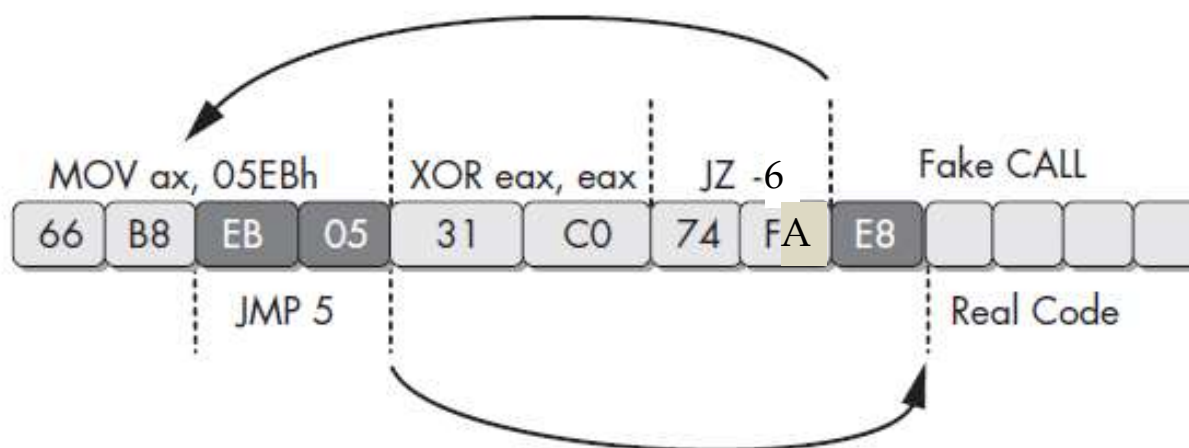
- קפוץ אחורה בית אחד, לקוד חופף (EBFF : 2 בטים)
  - הגדל את EAX באחד (FFC0 : 2 בטים)
  - הקטן את EAX באחד (48 : בית אחד)
- זה אינו nop בגלל ההשפעה על הדגלים

from Practical Malware Analysis



# אנטי-דזיס-אסמבלי - חפיפת קוד

- ומה עושה הקוד הזה?



- ה-disassembler לא ידע לתרגם את הפקודות נכון

from Practical Malware Analysis  
עם תיקון

## תזכורת: טיפול בחריגות

- תזכורת: SEH הוא מנגנון החריגות של מערכת ההפעלה
  - בעת חריגה מערכת ההפעלה ניגשת ל-TEB לקבל מצביע לרשימת פונקציות
    - הגישה לרשימה נעשית באמצעות FS:[0]
  - אלו הפונקציות שאחראיות על הטיפול בחריגה
  - אם הפונקציה הראשונה לא מטפלת, עוברים לבאה אחריה, עד שאחת מטפלת, או שהרשימה הסתיימה
- הפקודה "try" בשפת C ניגשת לרשימה (FS:[0]) ומוסיפה פונקציה חדשה לתחילתה
  - בסוף ה-try הפונקציה נמחקת מהרשימה
- דוגמא להוספת פונקציה:

```
push ExceptionHandler  
push fs:[0]  
mov fs:[0], esp
```



# אנטי-דאס-אפלי שימוש בחריצות

- ניתן לתכנן את התכנית כך שמהלכה יועבר לקוד במקום אחר על ידי חריגות
  - כגון חילוק באפס
  - כלומר, ללא פקודת jmp כלשהי





# אנטי-וירוס-אסמבלי שימוש בחריכות

## אנטי-וירוס

from Practical Malware Analysis

```
00401050  mov     eax, (offset loc_40106B+1)
00401055      add     eax, 14h
00401058      push    eax
00401059      push    large dword ptr fs:0 ; dwMilliseconds
00401060      mov     large fs:0, esp
00401067      xor     ecx, ecx
00401069  div     ecx
0040106B
0040106B loc_40106B:                                ; DATA XREF: sub_4010500
0040106B      call    near ptr Sleep
00401070      retn
00401070 sub_401050  endp ; sp-analysis failed
00401070
00401070 ; -----
00401071      align 10h
00401080      mov     esp, [esp+8]
00401084      mov     eax, large fs:0
0040108A      mov     eax, [eax]
0040108C      mov     eax, [eax]
0040108E      mov     large fs:0, eax
00401094      add     esp, 8
00401097      push    offset aMysteryCode ; "Mystery Code"
0040109C      call    printf
```



# עזר בולטאות פאנטי-דיס-אסמבלי

- קפיצות לכתובות שלא ידועות מראש
  - `call [eax]`
  - קשה להתגבר על השיטה ללא הרצה של הקוד עצמו
  - יש לציין שלקפיצות כאלו יש גם שימושים רגילים
    - למשל בשפות מונחות עצמים, כגון C++
    - או לקריאה לפונקציות מספריות דינמיות
- שיבוש זיהוי אוטומטי של משתנים לוקאליים
  - נעשה על ידי שינוי EBP (או ESP) באמצע הפונקציה
- ועוד

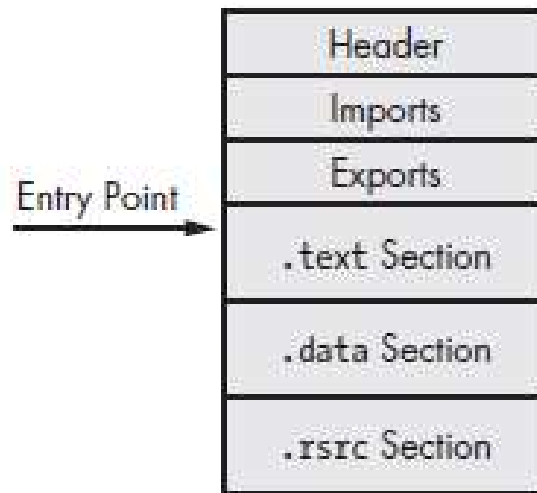


## דחיסת קובץ ההרצה

- ישנם כלים שמאפשרים דחיסת תוכנה
  - תוכנה דחוסה, בנוסף לקוד הרגיל, תכיל מתודה שאחראית על חילוץ הקוד והרצתו
  - במקרה כזה לא נוכל לנטר את התוכנה באופן סטטי
    - שכן הקוד לא חשוף לנו
    - כלומר לא ניתן להריץ IDA
  - הצפנה היא אחד הכלים המשמשים בזמן דחיסת קובץ ההרצה
    - אבל קוד הפענוח/מפתח נמצא בתוכנה, אחרת לא נוכל לפענח
    - לשם פשטות, נשתמש במלה דחיסה גם כאשר בפועל לא דוחסים
- טכניקה מקובלת לדחיסה נקראת packing
  - ותוכנה נפוצה נקראת UPX
- בשקפים הבאים נראה דוגמא כיצד לחלץ תוכנה שנדחסה ע"י UPX

# דחיסת קובץ ההרצה

תוכנה רגילה



תוכנה דחוסה



# דחיסת קובץ ההרצה

## Packing

- הטכניקה הכי נפוצה כיום למניעת Debugging
- הקוד האמיתי של התוכנה יכול להיות מכווץ או מוצפן באופן שלא ניתן לפתיחה ללא המתודה הרלוונטית
  - המתודה שאחראית על הפתיחה תכיל כמה שיותר דרכים שתפקידם להטעות על החוקר
- בתוכנות כאלו ניתן למצוא את מרבית שיטות ה-Anti-Debugging הנפוצות (שחלקן ראינו בהרצאה)
  - ושיטות נוספות שמטרתן להסתיר את הקוד הרלוונטי מפני חוקר
- ישנם Packers שדרכי החילוץ שלהם מתועדות בצורה נרחבת
  - ויש כאלו ייעודיים שמכילים טריקים פחות ידועים
  - אין זמן מוגדר להתמודדות
    - ישנן נוזקות שדרשו מספר שבועות מחקר עד להשגת הקוד



# דחיסת קובץ ההרצה

## דחיסת UPX

Code?:

C0	DB C0	CHAR 'd'
64	DB 64	
B0	DB B0	
2F	DB 2F	CHAR '/'
D8	DB D8	
2E	DB 2E	CHAR '.'
72	DB 72	CHAR 'r'
28	DB 28	CHAR '('
1F	DB 1F	
30	DB 30	CHAR '0'
27	DB 27	CHAR ''
12	DB 12	
F2	DB F2	
3D	DB 3D	CHAR '='
C7	DB C7	
8F	DB 8F	
60	DB 60	CHAR ''
40	DB 40	CHAR '@'
2E	DB 2E	CHAR '.'
62	DB 62	CHAR 'b'
62	DB 62	CHAR 'b'
5B	DB 5B	CHAR '['
9B	DB 9B	
40	DB 40	CHAR '@'
30	DB 30	CHAR '0'
09	DB 09	
DA	DB DA	
05	DB 05	
80	DB 80	
18	DB 18	
4F	DB 4F	CHAR 'O'
69	DB 69	CHAR 'i'
60	DB 60	CHAR ''
65	DB 65	CHAR 'e'
83	DB 83	
8D	DB 8D	
18	DB 18	
50	DB 50	
9B	DB 9B	CHAR 'P'
14	DB 14	
77	DB 77	
00	DB 00	CHAR 'w'
34	DB 34	
62	DB 62	CHAR '4'
B0	DB B0	CHAR 'b'
53	DB 53	CHAR 'S'
48	DB 48	CHAR 'H'
97	DB 97	

No strings:

```
004076C2 ASCII "rv"
004076F5 ASCII "siz"
004077FA ASCII "4JT\fp",0
00407810 ASCII "C6J",0
004078A5 ASCII "en"
```

No Imports:

```
00408050 UPX2 Import KERNEL32.ExitProcess
00408040 UPX2 Import KERNEL32.GetProcAddress
00408058 UPX2 Import msvcrt._iob
0040803C UPX2 Import KERNEL32.LoadLibraryA
004079B0 UPX1 Export <ModuleEntryPoint>
00408048 UPX2 Import KERNEL32.VirtualAlloc
0040804C UPX2 Import KERNEL32.VirtualFree
00408044 UPX2 Import KERNEL32.VirtualProtect
```

# דחיסת קובץ ההרצה

## דואלת UPX

004079B0	BE 15704000	PUSHAD
004079B1	8DBE EB9FFFFFFF	MOV ESI,crackU2.00407015
004079B6	57	LEA EDI,DWORD PTR DS:[ESI+FFFF9FEB]
004079BC	5B	PUSH EDI
004079BD	0B	JMP SHORT crackU2.004079CA
004079BF	90	NOP
004079C0	8A06	MOV AL,BYTE PTR DS:[ESI]
004079C2	46	INC ESI
004079C3	8B07	MOV BYTE PTR DS:[EDI],AL
004079C5	47	INC EDI
004079C6	01DB	ADD EBX,EBX
004079C8	75 07	JNZ SHORT crackU2.004079D1
004079CA	8B1E	MOV EBX,DWORD PTR DS:[ESI]
004079CC	83EE FC	SUB ESI,-4
004079CF	11DB	ADC EBX,EBX
004079D1	72 ED	JB SHORT crackU2.004079C0
004079D3	B8 01000000	MOV EAX,1
004079D8	01DB	ADD EBX,EBX
004079DA	75 07	JNZ SHORT crackU2.004079E3
004079DC	8B1E	MOV EBX,DWORD PTR DS:[ESI]
004079DE	83EE FC	SUB ESI,-4
004079E1	11DB	ADC EBX,EBX
004079E3	11C0	ADC EAX,EAX
004079E5	01DB	ADD EBX,EBX
004079E7	73 EF	JNB SHORT crackU2.004079D8
004079E9	75 09	JNZ SHORT crackU2.004079F4
004079EB	8B1E	MOV EBX,DWORD PTR DS:[ESI]
004079ED	83EE FC	SUB ESI,-4
004079F0	11DB	ADC EBX,EBX
004079F2	73 E4	JNB SHORT crackU2.004079D8
004079F4	31C9	XOR ECX,ECX
004079F6	83E8 03	SUB EAX,3
004079F9	72 0D	JB SHORT crackU2.00407A08
004079FB	C1E0 08	SHL EAX,8
004079FE	8A06	MOV AL,BYTE PTR DS:[ESI]
00407A00	46	INC ESI
00407A01	83F0 FF	XOR EAX,FFFFFFFF
00407A04	74 74	JE SHORT crackU2.00407A7A
00407A06	89C5	MOV EBP,EAX
00407A08	01DB	ADD EBX,EBX
00407A0A	75 07	JNZ SHORT crackU2.00407A13
00407A0C	8B1E	MOV EBX,DWORD PTR DS:[ESI]
00407A0E	83EE FC	SUB ESI,-4
00407A11	11DB	ADC EBX,EBX
00407A13	11C9	ADC ECX,ECX
00407A15	01DB	ADD EBX,EBX
00407A17	75 07	JNZ SHORT crackU2.00407A20
00407A19	8B1E	MOV EBX,DWORD PTR DS:[ESI]
00407A1B	83EE FC	SUB ESI,-4
00407A1E	11DB	ADC EBX,EBX
00407A20	11C9	ADC ECX,ECX
00407A22	75 20	JNZ SHORT crackU2.00407A44
00407A24	41	INC ECX
00407A25	01DB	ADD EBX,EBX
00407A27	75 07	JNZ SHORT crackU2.00407A30
00407A29	8B1E	MOV EBX,DWORD PTR DS:[ESI]
00407A2B	83EE FC	SUB ESI,-4
00407A2E	11DB	ADC EBX,EBX
00407A30	11C9	ADC ECX,ECX
00407A32	01DB	ADD EBX,EBX
00407A34	73 EF	JNB SHORT crackU2.00407A25
00407A36	7C 05	JMP SHORT crackU2.00407A11

הקוד הראשון שרץ:

ניתן לראות בבירור שלא מדובר במשהו טריוויאלי שנרצה לחקור.



# דחיסת קובץ ההרצה

## דחיסת UPX

```

00407B00 . 55      PUSH EBP
00407B01 . 57      PUSH EDI
00407B02 . FFDB   CALL EB3
00407B03 . 8D87 9F010000 LEA EAX,DWORD PTR DS:[EDI+19F]
00407B04 . 8020 7F    AND BYTE PTR DS:[EAX],7F
00407B05 . 8060 28 7F  AND BYTE PTR DS:[EAX+28],7F
00407B06 . 58      POP EAX
00407B07 . 50      PUSH EAX
00407B08 . 54      PUSH ESP
00407B09 . 50      PUSH EAX
00407B0A . 53      PUSH EBX
00407B0B . 57      PUSH EDI
00407B0C . FFDB   CALL EB3
00407B0D . 58      POP EAX
00407B0E . 61      POPAD
00407B0F . 8D4424 80  LEA EAX,DWORD PTR SS:[ESP-80]
00407B10 . 6A 00     PUSH 0
00407B11 . 3C04     CMP ESP,EAX
00407B12 . 75 FA     JNZ SHORT crackU2.00407B2A
00407B13 . 83EC 80   SUB ESP,-80
00407B14 . E9 F895FFFF JMP crackU2.00401130
00407B15 . 00      DB 00
00407B16 . 00      DB 00
00407B17 . 00      DB 00
00407B18 . 00      DB 00
00407B19 . 00      DB 00
00407B1A . 00      DB 00
00407B1B . 00      DB 00
00407B1C . 00      DB 00
00407B1D . 00      DB 00
00407B1E . 00      DB 00
00407B1F . 00      DB 00
00407B20 . 00      DB 00
00407B21 . 00      DB 00
00407B22 . 00      DB 00
00407B23 . 00      DB 00
00407B24 . 00      DB 00
00407B25 . 00      DB 00
00407B26 . 00      DB 00
00407B27 . 00      DB 00
00407B28 . 00      DB 00
00407B29 . 00      DB 00
00407B2A . 00      DB 00
00407B2B . 00      DB 00
00407B2C . 00      DB 00
00407B2D . 00      DB 00
00407B2E . 00      DB 00
00407B2F . 00      DB 00
00407B30 . 00      DB 00
00407B31 . 00      DB 00
00407B32 . 00      DB 00
00407B33 . 00      DB 00
00407B34 . 00      DB 00
00407B35 . 00      DB 00
00407B36 . 00      DB 00
00407B37 . 00      DB 00
00407B38 . 00      DB 00
00407B39 . 00      DB 00
00407B3A . 00      DB 00
00407B3B . 00      DB 00
00407B3C . 00      DB 00
00407B3D . 00      DB 00

```

UPX לא נועד למנוע Debugging ולכן קל להתמודד איתו.

קפיצה לקוד (לאחר החילוץ).





# דחיסת קובץ ההרצה

## חיפוש הקוד הדחוס

- כאשר יש לנו כתובת (בזיכרון) שמכילה את הקוד הרלוונטי, נרצה לחלץ אותו לקובץ EXE חדש, כדי שנוכל לנטר אותו
  - החילוץ בדרך כלל נעשה ע"י כלים שיוצרים קובץ PE שמכיל את הקוד
- הקובץ שנוצר אומנם בפורמט PE, אך לא בהכרח יכול Import table תקני
  - תזכורת: ל-Import table יש שני מצבים
    1. לפני פתיחת הקובץ – יכול טבלה עם שמות DLLs ופונקציות שעל מערכת ההפעלה לטעון
    2. לאחר פתיחת הקובץ – יכול טבלה עם כתובות. לא חייב להכיל שמות (לעיתים Packers ימחקו את השמות כחלק מתהליך ה-Anti-Debugging)
  - יש צורך לעבור ממצב 2 למצב 1, כלומר לעשות "Reverse" ל-Import table
    - במקרים מסובכים נצטרך לאתר בצורה ידנית את שמות הפונקציות ולתקן את ה-IAT בהתאם
    - במקרים אחרים ישנן תוכנות אוטומטיות שעושות עבורנו את העבודה



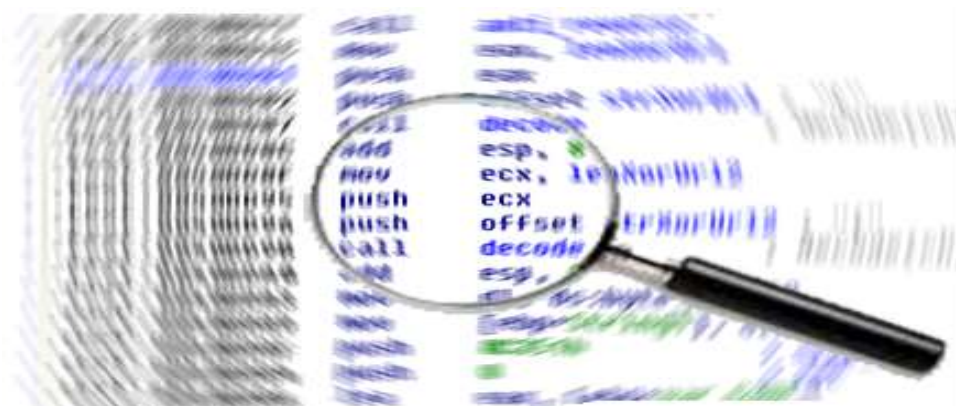
# דחיסת קובץ ההרצה

## סיכום

- ראשית נבדוק שאכן הקובץ דחוס/מוצפן
  - בדרך כלל מחסור ב-Import Table/Strings מרמז על כך
- ננסה לאתר את ה-Packer
  - האיתור יחשוף את הדרך הנכונה להתמודדות מולו
- במידה שה-Packer לא מוכר, נשתמש בכלים שלמדנו
  - נאתר שימוש ב-Anti-Debugging וננסה למנוע את מרבית השיטות
    - ע"י Hook, דיבוג איטי או דרכים אחרות
  - המטרה שלנו היא לתת למתודה המחלצת לרוץ
    - ולהתערב בקוד רק לאחר הסיום
  - אין דרך אחת לבצע זאת, יש להיות יצירתיים וסבלניים
  - דוגמאות לשיטות
    - תפיסת קפיצה/מעבר קוד בין Sections
      - הקוד האמיתי עלול להיות מחולץ בחלק אחר
    - Breakpoints על פונקציות בסיסיות שהקוד האמיתי עלול לקרוא להם
      - החזרה מהפונקציה מובילה לקוד



# עכירות?



# Anti-Debugging

- כאשר תוכנה מזהה שהיא רצה תחת כלי דיבוג או בוורטואליזציה, היא לעיתים תבחר
  - לעצור את הפעילות
  - לשנות אותה לחלוטין
  - להפעיל קוד לא רלוונטי
  - במטרה לבלבל את החוקר ולבזבז לו זמן יקר
- ישנם מספר הבדלים בין ריצה טבעית לבין ריצה בסביבת מחקר מסוימת
- Anti-Debugging מתייחס לניצול הבדלים אלו לטובת הזיהוי
  - נציג מספר שיטות בסיסיות וכיצד להתמודד מולם
  - ניתן למצוא את הטכניקות בחלק גדול מהנוזקות



# טכניקות מקובלות לזיהוי דיבוג

## דואליות

- קריאה ל-API:
  - isDebuggerPresent
  - CheckRemoteDebuggerPresent
  - NtQueryInformationProcess
- זיהוי לפי מדידת זמן
- חיפוש 3 int
- outputDebugString
- בדיקה מי התהליך האב (Parent Process)
- בדיקת קיום דיבגר
- זיהוי breakpoints



# IsDebuggerPresent

BOOL WINAPI IsDebuggerPresent(void)

If the current process is running in the context of a debugger, the return value is nonzero. (MSDN)

```

004042D4
004042D4 loc_4042D4:
004042D4 lea     eax, [ebp+var_8]
004042D7 push    eax
004042D8 push    esi
004042D9 mov     [ebp+var_8], esi
004042DC mov     [ebp+var_4], esi
004042DF call   ecx
004042E1 call   loc_402082
004042E6 call   ds:IsDebuggerPresent
004042EC xor     eax, eax
004042EE pop     edi
004042EF inc     eax
004042F0 pop     esi
004042F1 leave
004042F2 retn     8
    
```

IsDebuggerPresent (Windows 8)

```

MOV EAX,DWORD PTR FS:[30]
MOVZX EAX,BYTE PTR DS:[EAX+2]
RETN
    
```

IsDebuggerPresent (Windows XP SP3)

```

MOV EAX,DWORD PTR FS:[18]
MOV EAX,DWORD PTR DS:[EAX+30]
MOVZX EAX,BYTE PTR DS:[EAX+2]
RETN
    
```

typedef struct \_PEB {

```

BYTE Reserved1[2];
BYTE BeingDebugged;
BYTE Reserved2[1];
PVOID Reserved3[2];
PPEB_LDR_DATA Ldr;
PRTL_USER_PROCESS_PARAMETERS ProcessParameters;
BYTE Reserved4[104];
PVOID Reserved5[52];
PPS_POST_PROCESS_INIT_ROUTINE PostProcessInitRoutine;
BYTE Reserved6[128];
PVOID Reserved7[1];
ULONG SessionId;
} PEB, *PPEB;
    
```

הפתרון: תיקון הFlag.



# CheckRemoteDebuggerPresent

BOOL WINAPI CheckRemoteDebuggerPresent(  
    \_In\_ HANDLE hProcess,  
    \_Inout\_ PBOOL pbDebuggerPresent );

*pbDebuggerPresent* – A pointer to a variable that the function sets to **TRUE** if the specified process is being debugged, or **FALSE** otherwise.

```
BOOL isDebugged;  
CheckRemoteDebuggerPresent(GetCurrentProcess(), &isDebugged);  
If (isDebugged == TRUE) {
```

```
BOOL WINAPI CheckRemoteDebuggerPresent(HANDLE process, PBOOL DebuggerPresent)  
{  
    NTSTATUS status;  
    DWORD_PTR port;  
  
    if(!process || !DebuggerPresent)  
    {  
        SetLastError(ERROR_INVALID_PARAMETER);  
        return FALSE;  
    }  
  
    status = NtQueryInformationProcess(process, ProcessDebugPort, &port, sizeof(port), NULL);  
    if (status != STATUS_SUCCESS)  
    {  
        SetLastError(RtlNtStatusToDosError(status));  
        return FALSE;  
    }  
  
    *DebuggerPresent = !!port;  
    return TRUE;  
}
```

(ניתן להשתמש ישירות ב-  
(NtQueryInformationProcess

Source from Wine



# OutputDebugString

```
void WINAPI OutputDebugString( _In_opt_ LPCTSTR lpOutputString )
```

```
DWORD errorValue = 1234;  
SetLastError(errorValue);  
OutputDebugString("test");  
if(GetLastError() != errorValue) {  
    {  
        noDebugger on XP  
    }  
}
```

OutputDebugString שולחת מחרוזת להצגה ע"י הדיבגר, אם קיים כזה.

אם אין Debugger הפונקציה מחזירה שגיאה (lastError- את ה-).





# זיהוי לפי מדידת זמן

- ניתן לנצל את העובדה שריצה ב-debugger איטית יותר, לדוגמא

```
rdtsc // get time stamp counter to edx:eax
mov esi, eax
mov edi, edx
...
...
rdtsc
cmp edx,edi
ja Debugger
sub eax,edi
cmp eax,0x100
ja Debugger
Jmp noDebugger
```

- ניתן להשתמש גם ב-API למדידת זמנים  
▪ כגון `getLocalTime` / `getSystemTime` / `getTickCount` וכו'



## זיהוי קיום int 3

- כאשר מבצעים software breakpoint, ה-Debugger רושם את הפקודה int 3 לתוך הקוד
- אם התוכנה המדובגת תסרוק את הקוד בשביל לאתר בית המכיל 0xCC (כלומר int 3), היא תחשוף את ה-breakpoint

```
mov edi,0x400000  
mov ecx,size  
mov al,0xCC  
repne scasb  
cmp ecx,0  
jne Debugger
```



# בדיקה האם int 3 אורם לחריצה

- תזכורת: int 3 זו חריגה שבה מערכת ההפעלה מעבירה את השליטה ל Debugger

```
push handler
push fs:[0]
mov fs:[0],esp
mov eax,debh
int 3
pop fs:[0]
add esp,4
cmp eax,debh
je Debugger
jmp noDebugger
```

Handler:

```
mov eax, [esp+0x0c] // get the context record
mov [eax+0xb0],100h // eax in context record
inc [eax + 0xb8] // eip in context record
xor eax,eax
ret
```

- כלומר int 3 זה software breakpoint
- אם אין Debugger אז
- שימוש ב-int 3 יוצר חריגה רגילה



# שיטות נוספות לזיהוי דיבוק

- ניתן לעבור על כל התהליכים באמצעות CreateToolhelp32Snapshot על מנת לאתר תהליכים לא רצויים
  - תהליך לא רצוי כולל גם כלים נוספים, כגון wireshark
- רוב התהליכים במערכת נוצרים ע"י תהליך בשם explorer
  - אם התהליך שיצר את התוכנה אינו כזה, יש סיבה לחשוד שדיבגר יצר אותו
  - שיטה כזאת תגרום לעיתים קרובות ל-false-positive
    - לדוגמא הפעלה דרך cmd



# שיטות נוספות לזיהוי דיבוג

- ניתן לחפש האם מותקנות במחשב תוכנות מחקר שונות (למשל דרך סריקת תוכן מערכת הקבצים או ה-Registry)
- דוגמא למציאת תהליך ע"י שם החלון

```
if(FindWindow("OLLYDBG", 0) == NULL) {  
    //Debugger Not Found  
} else {  
    //Debugger Detected  
}
```

- כמובן שלא מדובר בשיטות מדויקות
- השיטות האלו נפוצות בעיקר לאיתור תוכנות שונות שאינן חלק מהדיבגר
  - שכן אין דרך אחרת לאתרן
  - לעומת Debugger שראינו רק חלק קטן מהשיטות לאתרן



# זיהוי מחשק וירטואלי

- רבים מניתוחי RE של נוזקות מבוצעים על מכונות וירטואליות
- לכן רבות מהנוזקות משנות את התנהגותן תחת וירטואליזציה
- תרגיל

- מצאו שתי דרכים לאיתור VMware
  - באמצעות פקודות פשוטות ב-cmd



# ליהוי מחשב וירטואלי

## • דוגמאות לשוני במחשב וירטואלי

- מחסור בדרייברים בסיסיים
  - כגון כרטיס קול
- קיום דרייברים ייחודיים
- חומרה מסוימת
- תוכן וקטור הפסיקות
- ערכים ב-Registry
- התנהגות שונה בפקודות מסוימות
  - בעיקר פקודות מורצות על המעבד
- תהליכים של מערכת הווירטואליזציה
- שינוי מבנים מסוימים במערכת ההפעלה
- זמני ריצה שונים



# זיהוי מחשב וירטואלי

- קל לזהות את VMware

- תהליכים לניהול המחשב הווירטואלי

- VMwareService, VMwareTray, VMware User

- כתובת MAC ייחודית

- כדי להקשות על זיהוי, אפשר ליצור patch עם שמות אחרים וכתובות אחרות

- ניתן להקשות על זיהוי, אך לא למנוע

- ישנן תוכנות וירטואליזציה שיותר קשה לאתרן

- במידה שרוצים למנוע לחלוטין אפשרות לזיהוי, צריך להריץ בסביבה טבעית

במחשב ייעודי

- אבל אז צריך להשקיע יותר בהתקנה מחדש כל פעם שנגרם נזק





# עכירות?



# הסוואה בתולעת הא'נטרנט (1988)

- כיצד התולעת הגנה על עצמה מפני מפעילי מחשב?
  - שינוי שם התהליך ל-sh ומחיקת הפרמטרים
    - בפלט של ps התהליך נראה כמו shell תמים
  - קריאת כל הקבצים לזיכרון ומחיקתם
    - אחרי ההדבקה אין סימנים על הדיסק
  - כל כמה דקות מתבצע `fork()`
    - הבן ממשיך לעבוד כעותק זהה, והאב מתאבד
      - לא ניתן לעקוב אחרי ה-pid
      - זמן החישוב של כל תהליך נשאר קטן
      - מניעת הקטנת עדיפות (כי תהליך שרץ הרבה זמן מאבד עדיפות)



# תלכורת: תהליך עיצוב מ"ה ק-PC

- הקוד הראשון שרץ במחשב הוא ה-BIOS
  - מכתובת מוגדרת מראש
  - תוכן ה-BIOS מגיע צרוב כחלק מהמחשב (ROM)
    - במחשבים מודרניים, ב-EEPROM
    - כלומר ניתן לצריבה חוזרת על פי האפשרויות שקבען היצרן
  - כולל מבחר פונקציות לניהול המחשב ולגישת קלט/פלט להתקנים
- ה-BIOS מבצע בדיקות אתחול, בודק אילו התקנים ודיסקים מחוברים, ומחליט מאיזה מהם לבצע הפעלה
  - יתכן שישאל את המשתמש



# תלכורת: תהליך עיצוב מ"ה ב-PC

- הוא פונה לדיסק הנבחר, וקורא ממנו את הסקטור הראשון בדיסק
  - נקרא Master Boot Record בדיסקים
    - או Boot Sector בדיסקים
  - בו יש פרטים נוספים לגבי הדיסק, חלוקתו למחיצות, וכו'
- ומפעיל ממנו קטע קוד קטן
  - שתפקידו להביא את קוד הטעינה של מערכת ההפעלה
  - בגלל קטנותו, בדרי"כ אינו מכיר את מבנה מערכת הקבצים
  - זה הקוד הראשון שמופעל, שניתן לשינוי על ידי מ"ה
    - ובתלות בהרשאות, על ידי תוכנות אחרות, למשל נוזקות
  - קוד זה קורא ממקום קבוע בדיסק את קוד הטעינה של מ"ה
    - ומפעיל אותו



# תלכורת: תהליך עליית מ"ה ק-PC

- קוד הטעינה של מ"ה ניגש למערכת הקבצים בדיסק
  - וטוען את מ"ה
  - יתכן שבכמה שלבים, בעזרת קודי טעינה נוספים
- מ"ה מפעילה תוכנה על פי מה שהוגדר לה
  - כולל הפעלה אוטומטית של תכניות בזמן עליית מ"ה
  - ובהמשך, על פי בחירת המשתמש



## הדלקת ה-Boot Sector

- וירוס שמדביק את ה-boot sector
  - מחליף את תוכן ה-boot sector בקוד משלו
  - שומר את התוכן המקורי של ה-boot sector
    - במקום אחר בדיסק
    - כדי שיוכל להפעיל באמצעותו את מ"ה
- בזמן הדלקת המחשב, הקוד החדש ב-boot sector מופעל
  - ומפעיל את הוירוס
  - שבתורו מפעיל את מ"ה
  - בעצם הוירוס הוא מערכת על
    - ומערכת ההפעלה מופעלת על ידו



# הדבקת ה-Boot Sector

- היות שהוא נטען לפני מ"ה

- היא לא יכולה למנוע את עלייתו

- ולא יודעת על קיומו

- הוא יכול לשנות את הטעינה שלה, ואת מה שהיא עושה

- היות שכל הקריאה שלה מהדיסק נעשית דרך שירותים שהוא יכול להשתלט עליהם לפני שהיא עלתה

- והיא לא יכולה לבדוק אם הוא שינה, בלי עזרתו

- "בעייתיות":

- כאן הוירוס מסתתר בתוך הזיכרון הנגיש למ"ה

- ואם היא תחשוד בקיומו, היא תוכל לזהות אותו



## הדבקה ה-Boot Sector

- צורת הדבקה זו הייתה מאד נפוצה בתקופה שמידע דיגיטלי הועבר בדיסקטים
  - בעלת יתרונות רבים
  - אבל קשה לביצוע כיום
    - יש הגנות על כתיבה ל-boot sector של דיסקים קשיחים
    - וכמעט שאין אתחול של מחשבים מהתקנים נתיקים





# שיטות הסתתרות נוספות

- וירוסים בנתונים : ב-Word, Postscript, וכו'
  - שהם שפת תכנות
- rootkits
- וירוסים פולימורפיים ומטאמורפיים
- ועוד



# וירוסים מפורסמים



# Stuxnet

- וירוס לתקיפת מערכות חלונות המחברים לבקרים תעשייתיים של סימנס
- כנראה יועד כנגד אירן
  - אבל הדביק גם תחנות גרעיניות ברוסיה ובעולם
  - אפילו הדביק את תחנת החלל הבינלאומית
    - כנראה דרך דיסק USB
- חיפש מחשבים עם תוכנת Step7 וחיבור ל-PLC של סימנס
  - מותנה בחיבור לצידים מסוימים, ובסיבוב בתדרים מסוימים
  - כשמצא כאלה – ניסה להשיג גרסה מעודכנת שלו
  - לא הפעיל את עצמו על מחשבים אחרים

# life מטרות הדבקה

- הדבקת מחשבי חלונות
  - במגוון דרכים
    - מתוכנן שתיים דרך הדבקה מדיסק USB
  - מטרה: לאפשר את שתי התקיפות הנוספות
- הדבקת מערכות ה-SCADA (תוכנה של סימנס)
  - כדי להציג כאילו הציוד פועל תקין, ולשלוט על הבקרים
  - על ידי התקנת DLL, ושימוש בחולשת 0-day של תוכנת סימנס
- שכתוב תוכנת בקרי ה-PLC
  - כדי לשלוט ישירות על הציוד, ולגרום נזק לציוד



PLC של סימנס - S7-300



הנדסה לאחור - חורף תשפ"א

© פרופ' אלי ביהם, אביעד כרמל, עמר קדמיאל



# Stuxnet

- מחולק על פני מספר רב של קבצים במחשב!!!
- משתלט על מערכת הקבצים
  - ומסתיר קבצי LNK בגודל 4171 בתים
  - ע"י שימוש בחולשה שמאפשרת להעלים אותם מסייר חלונות
- נחתם על ידי סרטיפיקט שנגנב מ-Verisign
- מפעיל בקרים תעשייתיים של סימנס
  - ושולט על צנטריפוגות
  - מסובב אותן מהר מידי לזמנים קצרים
    - באופן שלא יזוהה מה גורם לבעייה הזו
  - עד שהן מתקלקלות



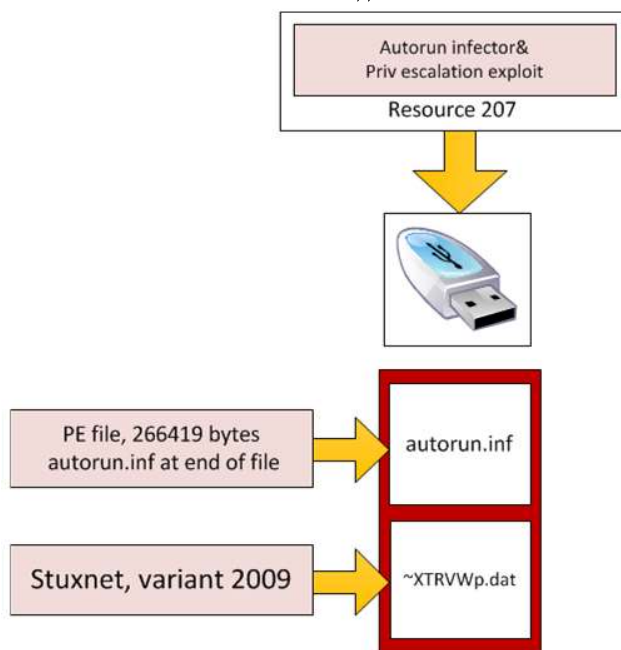
# הדבקה ע"י Autorun.inf

## • קובץ Autorun.inf

- מכיל קובץ PE
- שבהמשכו יש נתוני autorun.inf
- באופן שהוא מורץ בעצמו כקובץ הרצה בזמן חיבור ההתקן
- אנטי-וירוס חיפש נתוני בקרה של autorun רק בהתחלה
  - ולכן לא זיהה...

## • בנוסף קובץ שנראה קובץ זמני

- שם יש את התוכנה עצמה



Kaspersky Lab, [www.securelist.com](http://www.securelist.com)  
הנדסה לאחר - חורף תשפ"א

© פרופ' אלי ביהם, אביעד כרמל, עמר קדמיאל

29.06.2023



## שימוש P2P

- בנוסף לגישה לשרת C&C, יש ל-Stuxnet יכולת תקשורת בין עותקים של הוירוס
  - ברשת P2P
    - Peer-to-Peer
  - מאפשר לוירוס להתעדכן אם גילה שכן עם גרסה חדשה יותר
- באמצעות רשת כזו, ניתן להתגבר על גילוי שרת ה-C&C
  - שכן ניתן להעביר פקודות, ותוכנה מעודכנת, גם כשהשרת לא פועל
    - וכשהשרת נחטף על ידי גורם זר



# Flame

- וירוס שמטרתו לאסוף חומר ממחשבים, ולשלוח אותו לבעל הוירוס
  - דרך שרתי Command & Control
- אוסף חומר מסוגים שונים
  - קבצים
  - צילומי מסך
  - הקלטות אודיו מהמיקרופון
  - ועוד
- התפשט במדינות במזרח התיכון





# הסולאה

- Flame מחולק על פני כמה קבצי DLL
  - מקשה על זיהוי, ומקשה על דיבוג
- מסתיר את עצמו בדפי זיכרון של תהליכים אחרים
  - עם הזרקת קוד
  - והרשאות דפים שמונעות מאפליקציות ב-user mode לגשת אליהם או להריץ מהם
  - המודולים אינם מופיעים בליווח של המודולים של האפליקציות
    - וגם לא ברשימות התהליכים
- מזהה איזה תוכנות אנטי-וירוס מותקנות
  - ומתאים את עצמו כדי למנוע מהן לזהות אותו
    - למשל על ידי שינוי סיומות שמות הקבצים



# מאון ספריות תוכנה

- גודלו ענק : 20MB

- כולל מגוון ספריות לתמיכה

- בחמישה סוגי הצפנה,
- כמה סוגי דחיסה,
- SQLite,
- מכונה וירטואלית,
- ועוד

- התקנות במ"ה

- משתמש ב-registry
- מתקין דרייבר אודיו מזויף
  - משמש כעוגן נגד מחיקתו מהמחשב



## תכונות נוספות

- **Flame מסוגל לזהות התקני בלוטות פועלים בסביבתו**
  - ולאסוף עליהם נתונים
  - להתחבר אליהם ולקבל מידע
    - כולל גרסאות מעודכנות
    - כולל מכמה קילומטרים, אם משתמשים באנטנה כיוונית מיוחדת
- **מסוגל להפוך את הבלוטות שלו למזוהה**
  - ולקודד מידע על הסטטוס של עצמו בפרטי הזיהוי של הבלוטות
- **צילומי המסך נעשים**
  - מידי פעם בפעולה רגילה
  - וכן כאשר פועלות תוכנות "מעניינות"
    - למשל תוכנות מסרים מיידיים
- **מסוגל למחוק עצמו לבקשת בעליו**

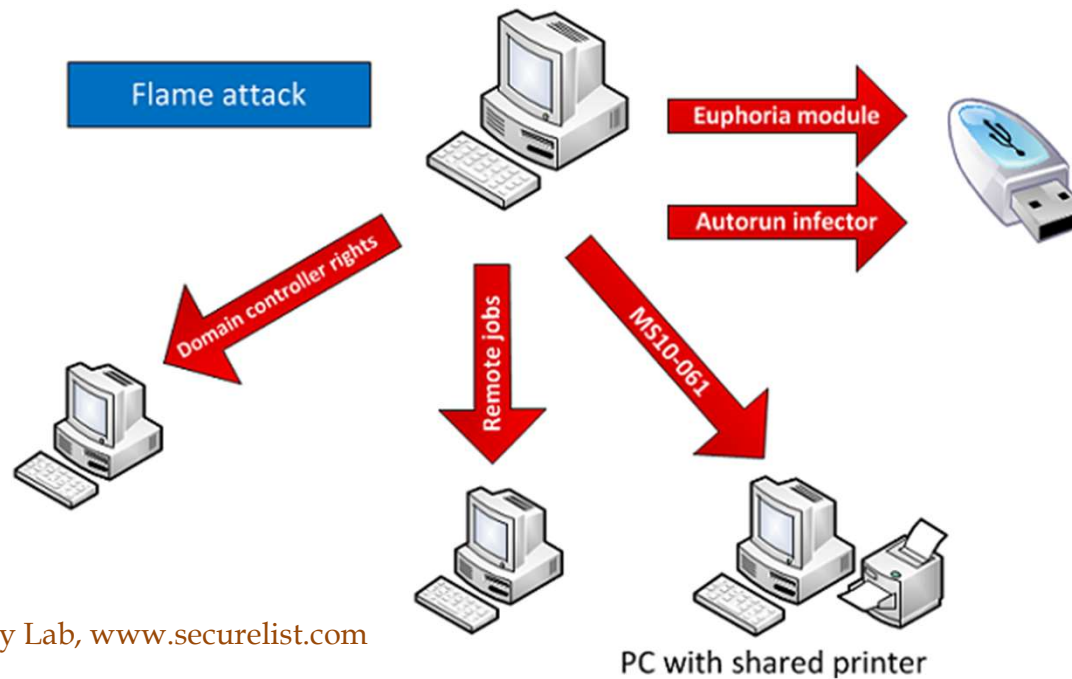


# דרכי ההדבקה Flame

- Flame כנראה הופץ במקור על ידי התחזות לשרת windows Update
- ואז השתמש בחמש דרכי הדבקה לצורך התפשטות
  - שתיים מהן להדבקה דרך דיסק USB (לא ברור אם היו בשימוש)
    - הדבקה דרך Autorun.inf
    - כמו ב-Stuxnet, עם shell32.dll
    - Euphoria
    - בשימוש בקבצי LNK לתיקיות
    - מאפשר להפעיל קבצי הדבקה מהתיקיות
  - ושלוש מהן לשכפול ברשת המקומית
    - בשימוש בחולשה של פרוטוקול הגישה למדפסות ברשת
    - vulnerability MS10-061
    - גם היא הייתה בשימוש על ידי Stuxnet
    - Remote jobs (כלומר RPC)
    - בשימוש בהרשאות ניהול מרחוק בדומיין (אם יש לו)
    - יוצר חשבון במחשבים אחרים ומעתיק עצמו לשם



# דרכי ההדבקה של Flame



# אניצת אילוי צי"י ליוף סרטיפיקט

- חלונות מגנה כנגד דרייברים ותוכנות מתחזות עי"י חתימה דיגיטלית
  - לכן, כדי לא להתגלות, צריך חתימה עם סרטיפיקט מורשה
    - אבל שאינו מזהה את התוקף
- Flame מתחזה לתוכנה שנחתמה דיגיטלית על ידי מיקרוסופט
  - מנצל שירות חתימות לא מעודכן של מיקרוסופט
    - שעדיין השתמש ב-MD5
    - ושמאפשר חתימה על תוכנה
  - דרכו הצליח לזייף סרטיפיקט לחתימה על תוכנה
    - בשימוש בהתקפה קריפטוגרפית
- לשם כך, זויף סרטיפיקט
  - בשימוש בפונקציית התמצות MD5 השבורה
  - בגרסת שבירה מיוחדת של MD5 לצורך זיוף סרטיפיקטים
  - הרעיון פורסם במקור על ידי חוקרים מהולנד
    - וכותבי Flame השתמשו בו לצרכיהם, עם שיפורים משלהם



# הסרטיקט האלוף – ע"י שפירט MD5

Version	V3
Serial number	3a ab 11 de e5 2f 1b 19 d0 56
Signature algorithm	md5RSA
Signature <a href="#">hash algorithm</a>	md5
<a href="#">Issuer</a>	CN = Microsoft Root Authority,OU = Microsoft Corporation,OU = Copyright (c) 1997 Microsoft Corp.
Valid from	Thursday,10 December 2009 11:55:35 AM
Valid to	Sunday,23 October 2016 6:00:00 PM
Subject	CN = Microsoft Enforced Licensing Intermediate PCA,OU = Copyright (c) 1999 Microsoft Corp.,O = Microsoft Corporation,L = Redmond,S = Washington,C = US
<a href="#">Public key</a>	30 82 01 0a 02 82 01 01 00 fa c9 3f 35 cb b4 42 4c 19 a8 98 e2 f4 e6 ca c5 b2 ff e9 29 25 63 9a b7 eb b9 28 2b a7 58 1f 05 df d8 f8 cf 4a f1 92 47 15 c0 b5 e0 42 32 37 82 99 d6 4b 3a 5a d6 7a 25 2a 9b 13 8f 75 75 cb 9e 52 c6 65 ab 6a 0a b5 7f 7f 20 69 a4 59 04 2c b7 b5 eb 7f 2c 0d 82 a8 3b 10 d1 7f a3 4e 39 e0 28 2c 39 f3 78 d4 84 77 36 ba 68 0f e8 5d e5 52 e1 6c e2 78 d6 d7 c6 b9 dc 7b 08 44 ad 7d 72 ee 4a f4 d6 5a a8 59 63 f4 a0 ee f3 28 55 7d 2b 78 68 2e 79 b6 1d e6 af 69 8a 09 ba 39 88 b4 92 65 0d 12 17 09 ea 2a a4 b8 4a 8e 40 f3 74 de a4 74 e5 08 5a 25 cc 80 7a 76 2e ee ff 21 4e b0 65 6c 64 50 5c ad 8f c6 59 9b 07 3e 05 f8 e5 92 cb d9 56 1d 30 0f 72 f0 ac a8 5d 43 41 ff c9 fd 5e fa 81 cc 3b dc f0 fd 56 4c 21 7c 7f 5e ed 73 30 3a 3f f2 e8 93 8b d5 f3 cd 0e 27 14 49 67 94 ce b9 25 02 03 01 00 01
Enhance key usage	Code Signing (1.3.6.1.5.5.7.3.3) Key Pack Licenses (1.3.6.1.4.1.311.10.6.1) License Server Verification (1.3.6.1.4.1.311.10.6.2)
Authority identifier	Certificate Issuer: CN=Microsoft Root Authority, OU=Microsoft Corporation, OU=Copyright (c) 1997 Microsoft Corp.   Certificate SerialNumber=00 c1 00 8b 3c 3c 88 11 d1 3e f6 63 ec df 40
Subject key identifier	6a 97 e0 c8 9f f4 49 b4 89 24 b3 e3 d1 a8 22 86 aa d4 94 43
Key usage	Digital Signature, Certificate Signing, Off-line CRL Signing, CRL Signing (86)
Basic constraints	Subject Type=CA, Path Length Constraint=None
Thumbprint algorithm	sha1
Thumbprint	2a 83 e9 02 05 91 a5 5f c6 dd ad 3f b1 02 79 4c 52 b2 4e 70

(ויקיפדיה)

הנדסה לאחור – חורף תשפ"א

© פרופ' אלי ביהם, אביעד כרמל, עמר קדמיאל

29.06.2023



# APT





# APT

## Where are all the 'A's in APT?

Posted by  on  Sep 20, 2018

*In a guest blog post by VB2018 gold partner Kaspersky Lab, Costin Raiu, Director of the company's Global Research and Analysis Team, looks critically at the 'A' in APT.*

### So what is missing?

Looking at the discussions and development of sophisticated attack techniques, there is a significant difference between the theory and in-the-wild observations. So what is missing? Here's a list of possible culprits:

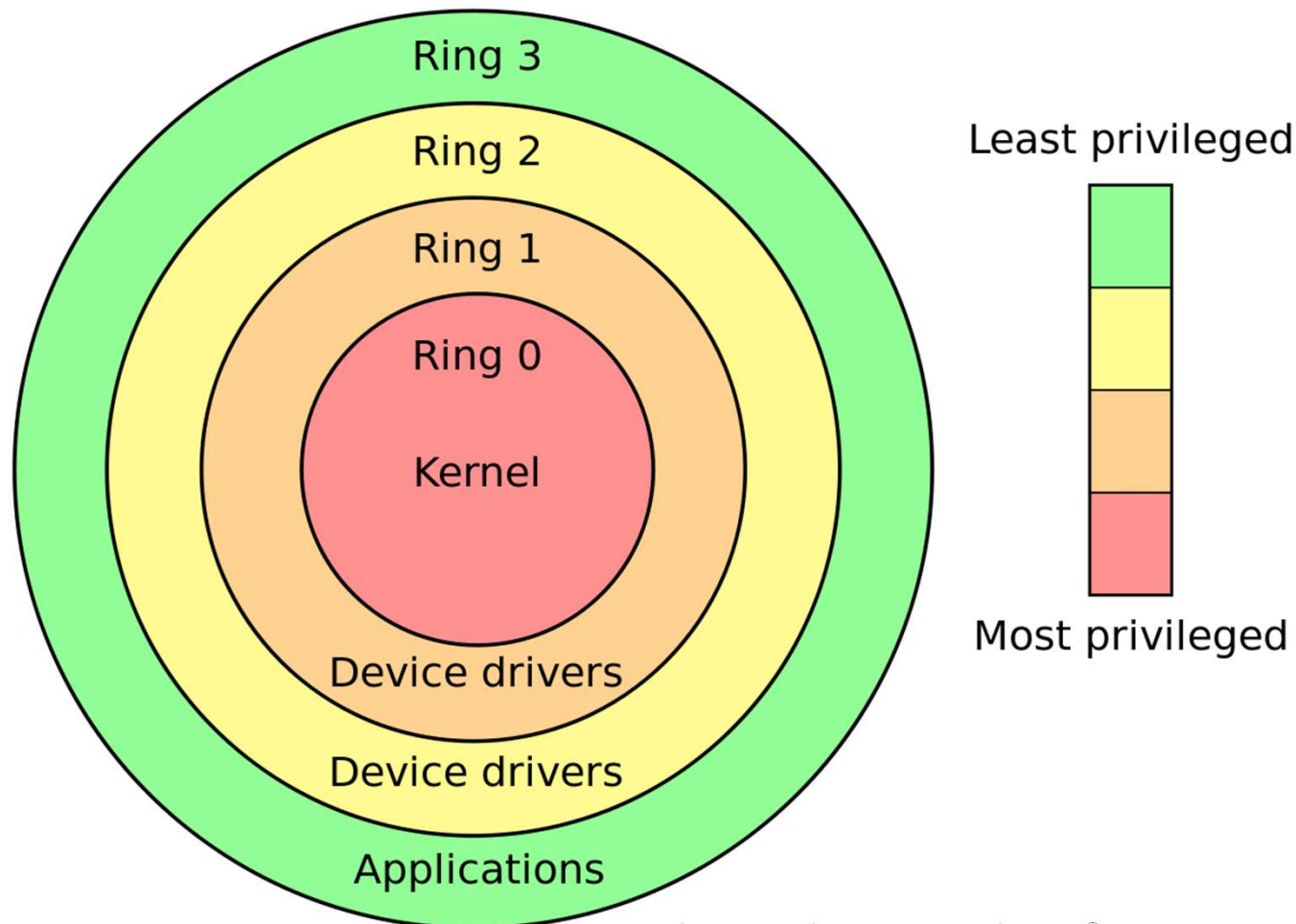
- Virtualization / hypervisor malware – although the infamous Blue Pill was discussed as far back as 2006, we haven't seen any in-the-wild (ItW) attacks leveraging this.
- SMM malware – although Dmytro Oleksiuk, a.k.a. Cr4sh, developed an **SMM backdoor** as far back as 2015, this is something yet to be seen in real-world attacks.
- UEFI malware – the hacking of HackingTeam revealed that a UEFI persistence module has been available since at least 2014, but we have yet to observe real-world UEFI malware.
- Hardware implants – although **Joe Fitzpatrick** and others have covered this subject in great detail, the number of real-world cases where hardware implants have been found is extremely low.

*Defensive attacks against hardware – the Stuxnet payload is an example, but also BlackEnergy attacks in which the firmware was overwritten with trash.*

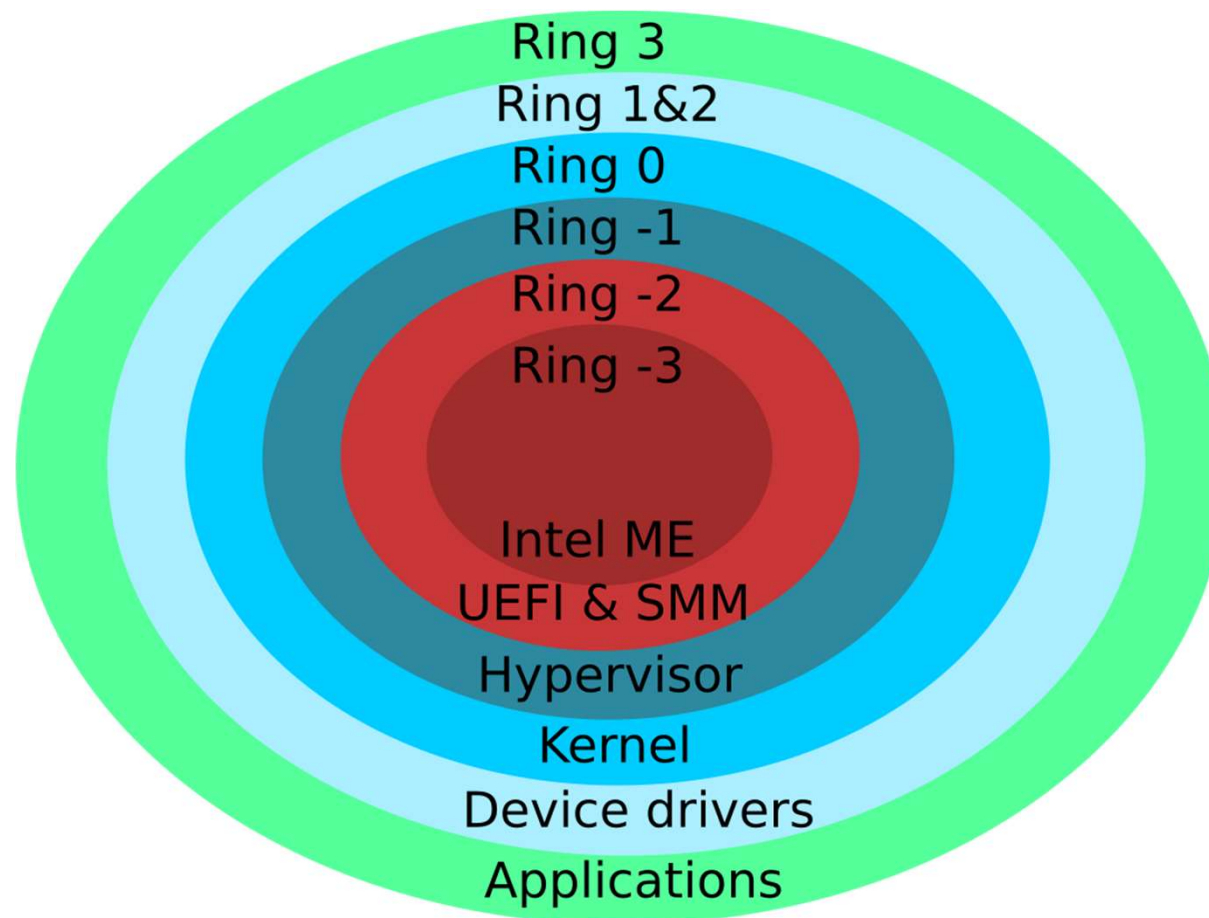
- Infection of pro-level network hardware such as core routers – **SYNful Knock** being a good example.
- Supply chain attacks – such as **Shadowpad** and the CCleaner compromise, both launched by the same APT group.
- The development of multi-platform malware – for instance WildNeutron using malware for *Windows*, *MacOS X* and *Linux*.
- World-class crypto attacks – as seen in Flame.



# Protection Rings



# Real Protection Rings



## תלכארת: היפרווייזור

- hypervisor הוא מערכת שמריצה תחתיה את המחשבים הווירטואליים (ומערכות ההפעלה שלהם) במערכות וירטואליות
  - למשל VMWARE
  - מקום מעולה לנוזקה להסתתר בו
- למשל, כשאתם מריצים בחוות המחשבים,
  - לכל אחד מכם יש מכונה וירטואלית משלו
    - "מתחזה" למחשב אמיתי עם מ"ה וכל הציוד הנדרש
  - כל המכונות הווירטואליות רצות תחת מחשב אחד
    - בחדר השרתים הפקולטי
    - בעל זיכרון גדול ומרובה ליבות
  - וההיפרווייזור זו מ"ה-על
    - שמריצה תחתיה את המכונות הווירטואליות
    - מחליטה מי מהן תרוץ באיזה ליבה ומתי
    - מבצעת context-switching בין המכונות הווירטואליות
    - מקצה להן זיכרון, ופונה עבורן להתקני קלט-פלט



# Blue Pill

- Blue Pill זה תוכנה שמסתתרת כהיפרווייזור
  - ולכן מ"ה ואנטי-וירוסים שרצים תחתיו לא רואים אותו

- כש-Blue Pill מופעל הוא הופך ל-hypervisor
  - כלומר משתלט על המחשב
  - משתלט על וקטור הפסיקות, והגישה להתקנים
  - מפעיל את שירותי הווירטואליזציה של המעבד
  - ומריץ תחתיו בווירטואליזציה את מערכת ההפעלה המקורית
  - תוך בקרה ושליטה על כל פעולותיה



## Blue Pill

- מערכת ההפעלה שומרת על הגישה הקיימת להתקנים ולקבצים
  - אבל הגישה בפועל מבוקרת ע"י ההיפרוייזור
  - כל גישה חדשה נעשית באמצעות ההיפרוייזור
    - כולל פסיקות חומרה
    - בקשות לנתונים
    - כולל גם שעון המערכת
- וכמובן ההיפרוייזור מזייף כלפי מ"ה כאילו הוא החומרה המקורית



# Blue Pill

- קשה מאד לזהות את קיום Blue Pill מתוך מ"ה שרצה תחתיו
  - אם היישום בוצע בזהירות המתבקשת
    - כי Blue Pill שולט על התשובות שמ"ה מקבלת בגישה לחומרה
  - אבל כנראה לא בלתי אפשרי





# Malware Buried Deep Down the SPI Flash: Sednit's First UEFI Rootkit Found in the Wild

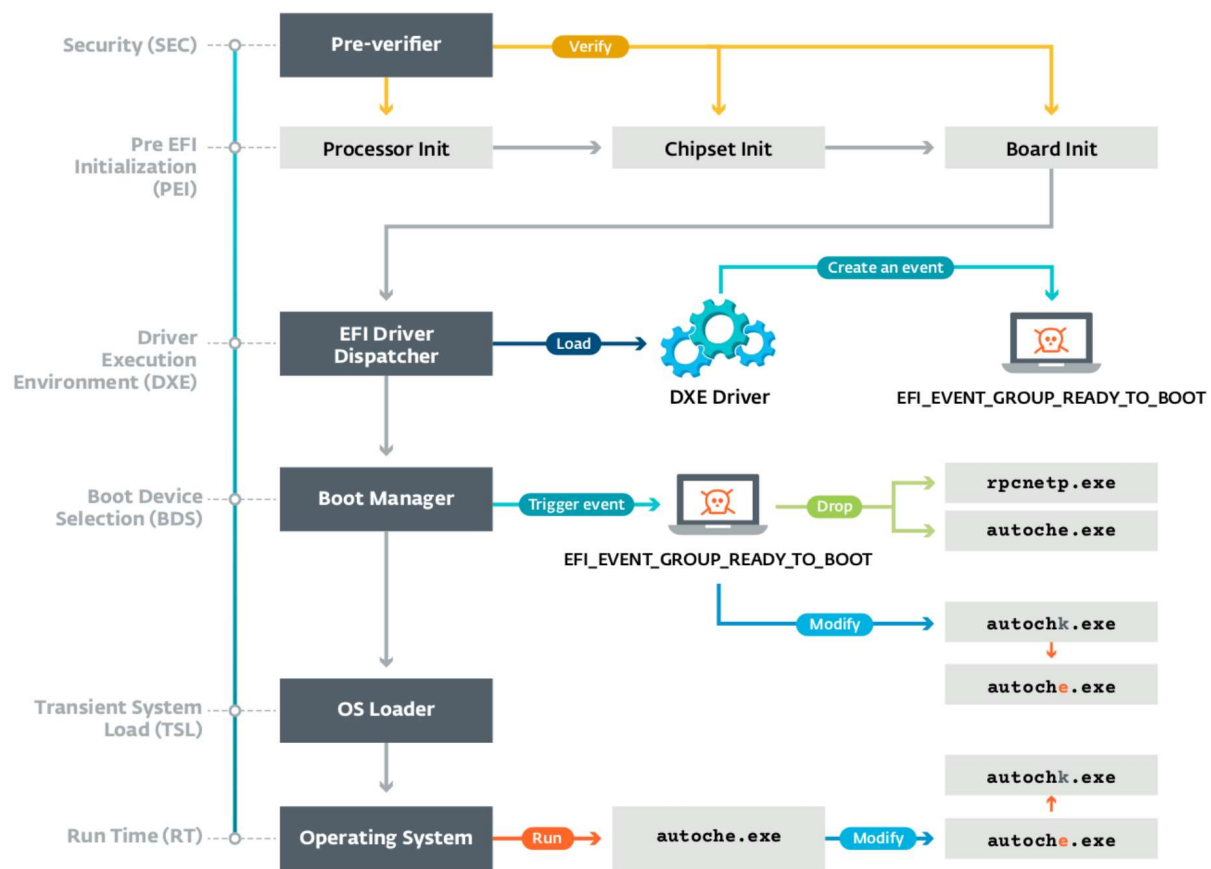
Jean-Ian Boutin | Senior Malware Researcher

Frédéric Vachon | Malware Researcher





# UEFI Rootkit Workflow



# הסתתרות בקדיסק קשיח

- הסתתרות בקושחה של דיסק קשיח
  - זו שיטה שאהובה על גופים בעלי יכולת לכך
  - מאפשרת החזרת תוכן דיסק לגרסה הדבוקה
    - גם אם מ"ה שלחה פקודת כתיבה לדיסק לתיקון התוכן
    - למשל תוכן boot sector או תוכן של קבצי מ"ה
- לשם שינוי הקושחה צריך יכולת כתיבת קושחה
  - והשתלתה בדיסק
  - תוך עקיפת אמצעי ההגנה על הקושחה (אם יש כאלה)
  - לא קל, אבל אפשרי



# הסתתרות בחיבור USB

- מתוך קטלוג מוצרי הריגול של ה-NSA
  - שהודלף ע"י Snowden

TOP SECRET//COMINT//REL TO USA, FVEY




## COTTONMOUTH-I

### ANT Product Data

(TS//SI//REL) COTTONMOUTH-I (CM-I) is a Universal Serial Bus (USB) hardware implant which will provide a wireless bridge into a target network as well as the ability to load exploit software onto target PCs.

08/05/08



**COTTONMOUTH - 1**

Status: Availability – January 2009      Unit Cost: 50 units: \$1,015K

POC: [REDACTED], S3223, [REDACTED], [REDACTED]@nsa.ic.gov  
ALT POC: [REDACTED], S3223, [REDACTED], [REDACTED]@nsa.ic.gov

Derived From: NSA/CSSM 1-52  
Dated: 20070108  
Declassify On: 20320108

TOP SECRET//COMINT//REL TO USA, FVEY

## הסתתרות בחיבור USB

- לא ברור מה בדיוק מופעל שם
  - האם זה רק חיבור לרשת שמאפשר גישה USB פנימה
  - או שיש בו רכיב ממוחשב עם תוכנה
- בכל מקרה רוב האנשים לא חושדים בכבל USB
- חיבור כזה מאפשר לבצע מה שכל חיבור USB יכול, למשל
  - התחזות למקלדת ועכבר
  - התחזות לצג
  - התחזות לדיסק
- עם התקנה אוטומטית של דרייברים ע"י מ"ה ברגע שההתקן הזה מעוניין לפעול
  - חלומו של כל וירוס...



# עכירות?

