הפקולטה למדעי המחשב
הטכניון

הנדסה לאחור

236496

מרצה: עמר קדמיאל
מתרגל: טל שנקר

© Eli Biham, Omer Kadmiel and Aviad Carmel

ידע נדרש

- שפות עיליות
  - C
- אסמבלי
  - Intel 32-bit IA32 assembly
    - ילמד בקצרה בתרגול
    - לא נתייחס לגרסאות 64 סיביות
- ידע בסיסי במערכות הפעלה
  - הבנת דרך הפעולה של מערכת הפעלה והמבנה שלה
  - השימוש בחלונות, כולל תכנות ודיבוג
  - ניסיון כמשתמש וירטואליזציה
- ידע בסיסי באבטחת מידע
  - וחולשות אבטחה

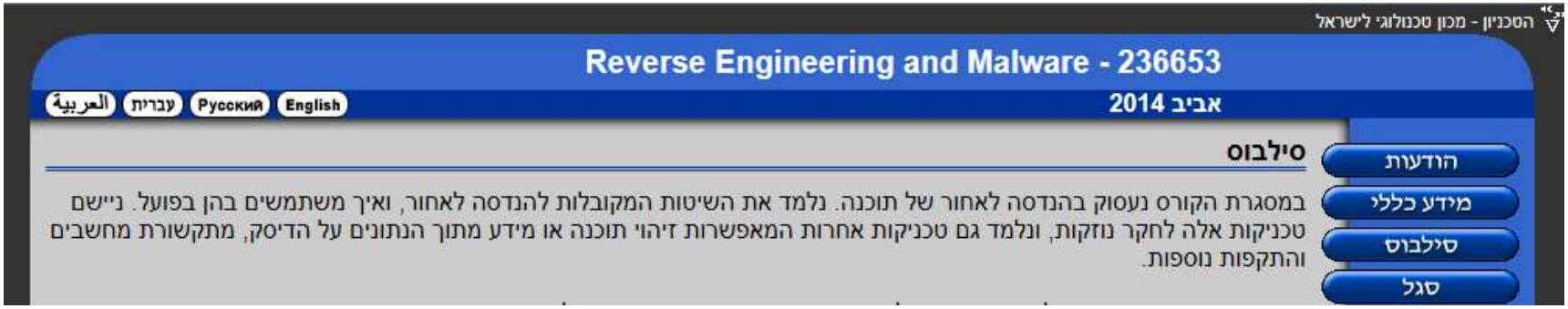
קדמים

- קדם חובה
  - מערכות הפעלה 234123 או 046209 או שקול
- צמוד חובה
  - הגנה במערכות מתוכנתות (הגנה ברשתות) 236350
  - או אבטחת מחשבים 236652 או 236653 או 236607
  - או השלמה עצמית של שיעור חולשות ב-236350
    - תוך נוכחות בהרצאה ובתרגול בנושא (בסביבות השבוע השלישי)
- מומלץ
  - קומפילציה
- כדאי לזכור
  - אתי"מ

הפצת מידע על המקצוע

- הודעות וציונים יועברו דרך מערכת GR
  - עליכם לוודא רישום במערכת GR כדי לקבל הודעות ושנוכל להזין לכם ציונים
    - זה אמור להתבצע אוטומטית למי שרשום רשמית למקצוע
  - הירשמו ליומן של GR, וכך הסדנאות, הבוחן, המבחנים ותרגילי הבית יופיעו לכם אוטומטית ביומן שלכם
- שקפי ההרצאות והתרגולים, פרטי צוות המקצוע ושעות קבלה, הודעות, ציונים, ופרטים נוספים באתר המקצוע

http://webcourse.cs.technion.ac.il/236653/



סדנאות

- יתקיימו מספר סדנאות בכיתה
  - בשעות של השיעורים והתרגולים
  - בשאיפה – אחת לשלושה שבועות

- הביאו את המחשבים הניידים שלכם
  - במקרה הצורך אפשר להפעיל את המחשוב הווירטואלי הפקולטי מהנייד שלכם

מטלות המקצוע

- המטלות במקצוע כוללות כ-4-6 תרגילי בית
  - כולם או רובם רטובים
  - כולל תרגילים שדורשים השקעה רבה
  - משקלים לא זהים
  - בזוגות
  - ציון תקף
  - אין הגשה באיחור
    - במקרים מוצדקים (מילואים וכו') חובה לבקש אישור מהמתרגל האחראי לפחות 24 שעות מראש
- מבחן – בשני חלקים בני שלוש שעות (עם הפסקה ביניהם)
  - יש לגשת לשני החלקים באותו מועד
- ציון סופי
  - מבחן 70%, תרגילים 30%
  - ציון מבחן נמוך מ-50 לא ישוקלל עם ציוני התרגילים

יושר אקדמי

- העבודה היא שלך – כתוב אותה בעצמך
  - אין להעתיק מאחרים
  - אל תשתמש בניתוחים קודמים של החומר אותו אתה מנתח
    - אחרת הציון יגיע למי שכתב אותו
  - אין להשתמש בדה-קומפיילרים
  - הפתרונות שלכם מהתחלה ועד הסוף
  - צטט מקורות
- ואם בכל זאת עשית משהו אסור – אמור זאת בפירוש

- העוברים על הכללים יוענשו בחומרה

התכוננות לשיעורים

- אתם מתבקשים להתכונן לשיעורים
  - על ידי קריאת שקפי השיעור הבא לפני השיעור
- כך נוכל לדון בעקרונות, ולא לעסוק בפרטים משניים

ספרות

- רשימת הספרות נמצאת באתר המקצוע
  - ספרים נמצאים בספריה

- בנוסף, קיים חומר רב באינטרנט
  - בפרט, תיאור מלא של אסמבלי של אינטל
    - לינק באתר המקצוע

פרק 1

מבוא להנדסה לאחור

מבוא

הנדסה לאחור היא תהליך של גילוי עקרונות טכנולוגיים והנדסיים של מוצר דרך ניתוח המבנה שלו ואופן פעולתו.

Reverse engineering is the process of discovering the technological principles of a device, object, or system through analysis of its structure, function, and operation.

הנדסה לאחור (RE) גם כוללת גילוי של מבני פרוטוקולים ושל קוד תוכנה, וכן גילוי מידע לא ידוע על מערכת ע"י ניתוח המבנה שלה ואופן פעולתה.

הנדסה לאחור של מכונות

- בעידן התעשייתי, עוד לפני עידן המחשב, הנדסה לאחור הייתה נפוצה מאד
  - עוד לפני חוקי הפטנטים וזכויות היוצרים
- קל יחסית לבצע הנדסה לאחור של מכונות
  - רכיבים פיסיים גדולים
  - אבל לעיתים מסתמכים על חומרים ייחודיים
    - בעלי תכונות מסוימות
    - למשל בשעונים

© פרופ' אלי ביהם, אביעד כרמל, עמר קדמיאל

הנדסה לאחור של חומרה

- בעידן המחשב – מסובך יותר – בגלל הקוטן והמורכבות
  - והקושי של פירוק למרכיבים
- אבל לא בלתי אפשרי
  - יתכן שהצורה של המוצר מדליפה מידע או שניתן לזהות כזה באופן ויזואלי
  - במעבדים מודרניים:
    - מיקרוסקופים ומיקרוסקופים אלקטרוניים
  - מדידת מתח בזמן הפעולה
  - הזרקת מתח למעבדים ומעקב אחרי התגובה
  - גרימה לשגיאות בחישוב, ומעקב אחרי השינויים
  - קריאת זיכרון, או האזנה ל-bus, אם אפשר
  - ועוד

הנדסה לאחור של תוכנה

- הבנת קוד תוכנה בשפה עילית
- הבנת קבצי הרצה (כגון EXE או a.out)
  - אנליזה סטטית – דה-קומפילציה
  - אנליזה סטטית – שפת מכונה
  - אנליזה דינמית – דיבגרים

נתרכז במקרים הללו רוב הזמן

- קוד ביניים ו-JIT – Java bytecode & .NET
- רלוונטי גם ל-
  - אופטימיזציה של קוד ע"י המהדר
    - בהינתן קוד המקור או ייצוג פנימי של המהדר
  - Post-link optimization
    - כלומר אופטימיזציה בהינתן קבצי הרצה
  - בדיקות אבטחה של קוד
    - תלויות במה הקוד עושה

הנדסה לאחור: פרוטוקולים וקבצים

- הבנת המבנה של פרוטוקולי תקשורת וקבצים
- טכניקות יכולות לשלב
  - האזנה לתקשורת (sniffing)
  - RE של התוכנה המתקשרת
  - שינוי נתונים שנשלחו, או משלוח חבילות חדשות, ומעקב אחרי התגובות
  - בחינת קבצים
- בכל המקרים הללו החוקר מתנהג כבלש: בוחן עדויות, חוקר, מחפש מידע חדש, מנתח מידע שהשיג, ואפילו משנה תוכן ועוקב אחרי השינויים

מניפולציות ו–Hooking

- מניפולציות מאפשרות לחוקר להבין טוב יותר את המערכת הנבחנת ע"י
  - הוספת קוד לצורך דיבוג
  - משלוח הודעות נוספות לבחון את התגובה
- אבל גם כדי לשנות את פעולת המערכת
  - תיקון באגים, חולשות אבטחה, או התנהגות אחרת במערכות בהן אין לנו קוד מקור
  - הדבקת קובץ הרצה קיים על ידי וירוס
  - הוספת פונקציונליות לקוד קיים או לספריות מערכת
    - עבורו אין לנו קוד מקור

למה הנדסה לאחור?

- מחקר תוכנות תקיפה לצורך פיתוח אמצעי הגנה כנגדם
  - למשל לצורך Antivirus
- חיפוש חולשות – לצורך פיתוח הגנות כנגדן
- Interoperability
  - צורכי פיתוח – לעיתים יש לבדוק את המערכת שעבורה כותבים קוד (לא הכל מתועד)
  - ניפוי הקוד שכתבנו
  - זיהוי מגבלות
  - הרחבה או שינוי בתוכנה שקשה לבצע דרך שינוי קוד המקור
  - תמיכה במוצר שהיצרן לא תומך בו או הפסיק לתמוך
  - בחינה אם אחרים העתיקו קוד שלך
- Forensics
  - זיהוי API של תוכנות, מ"ה, וכו'
  - וידוא שספריות מספקים אחרים עומדות בדרישות האבטחה של המוצר

למה הנדסה לאחור?

- סיבות לא חוקיות
  - עקיפת הגנה בתוכנה
    - פריצת תוכנה – Crack
  - חיפוש חולשות – מחקר לצורך פיתוח שיטות תקיפה
  - גניבת קוד ממוצר מתחרה

  - כמובן שבמקצוע זה לא נעסוק בפעולות לא חוקיות

מקרים ידועים של הנדסה לאחור

- RE של תוכנה
  - RE של ה-BIOS של IBM PC, יצר את תעשיית תואמי ה-PC 1980's
  - פרויקט SAMBA של שיתוף קבצים תואם מיקרוסופט 1990's
  - Wine (ביצוע RE ל-API של חלונות כדי להיות תואם לו) 1990-2000's
- RE של מבנה קבצים
  - Openoffice : פענוח מבנה קבצי DOC 1990's
- RE של מכונת הצפנה וצפנים
  - RE של אניגמה על ידי הפולנים ממידע חלקי והודעות מוצפנות 1932
  - פרסום צופן RC4 1994
  - פרסום צפני A5 של הטלפונים הניידים 1999
- פריצת אייפון ואנדרואיד
  - Jailbreaking, Rooting 2010's
- פריצת הגנות על זכויות יוצרים
  - DVD-CSS 2000's

ולאחרונה...

Intel AMT vulnerability. Life after CVE-2017-5689

The intention of this report is not only to show the story of "her majesty" Intel AMT vulnerability, or the CVE-2017-5689. This report describes possible ways and scenarios of exploiting the vulnerability as well. In addition this white paper outlines some new interesting "undocumented features" of Intel ME/AMT that can be used by an attacker. However, this will demonstrate how the capabilities of Intel ME/AMT can be used to their full extent to tinker with the very platform.

Beyond the Dark Portal:

Further research revealed, that there are some more things to worry about:

- Firmware (Intel ME/AMT) has security issues. Thus, web interface security weakness can be used to obtain remote access Intel AMT system.
- Hardware (Intel ME/AMT) has undocumented features. An illustrative example to it is MEI (HECI) communication protocol that can be reverse-engineered, thus making it possible for attackers to intercept the data and use it for their own benefit.
- There is a possible New stealth infecting computer system with malware that uses only common Intel AMT SoL capabilities to keep communication stealthy and evade security applications.
- If successfully exploited by attackers Intel ME/AMT capabilities become attackers' capabilities. Thus legit functionality will be used to perform non-legit actions

הגנה כנגד הנדסה לאחור

- קיימות מספר שיטות להגן כנגד RE של תוכנה (או חומרה)
- לדוגמא
  - Obfuscation – יצירת קוד מורכב וקשה להבנה
  - Anti-debugging – טכניקות לבלבול והקשיה על פענוח קוד
  - זיהוי האם הקוד רץ תחת דיבגר או מכונה וירטואלית
- משמש עבור
  - הגנת תוכנה נגד פריצה, העתקה, או שימוש לא חוקי
  - הגנה על נוזקה מפני זיהוי וניתוח

שימושים לנוזקות

- מפתחי נוזקות מחפשים אחרי חולשות לא מוכרות
  - הן צריכות לדעת
    - איך להסתיר את עצמן
    - איך להדביק תוכנות ומערכות אחרות
    - איך לגרום נזק
    - איך להגן כנגד RE
  - RE משמש לגילוי החולשות ולהבנה של התוכנות המותקפות
- מגנים נגד נוזקות צריכים להבין את דרך פעולת הנוזקה
  - וחולשותיה, כולל כל ה"איך" לעיל
  - RE משמש למחקר נוזקות, ולפיתוח שיטות זיהוי וניקוי שלהן

דוגמא – תולעת Blaster

- תולעת Blaster הייתה פעילה החל מאוגוסט 2003
- היא התפשטה למאות אלפי מחשבים עם נזק מוערך של כ-320 מיליון דולר

זוכרים?

Blaster תולעת – כיצד?

- כיצד התולעת הדביקה מחשבים?
  - MS03-026 מתייחס לחולשת אבטחה מטיפוס חריגה מחוצץ במנגנון ה-RPC במערכות חלונות השונות (XP ביניהן)

Microsoft Security Bulletin MS03-026

Buffer Overrun In RPC Interface Could Allow Code Execution (823980)

Originally posted: July 16, 2003
Revised: September 10, 2003

  - השירות היה פתוח כברירת מחדל בפורט 445, ולכן כל מחשב שהתחבר לאינטרנט היה בסיכון

- מציאת החולשה וכתיבת קוד שמנצל זאת (Exploit) הינו תהליך ארוך
  - שרובו נעשה באמצעות Reverse Engineering

דוגמא – תולעת Blaster

- כיצד עצרו את התולעת?
  - RE של התולעת איפשר לדעת
    - כיצד היא מתפשטת? (איזו חולשה צריך לסגור)
    - כיצד להסיר אותה מהמחשב?
    - איך לגלות בזמן אמת שמחשב מודבק על מנת למנוע זאת?
    - מה המטרה של התולעת?
    - כיצד היא מקבלת פקודות מהמפעיל?
    - וכו'

שימושים אפשריים של RE-

- תשתיות לאומיות
  - השתלטות על תחנות כוח והפסקת פעילותן
    - או שריפת הגנרטורים על ידי הגברת מהירותם מעבר למותר
  - השתלטות על אספקת המים
  - מערכות רמזורים, מחלפי רכבת, וכו'
  - מצלמות מהירות, ומצלמות אבטחה
  - נניח שהיה מאגר ביומטרי שנטען להיות מוגן...

Cyberattacks Put Russian Fingers on the Switch at Power Plants, U.S. Says

By NICOLE PERLROTH and DAVID E. SANGER MARCH 15, 2018

The Trump administration accused Russia on Thursday of engineering a series of cyberattacks that targeted American and European nuclear power plants and water and electric systems, and could have sabotaged or shut power plants off at will.

שימושים אפשריים של RE-

- נניח שבמכ"ם שלנו יש באג שמאפשר להחביא ממנו מטוסים או שניתן לכבות אותו מרחוק – חייבים לגלות ולתקן
- על פי סנואדן, ה-NSA גילה עשרות בעיות אבטחה במערכות הפעלה וטלפונים חכמים, ומשתמש בהם להאזנה
- דלתות אחוריות שהושתלו על ידי יצרן המערכת
  - או ש"טופלו" על ידי גורם אחר לפני או אחרי ההתקנה
- נפילת מזל"ט בידי האויב: מה הוא יכול להבין ממנו?
  - ונחיתת מטוס מיג בארץ... או התרסקות מזל"ט...
- השתלטות על מזל"ט מרחוק, או השתלטות על לווין

חדשות בעולם

איראן הציגה את המל"ט האמריקאי החמקן שהופל

מפקד חיל האוויר במשמרות המהפכה חשף את המל"ט הסודי שמרגל בשנים האחרונות אחר אתרי הגרעין: "יחידת הלוחמה האלקטרונית שלנו הפילה אותו"

הארץ | פורסם לראשונה: 08.12.2011 | 18:47 | עודכן ב: 09.12.2011 | 00:07 | 45 | הוסף תגובה

Ewen MacAskill in Washington

Sunday 22 April 2012 10.50 BST

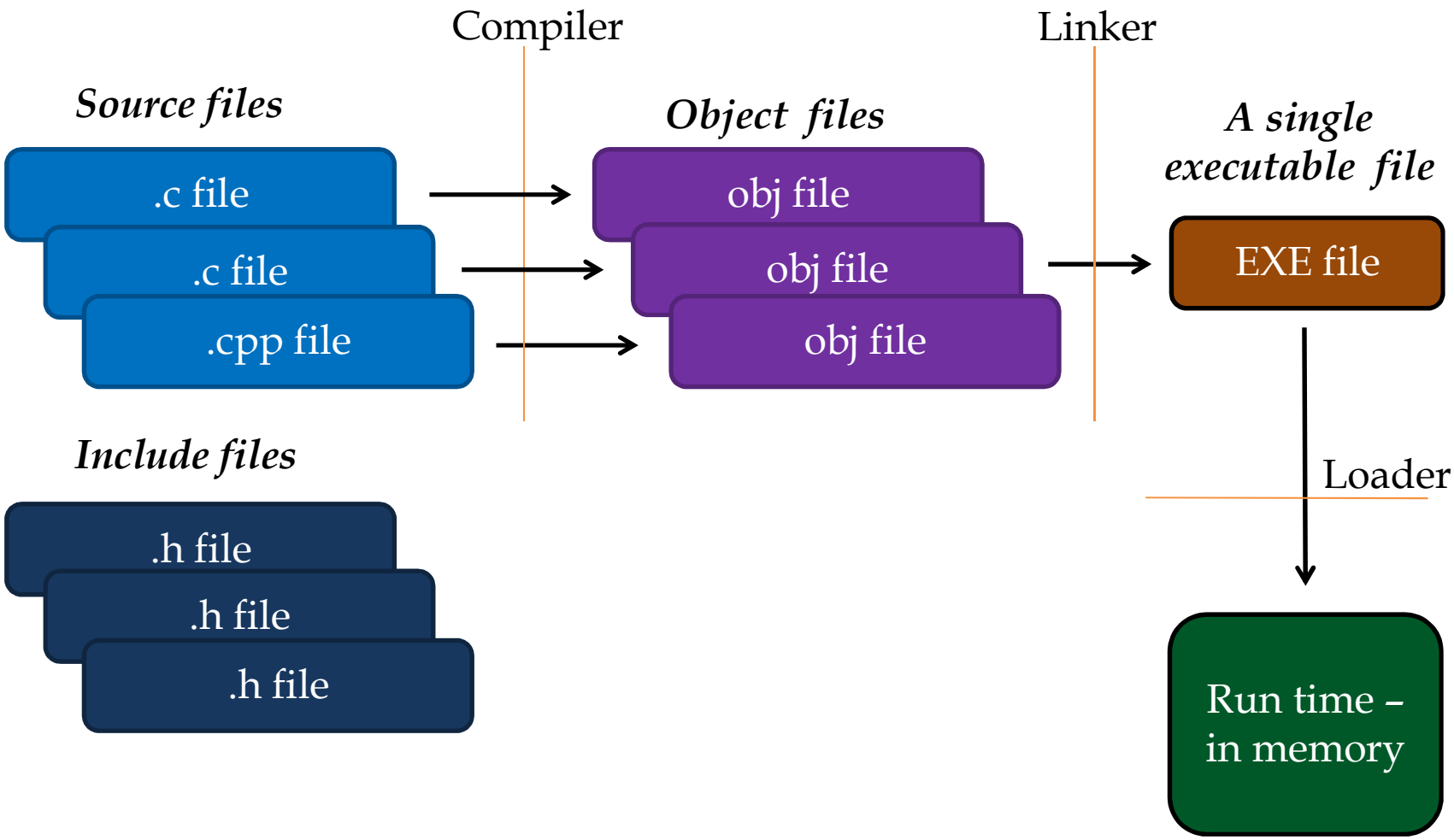
theguardian Winner of the Pulitzer prize

Iran claims to have reverse-engineered US spy drone

General says Tehran has extracted data and figured out workings of Sentinel craft captured last year

RE תוכנה

תהליך ההידור וההרצה ב-C ו-++C

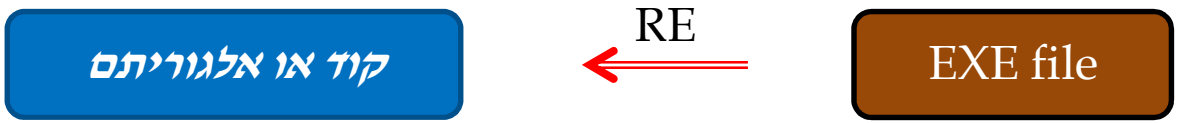


עוד על הידור

- למרות שקובץ ההרצה בחלונות הוא תמיד באותו פורמט ואותה שפת מכונה, כל מהדר יוצר קוד שונה
  - 3 שורות ב-C יכולות להתהדר למספר שורות באסמבלי, בעוד שאותו קוד שנכתב ב-VB יכול להתהדר למאות שורות
  - ואותו קוד ב-C יתורגם על ידי מהדר אחר באופן אחר
  - וכמובן, ניתן להורות למהדר לבצע אופטימיזציות מסוגים שונים וכך ליצור קוד שונה
  - (כפי שתראו בתרגיל הבית)

התהליך ההפוך

- כדי להבין קובץ הרצה, כדאי לדעת מהי השפה שבה נכתב הקוד ואיזה מהדר היה בשימוש
  - בהרבה מקרים קל לשלוף את המידע הזה מקובץ ההרצה, על פי הספריות שבשימוש, ומידע אחר הנמצא בקובץ
- לעיתים בקובץ ההרצה יש מידע נוסף המיועד ללינקר ו\או לדיבגר שיכול לכלול שמות פונקציות ושמות משתנים
- ידע זה עוזר לזהות מבנים באסמבלי ולתרגמם בקלות לקוד
- כמו כן, משתמשים בכלים שעוזרים לפענח את פקודות המכונה לקוד ברמה גבוהה יותר



דוגמא: מה קוד זה מבצע?

```
mov     ebp, esp
mov     eax, 186F8h
call    sub_4180E0
mov     eax, dword_41F094
xor     eax, ebp
mov     [ebp+var_10], eax
call    anti_reverse
mov     eax, lenXorUrl
push    eax
push    offset strXorUrl ; "("
call    decode           ; decode(char *buf, unsigned int len)
add     esp, 8
mov     ecx, lenXorUrl2
push    ecx
push    offset strXorUrl2 ; "("
call    decode           ; decode(char *buf, unsigned int len)
add     esp, 8
mov     dl, ds:byte_41ACF9
mov     [ebp+String1], dl
push    0C351h           ; size_t
push    0                ; int
lea     eax, [ebp+var_C38F]
push    eax              ; void *
call    _memset
add     esp, 0Ch
mov     cl, ds:byte_41ACFA
mov     [ebp+var_186F0], cl
push    0C351h           ; size_t
push    0                ; int
lea     edx, [ebp+var_186EF]
push    edx              ; void *
call    _memset
add     esp, 0Ch
mov     al, ds:byte_41ACFB
mov     [ebp+var_30], al
xor     ecx, ecx
```

כמה זמן לוקח להבין מה הקוד הבא עושה? (אל תנסו)

האם זה אפשרי למצוא את קטע הקוד הרלוונטי לנו מתוך מיליוני שורות אסמבלי?

האם כותב הוירוס יכול להוסיף קושי נוסף במכוון?

לפי שמות הפונקציות הנקראות מקוד זה, אפשר לנחש שהקוד כתוב ב-C (למשל קוראים ל-memset), ושהוא מנסה להגן כנגד RE על ידי פעולות נגד RE וע"י הצפנה.

Java Bytecode & .NET

- ישנן שפות שמהודרות לקוד ביניים (Bytecode)
  - לכל שפה כזאת יש מכונה וירטואלית
  - שמתרגמת בזמן ריצה מקוד הביניים לאסמבלי (JIT)
- אחד היתרונות בשיטה זאת הוא האפשרות להריץ את אותו הקוד במערכות הפעלה שונות וארכיטקטורות שונות בקלות
  - Java רצה בכל מערכות ההפעלה לרבות פלאפונים ושעונים
- תכניות .NET (למשל C#) מהודרות לקבצי הרצה (EXE)
  - הכוללים קוד אסמבלי וקוד ביניים (CIL, נקרא בעבר MSIL)
  - קוד האסמבלי שנמצא שם אחראי להרצת קוד הביניים כ-JIT
  - זיהוי: קבצי EXE שמשתמשים ב-msil.dll נכתבו ב-.NET.
- בדרך כלל מחקר Bytecode לא נחשב מסובך
  - המון מידע נשמר בקובץ
  - ניתן יחסית בקלות להבין מה היה הקוד בשפה העילית
    - יש כלים אוטומטיים שמקלים על כך

בעצם מה זה הנדסה לאחור?

- הנדסה לאחור היא בעצם ניסיון להבין למה המתכנת התכוון
- לצורך כך מאד חשוב להכיר את השפה שבה הקוד נכתב ואת מערכת ההפעלה לעומקן
- ואיך מהדר מייצר קוד בשפת מכונה לפקודות שונות בשפה העילית
- ומאד מועיל גם להבין את צורת החשיבה של המתכנת המקורי ואת סגנון כתיבתו, שיעזרו להבין את מה שהוא התכוון לעשות
- ולהשוות ל"איך אני הייתי כותב את זה"

דוגמא פשוטה להנדסה לאחור

מה עושה הקוד הבא?

```
      mov   DWORD PTR [ebp-16], 0   → s
      mov   DWORD PTR [ebp-12], 0
      jmp   L2
L1:   mov   eax, DWORD PTR [ebp-12] → i
      add   DWORD PTR [ebp-16], eax
      add   DWORD PTR [ebp-12], 1
L2:   cmp   DWORD PTR [ebp-12], 99
      jle   L1
```

דוגמא פשוטה להנדסה לאחור

הוא שקול לקוד הבא, בהחלפת הגישות לזיכרון ברגיסטרים:

```
      mov   s, 0
      mov   i, 0
      jmp   L2
L1:   nop                 (was: mov  i, i)
      add   s, i
      add   i, 1
L2:   cmp   i, 99
      jle   L1
```

דוגמא פשוטה להנדסה לאחור

בחירה מושכלת של שמות משתנים תקל עלינו את ההבנה של הקוד:

```
      mov   s, 0          sum=0;
      mov   i, 0          i=0;
      jmp   L2            goto L2
L1:   nop               L1:
      add   s, i          sum += i;
      add   i, 1          i++;
L2:   cmp   i, 99       L2:
      jle   L1            if(i<=99) goto L1;
```

→ Here i=100 and sum=0+1+2+3+…+99=4950

חוק

- במדינות רבות הנדסה לאחור אינה חוקית בעוד באחרות מותר
  - האיסור תלוי בין השאר בהסכמים של הלקוח עם הספק, חוקי זכויות יוצרים, וחוקים מיוחדים נגד RE
  - כמעט בכל הסכם תוכנה כתוב שאסור לבצע RE
- ברוב המדינות מותר לבצע RE לשם השגת interoperability
  - אבל לא ליצירת מוצר מתחרה המעתיק את המקור
  - בחלק מהמדינות חל איסור גורף לפרסם מידע שהושג על ידי RE
- שימו לב שקוד תוכנה עשוי להיות מוגן זכויות יוצרים
  - ולכן צוותים המבצעים RE לשם פיתוח מוצר אינם מתקשרים ישירות עם הגורמים שמשתמשים בתוצרי ה-RE
  - אלא דרך מסמכים בלבד המפרטים את הנדרש
    - וללא פירוט בצורת קוד

ארה"ב: חוק ה–DMCA

The Digital Millennium Copyright Act

- חוק אמריקאי המסדיר זכויות יוצרים של מידע דיגיטלי
  - למשל, מוסיקה, סרטים
- במקרים מסוימים, הוא אוסר על אנליזה של מערכות המיועדות להגנה על זכויות יוצרים, אפילו לצורך מציאת חולשות והבנת הנדרש לחיזוק המוצר
  - דוגמא: קוד ה-CSS לסרטי וידאו ב-DVD

מה נלמד במקצוע?

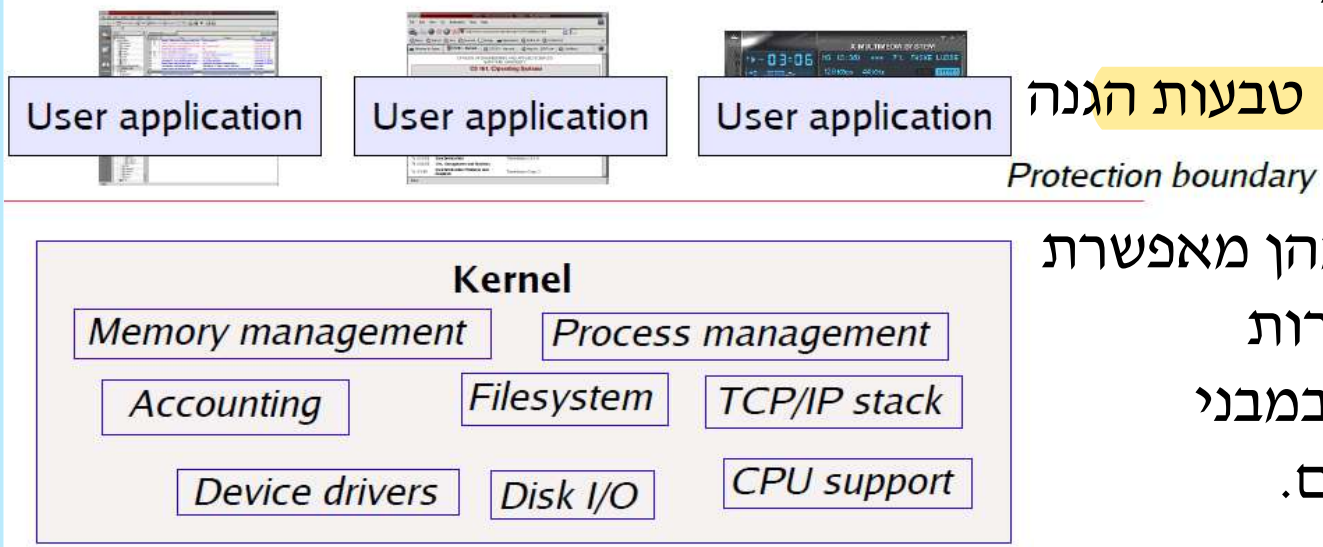
- המקצוע יקנה כלים בסיסיים ל-RE של תוכנות ונוזקות
  - בעיקר בהקשר של תוכנה שקובץ ההרצה שלה זמין
  - בחלקו הראשון של המקצוע נציג עקרונות וכלים
  - בחלקו השני ניישם את הידע ונחקור נוזקות פשוטות
  - בחלקו השלישי נרחיב גם למקרי RE בעלי טכנולוגיות אחרות
- המקצוע מתמקד בסביבת חלונות 32 סיביות
  - אך העקרונות זהים למערכות הפעלה ומעבדים אחרים
- נתייחס גם למניפולציות של קוד, ניתוח וירוסים, סביבות תוכנה שונות, ו-RE למקרים ייחודיים אחרים
  - וגם נתייחס ל-RE של חומרה והתקפות ערוץ צד

שאלות?

תזכורת: מערכת ההפעלה והזכרון

מערכת ההפעלה

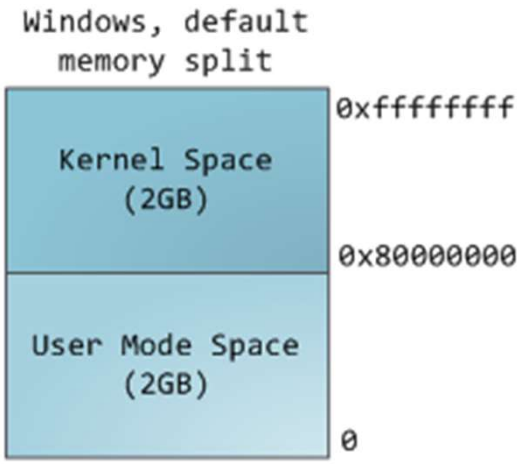
- מתווכת בין החומרה לבין התוכנה באופן "שקוף למשתמש"
  - אינה מאפשרת גישה ישירה לחומרה
- מנהלת משאבי המחשב
- מייצרת לכל תהליך סביבה שבה הוא רץ
  - הפרדה בין תהליכים
  - קשר בין תהליכים
- מגנה על עצמה
  - ארבע (שתי) טבעות הגנה



שכל אחת מהן מאפשרת פקודות אחרות ומשתמשת במבני זכרון אחרים.

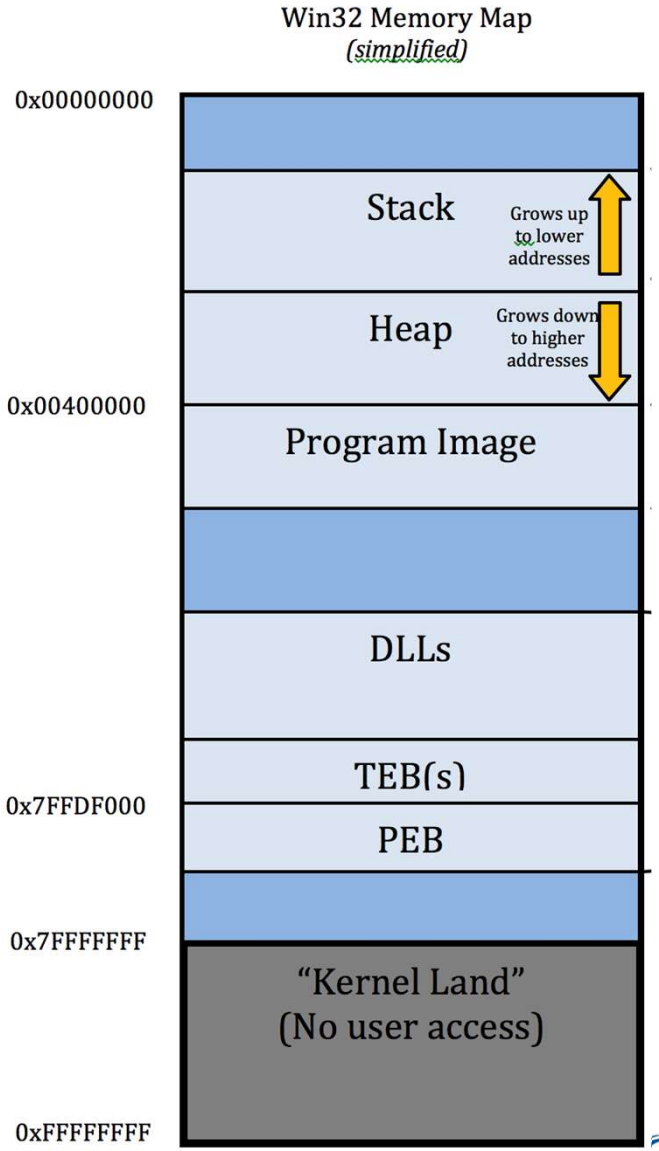
ניהול זכרון

- לכל תהליך "יוזרי" יש את מרחב הזכרון הוירטואלי שלו.
- כל קוד הקרנל חולק מרחב זכרון וירטואלי יחיד.
- במערכת 32 ביט מרחב הזכרון הוא 2^{32} בתים (4GB)
  - החלק התחתון הוא מרחב הזכרון של היוזר
  - החלק העליון משמש למרחב הזכרון של הקרנל.
- לקוד שרץ באזור היוזר אין גישה למרחב הזכרון של הקרנל
  - האם הדבר נכון גם לקרנל? למה?



מרחב הזכרון של תהליך

- קוד
  - Program Image – זהו קוד התהליך עצמו. מכיל את הקוד (.text) ואת המידע (.data) של התהליך.
  - DLL-ים – ספריות הנטענות למרחב הזכרון של התהליך.
- מבני ניהול זכרון
  - Stack (מחסנית) – מחזיקה משתנים מקומיים, פרמטרים וכתובות חזרה (עוד בהמשך).
  - Heap – משמש להקצאת זכרון בזמן ריצה, מיועד למידע שנשאר אחרי הפונקציה שיצרה אותו.
- טבלאות
  - לתהליך (PEB) ולthread (TEB) שמכילות את כל המידע הדרוש לצרכי ריצה.



פרק 2

ניתוח סטטי ודינמי

ניתוח סטטי

ניתוח סטטי

- ניתוח סטטי עוסק בניתוח התוכנה מתוך הקוד שלה
  - כפי שהוא מופיע בקובץ ההרצה
  - ללא הרצה של התוכנה
- תוכנות שתומכות בניתוח סטטי כוללות
  - דיס-אסמבלרים – מתרגמים את הקוד לאסמבלי
  - דה-קומפיילרים – מתרגמים לשפה עילית, אם ניתן
  - מפענחי מבני קובץ הרצה
    - מפרטים את המבנה של קובץ ה-PE
- גם מועיל שיש
  - Hex dump, strings וכו'

מה זה דיס-אסמבלר?

- דיס-אסמבלר הוא תכנית, שקוראת קובץ הרצה
  - ממירה את תוכן הקובץ משפת מכונה לאסמבלי
  - מנקודת ראות מ״ה והמעבד הוא סתם תכנית
- דיס-אסמבלר צריך
  - להכיר את מבנה קובץ ההרצה (למשל PE של חלונות)
  - לפרש פקודות אסמבלי
  - לקשר כתובות לשמות פונקציות ולשמות משתנים
    - כולל לקישורים אל DLL-ים
  - זיהוי מיקום פקודות המכונה
    - במעבדים בהם גודל פקודת מכונה אינו קבוע זה לא תמיד פשוט
  - לזהות בין קוד למשתנים, ובין סוגי משתנים שונים
    - למשל להדפיס נכון מחרוזות, שלמים, וכו׳

איך פועל דיס-אסמבלר?

- על פניו דיס-אסמבלר מזהה מיקומי פקודות מכונה באופן דומה לזיהוי על ידי המעבד בזמן ריצה
  - כלומר, פקודה מתחילה בבית שאחרי סוף הפקודה הקודמת
  - או, אם יש פקודת jmp, בבית שאליו ה-jmp מפנה

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

איך פועל דיס-אסמבלר?

- לאותו רצף של בתים יכולות להיות כמה משמעויות
  - לכן התרגום האוטומטי לא תמיד יזהה את המשמעות הנכונה
    - נראה דוגמאות בהמשך הקורס
  - ישנם אלגוריתמים ושיטות יותר מתקדמות לזיהוי האסמבלי
  - אחת השיטות היא לזהות קפיצות ולקבוע שיעד הקפיצה חייב להכיל קוד אסמבלי תקין (נקרא Recursive Traversal)
    - איזה שיטות נוספות אפשריות?

<table>
<tr><td>...</td><td>...</td><td>E8</td><td>74</td><td>48</td><td>66</td><td>B8</td><td>48</td><td>EB</td><td>E8</td><td>31</td><td>C0</td><td>...</td><td>...</td></tr>
<tr><td colspan="2">...</td><td colspan="5">call eip+0xb8664874</td><td>dec eax</td><td colspan="2">jmp eip-24</td><td colspan="2">xor eax, eax</td><td colspan="2">...</td></tr>
<tr><td colspan="4">...</td><td colspan="2">jz eip+72</td><td colspan="4">mov ax, 0xeb48</td><td colspan="4">call ...</td></tr>
<tr><td colspan="5">...</td><td>dec eax</td><td colspan="4">mov ax, 0xeb48</td><td colspan="4">call ...</td></tr>
</table>

איך פועל דיס-אסמבלר?

- כשיש תערובת של קוד ונתונים באותו section, זה לא תמיד קל
  - אחרת, שם ה-section יכול לעזור
    - מקובל ש-text. מציין קוד, וש-data. מציין משתנים
- דיס-אסמבלר יכול להשתמש במידע נוסף שקיים בקובץ ההרצה, למשל
  - נתונים מכותרות קובץ ההרצה
    - כגון כתובת תחילת התכנית
  - מידע על הספריות הדינמיות (DLL, ko)
    - כתובות הפונקציות מטבלת ה-export
  - מידע שמיועד לדיבגר
    - אם התכנית הודרה עם אופצית דיבגר
    - כולל שמות משתנים ומיקומם בזיכרון, מספרי שורות, שמות פונקציות

The Interactive Disassembler (IDA)

- Ida Pro הוא דיס-אסמבלר אינטראקטיבי
- מיועד ל-RE, עם יכולות דיבוג
- Ida Freeware 5.5 / 7.0
  - גרסה המתאימה לצרכינו
  - תוכנה חופשית
  - הגרסאות האחרות בעלות יכולות נוספות, לא חופשיות
  - אין להשתמש בגרסאות אחרות בקורס זה.
- מאפשרת רישום הערות ומתן שמות לתאי זיכרון
  - למשל משתנים וקוד
  - יכולת מומלצת ביותר
    - מאפשרת לעבוד עם קוד קריא יותר אחרי שזיהינו חלק מרכיביו
    - ואנו נבקש שכך תעשו

ניתוח דינמי

ניתוח דינמי

- ניתוח דינמי מפעיל את התוכנה הנחקרת, ועוקב אחרי פעולתה
- לצורך כך, תהליך אחר, הנקרא דיבגר, שולט על התהליך הנחקר
  - קובע מתי יעשה מה
  - יכול לעצור ולחדש את הפעולה
  - עוקב ויכול לשנות את כמעט כל משאב של התהליך המדובג
- בשונה מניתוח סטטי, הניתוח נעשה בזמן ריצה
  - ולא על קובץ ההרצה ללא ריצה
  - לכן, ניתן לבחון נתונים שאינם זמינים בניתוח סטטי

מה זה דיבאגר?

- Debugger מאפשר למשתמש
  - הרצת הקוד
    - תוך כדי בדיקת האוגרים והזיכרון של התהליך המדובג
  - שינוי כל אחד מהערכים האלו
    - כולל של הקוד עצמו
  - עצירת ריצת התוכנית (breakpoints) לפי מספר שיטות
    - נרחיב בהמשך
  - לעיתים גם פירוש מבנים מסוימים בזיכרון
    - למשל ניתוח מבנה קובץ ההרצה

כלים נפוצים

- כלים נפוצים בחלונות:
  - Ollydbg
  - Immunity Debugger
  - Windbg

מדבגים תהליך ב-user mode בלבד

מדבג הכל, הכלי היחידי שמדבג kernel בחלונות

- לינוקס: gdb

דיבוג גרעין מ"ה

- דיבוג גרעין מ"ה (kernel) אינו יכול להיעשות כמו דיבוג תהליך
  - כי הוא לא תהליך
  - אסור לעצור את פעולת הגרעין באמצע ביצוע
    - ואין משמעות להחלפת הקשר של גרעין מ"ה
  - הכלים שמ"ה מספקת לדיבגר לא קיימים במקרה זה
- לכן דיבוג של גרעין מ"ה דורש כלים יעודיים
- לא נעסוק בכך בקורס זה

Ollydbg/IMM

More stuff

Code

Registers

Memory

Stack
(some kind of memory view of 4 bytes starts from ESP).

איך פועל דיבגר?

- דיבגר הוא תהליך, השולט על פעולת תהליך אחר
- מנקודת ראות מ״ה הוא
  - יכול להריץ אותו
  - לעצור את פעולתו
    - כולל לעצור את פעולתו באופן מתוכנן בפקודה מסוימת
  - לשלוט על הזיכרון שלו
    - לקרא ולשנות את תוכן הזיכרון שלו
    - כולל טבלת הדפים שלו
  - לנהל את הטיפול בחריגות של התהליך המדובג
  - לדעת את מצבו ומצב כל הרגיסטרים בכל עצירה
    - ולשנות אותם לצורך המשך הרצה
- המעבד ומ״ה מתוכננים לתמוך בכך

תמיכת המעבד בדיבאגר

- דיבגר משתמש בשירותי המעבד על מנת לדבג תהליך
- לשם כך יש
  - פקודות מכונה ופסיקת תוכנה מיוחדות
    - int 3
  - מבחר exceptions יעודיים עבור breakpoints
  - רגיסטרים מיוחדים
  - צורות הפעלה יעודיות
    - למשל single step
  - ושימוש בתכונות אחרות הנתמכות ע"י המעבד
    - למשל זיכרון וירטואלי

תמיכת מ"ה בדיבאגר

- דיבגר משתמש בשירותי מערכת ההפעלה על מנת לדבג תהליך
- מ"ה מתייחסת שונה לתהליך מדובג, למשל
  - מדווחת ל-Debugger על אירועים שונים (events) שרלוונטיים לתהליך המדובג
  - בעת חריגה, מ"ה מעבירה את הטיפול ל-Debugger, ועוצרת את פעולת התוכנית
  - אם התוכנית לא מסוגלת לטפל בחריגה, מ"ה תיתן הזדמנות שנייה ל-Debugger לפני הקרסת התוכנית

- כאשר מדבגים תהליך, הוא עלול להתנהג בצורה שונה
- שיטות Anti-Debugging מנצלות זאת

מבנה הנתונים של דיבוג

- מבנה הנתונים שמייה שולחת לתהליך המדבג:

```
typedef struct _DEBUG_EVENT {
    DWORD dwDebugEventCode;     // מציין מה סוג האירוע בגללו נעצרה התכנית
    DWORD dwProcessId;          // process id (pid)
    DWORD dwThreadId;           // thread id in process
    union {                     // מידע נוסף לפי סוג האירוע
        EXCEPTION_DEBUG_INFO      Exception;
                // This includes exception info, e.g., which exception was
                // raised, in case of a page fault: which address failed, etc.
        CREATE_THREAD_DEBUG_INFO  CreateThread;
        CREATE_PROCESS_DEBUG_INFO CreateProcessInfo;
        EXIT_THREAD_DEBUG_INFO    ExitThread;
        EXIT_PROCESS_DEBUG_INFO   ExitProcess;
        LOAD_DLL_DEBUG_INFO       LoadDll;
        UNLOAD_DLL_DEBUG_INFO     UnloadDll;
        OUTPUT_DEBUG_STRING_INFO  DebugString;
        RIP_INFO                  RipInfo;
    } u;
} DEBUG_EVENT, *LPDEBUG_EVENT;
```

דוגמא למבנה של דיבאג

```
// Create the process
main () {
        ...
        CreateProcess (..., DEBUG_PROCESS , ...) ;
        // Or
        DebugActiveProcess( dwProcessId );
        ...
        EnterDebugLoop(...);
        ...
}
```

דוגמא למבנה של דיבאג

```
// Example based on MSDN

void EnterDebugLoop(const LPDEBUG_EVENT DebugEv)
{
   DWORD dwContinueStatus = DBG_CONTINUE;
                                          // exception continuation

   for(;;)
   {
      WaitForDebugEvent(DebugEv, INFINITE);
```

דוגמא למבנה של דיבאג

```
// Process the debugging event code.
    switch (DebugEv->dwDebugEventCode)
    {
      case EXCEPTION_DEBUG_EVENT:
      // Process the exception code. When handling
      // exceptions, remember to set the continuation
      // status parameter (dwContinueStatus). This value
      // is used by the ContinueDebugEvent function.

        switch(DebugEv->u.Exception.ExceptionRecord.ExceptionCode)
        {
```

דוגמא למבנה של דיבאג

```
  case EXCEPTION_ACCESS_VIOLATION:
        // u.Exception.ExceptionRecord.ExceptionInformation is an
        // array that contains the address of the memory that
        // could not be accessed, as well as why (read, write, ...)
    break;
  case EXCEPTION_BREAKPOINT:
    break;
  case EXCEPTION_DATATYPE_MISALIGNMENT:
    break;
  case EXCEPTION_SINGLE_STEP:
    break;
  case DBG_CONTROL_C:
    break;
  default:
    break;
}
break;
```

דוגמא למבנה של דיבאג

```
case CREATE_THREAD_DEBUG_EVENT:
   dwContinueStatus = OnCreateThreadDebugEvent(DebugEv);
   break;
case CREATE_PROCESS_DEBUG_EVENT:
   dwContinueStatus = OnCreateProcessDebugEvent(DebugEv);
   break;
case EXIT_THREAD_DEBUG_EVENT:
   dwContinueStatus = OnExitThreadDebugEvent(DebugEv);
   break;
case EXIT_PROCESS_DEBUG_EVENT:
   dwContinueStatus = OnExitProcessDebugEvent(DebugEv);
   break;
case LOAD_DLL_DEBUG_EVENT:
   dwContinueStatus = OnLoadDllDebugEvent(DebugEv);
   break;
```

דוגמא למבנה של דיבאג

```
        case UNLOAD_DLL_DEBUG_EVENT:
          dwContinueStatus = OnUnloadDllDebugEvent(DebugEv);
          break;
        case OUTPUT_DEBUG_STRING_EVENT:
          dwContinueStatus = OnOutputDebugStringEvent(DebugEv);
          break;
        case RIP_EVENT:
          dwContinueStatus = OnRipEvent(DebugEv);
          break;
      }
      // Resume executing the thread that reported the debugging event.
      ContinueDebugEvent(DebugEv->dwProcessId,
                         DebugEv->dwThreadId, dwContinueStatus);
    }
}
```

העברת החריגה לדיבאגר

היישום במערכת ההפעלה

(from reactos.org)

KiDispatchException

```
 if (PreviousMode == KernelMode)  { … }
else
   {
      if (FirstChance)      /* User mode exception, was it first-chance? */
      {
         /* Break into the kernel debugger unless a user mode debugger is present or user mode */
         /* exceptions are ignored, except if this is a debug service which we must always pass to KD */
         if ((!(PsGetCurrentProcess()->DebugPort) &&
             !(KdIgnoreUmExceptions)) ||
             (KdIsThisAKdTrap(ExceptionRecord, &Context, PreviousMode)))
         {
            KiPrepareUserDebugData();          /* Make sure the debugger can access debug directories */
            /* Call the kernel debugger */
          if (KiDebugRoutine(TrapFrame, ExceptionFrame, ExceptionRecord, &Context, PreviousMode, FALSE))
                goto Handled;                              /* Exception was handled */
         }
          /* Forward exception to user mode debugger */
          if (DbgkForwardException(ExceptionRecord, TRUE, FALSE))
              return;
         KiDispatchExceptionToUser()
         __debugbreak();
      }
```

שיטות לעצירת ריצה בדיבאגר

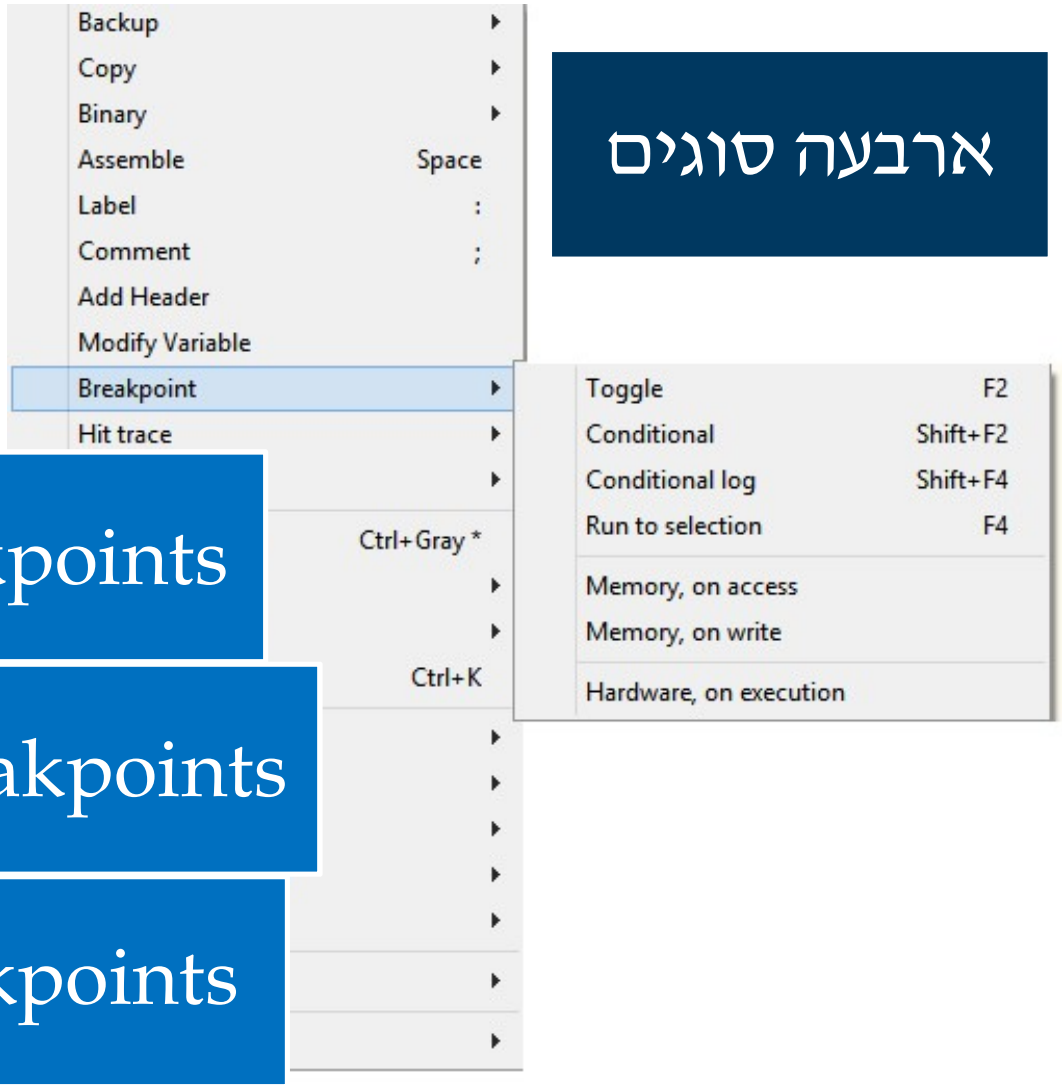
ארבעה סוגים

Single step

Memory breakpoints

Hardware breakpoints

Software breakpoints



שימושים

סוג	עצירה בקריאה מזיכרון	עצירה בכתיבה לזיכרון	עצירה בהרצת קוד
Single Step			☑
Memory BPT	☑	☑	☑
Hardware BPT	☑	☑	☑
Software BPT			☑

Single Step משמש גם ככלי עזר בישומי Memory BPT ו-Software BPT.

Single Step

- המעבד תומך בהפעלת פקודת מכונה בודדת בתהליך מדובג, על פי בקשת הדיבגר
- התהליך מופעל לפקודת מכונה אחת, ועוצר
- התהליך המדבג חוזר לפעולה ומחליט מה לעשות הלאה
  - הוא גם מקבל מידע על המצב של התהליך המדובג
  - למשל אם הייתה חריגה
  - וכמובן ערכי הרגיסטרים, וכו'
- Single Step שימושי לצורך
  - הרצה פקודה בודדת (פקודת Step של הדיבגר)
  - ולשליטה על הפעולה של שירותי דיבגר אחרים
    - בהם הדיבגר צריך לשנות את סביבת ההרצה לפקודת מכונה אחת
      - למשל Memory BPT ו-Software BPT

Memory Breakpoints

- ניתן להגדיר בתים בזיכרון שיגרמו לחריגה אם
  - היה ניסיון לקרוא אותם או לכתוב אליהם
  - היה ניסיון להריץ קוד שכתוב בהם
- ממומש באמצעות שינוי הרשאות הקריאה/כתיבה/ריצה של דף הזיכרון
  - באמצעות שינוי בטבלת הדפים של התהליך
  - גורם ל-EXCEPTION_GUARD_PAGE בזמן ריצה
- ה-breakpoint תקף לכל דף הזיכרון
  - כלומר ל-4096 בתים (תלוי בהגדרות כמובן)
  - הדיבגר צריך לבדוק לאן פקודת המכונה ניגשה בדף הזיכרון
    - ואם זה לא לאחד הבתים אחריהם עוקבים – אז הוא מתעלם
    - לצורך זה הוא מפרש בעצמו את פקודת המכונה בה קרתה החריגה
      - ולהבין מדוע קרתה ובאיזה גישה לזיכרון
        - ✓ מיקום קוד התכנית, קריאה או כתיבה לזיכרון, וכו'
- יתרון : אין הגבלה על מספר ה-memory breakpoints

Memory Breakpoints

- לאחר העצירה, לא ניתן להמשיך ריצה בלי תיקון ההרשאות בטבלת הדפים
- לשם כך מופעל הנוהל הבא
  - תיקון הרשאות הדף
  - הרצת התכנית ב-Single Step
    - כלומר הפעלת פקודת מכונה בודדת
      - זה נתמך בחומרה של המעבד (TF flag)
      - תמיכת המעבד הכרחית! אחרת אי אפשר להבטיח ריצת כל פקודה עם עצירה אחריה
    - יתכן שהתכנית תיעצר שוב באותה פקודת מכונה מסיבה אחרת
      - דורש תיקון כל הסיבות לפני הצלחת הפקודה
  - שיחזור הרשאות הדף לצורך המשך הדיבוג
  - המשך הרצת התכנית עד ה-breakpoint הבא
- כל זה נעשה גם אם עצרנו בגלל גישה לבית בזיכרון באותו הדף שאינו דורש מעקב
  - אבל, בלי שהדיבגר מציג את העצירה הזו למשתמש

Hardware Breakpoints

- ניתן להגדיר מספר מוגבל של בתים בזיכרון שיגרמו לחריגה אם
  - היה ניסיון לקרוא אותם או לכתוב אליהם
  - היה ניסיון להריץ קוד שכתוב בהם
- זה נעשה בעזרת אוגרים מיוחדים (DR0-DR7)
  - המעבד בודק את ערך האוגרים לפני כל גישה לתא בזיכרון
  - חפשו "debug registers" למידע נוסף על האוגרים
- ניתן לבצע את הפעולה רק על BYTE/WORD/DWORD
- חסם של ארבעה hardware breakpoints לכל היותר
- השיטה שקופה לתהליך המדובג
  - וגם יעילה יותר מהאחרות

DR0-DR3	• ארבע כתובות לעצירה
DR4-DR5	• מבוטלים
DR6	• סטטוס
DR7	• הפעלה ושליטה

Software Breakpoints

- מיושמים על ידי שינוי בקוד התכנית
  - שתילת פקודת המכונה int 3 במקום בו רוצים לעצור
  - זהה ל-BPT ב-PDP
- כשמגיעים לנקודת ה-breakpoint, פקודת int 3 מקפיצה חריגה שמועברת ל-Debugger
  - (בהנחה שהתהליך מדובג)
  - כדי להמשיך את הריצה, יש צורך
    - לשחזר את פקודת המכונה המקורית שהייתה שם
    - להריץ ב-single step
    - ואז להחזיר את פקודת int 3
      - כדי שבפעם הבאה שנגיע שוב תהיה עצירה

Software Breakpoints

- למה יש ל-int 3 אופקוד CC בן בית אחד?
  - בעוד ניתן היה להשתמש באופקוד CD 03?
  - כלומר איזה בעיות גורם אופקוד בן שני בתים?

- הדיבגר ממשיך להציג "נכון" את התרגום לאסמבלי
  - למרות ששינה את מקום ה-breakpoint ל-CC
  - כלומר לא מציג את ה-CC
  - אלא את הפקודה המקורית שהיתה שם

int 3 – שני ייצוגים	
CC	CD 03

Software Breakpoints–דוגמא

```
EIP-> xor eax,eax
      mov ebx,eax
      push esi
      mov esi , 200
      inc ebx
      xor edi , edi
      cmp edx , esi
      ja label1
      je label2
      mov eax,300
      cmp edx,eax
      sub eax,11
      and eax,FF0
      cmp eax,FF1
```

Software Breakpoints–נקודות

```
EIP->  xor eax,eax
       mov ebx,eax
       push esi
       mov esi , 200
       inc ebx
       xor edi , edi
       cmp edx , esi
       int 3          // put int 3. remember the overwritten opcodes.
       je label2
       mov eax,300
       cmp edx,eax
       sub eax,11
       and eax,FF0
       cmp eax,FF1
```

Software Breakpoints–נקודות

```
        xor eax,eax
        mov ebx,eax
        push esi
        mov esi , 200
        inc ebx
        xor edi , edi
        cmp edx , esi
EIP->   int 3
        je label2
        mov eax,300
        cmp edx,eax
        sub eax,11
        and eax,FF0
        cmp eax,FF1
```

Exception,
The debugger will handle it.

Before continuing execution, the debugger will restore the original opcode, use single step, and after executing the original opcode will put int 3 back.

Software Breakpoints–נקודות

```
        xor eax,eax
        mov ebx,eax
        push esi
        mov esi , 200
        inc ebx
        xor edi , edi
        cmp edx , esi
EIP->   ja label1
        je label2
        mov eax,300
        cmp edx,eax
        sub eax,11
        and eax,FF0
        cmp eax,FF1
```

Exception,
The debugger will handle it.

Before continuing execution, the debugger will restore the original opcode, use single step, and after executing the original opcode will put int 3 back.

חסרונות של Software Breakpoints

- מתערבים בקוד של תהליך
- לא יעבוד במקרה שתהליך לגיטימי ישנה את הקוד
  - נדיר, אבל קורה למשל ב-NET.
- תהליכים פחות לגיטימיים יכולים לסרוק את קוד התוכנה ולאתר את ה-breakpoint
  - לכן עדיף להשתמש ב-Hardware BP בניתוח נוזקות ותוכנות חשודות אחרות
- בעת ניתוח קוד שמשנה את עצמו, פקודת ה-int 3 עלולה להימחק...
  - למשל כאשר וירוס דחוס פותח את עצמו ומשנה את המקום בו עשינו breakpoint
  - אין בעיה כזו ב-hardware breakpoints – הם יעצרו כשנגיע למקום הזה בקוד החדש

שאלות?

פרק 3

הוקינג

מה זה הוקינג?

"In computer programming, the term hooking covers a range of techniques used to alter or augment the behavior of an operating system, of applications, or of other software components by intercepting function calls or messages or events passed between software components.

Code that handles such intercepted function calls, events or messages is called a "hook".

(Wikipedia)

שימושי הוקינג

- ניטור תוכנות
  - מעקב אחרי פונקציות מסוימות (קלט/פלט וזמן ריצה)
  - משמש מפתחים לצורך בדיקת פונקציות מסוימות
    - לדוגמא ניטור פונקציות הקצאה ושחרור זיכרון
      - בדומה לכלי valgrind
  - משמש לאיתור התנהגות לא תקינה של תוכנות מסוימות ע"י תוכנת אנטי וירוס
  - כלי עזר ל-Reverse Engineering
  - קוד עוין שמצורף לתכנית
    - למשל למעקב אחרי המשתמש

שימושי הוקינג

- שינוי תוכנה
  - שינוי קטע קוד/פונקציה מסוימת
  - שימוש לגיטימי – הרחבת האפשריות בתוכנה מסוימת
    - למשל Babylon מוסיפה אפשרות תרגום לכל התוכנות הרצות
  - וירוסים מסוימים ישנו את התנהגות המערכת על מנת להסתתר מהמשתמש
- יש עוד מספר שימושים

שימושי הוקינג

- עבורנו הוקינג חשוב משתי סיבות
  - האחת, נרצה להשתמש בהוקינג בזמן ניתוח קוד דינמי
    - להצגת ערכי ביניים וניטור
  - השנייה, לזהות הוקינג בקוד שאנחנו בוחנים
    - למשל כאשר נוזקות ביצעו הוקינג לצרכיהן
    - וכמובן גם לבטל הוקינג כזה

טכניקות הוקינג

- ניתן לחלק לשתי שיטות
  - שינוי תוכנה בזמן ריצה – מתייחס לשינוי התנהגות התוכנה כאשר היא רצה. כלומר שינוי חלק מסוים בזיכרון
  - שינוי הקובץ עצמו – שינוי הקוד בקובץ עצמו טרם הריצה
  - לעיתים נשלב את שתיהן
    - השינוי בקובץ יפעיל שינוי נוסף בזמן ריצה
- דרישות
  - לצורך שינוי קובץ נצטרך
    - פונקציות בסיסיות לקריאה/כתיבה לקובץ
  - לצורך שינוי בזמן ריצה נצטרך
    - פונקציות לקריאה וכתיבה מתהליך אחר
- נתחיל מהוקינג ע"י שינוי תוכנה בזמן ריצה
  - כשהיא כבר טעונה לזיכרון
  - ונניח לרגע שיש ביכולתנו לעשות זאת, בלי לפרט איך

הנרקת קוד

- נניח שיש לנו קטע קוד (פונקציה), באילו דרכים ניתן להשתיל בה קוד?

…
Our code
....

?

someFunction:

```
8BFF      MOV EDI,EDI
55        PUSH EBP
8BEC      MOV EBP,ESP
83E4 F8   AND ESP,FFFFFFF8
6A 01     PUSH 1
....
...
```

הזרקת קוד

- לצורך הזרקת קוד ניתן להשתמש ב-jmp בתחילת הפונקציה
  - הפקודה לוקחת 5 בתים (קפיצה למקום מרוחק)

someFunction:

```
8BFF        MOV EDI,EDI
55          PUSH EBP
8BEC        MOV EBP,ESP
83E4 F8     AND ESP,FFFFFFF8
6A 01       PUSH 1
....
...
```

```
E9xxxxxxxx  JMP OUR_CODE
83E4 F8     AND ESP,FFFFFFF8
6A 01       PUSH 1
....
...
```

הזרקת קוד

- לצורך הזרקה נשתמש ב-jmp בתחילת הפונקציה
  - שתקפוץ לקוד שלנו במקום פנוי אחר
  - הפקודה jmp לוקחת 5 בתים
- אם הפקודה לא מתאימה לקוד מבחינת הגודל, נשתמש ב-nop לריפוד (לא חובה)

someFunction2:

```
8BFF        MOV EDI,EDI
55          PUSH EBP
83E4 F8     AND ESP,FFFFFFF8
6A 01       PUSH 1
....
...
```

```
E9xxxxxxxx  JMP OUR_CODE
90          NOP
6A 01       PUSH 1
....
...
```

הנרקת קוד

- יש צורך לזכור את הקוד שנמחק ע״י ה-jmp
  - כדי להחזירם למקומם אם נרצה אי פעם
  - וכדי להריץ את הפקודות שהיו שם, לפני הרצת הקוד שלנו
- וכמובן לקפוץ בחזרה לפקודת המכונה הבאה בסוף הקוד שלנו

someFunction2

before the hook

```
MOV EDI,EDI
PUSH EBP
AND ESP,FFFFFFF8
PUSH 1
...
```

after the hook

```
JMP OUR_CODE
NOP
PUSH 1
....
...
```

OUR_CODE

```
// restore the function:
MOV EDI,EDI
PUSH EBP
AND ESP,FFFFFFF8
…
Our code
....
Jmp back
```

הנרקת קוד

- jmp קופצת יחסית לכתובת הפקודה הבאה
  - חוסך טיפול ב-relocation בזמן טעינת התכנית...
  - לכן xxxxxxxx יהיה

xxxxxxxx = OUR_CODE – (someFunction2 + 5);

someFunction2:

```
E9xxxxxxxx      JMP OUR_CODE
90              NOP
6A 01           PUSH 1
....
...
```

OUR_CODE

```
// restore the function:
8BFF        MOV EDI,EDI
55          PUSH EBP
83E4 F8     AND ESP,FFFFFFF8
…
Our code
....
Jmp back
```

שימו לב:
הדרך שהוצגה כאן (jmp) נחשבת אומנם נפוצה, אך יש עוד המון דרכים לבצע זאת.

הנרקה בעזרת HOT Patching

- Visual Studio תומך בהידור עם הדגל hotpatching
  - חלק גדול מה-DLL-ים קומפלו בשיטה זו
  - במצב זה הפונקציות "מוכנות" ל-hooking
- במצב Hot patching
  - כל פונקציה מתחילה ב-mov edi, edi
    - בעצם nop שניתן לדרוס
    - גודל של jmp short
  - לפניהם יהיו בדיוק חמישה nop's
    - גודל של jmp
    - לא מבוצעים בריצה רגילה של הפונקציה

הזרקה בעזרת HOT Patching

- לצורך הזרקת קוד נחליף את שבעת הבתים הללו בקפיצות

Original Code	Patched Code
nop nop nop nop nop	jmp xxxxxxxx to our code
mov edi, edi	jmp short 7 bytes backward to the jmp above
Rest of function	Unchanged

- כך בקריאה לפונקציה, נבצע את שני ה-jmp
  - שיפנו אותנו לקוד שלנו
- לא דרסנו פקודות חשובות – אין צורך לשחזר
- למה שני jmp כשאפשר היה אחד?
  - כי בריצה רגילה, ללא הוקינג, ביצוע mov edi, edi אחד יעיל יותר מלהפעיל 5 nop-ים בתחילת כל פונקציה

HOT Patching הנרקה בעזרת

```
90          NOP
90          NOP
90          NOP
90          NOP
90          NOP
8BFF        MOV EDI,EDI
55          PUSH EBP
8BEC        MOV EBP,ESP
83E4 F8     AND ESP,FFFFFFF8
51          PUSH ECX
53          PUSH EBX
56          PUSH ESI
57          PUSH EDI
E8 ADFFFFFF CALL USER32.7740B4B8
64:8B0D 18000000 MOV ECX,DWORD PTR FS:[18]
8BF0        MOV ESI,EAX
```

jmp to our code

jmp 7 bytes backward

תחילת הפונקציה

הנרקה בעזרת שינוי ה–IAT

- בזמן ריצה, ה-IAT מצביע לפונקציות ב-DLL-ים השונים
- כל קריאה לפונקציה של DLL עוברת דרך ה-IAT
- שינוי ה-IAT, כך שיצביע לקוד שלנו במקום לפונקציה ב-DLL, יפנה את הקריאה לפונקציה לקוד שלנו
  - הקוד שלנו יקרא לקוד המקורי, אם יש צורך בכך
  - כאשר הקוד שלנו נטען כ-DLL, אין צורך לזכור את הכתובת המקורית
    - נקרא לפונקציה ישירות דרך ה-IAT של ה-DLL שלנו
  - כך שהשינוי בזמן ריצה יהיה רק שינוי המצביע ב-IAT
- יתרון נוסף: ה-IAT נשמר באיזור זיכרון שאינו מוגן מכתיבה

שינוי קובץ ההרצה

- בשקפים הקודמים עסקנו בשינוי הקוד בזמן ריצה
  - כלומר שינוי מקומי בכל פונקציה ופונקציה
  - מאפשר גם לשנות פונקציות מ-DLL-ים

- לעיתים נעדיף לשנות את הקוד ישירות בקובץ ההרצה
  - אפשר לבצע הוקינג באופן דומה למבנה שהצגנו לזמן ריצה
  - אבל לא לשנות קוד בספריות נטענות דינמית
  - טכניקות מורכבות יותר מאפשרות לבצע שינויים גדולים בקוד הנמצא בקובץ ההרצה
    - למשל post-link optimizations

תמיכה בתוכנה

- קיימות תוכנות המאפשרות שינוי ישיר בקובץ ההרצה
  - Immunity
  - מאפשר לשנות קוד בזמן ניתוח התוכנה
  - ואז לשמור את התוצאה לקובץ הרצה חדש

סיכום

- דנו איך לשנות קוד על מנת להכניס פונקציה משלנו
  - כשקוראים לפונקציה אחרת, למשל פונקציית מערכת
  - לא הצגנו כיצד לבצע זאת באופן מעשי, נדגים בהמשך
- יש מגוון דרכים שבהם ניתן לעשות זאת
  - הצגנו דוגמאות פשוטות
  - ישנם אנטי-וירוסים שמזהים שינויי הוקינג
    - לכן וירוסים פיתחו שיטות "יצירתיות", ומנגנונים שטרם נבדקים ע"י אנטי-וירוסים

- הָציעו דרך לבצע הוקינג שלא ניתנת לזיהוי פשוט
- הָציעו דרך לבצע הוקינג בזמן ריצה
  - בלי לשנות בכלל את קובץ ההרצה

תצורת
ודוגמא לשימוש בהמשך

מה זה API?

Application Programming Interface

- מתייחס לקבוצת פונקציות מוכנות שמסופקות למתכנת באופן מובנה
  - בדרך כלל ע"י מערכת ההפעלה
  - בחלונות זה נקרא WinApi
  - גם שירותי ענן מספקים API לאפליקציות, למשל פייסבוק
- לדוגמא, פונקציות לטיפול בקבצים
  - מיוצאות ע"י kernel32.dll
    - שבגרסאות חדשות של חלונות מפנה ל-kernelbase.dll
  - CreateFile, ReadFile, WriteFile
  - fopen, fread, fwrite הן עטיפה לפונקציות הגישה לקבצים, ברמת המהדר, המשתמשת בפונקציות לעיל

הודעות בחלונות

- תוכנות בחלונות מקבלות קלט מהמשתמש דרך הודעות מהמערכת
- כאשר המשתמש מבצע פעולה מסוימת בתוכנה
  - הפעולה מתורגמת ע"י מערכת ההפעלה להודעה
  - וזו נשלחת לתוכנה המתאימה
- בכל תוכנה בחלונות יש לולאה שמטפלת בהודעות
- ביצוע הטיפול בהודעה נעשה ע"י פונקציית callback
  - שנקראת (בעקיפין) מהלולאה
  - ומוגדרת במבנה נתונים WNDCLASS

הודעות =

אירועים של מ"ה שמדווחים לתהליך

הקלדות במקלדת | אירועי עכבר | ועוד

הודעות בחלונות

```
WNDCLASS wc;
…
wc.lpfnWndProc = (WNDPROC) WndProc;
…
…
 RegisterClass(&wc))
…
 hwndMain = CreateWindow("MainWndClass", "Sample",
        WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT,
        CW_USEDEFAULT, CW_USEDEFAULT, (HWND) NULL,
        (HMENU) NULL, hinst, (LPVOID) NULL); ;
…
while( (bRet = GetMessage( &msg, NULL, 0, 0 )) != 0)
{
    if (bRet == -1)
    {
      // handle the error and possibly exit
    }
    else
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
}
```

```
LRESULT CALLBACK WndProc( // מטפלת בהודעות מהחלון
   HWND hwnd,        // handle to window
   UINT uMsg,        // message identifier
   WPARAM wParam,    // first message parameter
   LPARAM lParam)    // second message parameter
```

Message Loop

דוגמאות לסוגי הודעות בחלונות

למשל הודעות על שימוש בעכבר

...
WM_MOUSEFIRST = 0x200
WM_MOUSEMOVE = 0x200
WM_LBUTTONDOWN = 0x201
WM_LBUTTONUP = 0x202
WM_LBUTTONDBLCLK = 0x203
WM_RBUTTONDOWN = 0x204
WM_RBUTTONUP = 0x205
WM_RBUTTONDBLCLK = 0x206
WM_MBUTTONDOWN = 0x207
WM_MBUTTONUP = 0x208
WM_MBUTTONDBLCLK = 0x209
WM_MOUSEWHEEL = 0x20A
WM_MOUSEHWHEEL = 0x20E
...

פונקציית SetWindowsHookEx

קבלת הודעות של תהליכים אחרים

- חלונות מאפשרת לבצע Hook בתהליכים אחרים
  - באמצעות הפונקציה SetWindowsHookEx
- הפונקציה מאפשרת לנטר הודעות בתוכנית
  - כך שההודעה תגיע לקוד שלך בנוסף לקוד הרגיל של התוכנית
    - תומך במגוון סוגי הודעות, חלקן אפשר לפני ההעברה לתהליך או אחרי
- ניתן גם להזריק DLL באמצעות הפונקציה
  - על הזרקות נדון בהרחבה בהמשך
- הפונקציה יכולה לבצע Hook בתוכנית מסוימת או בכל התוכניות
- הפונקציה היא מעטפת לקוד שכותב לזיכרון של תהליך אחר
- כל המידע נמצא ב-MSDN

Key Logger–מאזין

```
int CALLBACK WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR lpCmdLine, int nCmdShow) {
    hHook=SetWindowsHookEx(WH_KEYBOARD_LL,
                           hookProc, hInstance, 0);
    // error checking…
    MessageBox(NULL, L"Press OK to exit", L"Note", MB_OK);
    UnhookWindowsHookEx(hHook);
    return 0;
}
```

- הערה: MessageBox מנהלת לולאת טיפול בהודעות פנימית משלה, כך שאין צורך לכתוב אחת כזו במפורש.

Key Logger–דוגמא

```
LRESULT CALLBACK hookProc(int nCode, WPARAM wParam, LPARAM lParam) {
        PKBDLLHOOKSTRUCT ptr = (PKBDLLHOOKSTRUCT) lParam;
        if (wParam == WM_KEYDOWN) {
                switch (ptr->vkCode) {
                        case VK_TAB:
                                //code...
                        case VK_SHIFT:
                                //code...
                        case VK_CONTROL:
                                //code...
                        default:
                                //code...  (ptr->vkCode is the key)
                }
        }
        // Chain to other hooks from other applications
        return CallNextHookEx(0,nCode,wParam,lParam);
}
```

ספריות דינמיות – DLL

- ספריות סטטיות משולבות בזמן לינק לתוך קובץ ההרצה
  - הקוד שקורא להן יודע את כתובתן היחסית כפי שהוא יודע לכל פונקציה שכתובה ישירות בקוד
- ספריות דינמיות נטענות ע"י ה-loader
  - ורק אז מקושרות הפונקציות שבספריה למקומות בהן קוראים להן
  - ה-loader עשוי לטעון אותן לכתובת אקראית
    - ע"י ASLR
- בלינוקס הן עם סיומת so
  - למשל
    - /usr/lib64/libc.so
    - /usr/lib64/libcrypto.so
    - /usr/lib64/libX11.so

איך טעונות ספריות דינמיות?

"copy-on-write"

- כדי לחסוך בזיכרון סביר שטבלאות הדפים של כל התהליכים מפנות ספריה דינמית לאותו שטח בזיכרון הפיסי
  - כך שנשמר רק עותק אחד של הספריה
- ומה קורה כשתהליך רוצה לכתוב בדף כזה?
  - ???

הוקינג

דוגמא מפורטת

דוגמא להוקינג

- מתכנת כתב קוד שמבצע מספר קריאות ל-MessageBox :

Program.c:

```
...
MessageBox(NULL, L"First MsgBox\n",NULL, NULL);
...
MessageBox(NULL, L"Second MsgBox\n",NULL, NULL);
...
```

- נניח שרוצים שהודעות ה-MessageBox יהיו מוצפנות
- ברשותנו הקובץ הבינארי בלבד
  - בלי אפשרות להשיג את קוד מקור
- איך נפתור את הבעיה?

שיקולי יישום

- תכנית הדוגמא משלבת את הקבצים הבאים
  - קוד התכנית נמצא נניח ב-program.exe
  - פונקציית MessageBox היא ב-user32.dll
  - והם כמובן גם קוראים בתורם לשירותי kernel32.dll
- האם לשנות את הקוד בקובץ
  - program.exe?
  - user32.dll?
  - kernel32.dll?

אך היכן נשנה?

- kernel32.dll?
  - נשמע לא רלוונטי
- user32.dll?
  - נשמע המקום הנכון
  - אבל ישנה גם בתכניות אחרות!
  - המקום הנכון כשרוצים לעקוב אחרי כל התהליכים
- program.exe?
  - מסובך. ואין לנו קוד מקור
  - אבל זה הקובץ היחיד שניתן לשנות ללא הרשאות מיוחדות
  - ודרכו נוכל לשנות את השאר בזמן ריצה!
    - ללא צורך בשינוי קבצי ה-DLL
    - כלומר ללא השפעה על שאר התהליכים במחשב

אך היכן נשנה?

Program.exe

```
...
....
...
Call MessageBox
...
Call MessageBox
...
...
...
```

User32.dll

```
int WINAPI MessageBox(
  _In_opt_  HWND hWnd,
  _In_opt_  LPCTSTR lpText,  //עלינו לקודד את lpText
  _In_opt_  LPCTSTR lpCaption,
  _In_      UINT uType
);
```

Kernel

...

האם כדאי לשנות את הקוד ב Program.exe ?

מה לגבי User32.dll?

או kernel32.dll?

שימו לב:

User32.dll ממופה ע"י ה Loader לזיכרון של Program.exe. לכן בזמן ריצה ניתן להתייחס ל-User32.dll שנטען לזיכרון כקוד בלעדי של Program – כל שינוי יהיה בלעדי לתהליך הזה. לעומת זאת שינוי הקובץ (User32.dll) עצמו , ישפיע על כל התהליכים במערכת.

אך היכן נשנה?

- אפשרות נוספת
  - מתהליך אחר שאנחנו מתכנתים בעצמנו
    - שישנה את הפעולה (או תוכן הזיכרון) של התהליך שרצינו
    - תוך שימוש בשירותי מערכת הפעלה שתומכים בכך
  - יתרונות
    - לא צריך לשנות שום קובץ!
    - כן צריך הרשאות מתאימות
    - ואי אפשר לזהות דבר מניתוח קבצי התכנית וה-DLL-ים
      - כי השינוי לא נעשה דרכם

המקרה הפשוט: יש קוד של התוכנית

- תחילה נדגים כיצד לשנות את MessageBox כאשר יש לנו גישה לקוד מקור (של Program)
- התהליך מתמקד בשינוי הפונקציה ב-user32.dll

שתי גרסאות של MessageBox

7513CDD2	.text	Export	MessageBeep
75185F69	.text	Export	MessageBoxA
75185FAF	.text	Export	MessageBoxExA
75185FD3	.text	Export	MessageBoxExW
7518618F	.text	Export	MessageBoxIndirectA
7514148C	.text	Export	MessageBoxIndirectW
7518607C	.text	Export	MessageBoxTimeoutA
75185FF7	.text	Export	MessageBoxTimeoutW
75185F8C	.text	Export	MessageBoxW
75196578	.idata	Import	GDI32.MirrorRgn
7518E4E1	.text	Export	ModifyMenuA
751682EC	.text	Export	ModifyMenuW
75139546	.text	Export	MonitorFromPoint
75136EE4	.text	Export	MonitorFromRect
75136C5D	.text	Export	MonitorFromWindow
75186ED9	.text	Export	mouse_event
7512AD9F	.text	Export	MoveWindow

- נחפש את הפונקציה ב-Export table של user32.dll:

- מה ההבדל בין MessageBoxA ל-MessageBoxW?
  - הראשון נועד לתווי ASCII והשני ל- Unicode
  - MessageBoxA ממיר את הקלט ל-Unicode, ואז קורא בתורו ל-MessageBoxW
  - כלומר MessageBoxW ייקרא בסוף בכל מקרה
    - מומלץ לעבור על הקוד של הפונקציה ולאתר את ההמרה

הוקינג של MessageBoxW

- הפונקציה מוכנה להוקינג עם Hot Patching

```
75185F87   90              NOP
75185F88   90              NOP
75185F89   90              NOP
75185F8A   90              NOP
75185F8B   90              NOP
75185F8C   8BFF            MOV EDI,EDI
75185F8E   55              PUSH EBP
75185F8F   8BEC            MOV EBP,ESP
75185F91   6A FF           PUSH -1
75185F93   6A 00           PUSH 0
75185F95   FF75 14         PUSH DWORD PTR SS:[EBP+14]
75185F98   FF75 10         PUSH DWORD PTR SS:[EBP+10]
75185F9B   FF75 0C         PUSH DWORD PTR SS:[EBP+C]
75185F9E   FF75 08         PUSH DWORD PTR SS:[EBP+8]
75185FA1   E8 51000000     CALL USER32.MessageBoxTimeoutW
75185FA6   5D              POP EBP
75185FA7   C2 1000         RETN 10
```

הוקינג ל MessageBoxW

- את mov edi, edi נשנה לקפיצה 7 בתים אחורנית
  - כלומר ל- EB F9 (הערך 7- מיוצג ע"י F9)
- משם נקפוץ לפונקציה שלנו ע"י שינוי התוכן של חמשת פקודות ה-nop ל-jmp
  - לא נוכל לדעת את כתובת הקפיצה משם מראש

```
75185F87  90          NOP
75185F88  90          NOP
75185F89  90          NOP
75185F8A  90          NOP
75185F8B  90          NOP
75185F8C  8BFF        MOV EDI,EDI
75185F8E  55          PUSH EBP
75185F8F  8BEC        MOV EBP,ESP
75185F91  6A FF       PUSH -1
75185F93  6A 00       PUSH 0
75185F95  FF75 14     PUSH DWORD PTR SS:[EBP
75185F98  FF75 10     PUSH DWORD PTR SS:[EBP
75185F9B  FF75 0C     PUSH DWORD PTR SS:[EBP
75185F9E  FF75 08     PUSH DWORD PTR SS:[EBP
75185FA1  E8 51000000 CALL USER32.MessageBox
75185FA6  5D          POP EBP
75185FA7  C2 1000     RETN 10
```

```
90          NOP
EB F9       JMP SHORT USER32.75185F87
55          PUSH EBP
8BEC        MOV EBP,ESP
6A FF       PUSH -1
6A 00       PUSH 0
FF75 14     PUSH DWORD PTR SS:[EBP+14]
FF75 10     PUSH DWORD PTR SS:[EBP+10]
FF75 0C     PUSH DWORD PTR SS:[EBP+C]
FF75 08     PUSH DWORD PTR SS:[EBP+8]
E8 51000000 CALL USER32.MessageBoxTimeoutW
5D          POP EBP
C2 1000     RETN 10
90          NOP
90          NOP
```

פונקציית ה-Hook

```
__declspec(naked) void msgHookW()
{
        __asm {
                mov eax , [esp+8]
                push a
                push 0x40
                push 0x100
                push eax
                call VirtualProtect
                //add esp , 0x10
                mov eax , [esp+8]
loop:
                inc WORD PTR [eax]
                inc eax
                inc eax
                cmp word ptr [eax] , 0
                jne loop
                nop
                …
        }
}
```

כמה דגשים:

- declspec(naked) מגדיר פונקציה ללא תוספות של הקומפיילר (כמו הקצאת frame בתחילת פונקציה)
- הפרמטר **lpText** נמצא בכתובת **esp+8**
- לא ידוע לנו מהן ההרשאות של ה-**Page** בו נמצא **lpText**, לכן נשתמש ב **VirtualProtect** לשינוי הרשאות
  - **VirtualProtect** דואג לשחזור המחסנית בסיומו, לכן אין צורך לעשות **add esp** לניקוי הפרמטרים
- **lpText** מקודדת ב-**Unicode**, כלומר 2 בתים לכל תו
- בסוף הפונקציה יש צורך לחזור ל-**MessageBoxW** (בתוספת שני בתים)
  - אך כרגע אנחנו לא יודעים את הכתובת
  - וכך גם במקרה הכללי
  - לכן נשאיר מקום לפקודת **jmp**
  - אם נדייק, בדוגמא הזו כן ניתן לדעת
    - רמז: נחשו איך אנחנו יודעים את הכתובת של **VirualProtect**

שינוי MessageboxW

חלק ראשון – חיפוש הכתובת

הסבר לחלק הראשון:

- שימוש בפונקציות GetModuleHandle ו-GetProcAddress להשגת הכתובת של MessageBoxW בתהליך שלנו
  - ניתן לקרוא עליהן ב-MSDN
- הכנת מחרוזת JmpOpcode (קפיצה חזרה)

```
void setHook() {
   LPVOID f;
   HMODULE h;
   CHAR  JmpOpcode[6] =    "\xE9\x90\x90\x90\x90";  //jmp (long)
   DWORD lpProtect;
   LPVOID CalculatedJump , JumpTo;
   h = GetModuleHandle(L"user32.dll");
   if (h == NULL) {
      return;
   }
   f = GetProcAddress(h , "MessageBoxW");
   if (f == NULL)
      return;
JumpTo = (char*)&msgHookW - (char*)f;
VirtualProtect((char*)f-5,0x7,PAGE_EXECUTE_READWRITE,&lpProtect);
memcpy(JmpOpcode+1,&JumpTo,0x4);
memcpy((char*)f-5,&JmpOpcode,0x5);
            *(char*)f = 0xEB;
            *((char*)(f)+1) = 0xf9;
VirtualProtect((char*)f-5,0x7,PAGE_EXECUTE_READ,&lpProtect);

            VirtualProtect((char*)msgHookW,0x100,PAGE_EXECUTE_READWRITE,&lpProtect);
            JumpTo = (char*)f + 2 - ((char*)&msgHookW+0x2e+0x5);
            memcpy(JmpOpcode+1,&JumpTo,0x4);
            memcpy((char*)&msgHookW+0x2e,&JmpOpcode,0x5);
VirtualProtect((char*)msgHookW,0x100,PAGE_EXECUTE_READ,&lpProtect);
}
```

שינוי MessageboxW

חלק שני – עדכון הקפיצות

```
void setHook() {
   LPVOID f;
   HMODULE h;
   CHAR  JmpOpcode[6] =    "\xE9\x90\x90\x90\x90";
   DWORD lpProtect;
   LPVOID CalculatedJump , JumpTo;
   h = GetModuleHandle(L"user32.dll");
   if (h == NULL) {
         return;
   }
   f = GetProcAddress(h , "MessageBoxW");
   if (f == NULL)
         return;
   JumpTo = (char*)&msgHookW - (char*)f;
   VirtualProtect((char*)f-5,0x7,
         PAGE_EXECUTE_READWRITE,
                &lpProtect);
   memcpy(JmpOpcode+1,&JumpTo,0x4);
   memcpy((char*)f-5,&JmpOpcode,0x5);
   *(char*)f = 0xEB;            // jmp (short)
   *((char*)(f)+1) = 0xf9;      // -7
   VirtualProtect((char*)f-5,0x7,PAGE_EXECUTE_READ,
                &lpProtect);
         VirtualProtect((char*)msgHookW,0x100,PAGE_EXECUTE_READWRITE,&lpProtect);
         JumpTo = (char*)f + 2 - ((char*)&msgHookW+0x2e+0x5);
         memcpy(JmpOpcode+1,&JumpTo,0x4);
         memcpy((char*)&msgHookW+0x2e,&JmpOpcode,0x5);
VirtualProtect((char*)msgHookW,0x100,PAGE_EXECUTE_READ,&lpProtect);
}
```

הסבר לחלק השני:

- ראשית מחשבים את המרחק בין הפונקציה שלנו ל-MessageBoxW
  - יש לשים לב שהקפיצה מחושבת החל מ-MessageBoxW עצמו מכיוון שהיא נמצאת 5 בתים מלפניו
- מאפשרים כתיבה לפונקציה ע"י VirtualProtect
- מעדכנים את jmpOpcode
- משנים את MessageBoxW

שינוי MessageboxW

```
void setHook() {
   LPVOID f;
   HMODULE h;
   CHAR  JmpOpcode[6] =     "\xE9\x90\x90\x90\x90";
   DWORD lpProtect;
   LPVOID CalculatedJump , JumpTo;
   h = GetModuleHandle(L"user32.dll");
   if (h == NULL) {
         return;
    }
    f = GetProcAddress(h , "MessageBoxW");
    if (f == NULL)
          return;
```

```
E9 74B0B8F4      JMP dllmsg.msgHookW
EB F9            JMP SHORT USER32.75185F87
55               PUSH EBP
8BEC             MOV EBP,ESP
6A FF            PUSH -1
6A 00            PUSH 0
FF75 14          PUSH DWORD PTR SS:[EBP+14]
FF75 10          PUSH DWORD PTR SS:[EBP+10]
FF75 0C          PUSH DWORD PTR SS:[EBP+C]
FF75 08          PUSH DWORD PTR SS:[EBP+8]
E8 51000000      CALL USER32.MessageBoxTimeoutW
5D               POP EBP
C2 1000          RETN 10
```

```
JumpTo = (char*)&msgHookW - (char*)f;
VirtualProtect((char*)f-5,0x7,
        PAGE_EXECUTE_READWRITE,
                &lpProtect);
memcpy(JmpOpcode+1,&JumpTo,0x4);
memcpy((char*)f-5,&JmpOpcode,0x5);
*(char*)f = 0xEB;
*((char*)(f)+1) = 0xf9;
VirtualProtect((char*)f-5,0x7,PAGE_EXECUTE_READ,
                &lpProtect);
        VirtualProtect((char*)msgHookW,0x100,PAGE_EXECUTE_READWRITE,&lpProtect);
        JumpTo = (char*)f + 2 - ((char*)&msgHookW+0x2e+0x5);
        memcpy(JmpOpcode+1,&JumpTo,0x4);
        memcpy((char*)&msgHookW+0x2e,&JmpOpcode,0x5);
VirtualProtect((char*)msgHookW,0x100,PAGE_EXECUTE_READ,&lpProtect);
}
```

שינוי MessageboxW

חלק שלישי – השלמת פונקציית ה–Hook

```
void setHook() {
   LPVOID f;
   HMODULE h;
   CHAR  JmpOpcode[6] =     "\xE9\x90\x90\x90\x90";
   DWORD lpProtect;
   LPVOID CalculatedJump , JumpTo;
   h = GetModuleHandle(L"user32.dll");
   if (h == NULL) {
          return;
    }
    f = GetProcAddress(h , "MessageBoxW");
    if (f == NULL)
           return;
   JumpTo = (char*)&msgHookW - (char*)f;
   VirtualProtect((char*)f-5,0x7,
                    PAGE_EXECUTE_READWRITE,
                                  &lpProtect);
   memcpy(JmpOpcode+1,&JumpTo,0x4);
   memcpy((char*)f-5,&JmpOpcode,0x5);
   *(char*)f = 0xEB;
   *((char*)(f)+1) = 0xf9;
   VirtualProtect((char*)f-5,0x7,PAGE_EXECUTE_READ,
                                  &lpProtect);
```

הסבר לחלק השלישי:

- VirtualProtect לפונקציה שלנו
- חישוב "כמה" לקפוץ:

נרצה לקפוץ ל-msgboxW+2 כאשר הקפיצה תמוקם בפונקציה שלנו החל מהבית 0x2e (זה מספר שרירותי, מה שחשוב זה למצוא מקום עם nop's אחרי הקוד שלנו)

מכיוון שהמרחק מחושב מהפקודה הבאה – נוסיף 5

```
VirtualProtect((char*)msgHookW,0x100,
            PAGE_EXECUTE_READWRITE,&lpProtect);
JumpTo = (char*)f + 2 - ((char*)&msgHookW+0x2e+0x5);
memcpy(JmpOpcode+1,&JumpTo,0x4);
memcpy((char*)&msgHookW+0x2e,&JmpOpcode,0x5);
 VirtualProtect((char*)msgHookW,0x100,
            PAGE_EXECUTE_READ,&lpProtect);
```

השלמת פונקציית ה-Hook

חלק שלישי – השלמת פונקציית ה-Hook

```
__declspec(naked) void msgHookW()
{
        __asm {
                mov eax , [esp+8]
                push a
                push 0x40
                push 0x100
                push eax
                call VirtualProtect
                //add esp , 0x10
                mov eax , [esp+8]
loop:
                inc WORD PTR [eax]
                inc eax
                inc eax
                cmp word ptr [eax] , 0
                jne loop
                nop
                …

                jmp back to MessageBoxW
        }
}
```

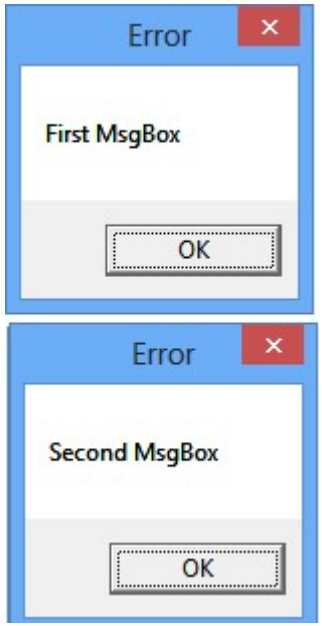
```
MOV EAX,DWORD PTR SS:[ESP+8]
PUSH DWORD PTR DS:[a]
PUSH 40
PUSH 100
PUSH EAX
CALL DWORD PTR DS:[<&KERNEL32.VirtualProtect>]
MOV EAX,DWORD PTR SS:[ESP+8]
INC WORD PTR DS:[EAX]
INC EAX
INC EAX
CMP WORD PTR DS:[EAX],0
JNZ SHORT dllmsg.69D1101C
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
JMP USER32.75185F8E
NOP
NOP
NOP
NOP
NOP
NOP
```

שתילת ה-Hook

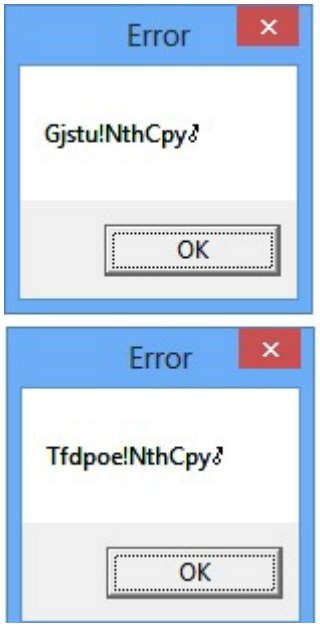
```
INT WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
            LPSTR lpCmdLine, int nCmdShow)
{
   setHook();
   MessageBox(NULL, L"First MsgBox\n",NULL, NULL);
   MessageBox(NULL, L"Second MsgBox\n",NULL, NULL);

   return 0;
}
```

without Hook:



With Hook:



שתילת ה-Hook

- ללא קוד מקור, נוכל לשתול את הקוד שלנו בקובץ ההרצה ע"י שינויו כך שיכלול
  - את הפונקציות msgHookW ו-setHook
    - למשל ב-section חדש, או בהמשך ל-section קיים
  - ולשנות את תחילת WinMain כך שתקרא ל-setHook
    - ע"י הוקינג
- אבל אפשר גם באופן "פחות פולשני"...

שתילת ה-Hook ב-DLL

- נרצה שה Hook שלנו יהיה גנרי, כך שיהיה ניתן להשתמש במודול שלנו בקלות מכל קובץ
- לכן נכתוב את setHook ב-DLL
  - מתכנת שרוצה להשתמש בקוד שלנו – יבצע LoadLibrary

```
INT WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
            LPSTR lpCmdLine, int nCmdShow)
{
   LoadLibrary("setHook.dll");
   MessageBox(NULL, L"First MsgBox\n",NULL, NULL);
   MessageBox(NULL, L"Second MsgBox\n",NULL, NULL);

   return 0;
}
```

שתילת ה-Hook ב-DLL

```
__declspec(naked) void msgHookW()
{
        ...
        ...
}
void sethook() {
        ...
        ...

}

BOOL APIENTRY DllMain( HMODULE hModule,
        DWORD  ul_reason_for_call,
        LPVOID lpReserved
                )
{
        switch (ul_reason_for_call)
        {
        case DLL_PROCESS_ATTACH:
                sethook();
                break;
        case DLL_THREAD_ATTACH:
        case DLL_THREAD_DETACH:
        case DLL_PROCESS_DETACH:
                break;
        }
        return TRUE;
}
```

- ניתן לקרוא ב-**MSDN** על **DLLs**
- חשוב להכיר – ברגע טעינת **DLL** המערכת קוראת ל-**DllMain**
- **DLL** הוא בפורמט **PE**
- בקוד של המתכנת במקום ל-**hook()** קוראים ל-**LoadLibrary**

הזרקת DLL

- נרצה לבצע את ה-Hook גם כאשר אין ברשותנו את קוד המקור
  - לכן נחפש דרך לטעון את ה-DLL לתוכנית בצורה אחרת
- זה נקרא הזרקת DLL
  - DLL Injection
- ישנן מספר שיטות לבצע זאת
  - למשל יש משתנה סביבה שמאפשר לטעון DLL בהרצת EXE
  - אנו נתמקד בשיטה הנפוצה המתבססת על WriteProcessMemory עם CreateRemoteThread
- בשקפים הבאים נראה מימוש דוגמא ל-DLL Injector

הזרקת DLL

```
int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine,int nCmdShow)
{
    PROCESS_INFORMATION pi;
    STARTUPINFOA Startup;
    ZeroMemory(&Startup, sizeof(Startup));
    ZeroMemory(&pi, sizeof(pi));
    CreateProcessA("msgbox.exe", NULL, NULL, NULL, NULL,
                   CREATE_SUSPENDED, NULL, NULL, &Startup, &pi);

    if(!(dllInjector("c:\\dllmsg.dll", pi.dwProcessId)))
        return 1;
    Sleep(1);
    ResumeThread(pi.hThread);
    return 0;
}
```

מידע נוסף?

.MSDN

הזרקת DLL

```
BOOL dllInjector(char * dllpath , DWORD pID)
{
  HANDLE pHandle , threadHandle;
  LPVOID remoteString;
  LPVOID remoteLoadLib;
  pHandle = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pID);
  if(!pHandle)
    return false;
  remoteLoadLib = (LPVOID)GetProcAddress(GetModuleHandle(L"kernel32.dll"),
                                         "LoadLibraryA");
  // error checking
  remoteString = (LPVOID)VirtualAllocEx(pHandle, NULL, strlen(dllpath)+1,
              MEM_RESERVE|MEM_COMMIT, PAGE_READWRITE);
 // error checking
  WriteProcessMemory(pHandle, remoteString, dllpath, strlen(dllpath)+1, NULL);
  threadHandle = CreateRemoteThread(pHandle, NULL, NULL,
              (LPTHREAD_START_ROUTINE)remoteLoadLib, remoteString, NULL, 0));
 // error checking
  return true;
}
```

הזרקת DLL

חשוב להקים את התהליך עם ההרשאות הנכונות.

openProcess יכול גם להיכשל במידה שאין לנו גישה להרשאות שביקשנו.
לעוד מידע בנושא הרשאות ניתן לעיין ב-MSDN בנושא Access Control Model.

```
BOOL dllInjector(char * dllpath , DWORD pID)
{
  HANDLE pHandle , threadHandle;
  LPVOID remoteString;
  LPVOID remoteLoadLib;
  pHandle = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pID);
  if(!pHandle)
    return false;
  remoteLoadLib = (LPVOID)GetProcAddress(GetModuleHandle(L"kernel32.dll"),
                                         "LoadLibraryA");
  // error checking
  remoteString = (LPVOID)VirtualAllocEx(pHandle, NULL, strlen(dllpath)+1,
              MEM_RESERVE|MEM_COMMIT, PAGE_READWRITE);
 // error checking
  WriteProcessMemory(pHandle, remoteString, dllpath, strlen(dllpath)+1, NULL);
  threadHandle = CreateRemoteThread(pHandle, NULL, NULL,
              (LPTHREAD_START_ROUTINE)remoteLoadLib, remoteString, NULL, 0));
 // error checking
  return true;
}
```

הזרקת DLL

השתמשנו בפונקציות הנ"ל בשקפים קודמים.
יש לשים לב שמבקשים את LoadLibrary**A** – כלומר פונקציה שמקבלת ASCII כקלט.

remoteLoadLib יחזיק את הכתובת של LoadLibrary בתהליך שלנו.
הכתובת הזאת זהה גם לתהליכים אחרים במחשב (כי kernel32.dll תמיד נטען לאותו המקום שנקבע מראש בזמן טעינת מ"ה).

```
BOOL dllInjector(char * dllpath , DWORD pID)
{
  HANDLE pHandle , threadHandle;
  LPVOID remoteString;
  LPVOID remoteLoadLib;
  pHandle = OpenProcess(PROCESS_ALL_ACC
  if(!pHandle)
    return false;
  remoteLoadLib = (LPVOID)GetProcAddress(GetModuleHandle(L"kernel32.dll"),
                                         "LoadLibraryA");
  // error checking
  remoteString = (LPVOID)VirtualAllocEx(pHandle, NULL, strlen(dllpath)+1,
              MEM_RESERVE|MEM_COMMIT, PAGE_READWRITE);
 // error checking
  WriteProcessMemory(pHandle, remoteString, dllpath, strlen(dllpath)+1, NULL);
  threadHandle = CreateRemoteThread(pHandle, NULL, NULL,
              (LPTHREAD_START_ROUTINE)remoteLoadLib, remoteString, NULL, 0));
 // error checking
  return true;
}
```

הזרקת DLL

– CreateRemoteThread
קריאה לפונקציה כ-thread חדש בתוכנית אחרת.
הפונקציה מקבלת כתובת לפונקציה וכתובת למשתנה – יש לשים לב שאלו כתובות בתוכנית אחרת – לא בזיכרון שלנו.
לכן צריך לדאוג שגם הכתובת של הפונקציה נכונה (בדקנו) וגם לדאוג שהכתובת של הפרמטר תקינה.

```
BOOL dllInjector(char * dllpath , DWORD pID)
{
  HANDLE pHandle , threadHandle;
  LPVOID remoteString;
  LPVOID remoteLoadLib;
  pHandle = OpenProcess(PROCESS_ALL_ACC
  if(!pHandle)
    return false;
  remoteLoadLib = (LPVOID)GetProcAddress(GetModuleHandle(L"kernel32.dll"),
                                         "LoadLibraryA");
  // error checking
  remoteString = (LPVOID)VirtualAllocEx(pHandle, NULL, strlen(dllpath)+1,
                 MEM_RESERVE|MEM_COMMIT, PAGE_READWRITE);
 // error checking
  WriteProcessMemory(pHandle, remoteString, dllpath, strlen(dllpath)+1, NULL);
  threadHandle = CreateRemoteThread(pHandle, NULL, NULL,
        (LPTHREAD_START_ROUTINE)remoteLoadLib, remoteString, NULL, 0));
 // error checking
  return true;
}
```

הזרקת DLL

```
BOOL dllInjector(char * dllpath , DWORD pID)
{
  HANDLE pHandle , threadHandle;
  LPVOID remoteString;
  LPVOID remoteLoadLib;
  pHandle = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pID);
  if(!pHandle)
    return false;
  remoteLoadLib = (LPVOID)GetProcAddress(GetModuleHandle(L"kernel32.dll"),
                                         "LoadLibraryA");
  // error checking
  remoteString = (LPVOID)VirtualAllocEx(pHandle, NULL, strlen(dllpath)+1,
                 MEM_RESERVE|MEM_COMMIT, PAGE_READWRITE);
 // error checking
  WriteProcessMemory(pHandle, remoteString, dllpath, strlen(dllpath)+1, NULL);
  threadHandle = CreateRemoteThread(pHandle, NULL, NULL,
                 (LPTHREAD_START_ROUTINE)remoteLoadLib, "file.dll", NULL, 0));
 // error checking
  return true;
}
```

מה היה קורה אילו במקום remoteString היינו כותבים "file.dll" ?

הזרקת DLL

שימוש ב-VirtualAllocEx (הקצאת זיכרון בתהליך מרוחק) ו-WriteProcessMemory (כתיבת זיכרון בתהליך מרוחק).
כך נבטיח שהמחרוזת "c:\\dllmsg.dll" נמצאת ב-Proccess המרוחק.

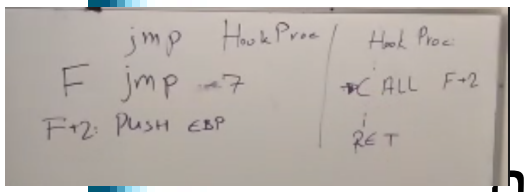
```
BOOL dllInjector(char * dllpath , DWORD pI
{
  HANDLE pHandle , threadHandle;
  LPVOID remoteString;
  LPVOID remoteLoadLib;
  pHandle = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pID);
  if(!pHandle)
    return false;
  remoteLoadLib = (LPVOID)GetProcAddress(GetModuleHandle(L"kernel32.dll"),
                                         "LoadLibraryA");
  // error checking
  remoteString = (LPVOID)VirtualAllocEx(pHandle, NULL, strlen(dllpath)+1,
                 MEM_RESERVE|MEM_COMMIT, PAGE_READWRITE);
  // error checking
  WriteProcessMemory(pHandle, remoteString, dllpath, strlen(dllpath)+1, NULL);
  threadHandle = CreateRemoteThread(pHandle, NULL, NULL,
                 (LPTHREAD_START_ROUTINE)remoteLoadLib,remoteString, NULL, 0));
  // error checking
  return true;
}
```

הזרקת DLL

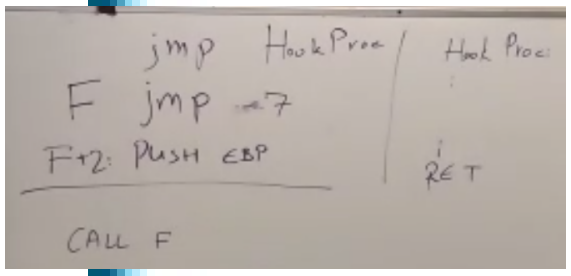
- הצגנו דוגמא לשינוי קוד ע"י הזרקת DLL
  - יש דרכים נוספות לבצע זאת
- הצגנו את הדוגמא על תוכנית ייעודית, אך ניתן כמובן להזריק את ה-DLL לכל תהליך
  - חפשו ב-MSDN כיצד לקבל רשימה של תהליכים רצים
  - רמז: CreateToolhelp32Snapshot

הוק בסוף פונקציה

- כל מה שעשינו עד עתה מבצע הוק בתחילת פונקציה
  - כלומר מאפשר לנו להריץ קוד לפני שהפונקציה רצה
  - כולל אפשרות לשנות את הפרמטרים שהיא מקבלת כקלט
- לעיתים נרצה להחליף את הפונקציה באחרת בלי לקרא למקורית
  - איך נעשה זאת?



- במקרים אחרים נרצה להריץ קוד בסוף הפונקציה
  - למשל קוד שמשנה את ערך החזרה שלה
  - חישבו על לפחות שלוש דרכים לעשות זאת...
    - חשוב לשים לב לשני דברים:
      - איך לקבל את הבקרה בסוף חישוב הפונקציה?
      - איך נוכל לקרא לה למרות ששתלנו בה הוק?



- במקרים מורכבים יותר נרצה
  - לקרא לפונקציה המקורית מההוק יותר מפעם אחת

הוק בסוף פונקציה בלי hot-patching

- במקרה שאין תמיכה ב-hot-patching הפתרון נעשה מסובך יותר
- מה הבעיות הנוספות שנוצרות?
- איך ניתן לפתור אותן?

- רמז 1: קריאה לפונקציה
- רמז 2: ביצוע פקודות המכונה שמוחלפות ע"י jmp אחרי פקודת ה-call לפונקציה

הוקינג בעולם האמיתי

אז מה היה לנו?

Dll Injection => SetHook() => HookProc()

- אפשר לשנות בקוד המקור
- אפשר לשנות בקובץ הבינארי
- יש פונקציות API המאפשרות hooking במקרים ספציפיים.
- קיימות עוד שיטות לon-the-fly hooking

Instrumentation

בהברה: הכלים שיידונו בסעיף זה אינם מותרים לשימוש בתרגילי הבית, בסדנא או במבחן

Pin Tutorial
(Robert Cohn, Intel)l

Detours: Binary Interception of Win32 Functions
(Galen Hunt and Doug Brubacher, MS Research

What is Instrumentation?

- A technique that inserts extra code into a program to collect runtime information

```
    counter++;
$0xff, %edx sub
    counter++;
%esi, %edx cmp
    counter++;
<L1> jle
    counter++;
$0x1, %edi mov
    counter++;
$0x10, %eax add
```

Instrumentation Approaches

- Source instrumentation:
  - Instrument source programs
- **Binary instrumentation**:
  - Instrument executables directly

Advantages for binary instrumentation

✓ Language independent

✓ Machine-level view

✓ Instrument legacy/proprietary software

Instrumentation Approaches

When to instrument:

- Instrument statically – before runtime
- Instrument dynamically – at runtime

Advantages for dynamic instrumentation

✓ No need to recompile or relink
✓ Discover code at runtime
✓ Handle dynamically-generated code
✓ Attach to running processes

How is Instrumentation used in Computer Architecture Research?

- **Trace Generation**
- **Branch Predictor and Cache Modeling**
- **Fault Tolerance Studies**
- **Emulating Speculation**
- **Emulating New Instructions**

How is Instrumentation used in Program Analysis?

- **Code coverage**
- **Call-graph generation**
- **Memory-leak detection**
- **Instruction profiling**
- **Data dependence profiling**
- **Thread analysis**
  - Thread profiling
  - Race detection

Detours

- Is a library for instrumenting and intercepting function calls in Win32 binaries.
- Replaces the first instructions of a *target* function with jmp to a *detour* function.
- Preserves original function semantics through a *trampoline* function.
- Enables interception and instrumentation of Win32 binary programs.

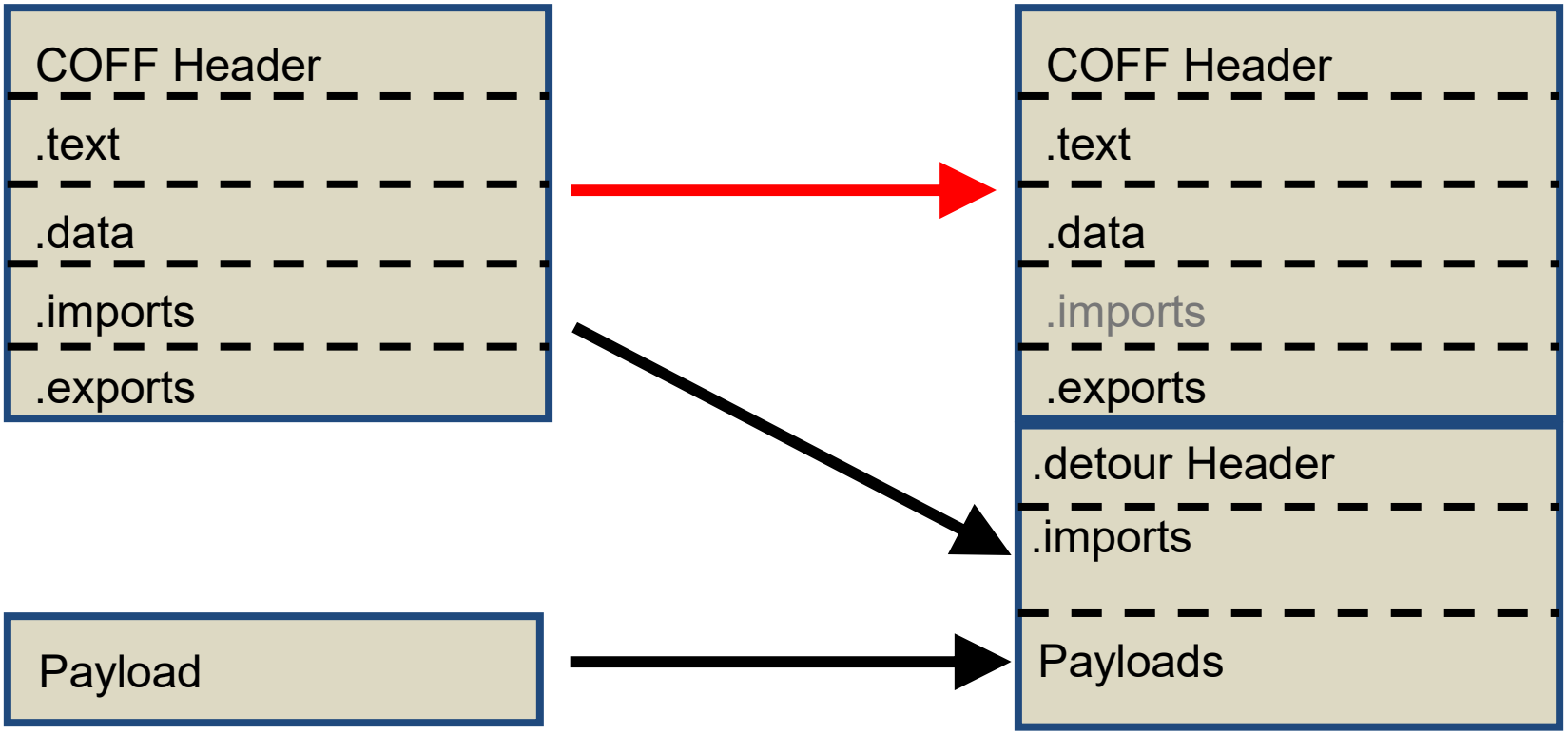
How do you get your code invoked?

- Replace first instructions of target with a jump to the detour.
- Insert replaced instructions into trampoline.
- Trampolines can be allocated and initialized either statically or dynamically (*see paper for dynamic*).

How do you get your code into an application's address space?

- First: Place code into a DLL.
- Then do one of the following:
  - Link application with your DLL.
    - Only works if you have .obj files.
  - Modify application .imports to include DLL.
    - Detours includes routines for editing .imports.
  - Inject DLL into running process.
    - Detours calls OpenProcess(), VirtualAllocEx(), WriteProcessMemory(), and CreateRemoteThread()
  - Inject DLL into process at creation time.
    - Detours calls CreateProcess() w/ CREATE_SUSPENDED.

Rewriting a Binary:



Detouring a Function:

Before:

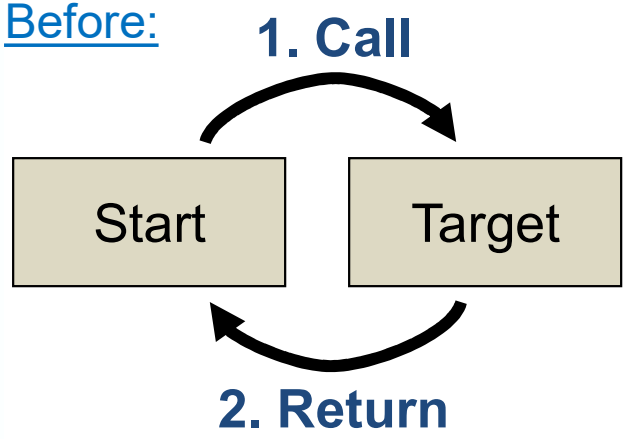
```
;; Target Function
Sleep:
  push   ebp        [1 byte]
  mov    ebp,esp    [2 bytes]
  push   ebx        [1 bytes]
  push   esi        [1 byte]
  push   edi
  ....
;; Trampoline Function
UntimedSleep:
  jmp    Sleep
;; Detour Function
TimedSleep:
  ....
```

After:

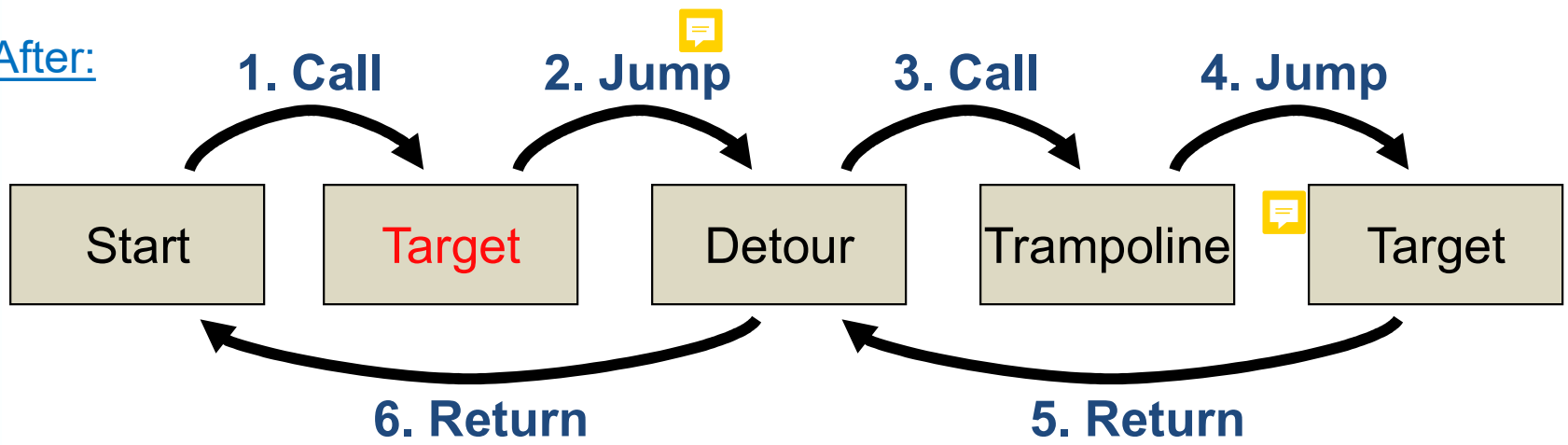
```
;; Target Function
Sleep:
  jmp    TimedSleep [5 bytes]
  push   edi
  ....
;; Trampoline Function
UntimedSleep:
  push   ebp
  mov    ebp,esp
  push   ebx
  push   esi
  jmp    Sleep+5
;; Detour Function
TimedSleep:
  ....
```

Invoking Your Code:

Before:



After:



Pin Instrumentation APIs

- Basic APIs are architecture independent:
  - Provide common functionalities like determining:
    - Control-flow changes
    - Memory accesses
- Architecture-specific APIs
  - e.g., Info about opcodes and operands
- Call-based APIs:
  - Instrumentation routines
  - Analysis routines

HOOKS

- The heart of Pin's approach to instrumentation
- Analysis and Instrumentation
- Can be placed on various events / objects, e.g:
  - Instructions
  - Context switch
  - Thread creation
  - Much more…

Instrumentation vs. Analysis

- Instrumentation routines define where instrumentation is inserted
  - e.g., before instruction
  - ☞ **Occurs *first time* an instruction is executed**
  - ☞ Usually defined in the tool "main"
  - ☞ Heavy lifting
- Analysis routines define what to do when instrumentation is activated
  - e.g., increment counter
  - ☞ **Occurs *every time* an instruction is executed**
  - ☞ Usually defined in instrumentation routine
  - ☞ As light as possible

ManualExamples/inscount0.cpp

```
#include <iostream>
#include "pin.h"

UINT64 icount = 0;

void docount() { icount++; }
```

analysis routine

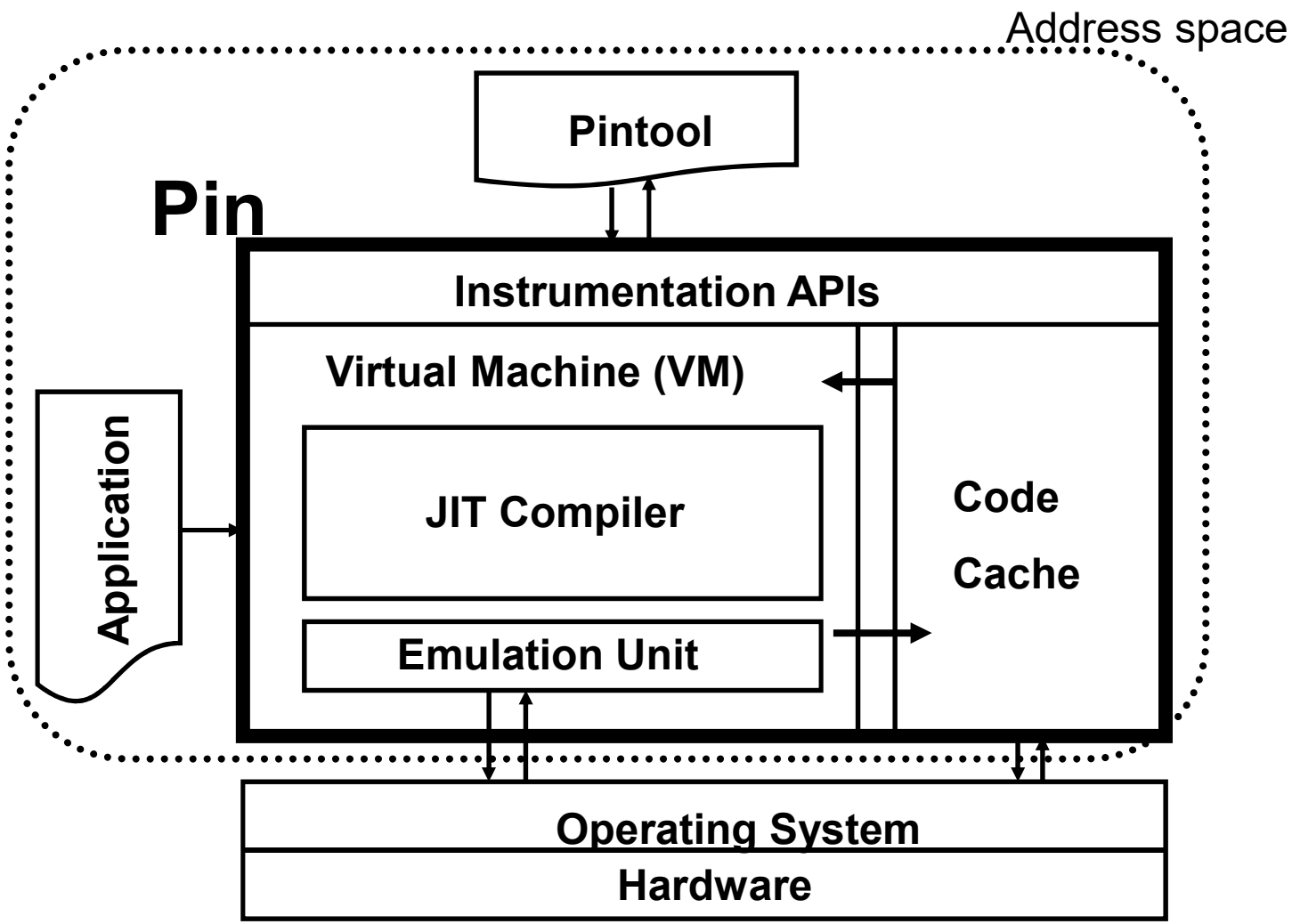
```
void Instruction(INS ins, void *v)
{
    INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)docount, IARG_END);
}
```

instrumentation routine

```
void Fini(INT32 code, void *v)
{ std::cerr << "Count " << icount << endl; }

int main(int argc, char * argv[])
{
    PIN_Init(argc, argv);
    INS_AddInstrumentFunction(Instruction, 0);
    PIN_AddFiniFunction(Fini, 0);
    PIN_StartProgram();
    return 0;
}
```

Pin's Software Architecture



Instrumentation Approaches

- JIT Mode
  - Pin creates a modified copy of the application on-the-fly
  - Original code never executes

    ➢ More flexible, more common approach

- Probe Mode
  - Pin modifies the original application instructions
  - Inserts jumps to instrumentation code (trampolines)

    ➢ Lower overhead (less flexible) approach

A Sample Probe

- **A probe is a jump instruction that overwrites original instruction(s) in the application**
  - Instrumentation invoked with probes
  - Pin copies/translates original bytes so probed functions can be called

Original function entry point:

```
0x400113d4:  push   %ebp
0x400113d5:  mov    %esp,%ebp
0x400113d7:  push   %edi
0x400113d8:  push   %esi
0x400113d9:  push   %ebx
```

Entry point overwritten with probe:

```
0x400113d4:   jmp   0x41481064
0x400113d9:   push  %ebx
```

Copy of entry point with original bytes:

```
0x50000004:  push   %ebp
0x50000005:  mov    %esp,%ebp
0x50000007:  push   %edi
0x50000008:  push   %esi
0x50000009:  jmp    0x400113d9
```

PinProbes Instrumentation

- Advantages:
  - Low overhead – few percent
  - Less intrusive – execute original code
  - Leverages Pin:
    - API
    - Instrumentation engine
- Disadvantages:
  - More tool writer responsibility
  - Routine-level granularity (RTN)

Instrumentation Approaches

- JIT Mode
  - Pin creates a modified copy of the application on-the-fly
  - Original code never executes

    ➢ More flexible, more common approach

- Probe Mode
  - Pin modifies the original application instructions
  - Inserts jumps to instrumentation code (trampolines)

    ➢ Lower overhead (less flexible) approach

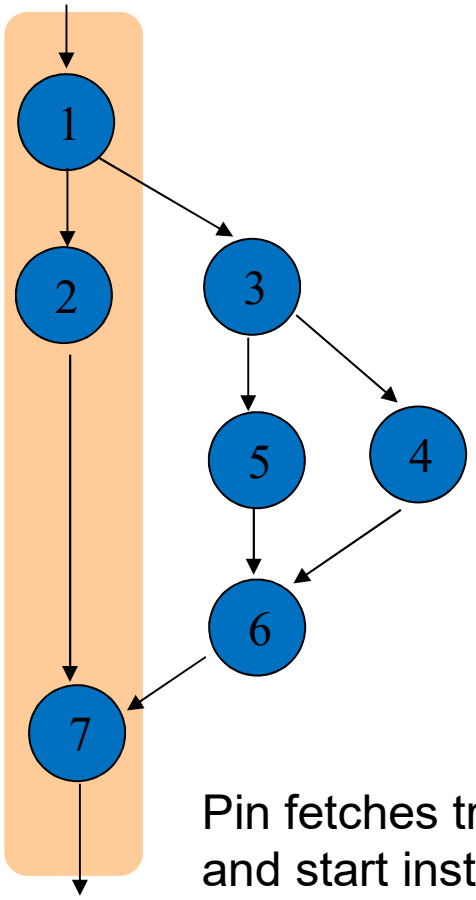
טכנולוגיית JIT

Just In Time compilation

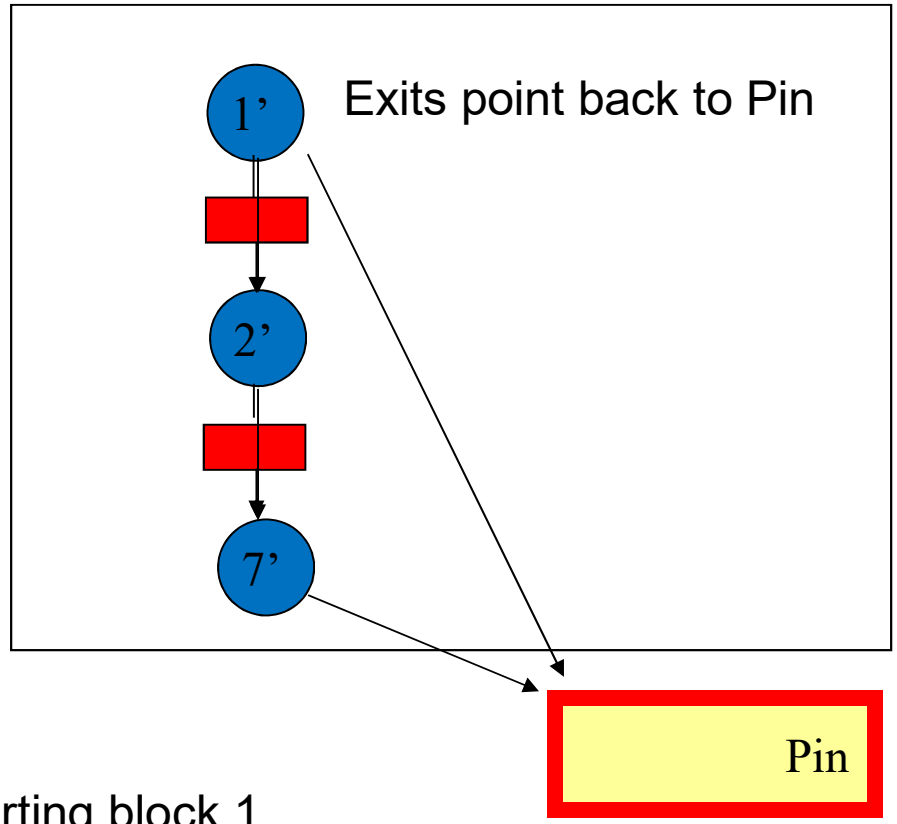
- הידור קוד בזמן ריצה
- יכולת לבצע אינטרפרטציה לקוד, ולבצע הידור רק לקוד שבשימוש רב
  - וללמוד מהאינטרפרטציה איך כדאי לבצע אופטימיזציות
- מאפשר לוודא שהקוד המהודר אינו מתוכנן לגרום נזק
  - קוד הביניים פשוט יותר לבדיקה
  - הקוד המהודר אינו ניתן לטיפול ע"י תוקף
  - כלומר, נוזקה אינה יכולה להשתלט על הקוד...

JIT-Mode Instrumentation

Original code



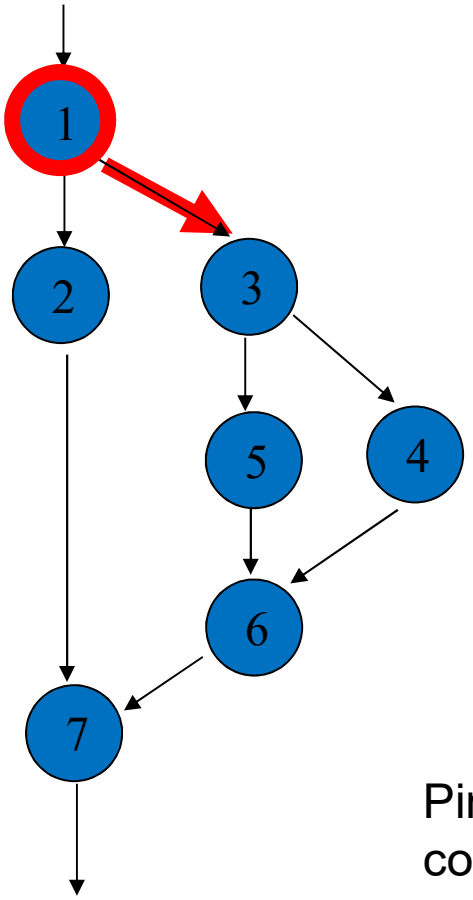
Code cache



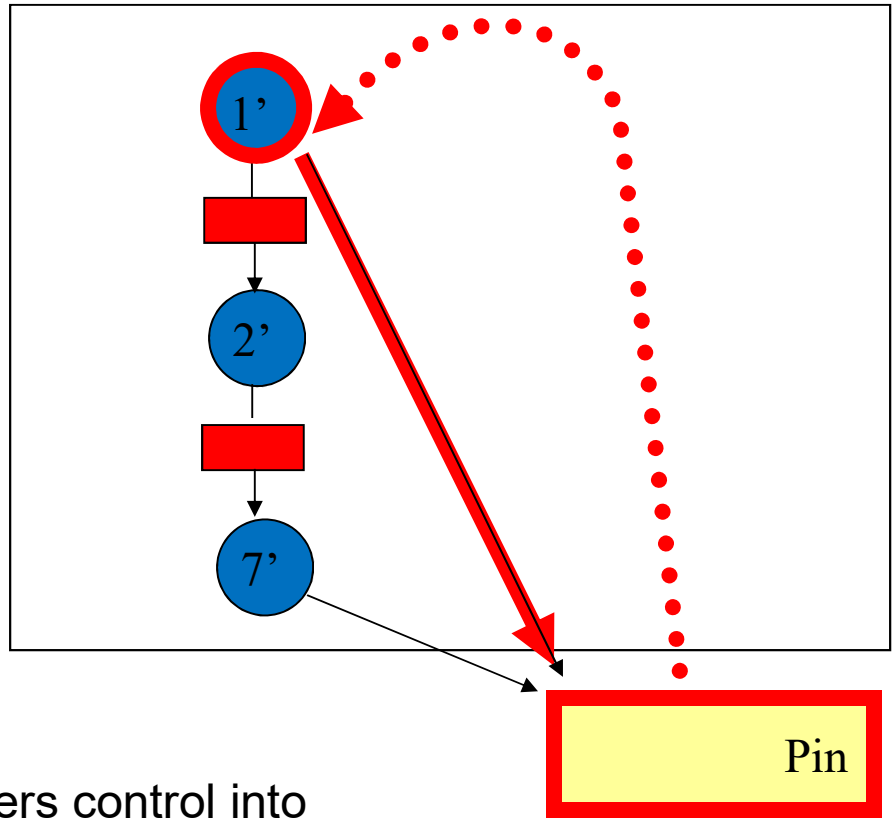
Pin fetches trace starting block 1
and start instrumentation

JIT-Mode Instrumentation

Original code



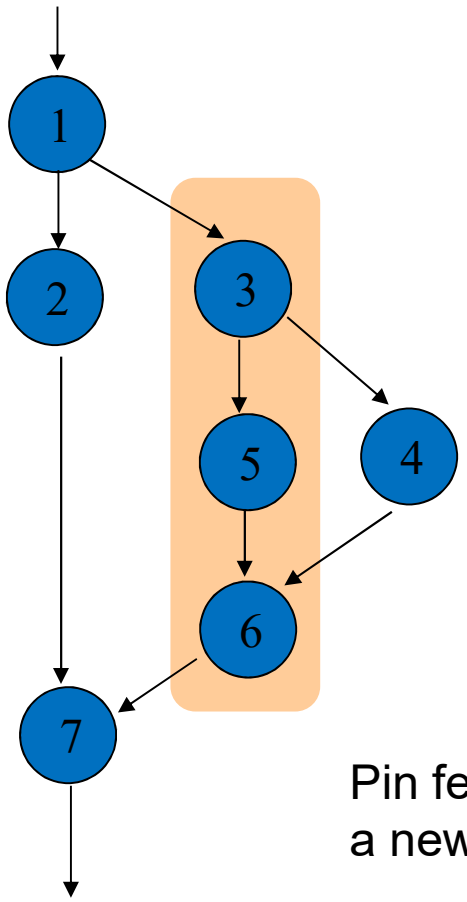
Code cache



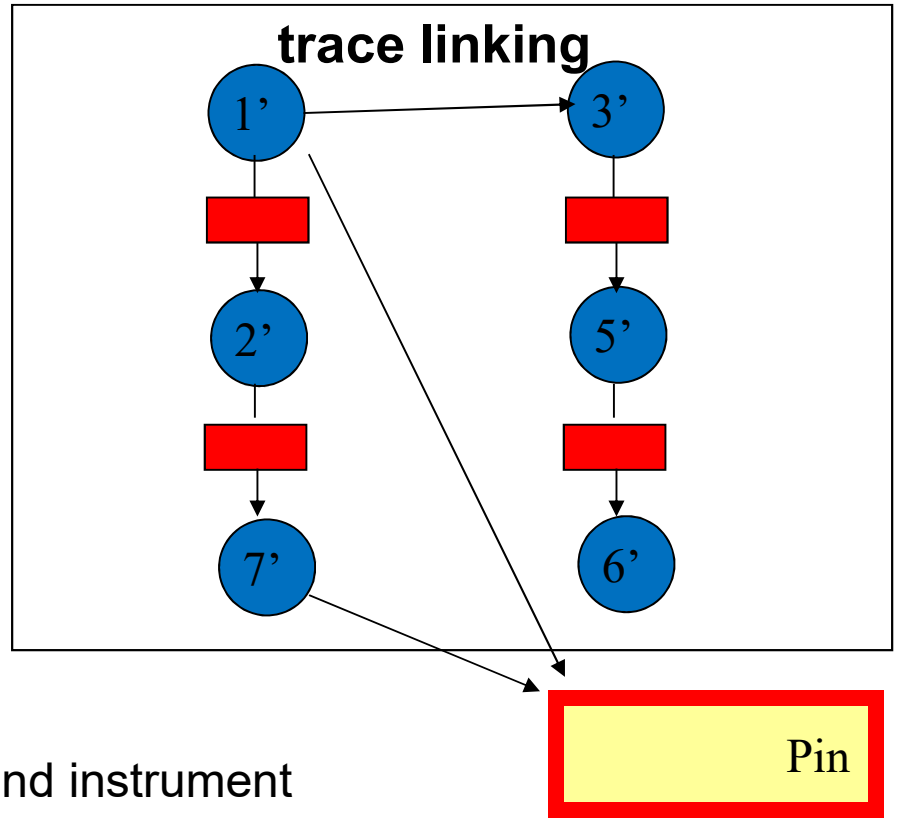
Pin transfers control into code cache (block 1)

JIT-Mode Instrumentation

Original code



Code cache



Pin fetches and instrument
a new trace

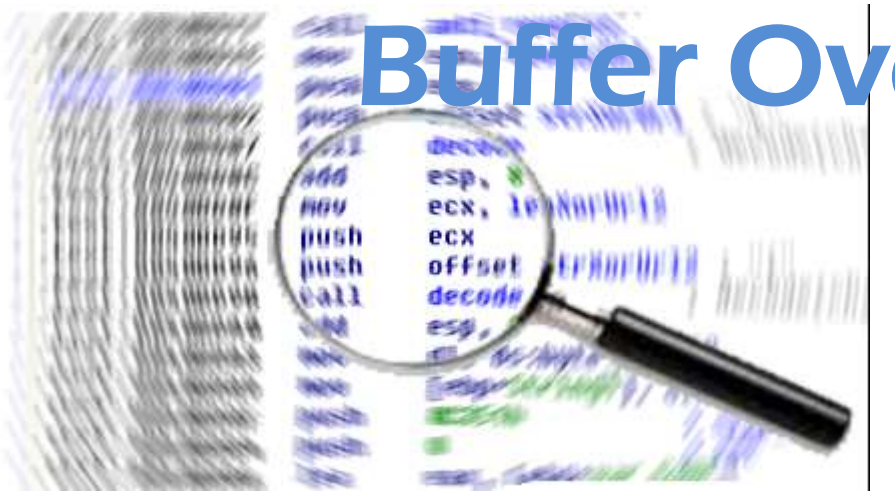
Pin Probes Summary

	PinProbes	PinClassic (JIT)
Overhead	Few percent	50% or higher
Intrusive	Low	High
Granularity	Function boundary	Instruction
Safety & Isolation	More responsibility for tool writer	High

שאלות?

חולשות ניכרון

Buffer Overflow



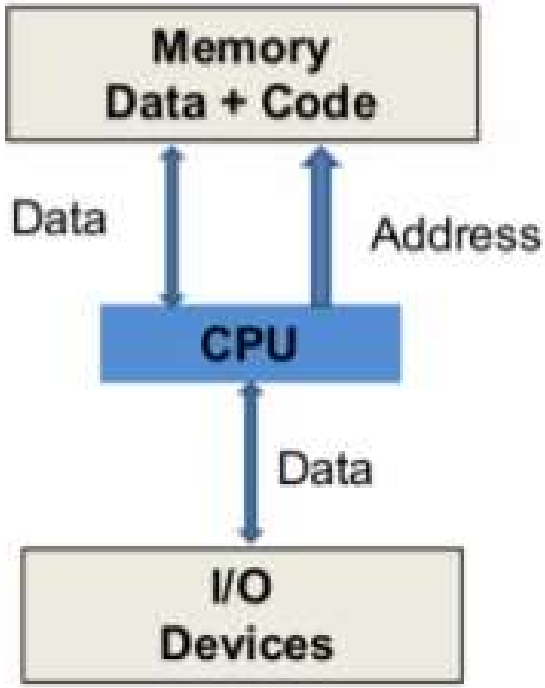
מה תוקף רודה?

להריץ קוד על המחשב של הנתקף

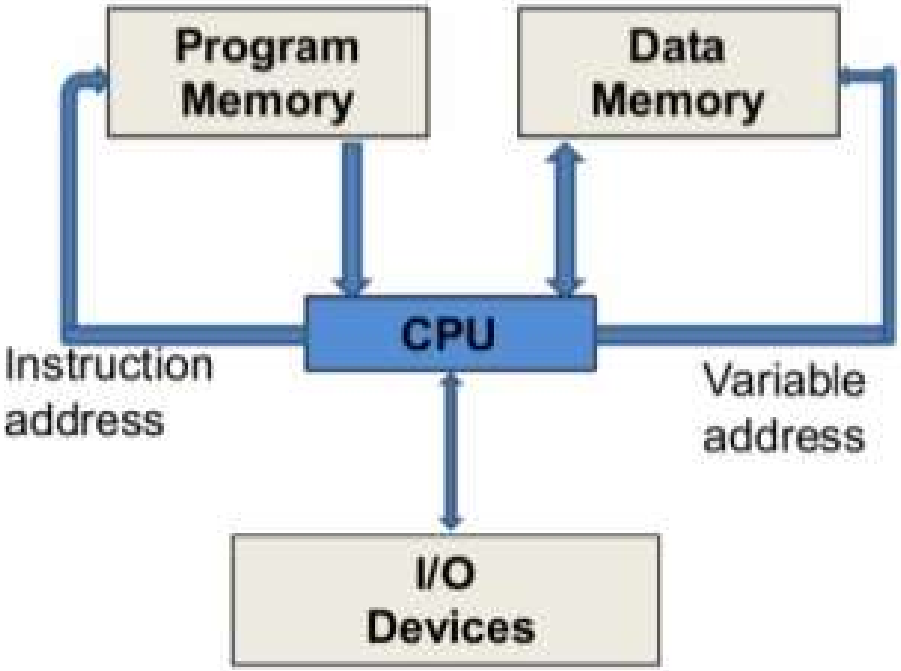
- איך אפשר לעשות את זה?
  - Social Engineering
  - "תקיפת המחשב" באמצעות חולשה
- מה זו חולשה?
  - באג לוגי / תכנוני / מימושי שניתן לנצל אותו

ארכיטקטורת המחשב

- Von Neumann



- Harvard

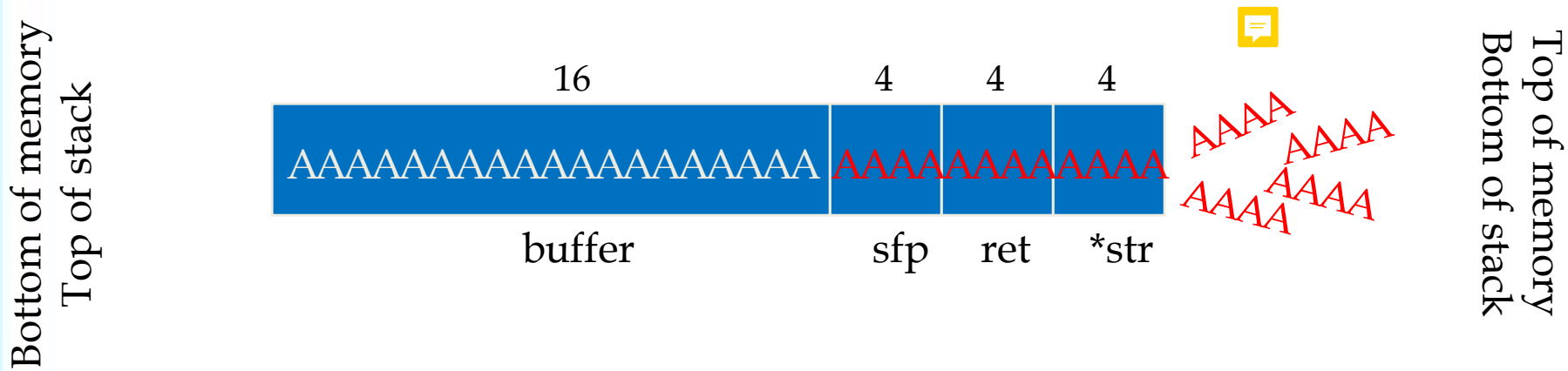


מה הבעיה?

```
4
5  void function(char *str){
6      char buffer[16];
7      strcpy(buffer, str);
8  }
9
10
11
12
13 int main(){
14     char large_string[256];
15     int i;
16
17     for (i = 0; i < 255; i++){
18         large_string[i] = 'A';
19     }
20
21     function(large_string);
22 }
```

Buffer Overflaw

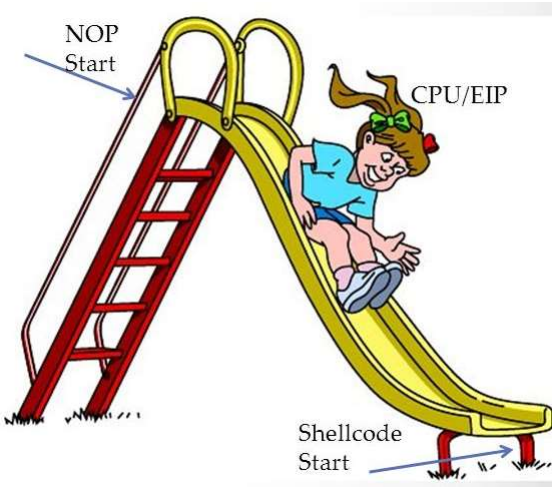
- כאשר התוכנית רצה נגרמת Seg-Fault



- כתובת החזרה נדרסת על ידי 'AAAA' (0x41414141)
- כשהפונקציה מסיימת היא מנסה לחזור לכתובת 0x41414141

אבל איך מנצלים את זה?

- דורסים את כתובת החזרה
  - לאן להצביע?
- כותבים קטע קוד שאינו תלוי במיקומו (Shellcode)
  - ושמים אותו על הstack
  - איך נתכונן לעובדה שאנחנו לא יודעים תמיד בדיוק מה יש על הStack? ומה המרחק הנכון?
    - NOOOOOOOOOOOOOOOOOOOOP



- הקוד מורץ!

File: archives/49/p49_0x0e_Smashing The Stack For Fun And Profit_by_Aleph1.txt
.oO Phrack 49 Oo. 1996

Volume Seven, Issue Forty-Nine

File 14 of 16

BugTraq, r00t, and Underground.Org
bring you

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Smashing The Stack For Fun And Profit
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

by Aleph One
aleph1@underground.org

`smash the stack` [C programming] n. On many C implementations it is possible to corrupt the execution stack by writing past the end of an array declared auto in a routine. Code that does this is said to smash the stack, and can cause return from the routine to jump to a random address. This can produce some of the most insidious data-dependent bugs known to mankind. Variants include trash the stack, scribble the stack, mangle the stack; the term mung the stack is not used, as this is never done intentionally. See spam; see also alias bug, fandango on core, memory leak, precedence lossage, overrun screw.

Introduction

Over the last few months there has been a large increase of buffer

איך ניתן למנוע את זה?

- קידוד מאובטח...
  - שימוש בספריות מאובטחות
  - שימוש בשפות שהן Type-Safe
  - Static Analysis

Dangerous C system calls

source: Building secure software, J. Viega & G. McGraw, 2002

Extreme risk
- gets

High risk
- strcpy
- strcat
- sprintf
- scanf
- sscanf
- fscanf
- vfscanf
- vsscanf

High risk (cntd)
- streadd
- strecpy
- strtrns
- realpath
- syslog
- getenv
- getopt
- getopt_long
- getpass

Moderate risk
- getchar
- fgetc
- getc
- read
- bcopy

Low risk
- fgets
- memcpy
- snprintf
- strccpy
- strcadd
- strncpy
- strncat
- vsnprintf

strncmp(computed_response, user_response, response_length)

```
NETSTACK_CODE:20431F80 loc_20431F80:                          # CODE XREF: NETSTACK_AuthDigestParseResponse+FE↑j
NETSTACK_CODE:20431F80         add     r14, sp, 0x10C+var_F4
NETSTACK_CODE:20431F82         mov     r0, r13
NETSTACK_CODE:20431F84         add     r0, r0, 0x55
NETSTACK_CODE:20431F86         mov     r1, r14
NETSTACK_CODE:20431F88         bl      NETSTACK_CODE_2043218C
NETSTACK_CODE:20431F8C         ld      r4, [sp,0x10C+nc.value_len]
NETSTACK_CODE:20431F90         ld      r5, [sp,0x10C+var_74]
NETSTACK_CODE:20431F94         ld      r0, [sp,0x10C+qop.value_len]
NETSTACK_CODE:20431F98         ld      r7, [sp,0x10C+qop]
NETSTACK_CODE:20431F9C         st      r0, [sp,0x10C+var_10C]
NETSTACK_CODE:20431F9E         ld      r0, [sp,0x10C+uri]
NETSTACK_CODE:20431FA2         st      a1, [sp,0x10C+var_108]
NETSTACK_CODE:20431FA6         st      r0, [sp,0x10C+var_104]
NETSTACK_CODE:20431FA8         ld      r0, [sp,0x10C+uri.value_len]
NETSTACK_CODE:20431FAC         ld      r6, [sp,0x10C+var_70]
NETSTACK_CODE:20431FB0         add     r13, sp, 0x10C+var_D0
NETSTACK_CODE:20431FB2         st      r0, [sp,0x10C+var_100]
NETSTACK_CODE:20431FB4         st      req_text, [sp,0x10C+var_FC]
NETSTACK_CODE:20431FB8         st      r13, [sp,0x10C+var_F8]
NETSTACK_CODE:20431FBA         ld      r1, [sp,0x10C+nonce]
NETSTACK_CODE:20431FBC         mov     r0, r14
NETSTACK_CODE:20431FBE         ld      r2, [sp,0x10C+nonce.value_len]
NETSTACK_CODE:20431FC0         ld      r3, [sp,0x10C+nc]
NETSTACK_CODE:20431FC4         bl      NETSTACK_CODE_204321D0
NETSTACK_CODE:20431FC8         ld      r1, [sp,0x10C+user_response]
NETSTACK_CODE:20431FCC         mov     r0, r13                 # computed_response
NETSTACK_CODE:20431FCE         ld      r2, [sp,0xA4]           # response_length
NETSTACK_CODE:20431FD2         bl      strncmp
NETSTACK_CODE:20431FD6         cmp     r0, 0
NETSTACK_CODE:20431FD8         bne     error
```

Silent Bob

- An authentication bypass vulnerability, which will be later known as CVE-2017-5689, was originally discovered in mid-February of 2017

- It seems quite obvious that the third argument of strncmp() should be the length of computed_response , but the address of the stack variable response_length, from where the length is to be loaded, actually points to the length of the **user_response!**

איך ניתן למנוע את זה?

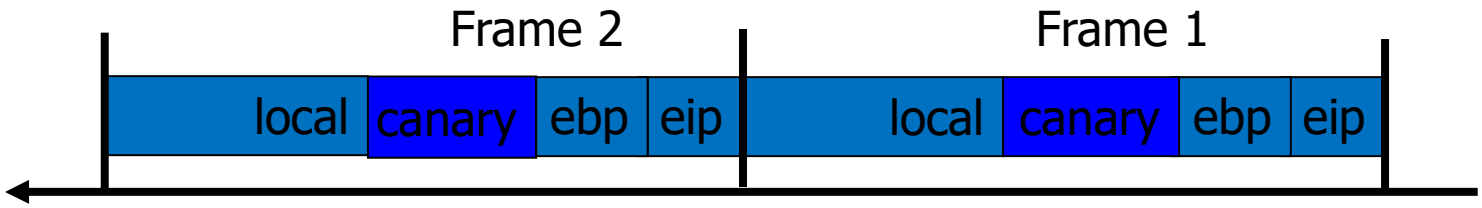
- קידוד מאובטח...
  - שימוש בספריות מאובטחות
  - שימוש בשפות שהן Type-Safe
  - Static Analysis

- בדיקות בזמן ריצה
  - Canaries \ Cookies

- Non Executable Stack
  - ATL Thunk: Legacy!

Stack Cookies /GS

- בדיקת "זמן ריצה".
- בזמן הידור מוכנס "משתנה" נוסף בין המשתנים המקומיים וכתובת החזרה.
  - איזה ערכים כדאי שיהיו למשתנה הזה? מה לא?
- מה המאפיינים שצריכים להיות למשתנה?
  - מה יכול תוקף ללמוד?
    - מריצה אחרת?
    - מראיית הזכרון?
    - מהכרת תהליכים אחרים?
- איפה עוד אפשר לשים קנריות?



How is it done?

```
; prologue

push    ebp
mov     ebp, esp
sub     esp, 214h
mov     eax, ___security_cookie ; random value, initialized at module startup
xor     eax, ebp                ; XOR it with the current base pointer
mov     [ebp+var_4], eax        ; store the cookie

...

; epilogue

mov     ecx, [ebp+var_4]        ; get the cookie from the stack
xor     ecx, ebp                ; XOR the cookie with the current base pointer
call    __security_check_cookie ; check the cookie
leave
retn    0Ch

; __fastcall __security_check_cookie(x)

cmp     ecx, ___security_cookie
jnz     ___report_gsfailure     ; terminate the process
rep retn
```

#pragme strict_gs_check

- Extra prologe & epiloge – significant overhead.
- So with /GS adds stack cookie only to functions that contain string buffers of allocate memory on the stack.
- A compiler directive (VS 20005 SP1+) that enables more aggressive GS heuristics
- Adds a cookie to all functions that use address of local variable.
- Result – more protection vs. runtime performance.

מה קורה כשהבדיקה נכשלת?

טיפול בחריגות

Exception Handling

מבוא לטיפול בחריגות

- טיפול בחריגות הינו דבר נפוץ שנמצא בשימוש כמעט בכל תוכנית.
- כחוקרים אנו ניתקל רבות בטיפול בחריגות, ועלינו לזהות ולהבין כיצד זה עובד.

- נתחיל משירותי מ״ה לחריגות ברמת התהליך.
- ונמשיך עם איך השירותים הללו ניתנים ע״י גרעין מ״ה.

טיפול בחריגות ב-C וב-++C

C

```
__try
{
        // guarded code
}
__except ( expression )
{
        // exception handler code
}
```

C++

```
try {

        //throw
}
catch (…)
 {
        // exception handler code
}
```

שירותי מ"ה לטיפול בחריגות

Structured Exception Handling (SEH)

- מערכת ההפעלה מאפשרת טיפול בחריגות ע"י מנגנון בסיסי המשותף לכל המהדרים והתהליכים.

- כל קומפיילר מממש Exception-Handling בצורה שונה.
  - לא ניכנס לפרטים לגבי המימושים השונים של הקומפיילרים.
  - קומפיילרים מרחיבים את המנגנון של מערכת ההפעלה.

מבוא ל-SEH

- תוכנית יכולה לקרוס מסיבות שונות (חלוקה ב-0 למשל).
- מערכת ההפעלה מיידעת את התוכנית על הקריסה, ואף נותנת לה הזדמנות לתקן את הבעיה.
  - מ"ה קוראת לפונקציה ששייכת לתוכנית שקרסה.
  - הפונקציה תעשה מה שתעשה, ותחזיר למ"ה ערך חזרה.
  - בתחילת ריצת התכנית או קטע ה-try..except התוכנית כמובן צריכה ליידע את מ"ה שיש לה פונקציה כזאת ומה הכתובת שלה.
- לפונקציה קוראים except_handler.

except_handler הפונקציה

- הפונקציה שנקראת בזמן חריגה.
  - בדרך כלל מדובר בפונקציה שהקומפיילר מספק.
  - אבל גם התכנית יכולה לספק אחת כזו.
- קלט:
  - מקבלת מידע לגבי סוג החריגה ומצב התהליך בזמן החריגה.
- ערכי החזרה:
  - ExceptionContinueExecution:
    - מסמל טיפול מוצלח בחריגה.
  - ExceptionContinueSearch:
    - אם הפונקציה לא יודעת כיצד לטפל בחריגה.
  - יש עוד שני ערכי חזרה אפשריים עליהם לא נרחיב:
    - ExceptionNestedException.
    - ExceptionCollidedUnwind.

except_handler הפונקציה

```
EXCEPTION_DISPOSITION
        except_handler(
                struct _EXCEPTION_RECORD *ExceptionRecord,
                void * EstablisherFrame,
                struct _CONTEXT *ContextRecord,
                 void * DispatcherContext
  );
```

EXCEPTION_RECORD

```
typedef struct _EXCEPTION_RECORD {
        DWORD ExceptionCode;
        DWORD ExceptionFlags;
        struct _EXCEPTION_RECORD *ExceptionRecord;
        PVOID ExceptionAddress;
        DWORD NumberParameters;
        DWORD ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS];
}  EXCEPTION_RECORD;
```

דוגמא לכמה ערכי ExceptionCode (מתוך אלפים אפשריים) :

0xC0000093
STATUS_FLOAT_UNDERFLOW
0xC0000094
STATUS_INTEGER_DIVIDE_BY_ZERO
0xC0000095
STATUS_INTEGER_OVERFLOW

ניתן למצוא רשימה מלאה ב-MSDN תחת ***NTSTATUS values***.

CONTEXT

```
typedef struct _CONTEXT
{
    ULONG ContextFlags;
    ULONG Dr0;
    ULONG Dr1;
    ULONG Dr2;
    ULONG Dr3;
    ULONG Dr6;
    ULONG Dr7;
    FLOATING_SAVE_AREA FloatSave;
    ULONG SegGs;
    ULONG SegFs;
    ULONG SegEs;
    ULONG SegDs;
    ULONG Edi;
    ULONG Esi;
    ULONG Ebx;
    ULONG Edx;
    ULONG Ecx;
    ULONG Eax;
    ULONG Ebp;
    ULONG Eip;
    ULONG SegCs;
    ULONG EFlags;
    ULONG Esp;
    ULONG SegSs;
    UCHAR ExtendedRegisters[512];
} CONTEXT, *PCONTEXT;
```

המבנה CONTEXT משמש גם כקלט וגם כפלט: בתחילה הוא מכיל את מצב האוגרים בעת החריגה ובהמשך הפונקציה except_handler יכולה לשנות אותו.
אם הפונקציה except_handler החזירה **ExceptionContinueExecution**, אז מ״ה תעדכן את האוגרים בהתאם למבנה CONTEXT.

דוגמה – except_handler

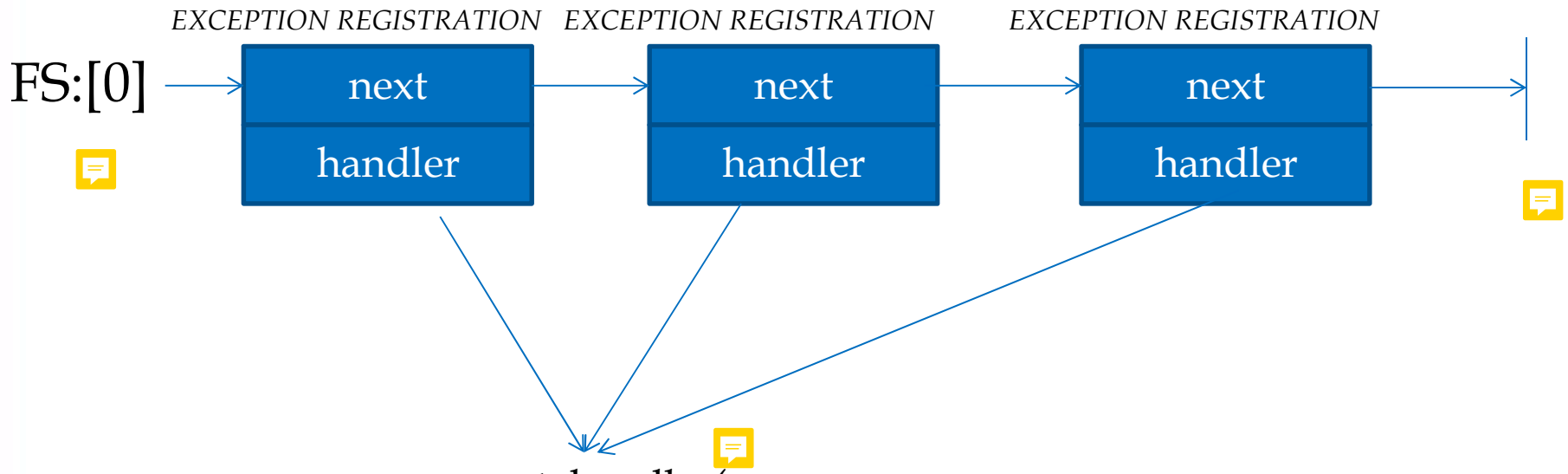
```
exit_label:
        ExitProcess();


_except_handler(struct _EXCEPTION_RECORD *ExceptionRecord,
                void * EstablisherFrame,
                 struct _CONTEXT *ContextRecord,
                void * DispatcherContext )
{
        ContextRecord->Eip = exit_label;
        return ExceptionContinueExecution;
}
```

EXCEPTION REGISTRATION

- ראינו כיצד נראית הפונקציה שמ״ה תקרא לה.
- אבל איך ניידע את מ״ה שיש לנו פונקציה כזאת?
  - איך מ״ה תמצא אותה בזיכרון התוכנית?

EXCEPTION REGISTRATION




```
except_handler(
        struct _EXCEPTION_RECORD *ExceptionRecord,
        void *     EstablisherFrame,
        struct _CONTEXT *ContextRecord,
         void * DispatcherContext );
```

Thread Environment Block

- מבנה שמכיל מידע על Thread מסוים.
  - לכל Thread יש אחד כזה.
- מגיעים אליו דרך FS.

```
typedef struct _TEB
{
   struct _NT_TIB
   {
      PEXCEPTION_REGISTRATION_RECORD ExceptionList;
      PVOID StackBase;
      PVOID StackLimit;
      PVOID SubSystemTib;
      …
   }

   PVOID EnvironmentPointer;
   CLIENT_ID ClientId;
            …
            …
```

EXCEPTION REGISTRATION

- המידע שמור במבנה הבא:

```
struct EXCEPTION_REGISTRATION {
        EXCEPTION_REGISTRATION * next;
        void *handler;
}
```

  - Handler – מצביע לפונקציה.
  - Next – מצביע ל-EXCEPTION REGISTRATION
    - יש מאמרים שמציגים את המבנה עם שדה prev, המשמעות זהה.
    - למעשה מדובר ברשימה מקושרת.
- **מקובל לשים את אברי הרשומות על המחסנית.**

KNCIP

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

Set the handler

```
... some code
```

The code in the “try” block

```
pop fs:[0]
add esp,4
```

Restore (delete) the exception record

```
except_handler:
…some code…
mov eax , ?
ret
```

0 – ExceptionContinueExecution
1 - ExceptionContinueSearch

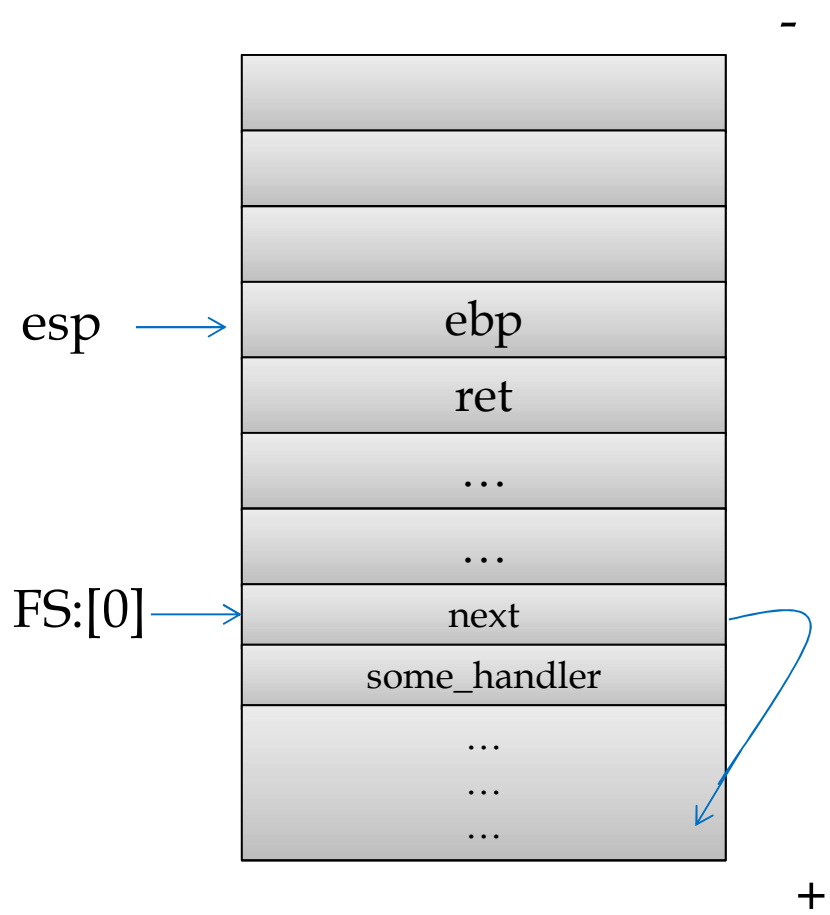
דוגמה

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

... some code

pop fs:[0]
add esp,4

except_handler:
…some code…
mov eax , ?
ret
```



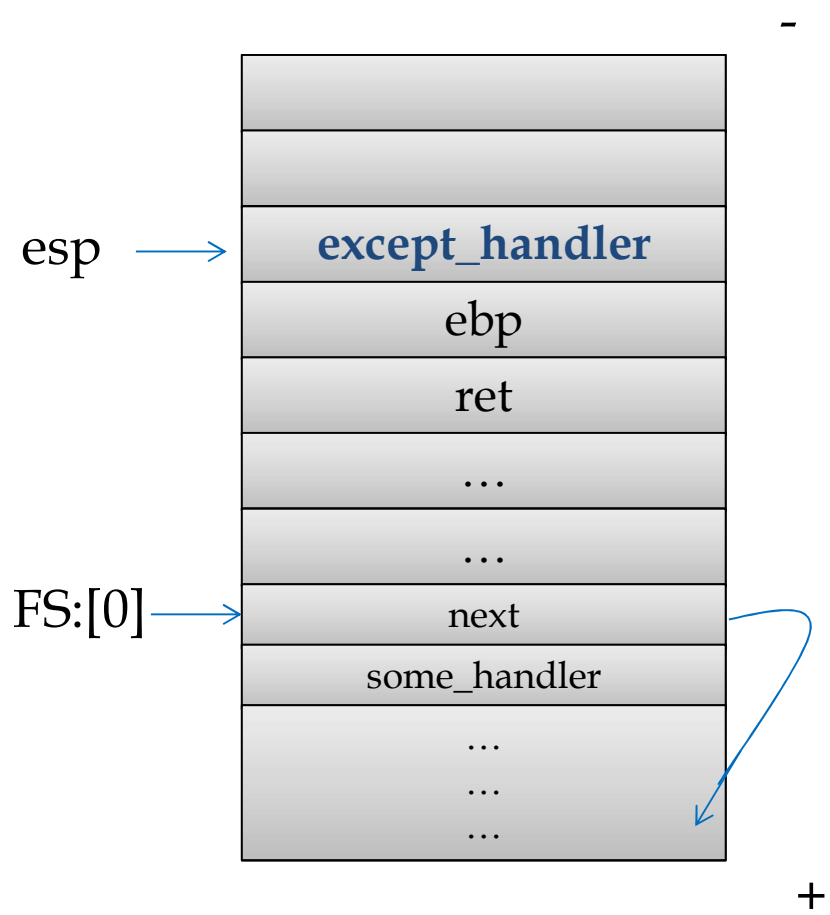
דוגמה

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

... some code

pop fs:[0]
add esp,4

except_handler:
…some code…
mov eax , ?
ret
```



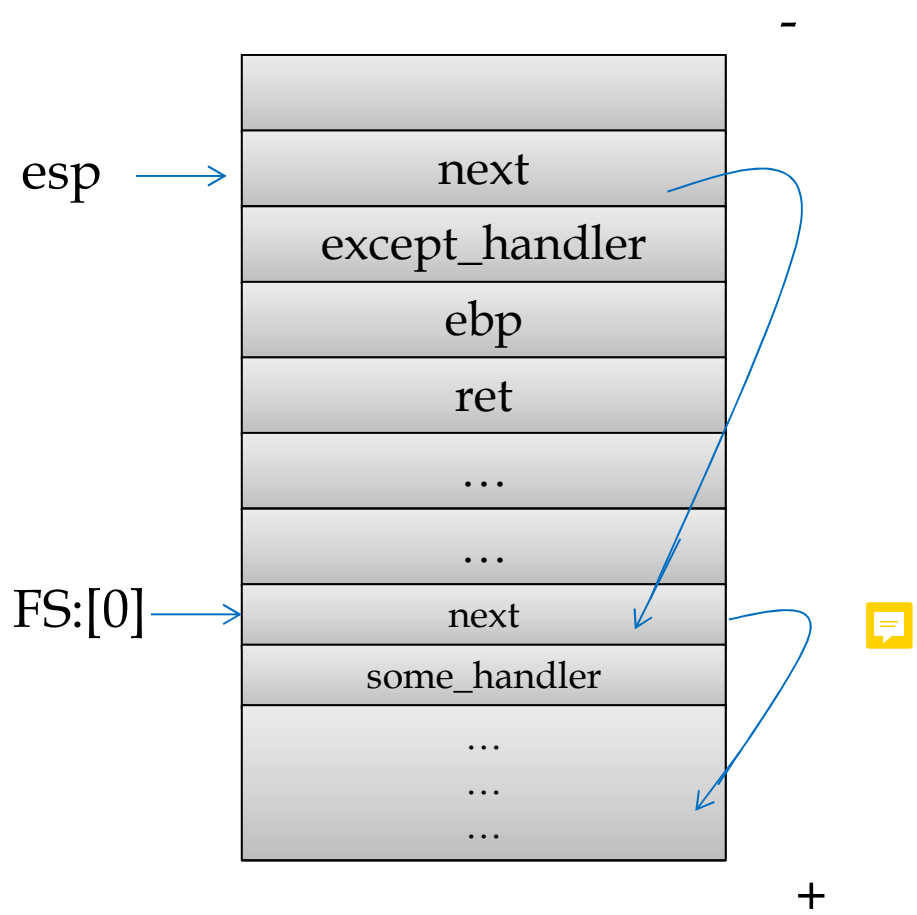
הגנה

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

... some code

pop fs:[0]
add esp,4

except_handler:
…some code…
mov eax , ?
ret
```



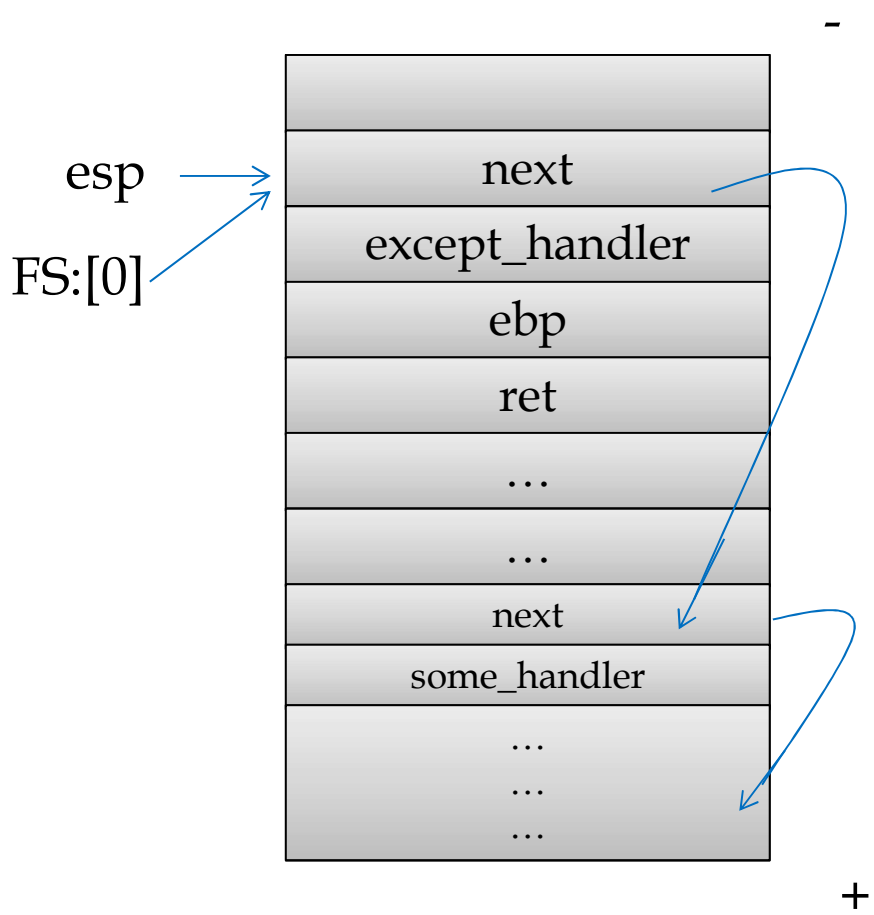
דוגמה

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

... some code

pop fs:[0]
add esp,4

except_handler:
…some code…
mov eax , ?
ret
```



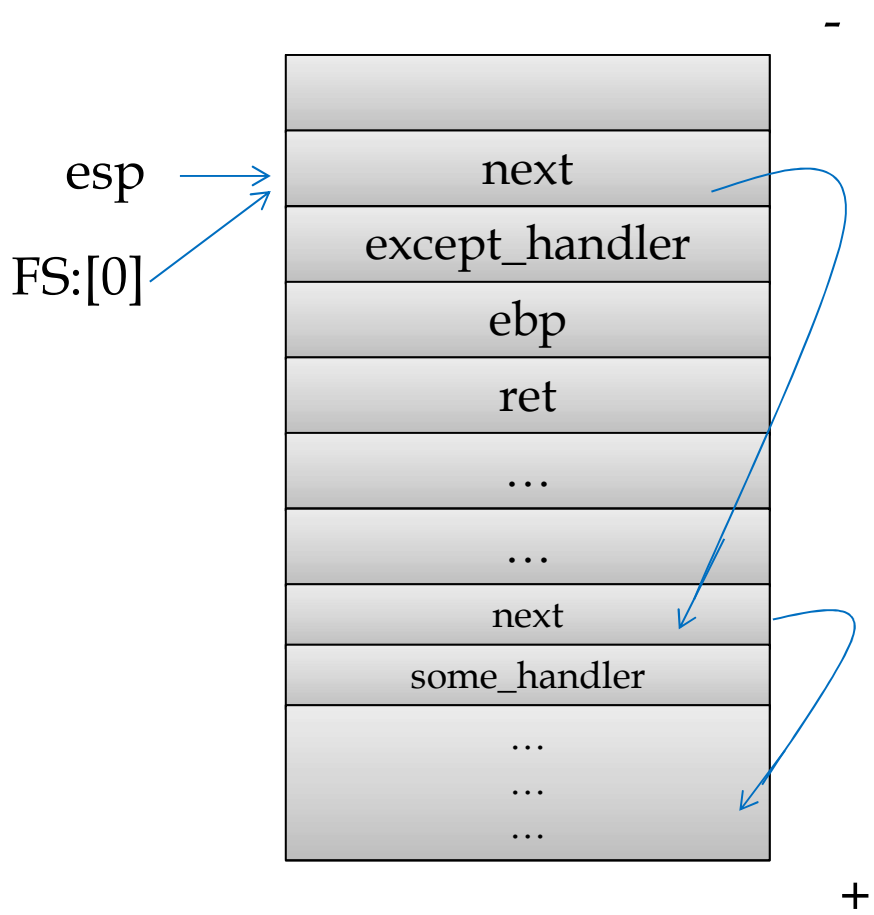
דוגמה

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

... some code

pop fs:[0]
add esp,4

except_handler:
…some code…
mov eax , ?
ret
```



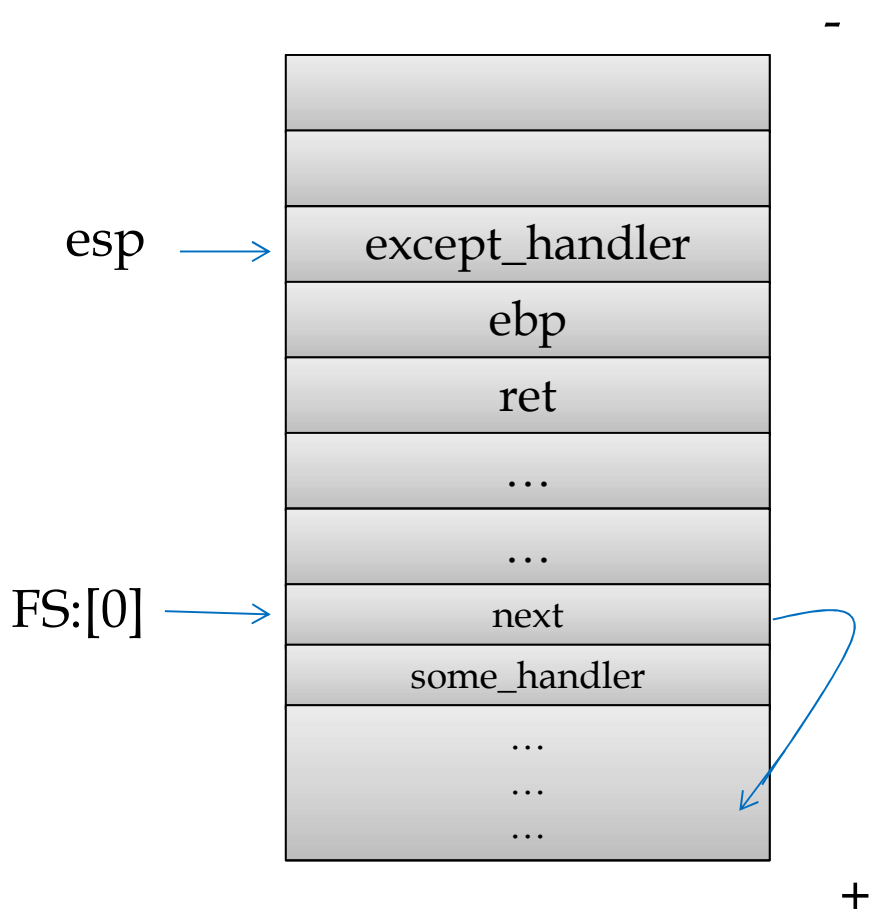
דוגמה

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

... some code

pop fs:[0]
add esp,4

except_handler:
…some code…
mov eax , ?
ret
```



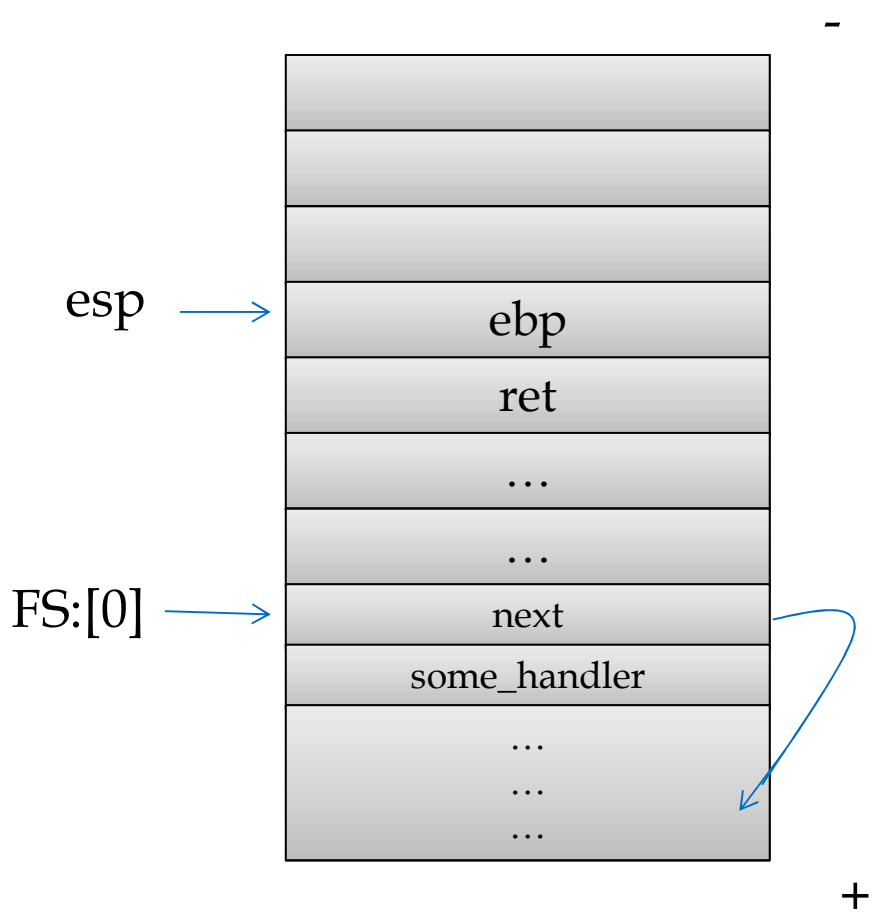
דוגמה

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

... some code

pop fs:[0]
add esp,4

except_handler:
…some code…
mov eax , ?
ret
```



שחרור משאבים

- כאמור, בעת חריגה מ״ה עוברת על הרשימה עד למציאת ה-handler שמצליח לטפל בחריגה.
- ה-handler שמצליח אחראי למחוק את יתרת המחסנית ואת רשומות ה-Exception Registration המיותרות.
  - יש לו חופש פעולה למחוק את מה שרלוונטי לאותו מקרה.
  - בדר״כ כל ה-handlers שסירבו לטפל בחריגה, כולל זה שהצליח.
  - אבל לעיתים ללא מחיקה.
    - למשל תיקון חילוק באפס שמחזיר ערך קבוע כתוצאת החילוק.
- במהלך המחיקה, כל handler שנמחק נקרא פעם נוספת לצורך שחרור משאבים.
  - למשל שחרור מחלקה ב-C++.
- לא נרחיב לעומק כיצד המנגנון עובד.

שאלה

- מה קורה אם נזרקת חריגה בקוד מבלי שהשתמשנו ב-try?
- תשובה:
  - בתחילת הריצה מ"ה מאתחלת את fs:[0] במצביע ל-handler שלה.
  - כאשר ה-handler נקרא, הוא מציג הודעה מתאימה.
  - הוא נקרא אם כל האחרים לא הצליחו לטפל בחריגה.

סיכום חריגות ברמת התהליך

- ה-Thread צריך לספק למערכת ההפעלה רשימה מקושרת של Exception Registration.
- הקומפיילר מממש את המנגנון שלו בשימוש ב-SEH.
  - כל try יוצר ExceptionRegistration.
  - ה-handler יהיה קוד פנימי של הקומפיילר שייקרא בסוף לקוד שלכם (מה שכתבתם בתוך ה-except).
  - הקומפיילר כאמור יכול להרחיב את המנגנון.
    - למשל באתחול ה-ExceptionRegistration נראה פקודות נוספות שקומפיילר יצר.

SEH Implementation

```
0189FF88  0189FF94
0189FF8C  764B1174  kernel32.764B1174
0189FF90  00000060
0189FF94  0189FFD4
0189FF98  777EB429  RETURN to ntdll.777EB429
0189FF9C  00000060
0189FFA0  7626C3EF
0189FFA4  00000000
0189FFA8  00000000
0189FFAC  00000060
0189FFB0  00000000
0189FFB4  00000000
0189FFB8  00000000
0189FFBC  0189FFA0
0189FFC0  00000000
0189FFC4  FFFFFFFF  End of SEH chain
0189FFC8  777AD555  SE handler
0189FFCC  00D1398B
0189FFD0  00000000
0189FFD4  0189FFEC
0189FFD8  777EB3FC  RETURN to ntdll.777EB3FC from ntdll.777EB402
0189FFDC  00401848  vulnserv.00401848
0189FFE0  00000060
0189FFE4  00000000
0189FFE8  00000000
0189FFEC  00000000
0189FFF0  00000000
0189FFF4  00401848  vulnserv.00401848
0189FFF8  00000060
0189FFFC  00000000
```

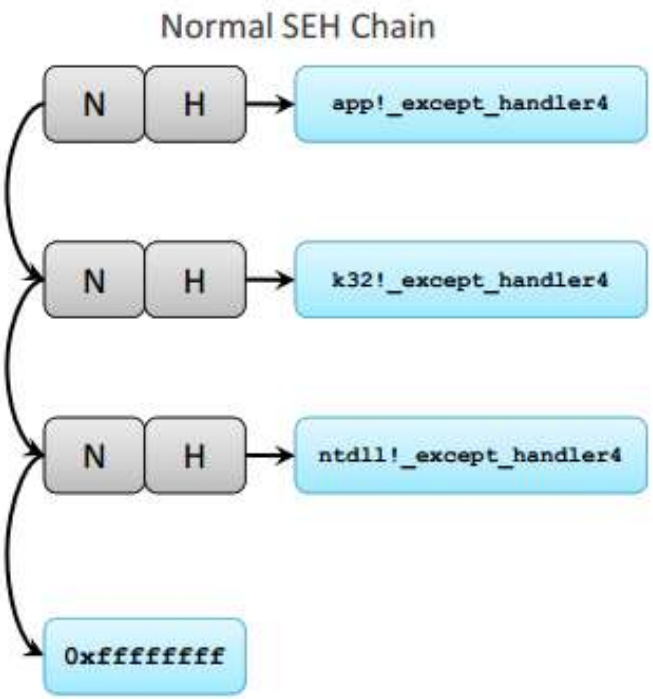
Did you say "STACK"?

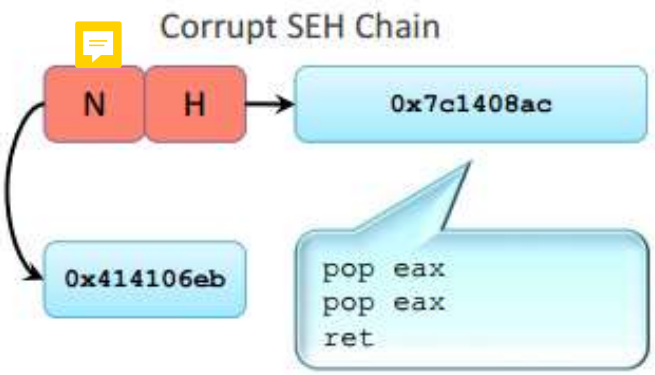
- AAAA

Buffer Overflow to the Rescue!

019FFF84	41414141	
019FFF88	41414141	
019FFF8C	41414141	
019FFF90	41414141	
019FFF94	41414141	
019FFF98	41414141	
019FFF9C	41414141	
019FFFA0	41414141	
019FFFA4	41414141	
019FFFA8	41414141	
019FFFAC	41414141	
019FFFB0	41414141	
019FFFB4	41414141	
019FFFB8	41414141	
019FFFBC	41414141	
019FFFC0	41414141	
019FFFC4	41414141	Pointer to next SEH record
019FFFC8	41414141	SE handler
019FFFCC	41414141	
019FFFD0	41414141	
019FFFD4	41414141	
019FFFD8	41414141	
019FFFDC	41414141	
019FFFE0	41414141	
019FFFE4	41414141	
019FFFE8	41414141	
019FFFEC	41414141	
019FFFF0	41414141	
019FFFF4	41414141	
019FFFF8	41414141	
019FFFFC	41414141	

Exploit: SEH Overwrite

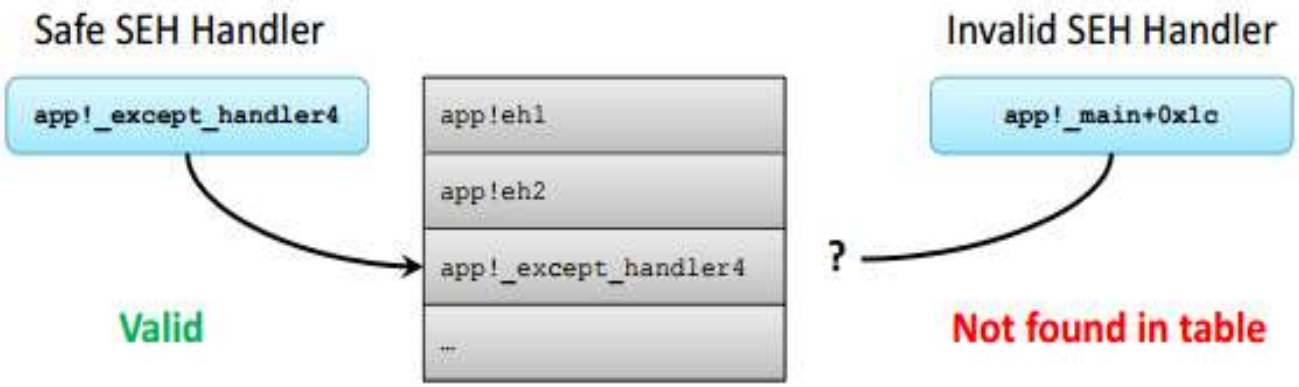




An exception will cause `0x7c1408ac` to be called as an exception handler as:

```
EXCEPTION_DISPOSITION Handler(
   PEXCEPTION_RECORD Exception,
   PVOID EstablisherFrame,
   PCONTEXT ContextRecord,
   PVOID DispatcherContext);
```

Mitigation: SafeSEH



- VS2003 linker change (/SAFESEH) [9]
- Binaries are linked with a table of safe exception handlers
  - Stored in program memory – not corruptible by an attacker
- Exception dispatcher checks if handlers are safe before calling

When SafeSEH Is Incomplete

[Sotirov and Dowd]

- If DEP is disabled, handler is allowed to be on any non-image page except stack
  - Put attack code on the heap, overwrite exception handler record on the stack to point to it
- If any module is linked without /SafeSEH, handler is allowed to be anywhere in this module
  - Overwrite exception handler record on the stack to point to a suitable place in the module
  - Used to exploit Microsoft DNS RPC vulnerability in Windows Server 2003

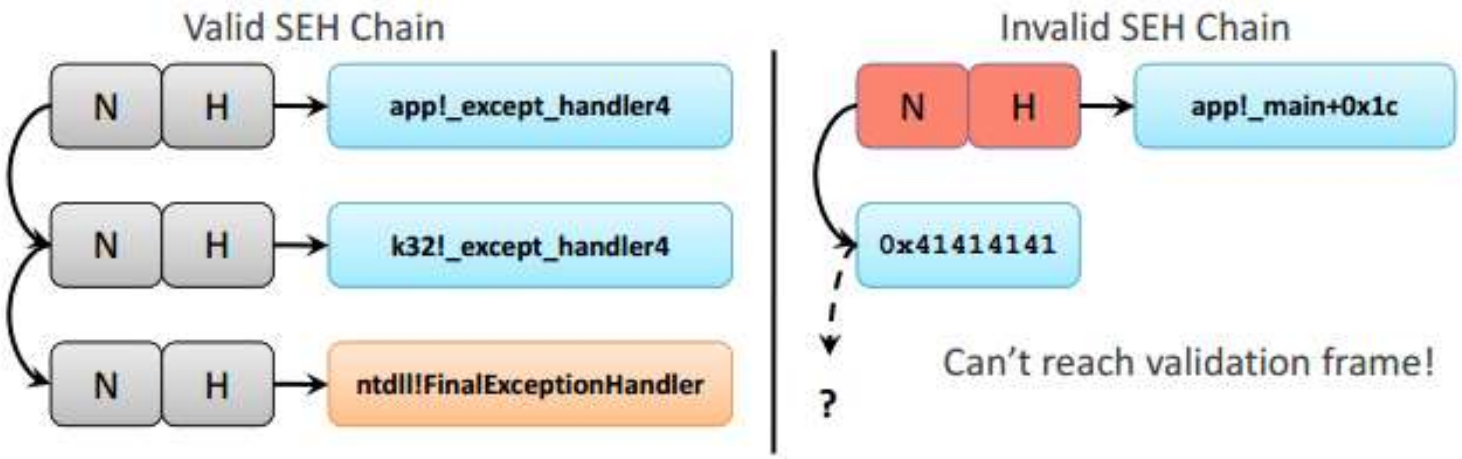
Safe Exception Handling

- Exception handler record must be on the stack of the current thread (why?)
- Must point outside the stack (why?)
- Must point to a valid handler
  - Microsoft's /SafeSEH linker option: header of the binary lists all valid handlers
- Exception handler records must form a linked list, terminating in FinalExceptionHandler
  - Windows Server 2008: SEH chain validation
  - Address of FinalExceptionHandler is randomized (why?)

Sentinel

- Sentinel: a fake value in a linked list whose only role is to be recognizable
- We insert the sentinel at the end of a list
- We also keep a copy of it
- When following the list, we compare every record to the sentinel
  - When we get to the sentinel, we know we reached the end of the list
  - If we never get to the sentinel, we know the list was tampered with

Mitigation: SEHOP



- Dynamic protection for SEH overwrites in Srv08/Vista SP1 [4]
  - No compile/link time hints required
- Symbolic *validation frame* inserted as final entry in chain
- Corrupt `Next` pointers prevent traversal to validation frame

SafeSEH

- /IMAGE_DLLCHARACTERISTICS_NO_SEH
  - A flag set on a DLL that prevents any addresses from the DLL being used as SEH handlers.
- /SafeSEH
  - linker option.
  - Only addresses listed as on a registered SEH handlers list can be used within that module
- SEHOP – SEH Overwrite Protection.
  - Vista an onwards.
  - Checks that the linked list of SEH handlers is valid and -1 terminated.

It is probably not enough...

אולי כדאי לנסות את הפתרון הטבעי?

- האם ניתן להבחין בין איזורי זיכרון שונים?
- דורש תמיכת חומרה ושל מערכת ההפעלה..

x86 processors, since the 80286, included a similar capability implemented at the segment level. However, almost **all operating systems for the 80386 and later x86 processors implement the flat memory model, so they cannot use this capability**

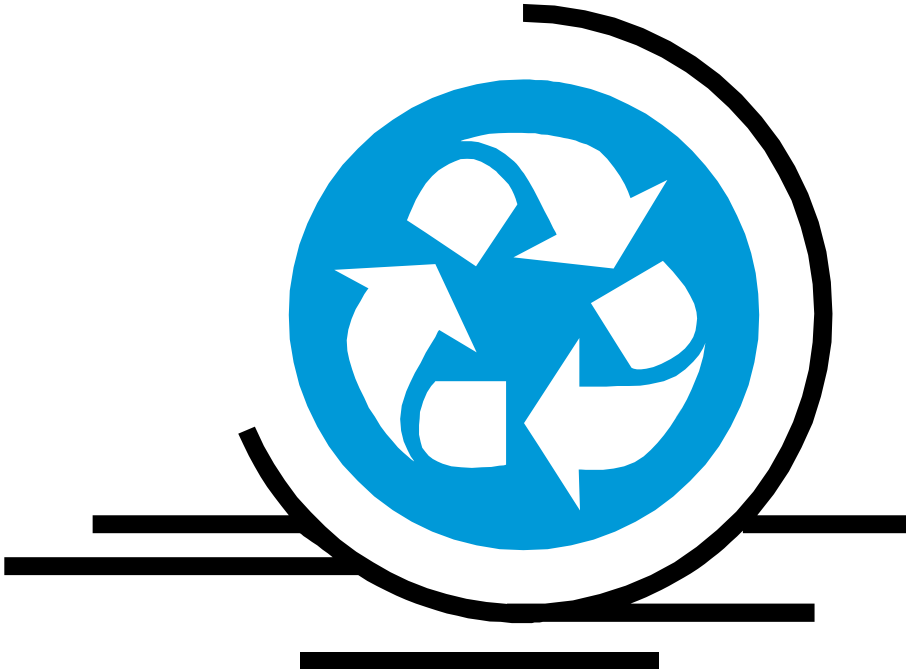
Windows

- Window XP (Service Pack 2)
- Microsoft uses NX bit to: ”prevents the execution of code in memory regions that are marked as data storage”
  - This will NOT prevent an attacker from overrunning the data buffer, but will prevent him from executing his attack (generate an exception)
- Some problems with legitimate code
  - a ”Data Execution Prevention" error message – for legitimate code
  - Workaround - Microsoft allow exceptions, per application. (I.e. turn DEP off for specific apps.)

Breaking DEP

- For years, off by default for compatibility reasons…
- Compatibility problems with plugins:
  - Internet Explorer 8 finally turned on DEP
- Sun JVM allocated its heap memory RWX, allowing us to write shellcode there

Now What?



Ret2Libc => ROP

- Eip made to "return to a function".
  - Create a fake frame on the stack.
- Series of function return
  - "ESP is the new EIP"

The Geometry of Innocent Flesh on the Bone:
Return-into-libc without Function Calls (on the x86)

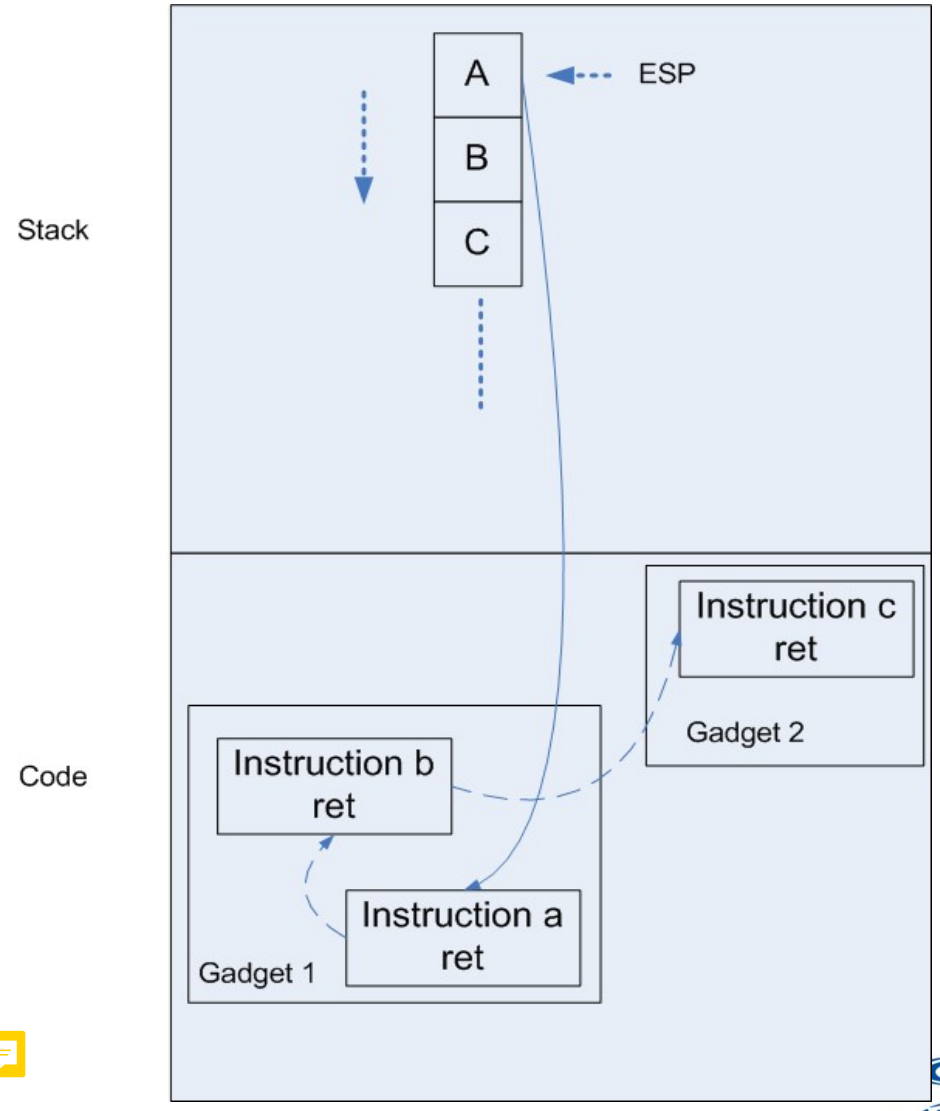
Hovav Shacham*
hovav@cs.ucsd.edu

Abstract

We present new techniques that allow a return-into-libc attack to be mounted on x86 executables that calls *no functions at all*. Our attack combines a large number of short instruction sequences to build *gadgets* that allow arbitrary computation. We show how to discover such instruction sequences by means of static analysis. We make use, in an essential way, of the properties of the x86 instruction set.

Return Oriented Programming

- "Reuse" small segments of code ("Gadgets") by placing them in the correct order on the stack.
- Theoretically possible to create any functionality using ROP.



ROP - Problems

- Need to find the correct "gadgets" for the job.
  - And correlate them.
  - Possible, but takes a lot of time.
- Feasible, but only for short segments of code
  - What do you have\want to use ROP for?
- Standard (Advanced\Theoretical) Exploitation method for the last years.

Gadgets

```
Mem[v2] = v1
```

Desired Logic

Suppose a_2 and a_3 on stack

a_5
v_2
a_3
v_1

← esp

Stack

eax	v_1
ebx	
eip	a_1

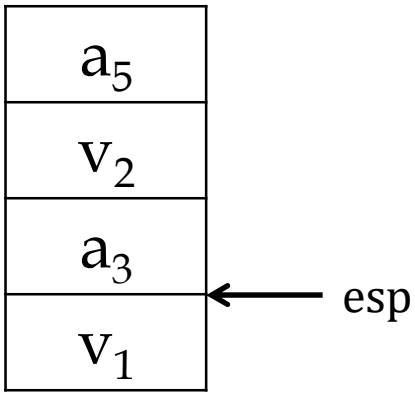
```
a1: pop eax;
a2: ret
a3: pop ebx;
a4: ret
a5: mov [ebx], eax
```

Implementation 2

Gadgets

```
Mem[v2] = v1
```

Desired Logic



Stack

eax	v_1
ebx	
eip	a_3

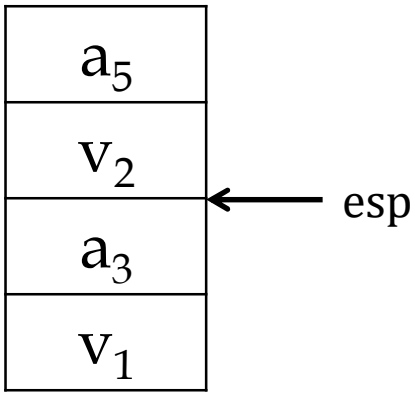
```
a1: pop eax;
a2: ret
a3: pop ebx;
a4: ret
a5: mov [ebx], eax
```

Implementation 2

Gadgets

```
Mem[v2] = v1
```

Desired Logic



Stack

eax	v_1
ebx	v_2
eip	a_3

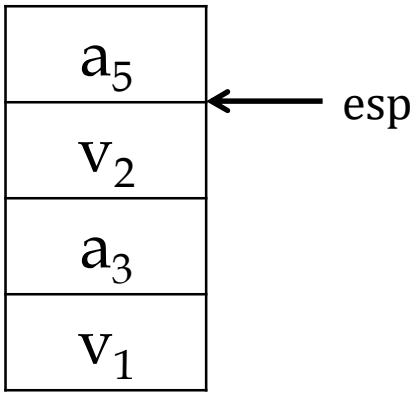
```
a1: pop eax;
a2: ret
a3: pop ebx;
a4: ret
a5: mov [ebx], eax
```

Implementation 2

Gadgets

```
Mem[v2] = v1
```

Desired Logic



Stack

eax	v_1
ebx	v_2
eip	a_4

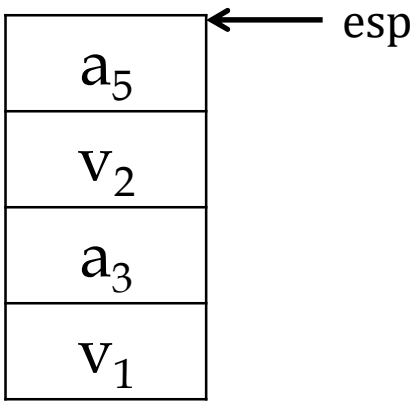
```
a1: pop eax;
a2: ret
a3: pop ebx;
a4: ret
a5: mov [ebx], eax
```

Implementation 2

Gadgets

```
Mem[v2] = v1
```

Desired Logic



Stack

eax	v_1
ebx	v_2
eip	a_5

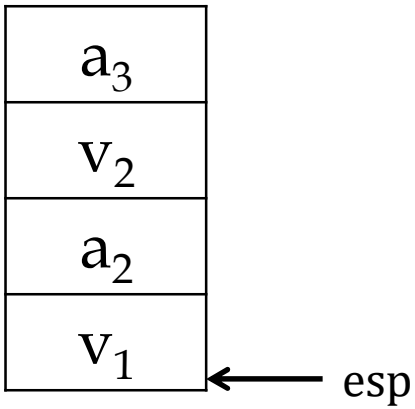
```
a1: pop eax;
a2: ret
a3: pop ebx;
a4: ret
a5: mov [ebx], eax
```

Implementation 2

Equivalence

```
Mem[v2] = v1
```

Desired Logic



Stack

semantically equivalent

"Gadgets"

```
a1: pop eax; ret
a2: pop ebx; ret
a3: mov [ebx], eax
```

Implementation 2

Equivalence

```
Mem[v2] = v1
```

Desired Logic

a_3
v_2
a_2
v_1

Stack

```
a1: pop eax; ret
...
a3: mov [ebx], eax
...
a2: pop ebx; ret
```

Address independent!

```
a1: pop eax; ret
a2: pop ebx; ret
a3: mov [ebx], eax
```

Implementation 2

Address Space Layout Randomization (ASLR)

- You cannot use code if you don't know where it is…
- /DynamicBase option in VS.
  - Windows 7: 8 bits of randomness for DLLs
    - aligned to 64K page in a 16MB region ⇒ 256 choices
  - Windows 8: 24 bits of randomness on 64-bit processors
- Not all DLLs and EXE support this flag.

שאלות?

פרק 7

שפות קודי ביניים

Java Bytecode & CIL

קודי ביניים

- תרגום שפה עילית לקוד ביניים שאינו שפת מכונה
  - קוד הביניים הינו שפה פשוטה
    - בדר"כ ממודלת כמכונת מחסנית אבסטרקטית
    - שקל לפרש אותה ולהריץ אותה, ללא צורך במהדר נוסף
  - בכל מחשב יש תוכנה שמריצה את קוד הביניים
    - באינטרפרטציה, ללא צורך בהידור נוסף
    - בחלק מהמקרים קוד הביניים מתורגם בזמן ריצה לשפת מכונה
      - Just In Time Compiler (JIT)
- שתי דוגמאות עיקריות

שפת קוד ביניים	שפת תכנות	הערות
Java Bytecode	Java	מתוכננת במיוחד ל-Java
CIL (ידועה גם כ-MSIL)	.NET (#C ועוד)	תומכת במגוון שפות

- נתמקד ב-Java Bytecode – בקיצור JB

מבנה קובץ הרצה ב-Java

- קובץ הרצה jar הינו ארכיון ZIP
  - כלומר ניתן לפתוח אותו עם WinZip
- קובץ jar כולל
  - קובץ ראשי
    - META-INF/MANIFEST.MF
    - מתאר היכן התכנית הראשית, מיקום ספריות, וכו'
  - קבצי מידע כללי
    - author.txt
  - וקבצי class
    - הכוללים Bytecode
    - program.class, lib.class, ...
- ייצוגים פנימיים תמיד ב-big-endian

מבנה קובץ Class

- קובץ class כולל
  - תיאור התוכן הפומבי של המחלקה
  - רשימת קבועים (Constant pool)
    - כוללת את כל הקבועים וסוגם
    - כלומר
      - הגדרות המשתנים (בפרט הפומביים של המחלקה), עם הפניות לטיפוסים
        - ✓ ההפניה היא לאיבר אחר ברשימה
      - מחרוזות
        - ✓ מחרוזות שמופיעות בתכנית
        - ✓ מחרוזות של שמות מחלקות ושמות פונקציות שבשימוש
      - כל הטיפוסים שיש בתכנית
    - מבנה של ענפים של עץ
  - הקוד (bytecode) של כל הפונקציות של המחלקה
    - בשפה דמוית אסמבלי

טיפוסים בסיסיים בשפת Java

- להלן רשימת הטיפוסים הבסיסיים בשפת Java וקיצוריהם
  - הקיצורים משמשים בקובץ ה-class

	טיפוס	קיצור
signed	integer	I
	long	L
	short	S
	byte	B
Unicode (16 bit)	character	C
	float	F
	double	D
	boolean	Z
	reference	A

  - למערך יש תוספת "[" לפני האות
  - אין טיפוסי unsigned int/long/byte...

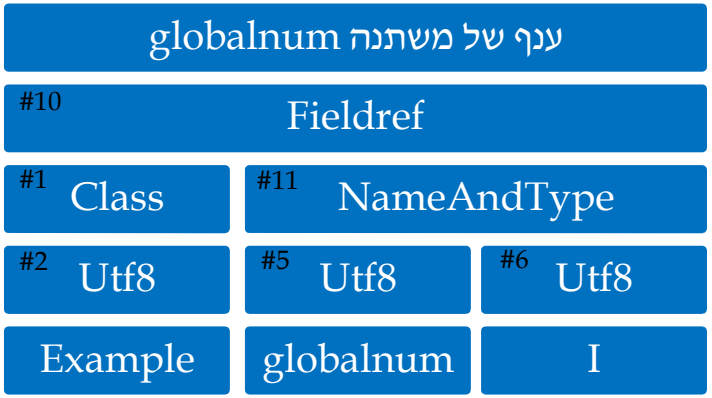
מבנה רשימת הקבועים

- רשימת הקבועים (Constant Pool) היא מערך
  - עם הפניות בצורת ענפים של עץ

- כל כניסה במערך מכילה
  - אינדקס (מאחד עד מספר הקבועים)
  - טיפוס
    - Utf8 : טקסט מקודד ב-UTF-8
      - משמש לכל הטיפוסים האחרים
      - קידוד Utf8 אינו קידוד UTF-8 תקני
        - ✓ NUL מקודד בשני בתים, ותווי UTF-8 ארוכים מקודדים שונה מבתקן
    - String : מחרוזת, בצירוף האינדקס של הטקסט UTF-8 שלה
    - Class : מחלקה, בצירוף האינדקס של הטקסט UTF-8 של השם שלה

מבנה רשימת הקבועים

- NameAndType: שם וטיפוס, עם שני אינדקסים ל-UTF-8, האחד לשם המזהה, והשני לקיצור שם הטיפוס (אות אחת) או טיפוס פונקציה
- Methodref: מתודה, עם אינדקס של מחלקה ואינדקס NameAndType
- Fieldref: שם שדה או שם משתנה, עם אינדקס של מחלקה ואינדקס NameAndType
- Integer
- Float
- Long
- ...



דוגמת תכנית

```
public class Example
{
    public static int globalnum=5;

    public static void main (String args[])
    {
      int num=3+globalnum;

      if(globalnum<10) { num++; };

      System.out.println("The sum is "+num);
    }
}
```

Constant pool

```
#1 = Class              #2                  // Example
#2 = Utf8               Example
#3 = Class              #4                  // java/lang/Object
#4 = Utf8               java/lang/Object
#5 = Utf8               globalnum
#6 = Utf8               I
#7 = Utf8               <clinit>
#8 = Utf8               ()V
#9 = Utf8               Code
#10 = Fieldref          #1.#11              // Example.globalnum:I
#11 = NameAndType       #5:#6               // globalnum:I
#12 = Utf8              LineNumberTable
#13 = Utf8              <init>
#14 = Methodref         #3.#15              // java/lang/Object."<init>":()V
#15 = NameAndType       #13:#8              // "<init>":()V
#16 = Utf8              main
#17 = Utf8              ([Ljava/lang/String;)V
#18 = Fieldref          #19.#21             // java/lang/System.out:Ljava/io/PrintStream;
#19 = Class             #20                 // java/lang/System
#20 = Utf8              java/lang/System
#21 = NameAndType       #22:#23             // out:Ljava/io/PrintStream;
#22 = Utf8              out
#23 = Utf8              Ljava/io/PrintStream;
#24 = Class             #25                 // java/lang/StringBuilder
#25 = Utf8              java/lang/StringBuilder
#26 = String            #27                 // The sum is
#27 = Utf8              The sum is
#28 = Methodref         #24.#29             // java/lang/StringBuilder."<init>":(Ljava/lang/String;)V
#29 = NameAndType       #13:#30             // "<init>":(Ljava/lang/String;)V
#30 = Utf8              (Ljava/lang/String;)V
#31 = Methodref         #24.#32             // java/lang/StringBuilder.append:(I)Ljava/lang/StringBuilder;
#32 = NameAndType       #33:#34             // append:(I)Ljava/lang/StringBuilder;
#33 = Utf8              append
#34 = Utf8              (I)Ljava/lang/StringBuilder;
#35 = Methodref         #24.#36             // java/lang/StringBuilder.toString:()Ljava/lang/String;
#36 = NameAndType       #37:#38             // toString:()Ljava/lang/String;
#37 = Utf8              toString
#38 = Utf8              ()Ljava/lang/String;
#39 = Methodref         #40.#42             // java/io/PrintStream.println:(Ljava/lang/String;)V
#40 = Class             #41                 // java/io/PrintStream
#41 = Utf8              java/io/PrintStream
#42 = NameAndType       #43:#30             // println:(Ljava/lang/String;)V
#43 = Utf8              println
#44 = Utf8              SourceFile
#45 = Utf8              Example.java
```

המשתנה globalnum
מופנה ע"י #10

המחרוזת "The sum is "
מופנה ע"י #26

אינדקס

טיפוס

ערך או הפניה

הערות

פלט של הרצת javap על קובץ המחלקה

מבנה בסיסי של שפת Java Bytecode

- שפה מבוססת מחסנית
  - פקודות JB לוקחות פרמטרים מהמחסנית
  - ומחזירות את תוצאתן בראש המחסנית
  - גם המשתנים המקומיים על המחסנית
- ה-opcode תמיד באורך בית אחד
  - יש מקרה בודד של prefix
    - wide – מאפשר אופרנדים בגודל שני בתים בפקודות מסוימות שבדר"כ מקבלות אופרנדים בני בית אחד
- פקודות JB יכולות לקבל מספר קטן של אופרנדים
  - כך שאורך פקודת JB משתנה
  - אופרנדים יכולים להיות בגודל בית אחד או שניים
    - לעיתים גם ארבעה

מבנה בסיסי של שפת Java Bytecode

- JB רץ בתוך מכונה וירטואלית
  - מתוכננת במיוחד להגנה כנגד חולשות
  - למשל, המכונה בודקת חריגה ממערכים
    - מייצרת exception במקרה של חריגה
  - המכונה הווירטואלית מוודאת שקפיצות הן תמיד לתחילת פקודות
    - נבדק על ידי ה-verifier בתחילת ההרצה

- כל כניסה על המחסנית היא בת 32 סיביות
  - שלם, מצביע, וכו'
  - ייצוג של long הוא כשתי כניסות רצופות (64 סיביות)

דוגמת JB של התכנית

```
static {};                                  // Constructor
  stack=1, locals=0, args_size=0
     0: iconst_5
     1: putstatic      #10                  // Field globalnum:I
     4: return

public static void main(java.lang.String[]);
   stack=4, locals=2, args_size=1
     0: iconst_3                            // Short form for   iconst  3 (single byte instead of 3)
     1: getstatic      #10                  // Field globalnum:I
     4: iadd
     5: istore_1                            // Short form for   istore  1 (single byte instead of 3)
     6: getstatic      #10                  // Field globalnum:I
     9: bipush         10
    11: if_icmpge      17
    14: iinc           1, 1
    17: getstatic      #18                  // Field java/lang/System.out:Ljava/io/PrintStream;
    20: new            #24                  // class java/lang/StringBuilder
    23: dup
    24: ldc            #26                  // String The sum is
    26: invokespecial  #28                  // Method java/lang/StringBuilder."<init>":(Ljava/lang/String;)V
    29: iload_1
    30: invokevirtual  #31                  // Method java/lang/StringBuilder.append:(I)Ljava/lang/StringBuilder;
    33: invokevirtual  #35                  // Method java/lang/StringBuilder.toString:()Ljava/lang/String;
    36: invokevirtual  #39                  // Method java/io/PrintStream.println:(Ljava/lang/String;)V
    39: return
```

כתובת | opcode | אופרנדים | הערות

טכנולוגיית JIT

Just In Time compilation

- הידור קוד בזמן ריצה
- יכולת לבצע אינטרפרטציה לקוד, ולבצע הידור רק לקוד שבשימוש רב
  - וללמוד מהאינטרפרטציה איך כדאי לבצע אופטימיזציות
- מאפשר לוודא שהקוד המהודר אינו מתוכנן לגרום נזק
  - קוד הביניים פשוט יותר לבדיקה
  - הקוד המהודר אינו ניתן לטיפול ע"י תוקף
  - כלומר, נוזקה אינה יכולה להשתלט על הקוד...

מודל האבטחה ב-Java

- תמיכה במגוון מבנים מאובטחים ע"י שפת התכנות
  - Strong data typing
  - ניהול זיכרון אוטומטי (ושחרור אוטומטי)
  - אין cast ללא בדיקה
  - אין אריתמטיקה של מצביעים
  - מערכים שמורים בזיכרון הדינמי
    - מוקצים בזמן ריצה (ע"י new), לא על המחסנית

- מכונה וירטואלית שבודקת כל מה שאפשר
  - מגוון בדיקות בזמן טעינת מחלקה
    - בפרט, Bytecode verification
      - למשל, כל הקפיצות למיקומים חוקיים (לא לאמצע פקודות JB)
  - בדיקות בזמן ריצה
    - למשל בדיקות חריגה ממערך
    - בדיקות שימוש במצביע בעל ערך NULL

מודל האבטחה ב-Java

- sandbox
  - מאפשר קביעת הרשאות לקוד שרץ
  - כלומר, לא כל ההרשאות של מריץ הקוד מאופשרות
    - למשל, מניעת גישה לקבצים, גישה לרשת או גישה לשרתים מסוימים ברשת, יציאה מהמכונה הווירטואלית, יצירת תהליך חדש, וכו'
- JIT
  - תוקף לא יכול לשתול קוד כרצונו
- לסיכום, מניעת בעיות אבטחה מקוד עוין על ידי מניעת היכולת של הקוד לבצע פעולות בעייתיות
  - ברמת שפת התכנות, המכונה הווירטואלית, וקריאות לפעולות מסוימות
  - אולם, ההגנות לא מושלמות
    - בשנים האחרונות התגלו מגוון בעיות אבטחה ב-Java שגרמו להפסקת השימוש ב-Java במספר ארכיטקטורות

מודל האבטחה ב-Java

- מה לא נעשה?
  - אין בדיקת integer overflow
- כל שמות המשתנים ברמת המחלקה רשומים ברשימת הקבועים
  - כך שהם נגישים לכל מי שמעוניין לבצע RE
  - בפרט נגישים לדה-קומפיילרים
- ואפילו מספרי השורות כלולים בקובץ...
  - במקור לצרכי דיבוג

דיס-אסמבלרים ודה-קומפיילרים

- קיימים מגוון של דיס-אסמבלרים ודה-קומפיילרים
  - הלוקחים קוד JB ומתרגמים לקוד קריא

- למשל
  - דיס-אסמבלרים
    - javap
    - גם גרסאות מסוימות של IDA
  - דה-קומפיילרים
    - JD
    - JAD
    - Krakatau

- בדרייכ מייצרים קוד די טוב

דה-קומפילציה ע"י JD מקובץ ה-Class

```
import java.io.PrintStream;

public class Example
{
  public static int globalnum = 5;

  public static void main(String[] paramArrayOfString)
  {
    int i = 3 + globalnum;

    if (globalnum < 10) i++;

    System.out.println("The sum is " + i);
  }
}
```

Java bytecode fe Obfuscation

- העלמת מחרוזות מרשימת הקבועים
  - למשל יצורן בקוד
- שינוי שמות משתנים ומתודות
- ניקוי נתוני דיבוג
- וכמובן, סיבוך הקוד

- יש מגוון תוכנות obfuscation ייעודיות ל-Java

קצת על המבנה של קובץ Class

- רשימת הקבועים
  - הרשומות מסודרות לפי סדר
    - ללא אינדקס מפורש בפנים
  - כל רשומה מתחילה בבית המתאר טיפוס
    - ומידע באורך קבוע
    - במקרה של UTF-8 – אורך משתנה
      - לאחר הטיפוס, אורך בשני בתים ואז המחרוזת
  - כלומר קל לשנות תוכן של רשומה
    - כולל לשנות אורך
    - בלי צורך לשנות דבר ברשומות אחרות
    - כל עוד לא מוחקים ולא מוסיפים רשומות באמצע

- JB
  - קפיצות נעשות באופן יחסי למיקום תחילת הפקודה
  - שינוי קוד אינו דורש תיקון בקפיצות שאינן קופצות מעל השינוי
  - עדיין עדיף לשנות בלי לשנות אורך

הוקינג של קבצי Class

- דוגמאות לשינויים ב-JB
  - דוגמא לשינוי מחרוזת ישירות ב-class
    - כולל שינוי אורך
  - דוגמא לשינוי opcode ישירות ב-class
    - למשל שינוי חיבור לכפל
  - שינויים יותר גדולים
- ה-bytecode verifier מונע מאיתנו לשנות בדרכים לא קבילות
  - למשל קפיצה לאמצע פקודה
- לא ניתן לבצע הוקינג לקוד בזיכרון בזמן ריצה
  - אין לתכנית גישה לקוד...

הבדלים עיקריים בין JB ל-CIL

- CIL תוכננה על סמך העקרונות של JB

- JB תוכננה במיוחד עבור Java
- CIL תוכננה לתמוך במגוון שפות
  - לכן כוללת מגוון גדול יותר של טיפוסים ומבנים

- המבנה העקרוני של השפות דומה
  - שתיהן מבוססות מחסנית
  - ואפילו עם פקודות דומות רבות

- אבל CIL למדה מהניסיון של JB
  - עם שיפורים
  - קוד הביניים תוכנן ל-JIT
    - JB תוכננה לתמוך באינטרפרטציה וב-JIT

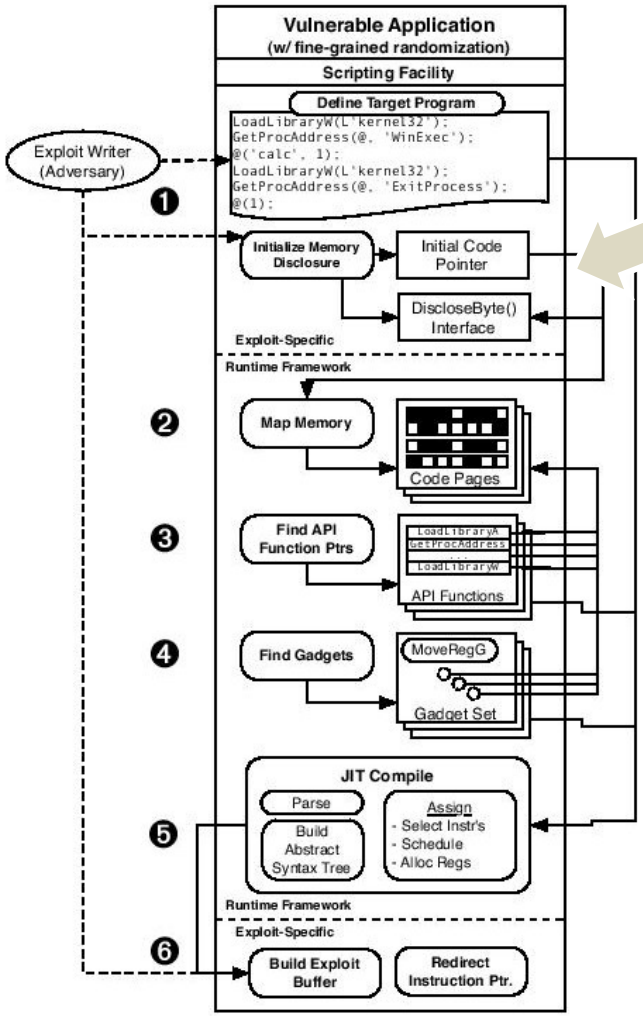
Java באנדרואיד

- באנדרואיד הפעלת Java מורכבת יותר
  - כדי לאפשר את יתרונות Java מצד אחד ופעולה יעילה מצד שני
- עד אנדרואיד 4.4 – שימוש ב-Dalvik
  - אפליקציה נשמרת בקובץ APK שבתוכו נשמר קובץ DEX
    - Dalvik executable
    - קוד ביניים ייעודי שיותר קל לבצע לו JIT
    - נוצר ע"י המפתח מקבצי ה-class וה-JAR בזמן יצירת האפליקציה
  - האפליקציה מורצת כ-JIT ע"י הידור ה-DEX בזמן ריצה
- אנדרואיד 5 – שימוש ב-ART
  - Android run time
  - המרת ה-APK לקובץ הרצה בזמן התקנת האפליקציה ובשדרוג מ"ה
    - אין שינוי במבנה קובץ ה-APK
  - האפליקציה מופעלת מקובץ ההרצה, ללא צורך בשום הידור נוסף
- אנדרואיד 6 – ART משולב JIT
  - כדי לחסוך בזמן התקנה ובזמן שדרוג מ"ה
  - ההמרה לקובץ הרצה נעשית בזמן פנוי של המערכת
  - עד אז מבוצע JIT

JIT Reuse

Just-in-Time Code Reuse (1)

[Snow et al., Oakland 2013]



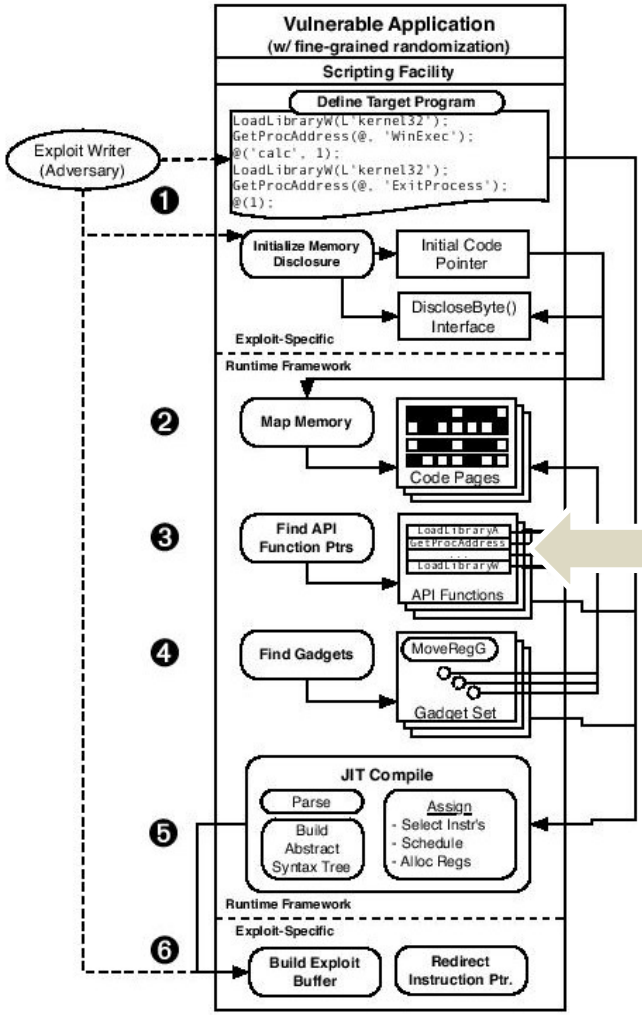
Find one code pointer
(using any disclosure vulnerability)

The entire page must be code…
Analyze the instructions to find
jumps and calls to other code pages…

Map out a big portion of
the application's code pages

Just-in-Time Code Reuse (2)

[Snow et al., Oakland 2013]

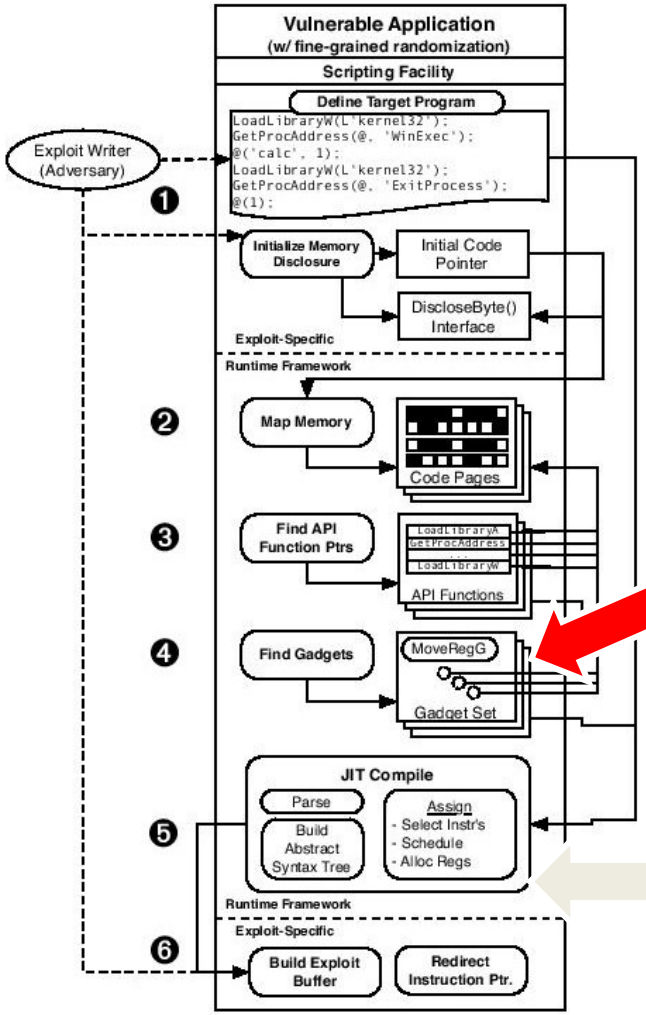


Use typical opcode sequences to find calls to LoadLibrary() and GetProcAddr()…

These can be used to invoke any library function by supplying the right arguments - don't need to discover the function's address!

Just-in-Time Code Reuse (3)

[Snow et al., Oakland 2013]



Collect gadgets in runtime by analyzing the discovered code pages (dynamic version of Shacham's "Galileo" algorithm)

Compile on the fly into shellcode

פרק 5

טכניקות נגד RE

Anti-Reversing

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
  - Anti-Debugging
    - טכניקות לזיהוי סביבת ריצה עוינת
      - ולשינוי (או שיבוש או עצירת) פעילות התוכנית כתוצאה מכך
  - דחיסת והצפנת הקוד בקובץ ההרצה

- נציג כמה רעיונות בסיסיים על מנת להדגים את הנושא
  - כל רעיון בסיסי ניתן להרחבה לשיטה מתקדמת יותר

מה מחשב הקוד הבא?

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

והקוד הנה?

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

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

דוגמא פשוטה ל-Obfuscation

```
#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" insturctions.

        popad
    }
}
```

```
013D1020  $  53            PUSH EBX
013D1021  .  56            PUSH ESI
013D1022  .  57            PUSH EDI
013D1023  .  60            PUSHAD
013D1024  .  B8 01000000   MOV EAX,1
013D1029  .  83F8 01       CMP EAX,1
013D102C  .  75 02         JNZ SHORT RE_Testi.013D1030
013D102E  .  EB 01         JMP SHORT RE_Testi.013D1031
013D1030  >  0FE8CA        PSUBSB MM1,MM2
013D1033  ?  FFFF          [illegible]
013D1035  ?  FF61 5F       JMP NEAR DWORD PTR DS:[ECX+5F]
013D1038  .  5E            POP ESI
013D1039  .  5B            POP EBX
013D103A  .  C3            RETN
```

From “leetMatrix” blog

דוגמא פשוטה ב-C :

```
static int two=1;

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

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-דיס-אסמבלי – ריבוי משמעויות

- מה עושה הקוד הבא?

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

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

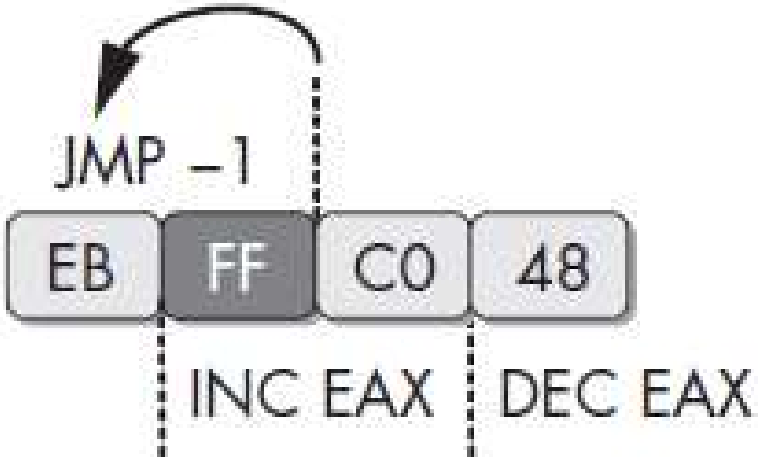
- והקוד הזה?

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

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

אנטי-דיס-אסמבלי – חפיסת קוד

- מה עושה הקוד הבא?

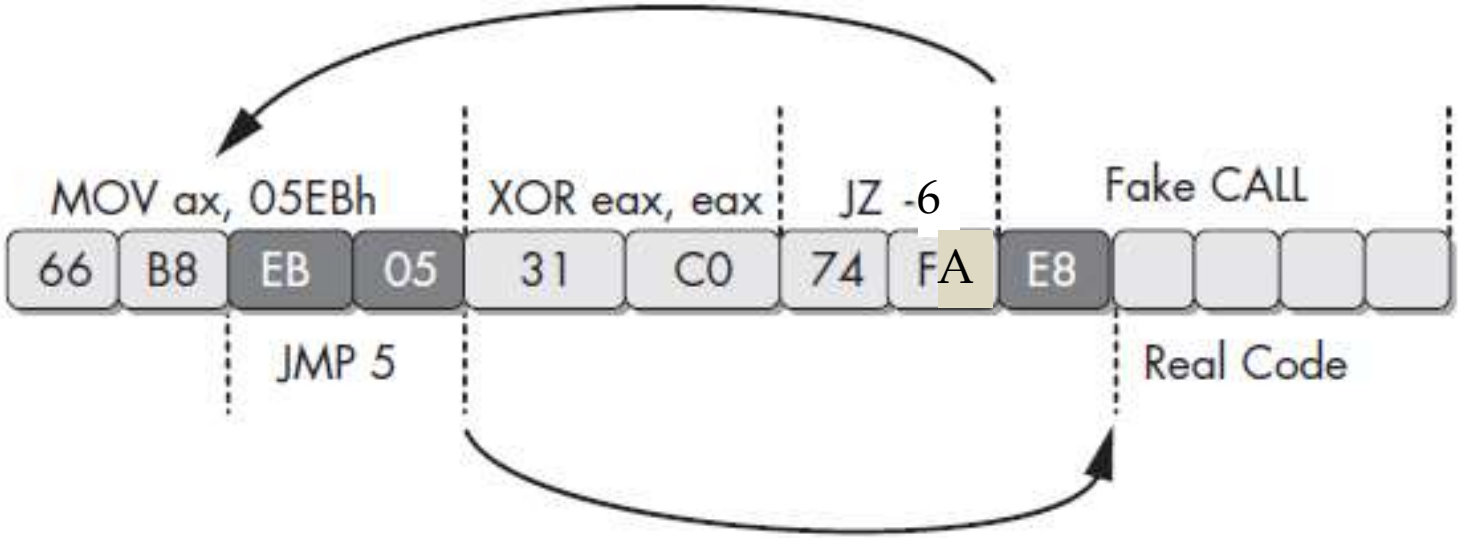


- זה למעשה (כמעט) פקודת nop בת 4 בתים
  - קפוץ אחורה בית אחד, לקוד חופף (2 בתים: EBFF)
  - הגדל את EAX באחד (2 בתים: FFC0)
  - הקטן את EAX באחד (בית אחד: 48)
    - זה אינו nop בגלל ההשפעה על הדגלים

from Practical Malware Analysis

אנטי-דיס-אסמבלי – חפיסת 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-דיס-אסמבלי בשימוש בחריגות

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

אנטי-דיס-אסמבלי בשימוש בחריגות

דוגמא

from Practical Malware Analysis

```
00401050                ❷mov     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                ❸div     ecx
0040106B
0040106B loc_40106B:                             ; DATA XREF: sub_401050o
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
```

עוד דוגמאות לאנטי-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 לוקאלים
  - נעשה על ידי שינוי EBP (או ESP) באמצע הפונקציה
- ועוד

Anti-Debugging

- כאשר תוכנה מזהה שהיא רצה תחת כלי דיבוג או בווירטואליזציה, 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

דוגמאות

- קריאה ל-API:
  - isDebuggerPresent
  - CheckRemoteDebuggerPresent
  - NtQueryInformationProcess
- זיהוי לפי מדידת זמן
- חיפוש int 3
- 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 (משנה את ה-lastError).

ניהוי 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'

int 3 ניהוי קיום

- 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 גורם לחריגה

- תזכורת: int 3 זו חריגה שבה מערכת ההפעלה מעבירה את השליטה ל Debugger
  - כלומר int 3 זה software breakpoint
- אם אין Debugger אז
  - שימוש ב-int 3 יוצר חריגה רגילה

```
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
```

טיפול בחריגות ע"י צרצין מ"ה

וקטור הפסיקות

Interrupt Descriptor Table (IDT)

- מדובר בטבלה שמכילה (בין היתר) כתובות לפונקציות.
- המעבד משתמש בטבלה על מנת להחליט כיצד להגיב לפסיקה או חריגה מסוימת.
- הטבלה לא נגישה מ-User mode.

וקטור הפסיקות

0	Division by zero
1	Debugger
2	NMI
3	Breakpoint
4	Overflow
5	Bounds
6	Invalid Opcode
..	…

Code…

Code…

Code…

- אם המעבד ביצע חלוקה באפס, הוא יקרא לכתובת אשר נמצאת ברשומה 0 בווקטור הפסיקות.

- ניתן גם לקרוא לפסיקה בצורה מפורשת ע״י פקודת האסמבלי int val.
  - לדוגמא int 0.

וקטור הפסיקות

```
KDx86> !idt 0
   00: 801753fc (_KiTrap00)
   01: 80175580 (_KiTrap01)
   02: 000012de
   03: 80175890 (_KiTrap03)
   04: 80175a08 (_KiTrap04)
   05: 80175b68 (_KiTrap05)
   06: 80175ce0 (_KiTrap06)
  …
```

(שימו לב שלא כל הרשומות ב-IDT קשורות לחריגות)

חריגות

- חריגה יכולה להתרחש בשני מצבים:
  - זריקת Exception בצורה מכוונת ע"י throw.
  - פקודה לא חוקית במעבד (ובדרך כלל מדובר במשהו לא צפוי), למשל:
    - חלוקה ב-0.
    - גישה לכתובת לא חוקית.

- זריקה מכוונת של Exception מתורגמת לקריאה לפונקציה RaiseException.

- במקרה של פקודה לא חוקית המעבד משתמש בטבלת ה-IDT על מנת למצוא את השגרה המתאימה לטיפול בפסיקה.
  - לכל סוג של חריגה יש רשומה מתאימה בטבלת ה-IDT (ראינו שחלוקה ב-0 זה אינדקס 0 בטבלה).
  - כל שגרה כזאת קוראת ל-kiDispatchException.

KiDispatchException

- מדובר בפונקציה המרכזית של מערכת ההפעלה לטיפול בחריגות.
- הפונקציה מחפשת את ה-handler המתאים כפי שממומש בthread שזרק את החריגה.
- אם התהליך נמצא תחת דיבוג הפונקציה kiDispatchException תקרא קודם ל-handler של הדיבאגר.
- ניתן למצוא מידע נוסף על הפונקציה באתר reactos.org.

KiDispatchException

```
/* User mode exception, was it first-chance? */
if (FirstChance)
{
    /*
     * Break into the kernel debugger unless a user mode debugger
     * is present or user mode exceptions are ignored, except if this
     * is a debug service which we must always pass to KD
     */
    if ((!(PsGetCurrentProcess()->DebugPort) &&
         !(KdIgnoreUmExceptions)) ||
         (KdIsThisAKdTrap(ExceptionRecord, &Context, PreviousMode)))
    {
        /* Make sure the debugger can access debug directories */
        KiPrepareUserDebugData();

        /* Call the kernel debugger */
        if (KiDebugRoutine(TrapFrame,
                           ExceptionFrame,
                           ExceptionRecord,
                           &Context,
                           PreviousMode,
                           FALSE))
        {
            /* Exception was handled */
            goto Handled;
        }
    }

    /* Forward exception to user mode debugger */
    if (DbgkForwardException(ExceptionRecord, TRUE, FALSE)) return;

    //KiDispatchExceptionToUser()
    __debugbreak();
}
```

reactos.org

קריאה נוספת

- A Crash Course on the Depths of Win32 Structured Exception Handling.
- Windows Internals.
- Windows NT/2000 Native API Reference.
  - Contains a chapter on exceptions.

שיטות נוספות לגילוי דיבוג

- 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 לגילוי דיבוג

- ניתן לחפש האם מותקנות במחשב תוכנות מחקר שונות (למשל דרך סריקת תוכן מערכת הקבצים או ה-Registry)
- דוגמא למציאת תהליך עייי 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 לאתרו

ניהוי מחשב 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

ניהוי 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

ניהוי מחשב וירטואלי

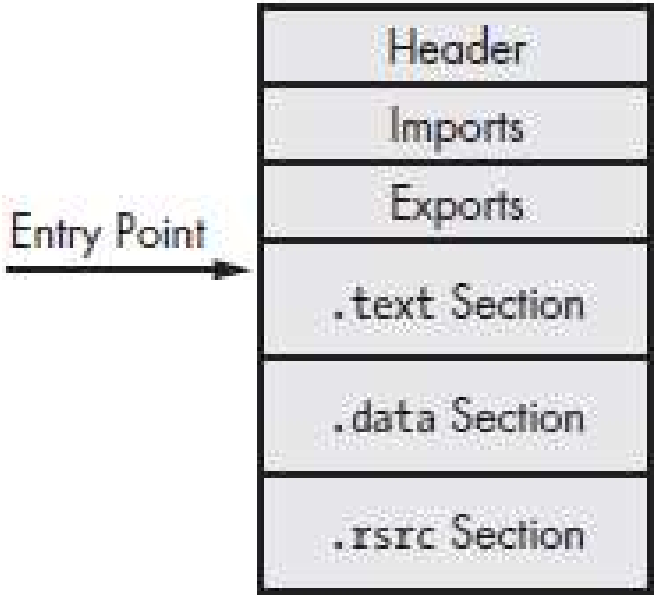
- קל לזהות את VMware
  - תהליכים לניהול המחשב הווירטואלי
    - VMwareService, VMwareTray, VMware User
  - כתובת MAC ייחודית
- כדי להקשות על זיהוי, אפשר ליצור patch עם שמות אחרים וכתובות אחרות
  - ניתן להקשות על זיהוי, אך לא למנוע
- ישנן תוכנות וירטואליזציה שיותר קשה לאתרן
- במידה שרוצים למנוע לחלוטין אפשרות לזיהוי, צריך להריץ בסביבה טבעית במחשב ייעודי
  - אבל אז צריך להשקיע יותר בהתקנה מחדש כל פעם שנגרם נזק

דחיסת קובץ ההרצה

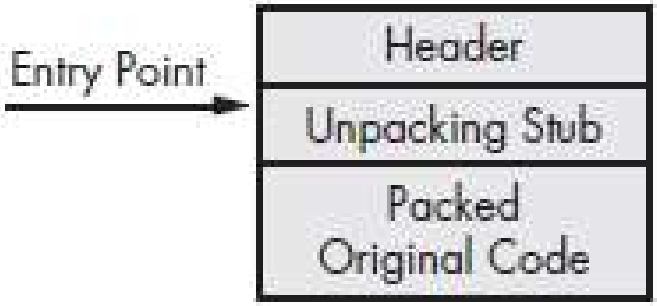
- ישנם כלים שמאפשרים דחיסת תוכנה
  - תוכנה דחוסה, בנוסף לקוד הרגיל, תכיל מתודה שאחראית על חילוץ הקוד והרצתו
  - במקרה כזה לא נוכל לנטר את התוכנה באופן סטטי
    - שכן הקוד לא חשוף לנו
    - כלומר לא ניתן להריץ IDA
  - הצפנה היא אחד הכלים המשמשים בזמן דחיסת קובץ ההרצה
    - אבל קוד הפענוח/מפתח נמצא בתוכנה, אחרת לא תוכל 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

דוגמת UPX

Code?:

```
C0      DB C0
64      DB 64      CHAR 'd'
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'
D9      DB D9
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
1B      DB 1B
50      DB 50      CHAR 'P'
9B      DB 9B
14      DB 14
77      DB 77      CHAR 'w'
00      DB 00
34      DB 34      CHAR '4'
62      DB 62      CHAR 'b'
B0      DB B0
53      DB 53      CHAR 'S'
48      DB 48      CHAR 'H'
97      DB 97
```

No strings:

```
004076C2 ASCII "ru"
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

דוגמת UPX

```
004079B0  $ 60              PUSHAD
004079B1  . BE 15704000     MOV ESI,crackV2.00407015
004079B6  . 8DBE EB9FFFFF   LEA EDI,DWORD PTR DS:[ESI+FFFF9FEB]
004079BC  . 57              PUSH EDI
004079BD  . EB 0B           JMP SHORT crackV2.004079CA
004079BF    90              NOP
004079C0  > 8A06            MOV AL,BYTE PTR DS:[ESI]
004079C2  . 46              INC ESI
004079C3  . 8807            MOV BYTE PTR DS:[EDI],AL
004079C5  . 47              INC EDI
004079C6  > 01DB            ADD EBX,EBX
004079C8  . 75 07           JNZ SHORT crackV2.004079D1
004079CA  > 8B1E            MOV EBX,DWORD PTR DS:[ESI]
004079CC  . 83EE FC         SUB ESI,-4
004079CF  . 11DB            ADC EBX,EBX
004079D1  >^72 ED           JB SHORT crackV2.004079C0
004079D3  . B8 01000000     MOV EAX,1
004079D8  > 01DB            ADD EBX,EBX
004079DA  . 75 07           JNZ SHORT crackV2.004079E3
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 crackV2.004079D8
004079E9  . 75 09           JNZ SHORT crackV2.004079F4
004079EB  . 8B1E            MOV EBX,DWORD PTR DS:[ESI]
004079ED  . 83EE FC         SUB ESI,-4
004079F0  . 11DB            ADC EBX,EBX
004079F2  .^73 E4           JNB SHORT crackV2.004079D8
004079F4  > 31C9            XOR ECX,ECX
004079F6  . 83E8 03         SUB EAX,3
004079F9  . 72 0D           JB SHORT crackV2.00407A08
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 crackV2.00407A7A
00407A06  . 89C5            MOV EBP,EAX
00407A08  > 01DB            ADD EBX,EBX
00407A0A  . 75 07           JNZ SHORT crackV2.00407A13
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 crackV2.00407A20
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 crackV2.00407A44
00407A24  . 41              INC ECX
00407A25  > 01DB            ADD EBX,EBX
00407A27  . 75 07           JNZ SHORT crackV2.00407A30
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 crackV2.00407A25
```

הקוד הראשון שרץ:

ניתן לראות בבירור שלא מדובר במשהו טריוויאלי שנרצה לחקור.

דחיסת קובץ ההרצה

דוגמת UPX

```
00407B0B  . 53            PUSH EBX
00407B0C  . 57            PUSH EDI
00407B0D  . FFD5          CALL EBP
00407B0F  . 8D87 9F010000 LEA EAX,DWORD PTR DS:[EDI+19F]
00407B15  . 8020 7F       AND BYTE PTR DS:[EAX],7F
00407B18  . 8060 28 7F    AND BYTE PTR DS:[EAX+28],7F
00407B1C  . 58            POP EAX
00407B1D  . 50            PUSH EAX
00407B1E  . 54            PUSH ESP
00407B1F  . 50            PUSH EAX
00407B20  . 53            PUSH EBX
00407B21  . 57            PUSH EDI
00407B22  . FFD5          CALL EBP
00407B24  . 58            POP EAX
00407B25  . 61            POPAD
00407B26  . 8D4424 80     LEA EAX,DWORD PTR SS:[ESP-80]
00407B2A  > 6A 00         PUSH 0
00407B2C  . 39C4          CMP ESP,EAX
00407B2E  .^75 FA         JNZ SHORT crackV2.00407B2A
00407B30  . 83EC 80       SUB ESP,-80
00407B33  . E9 F895FFFF   JMP crackV2.00401130
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

חילוץ 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 יכיל Import table תקני
  - תזכורת: ל-Import table יש שני מצבים
    1. לפני פתיחת הקובץ – יכיל טבלה עם שמות DLLs ופונקציות שעל מערכת ההפעלה לטעון
    2. לאחר פתיחת הקובץ – יכיל 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

הצפנת פקודות מכונה
למשל SGX של אינטל

- אינטל מאפשרים במסגרת SGX להקיץ קוד בסביבה בה
  - הזיכרון מוצפן כולו
    - כלומר כל פקודות המכונה והמשתנים מגיעים למעבד מוצפנים
  - התקשורת פנימה והחוצה מוצפנת
- כך שגם האזנה על הערוץ הפנימי של המחשב לא תסגיר דבר על מה שמתבצע
- זה לא מיועד לחישוב כללי, אלא לסביבות ייעודיות
  - בהן נדרשת קריפטוגרפיה להגנה על מידע על בעלים מרוחק
  - למשל הפצה אינטראקטיבית של סרטים בעלי זכויות יוצרים
- ל-AMD יש ארכיטקטורה משלהם שעושה דבר דומה

סיכום

- ראינו בהרצאה מספר טכניקות נפוצות להתמודדות נגד RE
- אין נוסחה או דרך אחת להתמודד נגדן
  - יש להיות יצירתיים ועקשנים
- כאשר נתקלים במכשול, כדאי לבדוק האם מישהו נתקל בזה לפנינו
  - למשל בגוגל
- כשמשקיעים שעות רבות על הבנת קוד, כדאי לשאול האם להמשיך, או למצוא דרך אחרת
  - לא נרצה למשל לחקור Packer במקום לחקור את החלק שמעניין אותנו
- הבנת חלקים שונים במערכת ההפעלה (Internals ו-API) הינה הכרחית
- מדובר על תחום מאוד רחב שהוצג רק בחלקו
  - מאוד מועיל להכיר טכניקות נוספות וכלים אוטומטיים

פרק 9

הסוואה בנוקות

גילוי ומניעת גילוי

נוזקות: וירוסים וחיות אחרות

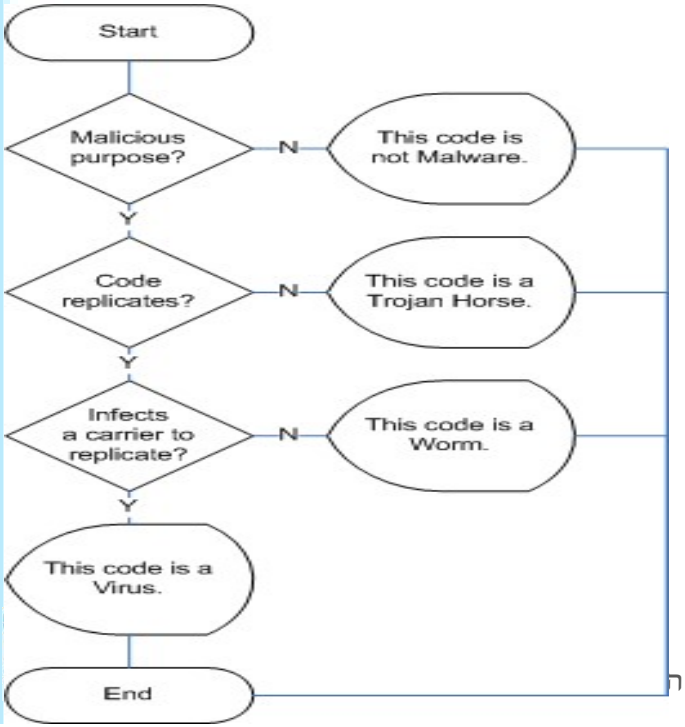
ביצי פסחא	סוסים טרויאניים	פצצות לוגיות
רוגלות	בוטנטים	כופר
דלתות סתרים	Rootkits	תוכנות פרסום
וירוסים	תולעים	

כיום מקובל לקרא לכולם וירוסים

או נוזקות

נוזקה

- נוזקה הינה מונח כוללני המתייחס למגוון שונים של **תוכנה עוינת או מזיקה**, ובכלל זאת וירוסים, תולעים, סוסים טרויאניים, סוגי פרסומות, נוזקות כופר, נוזקות שמטרתן להפחיד את המשתמש ועוד.
- נוזקה יכולה להיות קוד בינארי, סקריפט, active content ועוד. **הנוזקה מוגדרת על ידי כוונתה העוינת, והתנהגותה כנגד המשתמש**. לפיכך, המונח נוזקה אינו כולל תוכנה אשר גורמת לנזק ללא כוונה.
- נוזקה יכולה להיות מכוונת לתפוצה רחבה, או להיות ממוקדת ומוכוונת למטרה ספציפית...



כמה הגדרות

- חדירה
  - הפעלה ראשונית של קוד זדוני במחשב
  - נעשה באמצעות ווקטורי תקיפה
- התקבעות
  - הבטחת המשך ההפעלה העתידית של הנוזקה
    - גם אחרי אתחול
  - הפיכה לשוהה
- 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 פעילות נוזקה

חדירה
- מטרה: הרצת קוד על המכונה הנתקפת

הגנה

העלאת הרשאות
- מטרה: העלאת זכויות/הרשאות
- יתכן שהחדירה כבר הייתה עם הרשאות גבוהות; ויתכן שהרשאות רגילות מספיקות

אבטחת מחשבים

התקבעות
- מטרה: שרידת אתחול
- אם נדרש ואפשר: עם הרשאות גבוהות

התקשרות ל-C&C
- בוטנטים
- מטרה: קבלת פקודות ועדכונים, משלוח נתונים שנאספו

סריקה
- מטרה:
  - תולעת: מציאת מכונות נגישות שניתנות לתקיפה
  - וירוס: מציאת קבצים להדבקה

התפשטות
- מטרה: הרצת קוד על מכונה נגישה אחרת

חבלה וגרימת נזק
- מטרה:
  - איסוף והזלגת מידע
  - שיבוש, חבלה וגרימת נזק
  - משלוח דואר זבל, DOS, ...

הקשרים בזמן ריצה

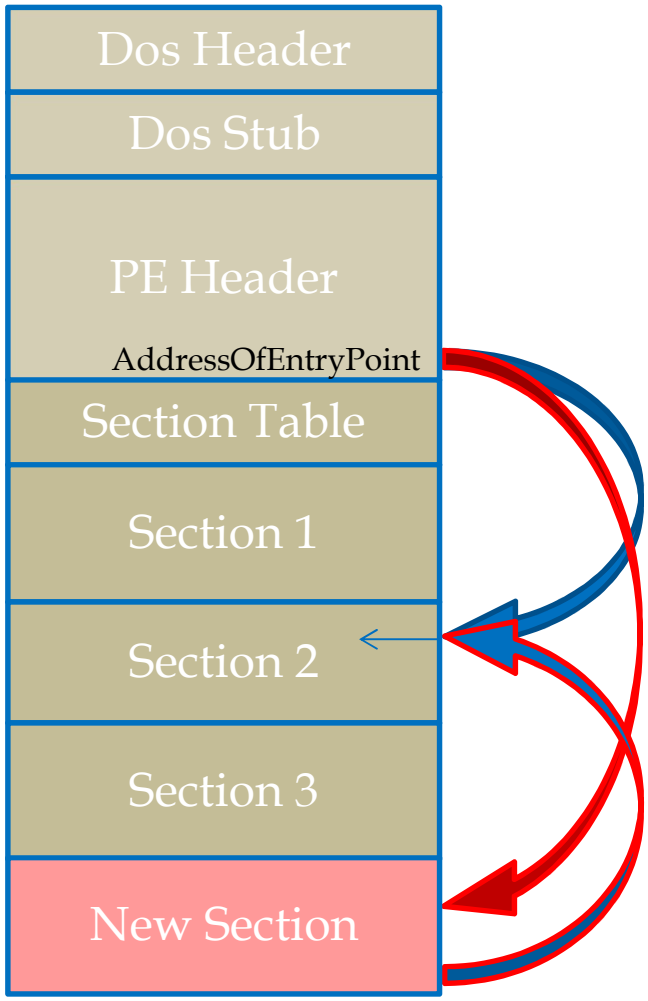
- נוזקה יכולה לרוץ במחשב במגוון הקשרים
- כתהליך
- כדרייבר
  - או DLL
  - או ברכיב של מ"ה
- כקובץ הרצה
  - או ע"י הדבקת קובץ הרצה קיים
  - בתקוה שמישהו יריץ אותו
- ב-DOS : TSR (terminate and stay resident)
  - תוך כדי השתלטות על וקטור הפסיקות
- כ-plugin של תוכנה אחרת
  - או סקריפט שמורץ על ידי תוכנה אחרת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

הדבקת קבצי הרצה

- 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.

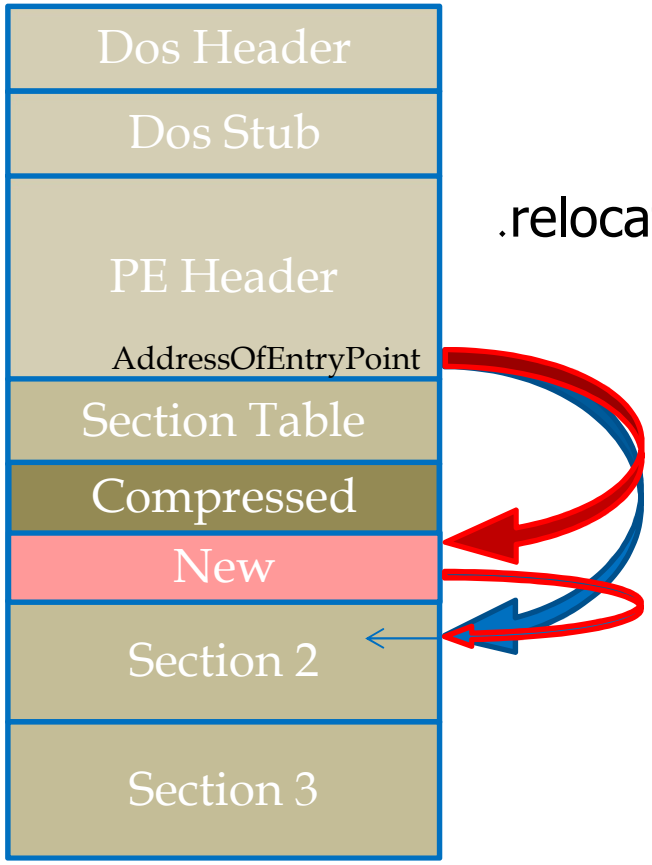
ובביצוע הוקינג...

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 הkernel 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 פעילות נוזקה

חדירה
- מטרה: הרצת קוד על המכונה הנתקפת

הגנה

העלאת הרשאות
- מטרה: העלאת זכויות/הרשאות
- יתכן שהחדירה כבר הייתה עם הרשאות גבוהות; ויתכן שהרשאות רגילות מספיקות

אבטחת מחשבים

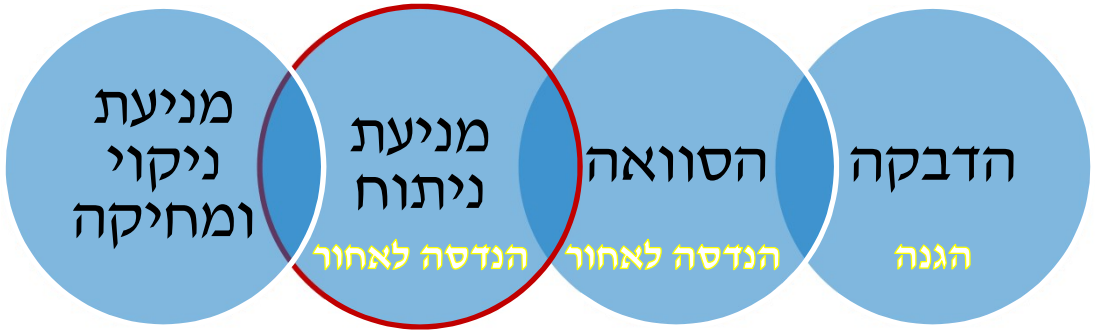
התקבעות
- מטרה: שרידת אתחול
- אם נדרש ואפשר: עם הרשאות גבוהות

התקשרות ל-C&C
- בוטנטים
- מטרה: קבלת פקודות ועדכונים, משלוח נתונים שנאספו

סריקה
- מטרה:
  - תולעת: מציאת מכונות נגישות שניתנות לתקיפה
  - וירוס: מציאת קבצים להדבקה

התפשטות
- מטרה: הרצת קוד על מכונה נגישה אחרת

חבלה וגרימת נזק
- מטרה:
  - איסוף והזלגת מידע
  - שיבוש, חבלה וגרימת נזק
  - משלוח דואר זבל, DOS, ...



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 הבא?

```
#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 הנה?

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

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

דוגמא פשוטה ל-Obfuscation

```
#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" insturctions.

        popad
    }
}
```

```
013D1020  $  53            PUSH EBX
013D1021  .  56            PUSH ESI
013D1022  .  57            PUSH EDI
013D1023  .  60            PUSHAD
013D1024  .  B8 01000000   MOV EAX,1
013D1029  .  83F8 01       CMP EAX,1
013D102C  .  75 02         JNZ SHORT RE_Testi.013D1030
013D102E  .  EB 01         JMP SHORT RE_Testi.013D1031
013D1030  >  0FE8CA        PSUBSB MM1,MM2
013D1033  ?  FFFF          [illegible]
013D1035  ?  FF61 5F       JMP NEAR DWORD PTR DS:[ECX+5F]
013D1038  .  5E            POP ESI
013D1039  .  5B            POP EBX
013D103A  .  C3            RETN
```

From “leetMatrix” blog

דוגמא פשוטה ב-C :

```
static int two=1;

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

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-דיס-אסמבלי – ריבוי משמעויות

- מה עושה הקוד הבא?

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

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

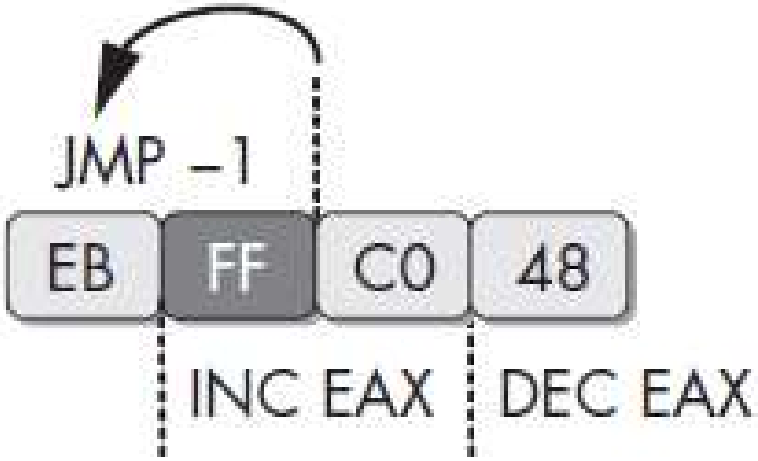
- והקוד הזה?

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

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

אנטי-דיס-אסמבלי – חפיסת קוד

- מה עושה הקוד הבא?

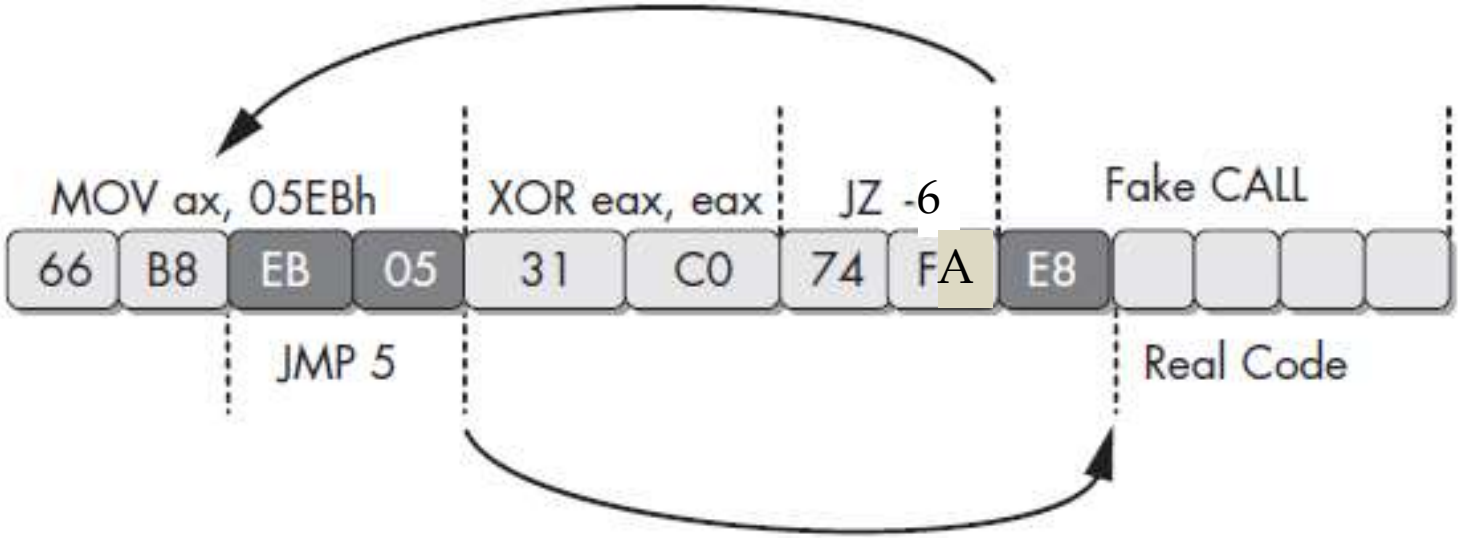


- זה למעשה (כמעט) פקודת nop בת 4 בתים
  - קפוץ אחורה בית אחד, לקוד חופף (2 בתים: EBFF)
  - הגדל את EAX באחד (2 בתים: FFC0)
  - הקטן את EAX באחד (בית אחד: 48)
    - זה אינו nop בגלל ההשפעה על הדגלים

from Practical Malware Analysis

אנטי-דיס-אסמבלי – חפיסת 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-דיס-אסמבלי בשימוש בחריגות

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

אנטי-דיס-אסמבלי בשימוש בחריגות

דוגמא

from Practical Malware Analysis

```
00401050                ❷mov     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                ❸div     ecx
0040106B
0040106B loc_40106B:                             ; DATA XREF: sub_401050o
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
```

עוד דוגמאות לאנטי-דיס-אסמבלי

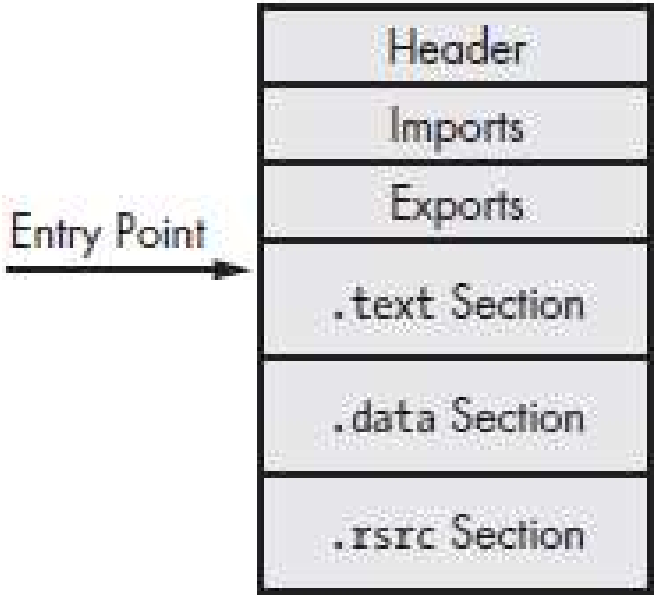
- קפיצות לכתובות שלא ידועות מראש
  - call [eax]
  - קשה להתגבר על השיטה ללא הרצה של הקוד עצמו
  - יש לציין שלקפיצות כאלו יש גם שימושים רגילים
    - למשל בשפות מונחות עצמים, כגון C++
    - או לקריאה לפונקציות מספריות דינמיות
- שיבוש זיהוי אוטומטי של משתנים לוקאלים
  - נעשה על ידי שינוי EBP (או ESP) באמצע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

דחיסת קובץ ההרצה

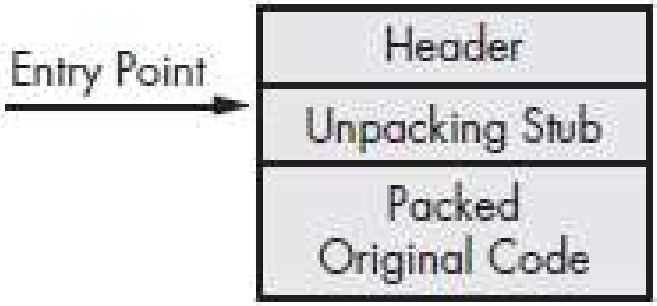
- ישנם כלים שמאפשרים דחיסת תוכנה
  - תוכנה דחוסה, בנוסף לקוד הרגיל, תכיל מתודה שאחראית על חילוץ הקוד והרצתו
  - במקרה כזה לא נוכל לנטר את התוכנה באופן סטטי
    - שכן הקוד לא חשוף לנו
    - כלומר לא ניתן להריץ IDA
  - הצפנה היא אחד הכלים המשמשים בזמן דחיסת קובץ ההרצה
    - אבל קוד הפענוח/מפתח נמצא בתוכנה, אחרת לא תוכל 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

דוגמת UPX

Code?:

```
C0        DB C0
64        DB 64          CHAR 'd'
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'
D9        DB D9
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
1B        DB 1B
50        DB 50          CHAR 'P'
9B        DB 9B
14        DB 14
77        DB 77          CHAR 'w'
00        DB 00
34        DB 34          CHAR '4'
62        DB 62          CHAR 'b'
B0        DB B0
53        DB 53          CHAR 'S'
48        DB 48          CHAR 'H'
```

No strings:

```
004076C2 ASCII "ru"
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

דוגמת UPX

```
004079B0  $ 60              PUSHAD
004079B1  . BE 15704000     MOV ESI,crackV2.00407015
004079B6  . 8DBE EB9FFFFF   LEA EDI,DWORD PTR DS:[ESI+FFFF9FEB]
004079BC  . 57              PUSH EDI
004079BD  . EB 0B           JMP SHORT crackV2.004079CA
004079BF    90              NOP
004079C0  > 8A06            MOV AL,BYTE PTR DS:[ESI]
004079C2  . 46              INC ESI
004079C3  . 8807            MOV BYTE PTR DS:[EDI],AL
004079C5  . 47              INC EDI
004079C6  > 01DB            ADD EBX,EBX
004079C8  . 75 07           JNZ SHORT crackV2.004079D1
004079CA  > 8B1E            MOV EBX,DWORD PTR DS:[ESI]
004079CC  . 83EE FC         SUB ESI,-4
004079CF  . 11DB            ADC EBX,EBX
004079D1  >^72 ED           JB SHORT crackV2.004079C0
004079D3  . B8 01000000     MOV EAX,1
004079D8  > 01DB            ADD EBX,EBX
004079DA  . 75 07           JNZ SHORT crackV2.004079E3
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 crackV2.004079D8
004079E9  . 75 09           JNZ SHORT crackV2.004079F4
004079EB  . 8B1E            MOV EBX,DWORD PTR DS:[ESI]
004079ED  . 83EE FC         SUB ESI,-4
004079F0  . 11DB            ADC EBX,EBX
004079F2  .^73 E4           JNB SHORT crackV2.004079D8
004079F4  > 31C9            XOR ECX,ECX
004079F6  . 83E8 03         SUB EAX,3
004079F9  . 72 0D           JB SHORT crackV2.00407A08
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 crackV2.00407A7A
00407A06  . 89C5            MOV EBP,EAX
00407A08  > 01DB            ADD EBX,EBX
00407A0A  . 75 07           JNZ SHORT crackV2.00407A13
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 crackV2.00407A20
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 crackV2.00407A44
00407A24  . 41              INC ECX
00407A25  > 01DB            ADD EBX,EBX
00407A27  . 75 07           JNZ SHORT crackV2.00407A30
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 crackV2.00407A25
```

הקוד הראשון שרץ:

ניתן לראות בבירור שלא מדובר במשהו טריוויאלי שנרצה לחקור.

דחיסת קובץ ההרצה

דוגמת UPX

```
00407B0B  . 53            PUSH EBX
00407B0C  . 57            PUSH EDI
00407B0D  . FFD5          CALL EBP
00407B0F  . 8D87 9F010000 LEA EAX,DWORD PTR DS:[EDI+19F]
00407B15  . 8020 7F       AND BYTE PTR DS:[EAX],7F
00407B18  . 8060 28 7F    AND BYTE PTR DS:[EAX+28],7F
00407B1C  . 58            POP EAX
00407B1D  . 50            PUSH EAX
00407B1E  . 54            PUSH ESP
00407B1F  . 50            PUSH EAX
00407B20  . 53            PUSH EBX
00407B21  . 57            PUSH EDI
00407B22  . FFD5          CALL EBP
00407B24  . 58            POP EAX
00407B25  . 61            POPAD
00407B26  . 8D4424 80     LEA EAX,DWORD PTR SS:[ESP-80]
00407B2A  > 6A 00         PUSH 0
00407B2C  . 39C4          CMP ESP,EAX
00407B2E  .^75 FA         JNZ SHORT crackV2.00407B2A
00407B30  . 83EC 80       SUB ESP,-80
00407B33  . E9 F895FFFF   JMP crackV2.00401130
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

חילוץ 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 יכיל Import table תקני
  - תזכורת: ל-Import table יש שני מצבים
    1. לפני פתיחת הקובץ – יכיל טבלה עם שמות DLLs ופונקציות שעל מערכת ההפעלה לטעון
    2. לאחר פתיחת הקובץ – יכיל 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

שאלות?

Anti-Debugging

- כאשר תוכנה מזהה שהיא רצה תחת כלי דיבוג או בווירטואליזציה, 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

דוגמאות

- קריאה ל-API:
  - isDebuggerPresent
  - CheckRemoteDebuggerPresent
  - NtQueryInformationProcess
- זיהוי לפי מדידת זמן
- חיפוש int 3
- 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 (משנה את ה-lastError).

ניהוי 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'

int 3 ניהוי קיום

- 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 גורם לחריגה

- תזכורת: int 3 זו חריגה שבה מערכת ההפעלה מעבירה את השליטה ל Debugger
  - כלומר int 3 זה software breakpoint
- אם אין Debugger אז
  - שימוש ב-int 3 יוצר חריגה רגילה

```
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
```

שיטות נוספות לגילוי דיבוג

- 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 לגילוי דיבוג

- ניתן לחפש האם מותקנות במחשב תוכנות מחקר שונות (למשל דרך סריקת תוכן מערכת הקבצים או ה-Registry)
- דוגמא למציאת תהליך עייי 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 לאתרו

ניהוי מחשב 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

ניהוי 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

ניהוי 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

שאלות?

הסוואה בתולעת האינטרנט (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)

שיטות הסתתרות נוספות

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

פרק 10

ניהוי ומחקר נוקות

מערכות אנטי-וירוס

- מערכות אנטי-וירוס מגנות נגד מגוון נוזקות נפוצות
  - וירוסים, תולעים
  - רוגלות, תוכנות פרסום
  - סוסים טרויאניים
  - malicious Browser Helper Objects (BHOs)
  - browser hijackers, key-loggers
  - ransomware (כופרה)
  - backdoors, rootkit
  - malicious LSPs, dialers, fraudtools
- בנוסף, הן מגנות נגד בעיות אבטחה נוספות, כגון
  - הנדסה חברתית
  - עוגיות ריגול
- הן מוגבלות ביכולתן להגן נגד נוזקות לא מוכרות
  - ובפרט נגד נוזקות ייעודיות שנכתבו במיוחד למטרה מסוימת

גילוי נוזקות ע"י חברות האנטי-וירוס

- זיהוי הוירוסים על ידי חברות האנטי-וירוס
- ויצירת בסיס הנתונים שמשמש לזיהוי במחשבי הלקוחות

איסוף
- איסוף דוגמאות של וירוסים ונוזקות
- איסוף "מודיעין"

ניתוח
- תשתית ניתוח נוזקות
- ויצירת חתימות
- ופיתוח אלגוריתמי זיהוי וניקוי

פרסום
- וכמובן כתיבת תוכנת אנטי-וירוס
- וקבצי חתימות
- מאפשר זיהוי וניקוי אצל הלקוח

- בסה"כ השקעה ענקית...

גילוי נוזקות ע"י חברות האנטי-וירוס

איסוף

- חיפוש ברשת
- קבצים חשודים שנשלחים מלקוחות מודאגים
- מלכודות דבש
- מידע שנאסף מתוכנות אנטי-וירוס שהותקנו ע"י לקוחות
  - לקספרסקי יש מיליוני מחשבים מהם הם מקבלים מידע
- איסוף "מודיעין"
- virustotal.com

גילוי נוזקות

- זיהוי נוזקות נעשה במגוון של אלגוריתמים היוריסטיים
  - זיהוי שינויים בקובץ
  - בדיקת תוכן קובץ
  - זיהוי התנהגות
  - בדיקת שינויים במערכת הקבצים (white list)
  - וכו'
- שני מקרים עקרוניים
  - זיהוי סטטי – בדיקת קובץ ההרצה
    - למשל אחרי הורדת קובץ מהרשת
    - או בהכנסת דיסק USB
  - זיהוי דינמי
    - בזמן ריצה
  - וכמובן אפשר לשלב
    - למשל זיהוי חשד באופן סטטי, ווידוא דינמי
      - ואז מציאת חתימה לוירוס שהתגלה, וזיהויו באופן סטטי
    - או קבצי הרצה באמצעות חתימות, וקבצי מערכת באמצעות white list

זיהוי חיובי ושלילי

זיהוי חיובי
- זיהוי מקרים נקיים
- ע"י השוואה לתוכן ידוע
- white list

זיהוי שלילי
- זיהוי מקרים חשודים
- חיפוש פעולות לא חוקיות
- מקובל באנטי-וירוס

מחקר ויירוסים – אנלינת Black Box

- ניתוח קוד לוקח זמן רב
  - וודאי אם מדובר בניתוח סטטי
  - וגם ניתוח דינאמי ...
- לפעמים מספיק להבין מה תוצרי הפעולה ללא להבין בדיוק מה נעשה
  - שינוי קבצים אחרים
  - שינוי ה-registry
  - קריאה לפקודות מערכת הפעלה "משונות"
  - תקשורת החוצה
- הרבה מאוד מניתוח הנוזקות נעשה מ"בחוץ" ולא "מבפנים"
  - בדרך כלל מספיק להבין את התוצרים על מנת להבין שהמדובר בנוזקה.
  - אז מתי בכל זאת עושים (או מנסים לעשות) ניתוח קוד?

בעיה מרכזית – רעשי רקע

- בכל רגע נתון יש אלפי גישות למערכת קבצים וכמו כן לא מעט תקשורת יוצאת/נכנסת.
  - הוירוס מהווה חלק קטן מהגישות האלו.
- יש צורך להכיר את המצב הנורמלי של המערכת על מנת להצליח לזהות את הפעולות החשודות.

ניתוח שינויים במחשב

- SysInternals
  - סט כלים ותוכנות לניתוח וניהול מחשבי windows :
    - Process Explorer
    - Process Monitor
    - Autoruns
    - RootkitRevealer
    - ועוד כ-60 כלים.
  - חברה שנקנתה על ידי MS
- RegShot
  - הקלטת מצב המערכת registry וגם קבצים והשוואה למול מצב הבא.
- WINDBG Detours או כלי מעקב אחרים על מערכת ההפעלה
  - בדיקה אלו פונקציות מערכת הפעלה הופעלו..

דוגמא: קובץ

TextOutA	GDI32
GetStockObject	GDI32
VirtualAlloc	KERNEL32
GetCurrentProcess	KERNEL32
GetStartupInfoA	KERNEL32
GlobalAlloc	KERNEL32
GetWindowsDirectoryA	KERNEL32
GetWindowsDirectoryW	KERNEL32
lstrcatW	KERNEL32
GetProcessHeap	KERNEL32
CreateFileW	KERNEL32
CreateWindowExA	USER32
RegisterClassExA	USER32
LoadCursorA	USER32
DefWindowProcA	USER32
ShowWindow	USER32
EndPaint	USER32
BeginPaint	USER32
InvalidateRect	USER32
SendMessageA	USER32
DestroyCaret	USER32
HideCaret	USER32
ShowCaret	USER32
CreateCaret	USER32
SetCaretPos	USER32
GetFocus	USER32
MessageBoxA	USER32
ReleaseDC	USER32
GetDC	USER32
UpdateWindow	USER32
GetMessageA	USER32
TranslateMessage	USER32
DispatchMessageA	USER32
LoadIconA	USER32

- הפונקציות אליהן הקובץ עושה Import :

(נראה כי מדובר בתהליך עם ממשק משתמש. לא אופייני לנוזקה)

Process Explorer

- נשתמש ב-Process Explorer:

explorer.exe		33,840 K	30,100 K	1856 Windows Explorer	Micr
vmtoolsd.exe		9,960 K	7,920 K	1980 VMware Tools Core Service	VMv
PWRISOVM.EXE		904 K	544 K	2000 PowerISO Virtual Drive Man...	Pow
ctfmon.exe		1,056 K	1,632 K	160 CTF Loader	Micr
cmd.exe		2,280 K	72 K	2084 Windows Command Processor	Micr
procexp.exe	2.94	10,004 K	16,452 K	5940 Sysinternals Process Explorer	Sysi
Procmon.exe		16,092 K	1,372 K	2068 Process Monitor	Sysi
7faad9ed4a2e68d77a0d32ca2917e39f.exe	3.13	5,964 K	2,000 K	2632 Назначенные задания	Kop
svchost.exe	8.82	12,596 K	5,720 K	5964 Generic Host Process for Wi...	Micr
TPAutoConnect.exe		12,392 K	4,240 K	2496 ThinPrint AutoConnect comp...	Cort
IEXPLORE.EXE		7,416 K	17,252 K	4732 Internet Explorer	Micr
IEXPLORE.EXE		67,924 K	81,116 K	3220 Internet Explorer	Micr

- התהליך מפעיל את svchost.exe ונסגר.
  - חלק מהנוזקות מתחזות ל-svchost.exe.
  - אבל במקרה הזה הנוזקה מפעילה את ה-svchost המקורי שבא עם הווינדוס.

הדגמת sysinternals

- נשתמש ב-Process Monitor:

svchost.exe	4788	CreateFile	C:\Documents and Settings\Administrator\Desktop\sql\7faad9ed4a2e68d77a0d32ca2917e39f\7faad9ed4a2e68d77a0d
svchost.exe	4788	QueryInformatio...	C:\Documents and Settings\Administrator\Desktop\sql\7faad9ed4a2e68d77a0d32ca2917e39f\7faad9ed4a2e68d77a0d
svchost.exe	4788	QueryAllInforma...	C:\Documents and Settings\Administrator\Desktop\sql\7faad9ed4a2e68d77a0d32ca2917e39f\7faad9ed4a2e68d77a0d
svchost.exe	4788	CreateFile	C:\Documents and Settings\Administrator\Local Settings\Application Data\gapvfbsv.exe
svchost.exe	4788	CreateFile	C:\Documents and Settings\Administrator\Local Settings\Application Data
svchost.exe	4788	WriteFile	C:\Documents and Settings\Administrator\Local Settings\Application Data\gapvfbsv.exe

- Svchost.exe העתיק את הוירוס למקום אחר בשם אקראי.
  - שם קובץ gapvfbsv.exe.
- אפשר למצוא עוד דברים.

הדגמת sysinternals

- נשתמש ב-Autoruns:

Winlogon | Winsock Providers | Print Monitors | LSA Providers | Network Pro
Everything | Logon | Explorer | Internet Explorer | Scheduled Tasks | Services | Drivers | Codecs | Boot Execute | Ima

Autorun Entry	Description	Publisher	Image Path	Timestamp
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run				8/4/2013 1:46 PM
Command Pro...			c:\windows\system32\cmdprompt.pif	5/30/2014 6:06 PM
fpro32			c:\windows\system32\fpro32.pif	5/30/2014 6:06 PM
Norton anti virus			c:\windows\rsv32.pif	5/30/2014 6:06 PM
PWRISOVM.E...	PowerISO Virtual Drive Man...	Power Software Ltd	c:\program files\poweriso\pwrisovm.exe	7/20/2013 3:18 PM
VMware User ...	VMware Tools Core Service	VMware, Inc.	c:\program files\vmware\vmware tools\vmtoolsd.exe	2/26/2013 5:30 AM
HKLM\SOFTWARE\Microsoft\Active Setup\Installed Components				8/4/2013 1:30 PM
Address Book 6	Outlook Express Setup Libr...	Microsoft Corporation	c:\program files\outlook express\setup50.exe	4/13/2008 9:30 PM
Microsoft Outlo...	Outlook Express Setup Libr...	Microsoft Corporation	c:\program files\outlook express\setup50.exe	4/13/2008 9:30 PM
HKCU\Software\Microsoft\Windows\CurrentVersion\Run				9/22/2013 3:00 PM
nvjfrpja	[redacted]	[redacted]	c:\documents and settings\administrator\local settings\application data\gapvfbsv.exe	2/6/2014 4:51 PM
[redacted]				[redacted]
FlashPlayerUp...	Adobe® Flash® Player Inst...	Adobe Systems Incorporated	c:\windows\system32\macromed\flash\flashutil32_11_5_502_110_activex.exe	10/29/2012 4:55 AM
HKCU\SOFTWARE\Microsoft\Internet Explorer\Desktop\Components				8/4/2013 1:46 PM
0			File not found: About:Home	
HKLM\Software\Classes*\ShellEx\ContextMenuHandlers				8/4/2013 4:20 PM
ANotepad++	ShellHandler for Notepad++		c:\program files\notepad++\nppshell_05.dll	6/18/2012 6:24 PM

- gapvfbsv.exe רץ עם אתחול המערכת.

סיכום הדוגמא

- נסיק את המסקנות הבאות:
  - הוירוס הזריק קוד ל-svchost בזמן ריצה.
    - כי רץ ה-svchost המקורי אך הוא התנהג בצורה שונה.
  - הוירוס הסתיר את ה-Import table שלו וייתכן שהיה שימוש ב-Packer.
    - הרי אנחנו יודעים שהיה שימוש ב-CreateProcess וזה לא הופיע ב-Import table.
  - הוירוס ביצע עוד פעולות:
    - העתיק את עצמו למיקום אחר בשם אקראי.
    - הגדיר את עצמו לרוץ עם אתחול המערכת.
    - ועוד פעולות שהיה ניתן לגלות באמצעות Procmon.
  - לקח לנו מספר דקות לגלות את כל זה.

ניתוח תקשורת

- כל Malware חייב לתקשר עם האינטרנט.
  - בשביל לשלוח מידע לתוקף.
  - בשביל להתפשט ברשת.
  - ושלל סיבות אחרות...
- בדרך כלל ההתנהגות של נוזקה שונה מהתנהגות של תוכנה רגילה. דוגמאות:
  - נסיון להתחבר לעשרות מחשבים ברשת ע"י סריקת פורטים.
  - שליחת מידע בשעות לא סטנדרטיות.
  - שליחת מידע בעל אופי חריג.
    - למשל ניצול חולשה.
- בעזרת ניטור חבילות המידע שנכנסות/יוצאות מן המחשב ניתן:
  - לקבל תמונה כללית על הנוזקה.
  - ויותר מזה - לגלות קיום של נוזקות.
- Wireshark היא התוכנה המובילה כיום לניטור תעבורת הרשת שעוברת במחשב מסוים.
  - פורמט הקובץ - PCAP
- האם ניתן לנטר תעבורה באופן בטוח? מה הסימנים המעידים?

דוגמה לגלישה

- דוגמה לגלישה באתר אינטרנט (עם Wireshark):

128	19.3288070	192.168.2.102	62.219.78.115	HTTP	582 GET / HTTP/1.1
134	19.4334850	192.168.2.102	62.219.78.115	HTTP	627 GET /style.css HTTP/1.1
145	19.4584920	192.168.2.102	62.219.78.115	HTTP	638 GET /images/logo_01.gif HTTP/1.1
148	19.4588960	192.168.2.102	62.219.78.115	HTTP	638 GET /images/logo_02.gif HTTP/1.1
151	19.4606560	192.168.2.102	62.219.78.115	HTTP	638 GET /images/logo_03.gif HTTP/1.1
159	19.4643110	192.168.2.102	62.219.78.115	HTTP	633 GET /images/bg.png HTTP/1.1
181	19.5093640	62.219.78.115	192.168.2.102	HTTP	257 HTTP/1.1 200 OK (text/css)
183	19.5097570	192.168.2.102	62.219.78.115	HTTP	638 GET /images/logo_05.gif HTTP/1.1
233	19.5677730	62.219.78.115	192.168.2.102	HTTP	162 HTTP/1.1 200 OK (GIF89a)
235	19.5687120	192.168.2.102	62.219.78.115	HTTP	638 GET /images/logo_06.gif HTTP/1.1

```
⊞ Frame 128: 582 bytes on wire (4656 bits), 582 bytes captured (4656 bits) on interface 0
⊞ Ethernet II, Src: Giga-Byt_97:a0:7a (90:2b:34:97:a0:7a), Dst: EdimaxTe_c3:27:ae (00:0e:2e:c3:27:ae)
⊞ Internet Protocol Version 4, Src: 192.168.2.102 (192.168.2.102), Dst: 62.219.78.115 (62.219.78.115)
⊞ Transmission Control Protocol, Src Port: 55766 (55766), Dst Port: http (80), Seq: 1, Ack: 1, Len: 52
⊟ Hypertext Transfer Protocol
  ⊞ GET / HTTP/1.1\r\n
    Host: www.trythisOne.com\r\n
    Connection: keep-alive\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/28.0.
    Accept-Encoding: gzip,deflate,sdch\r\n
    Accept-Language: he-IL,he;q=0.8,en-US;q=0.6,en;q=0.4\r\n
    Cookie: __utma=187449709.1628208541.1390075373.1397578245.1398548363.8; __utmz=187449709.139673241
    \r\n
```

נוזקה ברשת

- נוזקה כאמור, מתנהגת שונה מתוכנה רגילה.
- דוגמאות לפעולות חשודות של נוזקה:
  - סריקת מחשבים – אם הסריקה היא בתוך הרשת אנחנו נראה המון בקשות ARP יוצאות מהמחשב (הדגמה בהמשך).
  - סריקת פורטים – נראה התחברות לאותו IP אך עם פורטים שונים.
    - בדרך כלל חיבורים שנכשלים מרמזים על פעולה חשודה. שכן תוכנה רגילה לא מנחשת לאילו פורטים להתחבר.
  - בקשות HTTP מוזרות – למשל שליחת כמויות מסוימות של מידע לשרת C&C.
  - שימוש בפרוטוקול מסוים בצורה חריגה.
    - יש צורך להכיר מאות פרוטוקלים בשביל להגדיר מה זה חריג, אבל ההיכרות הכרחית.
  - ועוד

זיהוי התנהגות ב–Sandbox

- מערכת זיהוי התנהגות (Sandbox) אוספת מגוון נתונים ואירועים ממהלך ריצה
  - ובודקת אם הם צפויים או לא
  - כלומר אם התוכנה שרצה מבצעת סדרת אירועים כזו
  - או שתוכנה שאינה נוזקה צפויה גם כן לבצע סדרה כזו
- אירועים שנאספים יכולים לכלול
  - קריאה לקריאות מ״ה חשודות
    - כמו כתיבה לזיכרון של תהליך אחר
  - פתיחת קבצים
    - למשל כתיבה לקובץ הרצה אחר
  - יצירת תהליכים חדשים
  - שינוי בווקטור הפסיקות
  - ועוד – דוגמאות?
- דוגמא - Cuckoo

וננים ומצאתי ויروס – מה עכשיו?

או מה יש באנטי ויروס שלי?

חתימות של וירוסים

- חתימה של וירוס: מחרוזת ייחודית המופיעה בו, ולא מופיעה בתוכנות אחרות
  - השיטה הקלאסית לזיהוי וירוסים ונוזקות היא על ידי זיהוי ה"חתימות" שלהם
  - יצור החתימות דורש השוואה לכל התוכנות המוכרות!
    - היו מקרים שיצרני אנטי-וירוס בחרו בטעות חתימות שכן מופיעות בתוכנות אחרות, למשל WORD או מ"ה Windows עצמה
  - שיפור: המחרוזת נבחרת כביטוי רגולרי
    - למה זה משפר?

- סריקת כל הקבצים במערכת
  - וגם כל התכניות בזיכרון
    - כולל בזמן טעינת תוכנית חדשה
    - וכשהיא כבר רצה (למשל, אחרי שוירוס פענח את הקוד שלו)

חתימות של וירוסים

- אנטי וירוסים מקבלים מידי פעם קובץ חתימות חדש
  - שישמש מאותו רגע להשוואה עם כל הקבצים במערכת
- יצרן האנטי-וירוס מחשב אותו על סמך
  - כל הוירוסים המוכרים
  - וכל התוכנות החוקיות המוכרות

- הבדיקה נעשית מול התוכן הנוכחי בלבד
  - אין השוואה לתוכן היסטורי
    - למשל לגרסה מקורית של הקבצים
  - בדיקה מאד יעילה (מעבר אחד מהיר על כל קובץ)

אלגוריתמים היוריסטיים להגנה

- מסוגלים לזהות וירוסים לא מוכרים
  - אבל בלי לדעת מה הם מבצעים
- טכנולוגיה – חיפוש היוריסטי בקוד
  - חיפוש פעולות מחשידות בקוד
  - חיפוש מחרוזות קוד מחשידות
    - מחרוזות שוירוסים נוטים להפעיל
  - ביצוע RE של קוד באופן אוטומטי
    - כדי לזהות פעולות חשודות

- עלולים לטעות...
  - תוכנה חוקית עשויה להפעיל קוד מחשיד

חסימת התנהגות אסורה

- תוכנות אנטי-וירוס יכולות לחסום קריאה לקריאות מ״ה
  - או פונקציות ספריה מסוימות
  - כשצפוי שתוכנה חוקית לא תקרא להן
- בזמן קריאה לקריאות מ״ה כאלה
  - או אם קריאות מסוימות מופיעות ב-Import Table
- האנטי-וירוס יעצור את התוכנית
  - או יפתח חלון עם שאלה למשתמש
  - או ישלח דוגמא ״הביתה״

White List

- במערכות מסוימות התוכנות להן מותר לרוץ ידועות מראש
  - למשל מערכות ייעודיות, בהן המשתמש לא מתקין תוכנה נוספת
  - או שהתקנות הן נדירות
- במקרים כאלה, במקום לחפש וירוסים ונוזקות
  - נבדוק אם הקבצים במערכת זהים למקוריים שהיו בהתקנה
  - כלומר ששום קובץ לא השתנה
    - ולא נוסף שום קובץ חדש
  - השוואה לתוכן היסטורי
    - השתנה או לא השתנה?
- ניתן לעשות זאת ביעילות בעזרת פונקציות תמצות
  - מספיק להשוות תמציות
  - ואין צורך להשוות לעותק מקורי מלא
- תוכנות: Tripwire, McAfee ePO ואחרות

IPS/IDS

- Intrusion prevention/detection systems.
- רכיבים שמטרתם לנטר את התעבורה ולחפש פעולות שנחשבות זדוניות.
  - יכולים לתפקד כרכיבים נפרדים או בתוך רכיבים קיימים (למשל ראוטרים).
- IPS מנסה לחסום את הפעולות האלו בעוד IDS תפקידו רק לדווח.
- איך זה עובד?
  - זיהוי סריקות או מתקפות רשת (מתקפות על פרוטוקלים).
  - חתימות על חבילות שמוגדרות כחשודות.
    - למשל חתימה על אקספלויט מסוים.
  - ניתוח פעולות חריגות (אנומליות).
    - כלומר ללמוד איך הרשת צריכה להתנהג במצב נורמלי ולזהות פעולות שחורגות מן המצב הנורמלי. כל רשת מתנהגת בצורה שונה.
    - ועוד שיטות

אנטי-וירוס – סיכום

ניתוח: בדיקה וגילוי

- ניתוח : בדיקה וגילוי
  - על ידי ניתוח אנושי
  - מערכות אוטומטיות הבודקות פרמטרים חשודים
    - שינוי קבצים, הדבקת קבצים אחרים, פעילויות לא מקובלות

- טכנולוגיות בשימוש
  - ניתוח סטטי ודינמי
  - הרצה במכונות וירטואליות
    - מונע גרימת נזק, קל לבטל הנזקים
  - אמולציה
    - תוך חיפוש פעולות שתוכנות רגילות לא נוהגות לבצע

- מטרה
  - מציאת מאפיינים ייחודיים על פיהם תוכנות ה-AV יפעלו
  - העדפה מובנית למניעת false-positives מלמציאת יותר נוזקות

שאלות?

APT

APT

Where are all the 'A's in APT?

Posted by on *Sep 20, 2018*

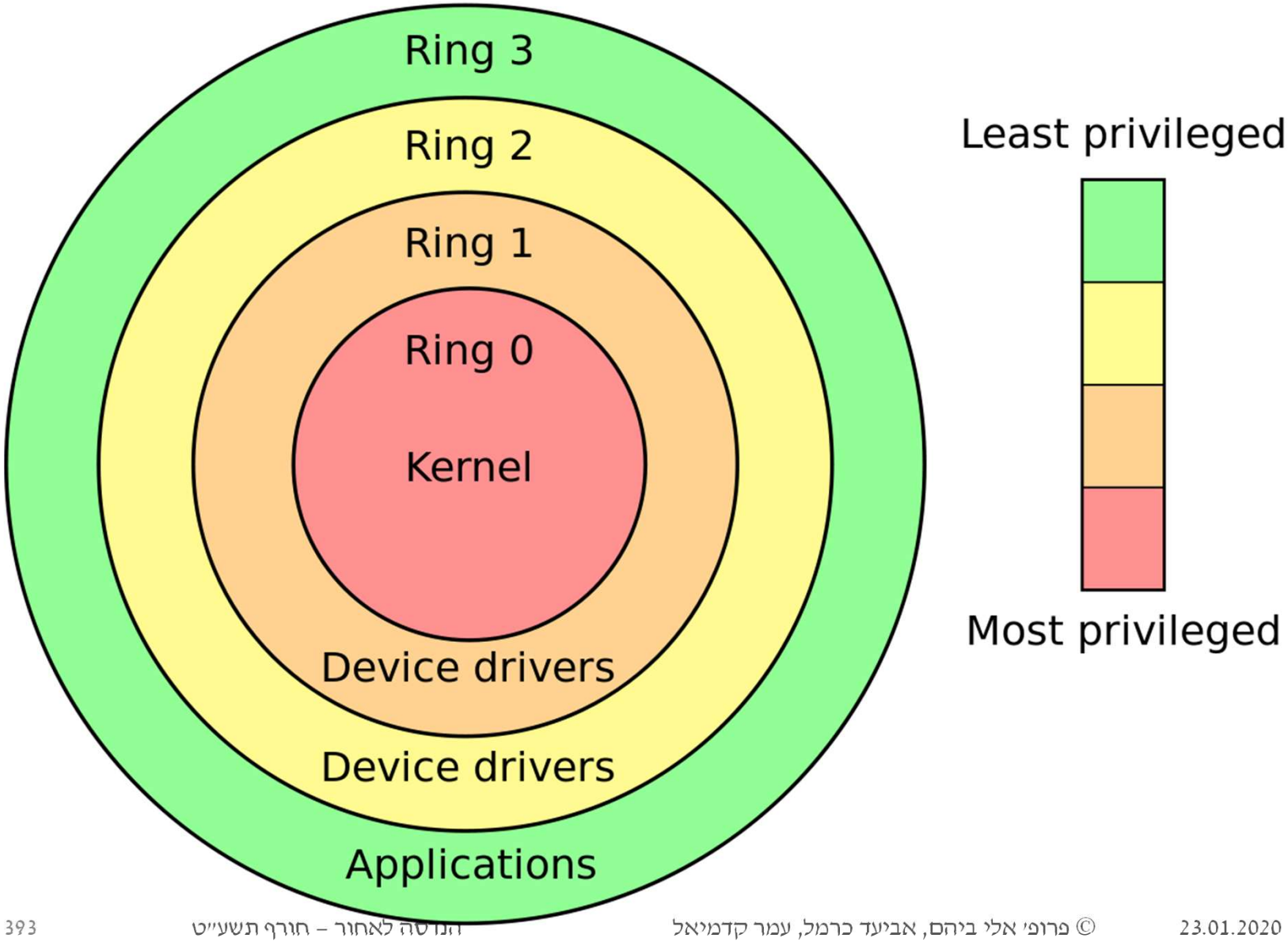
In a guest blog post by VB2018 gold partner Kaspersky Lab, Costin Raiu, Director of the company's Global Research and Analysis Team, looks critically at the 'A' in APT.

So what is missing?

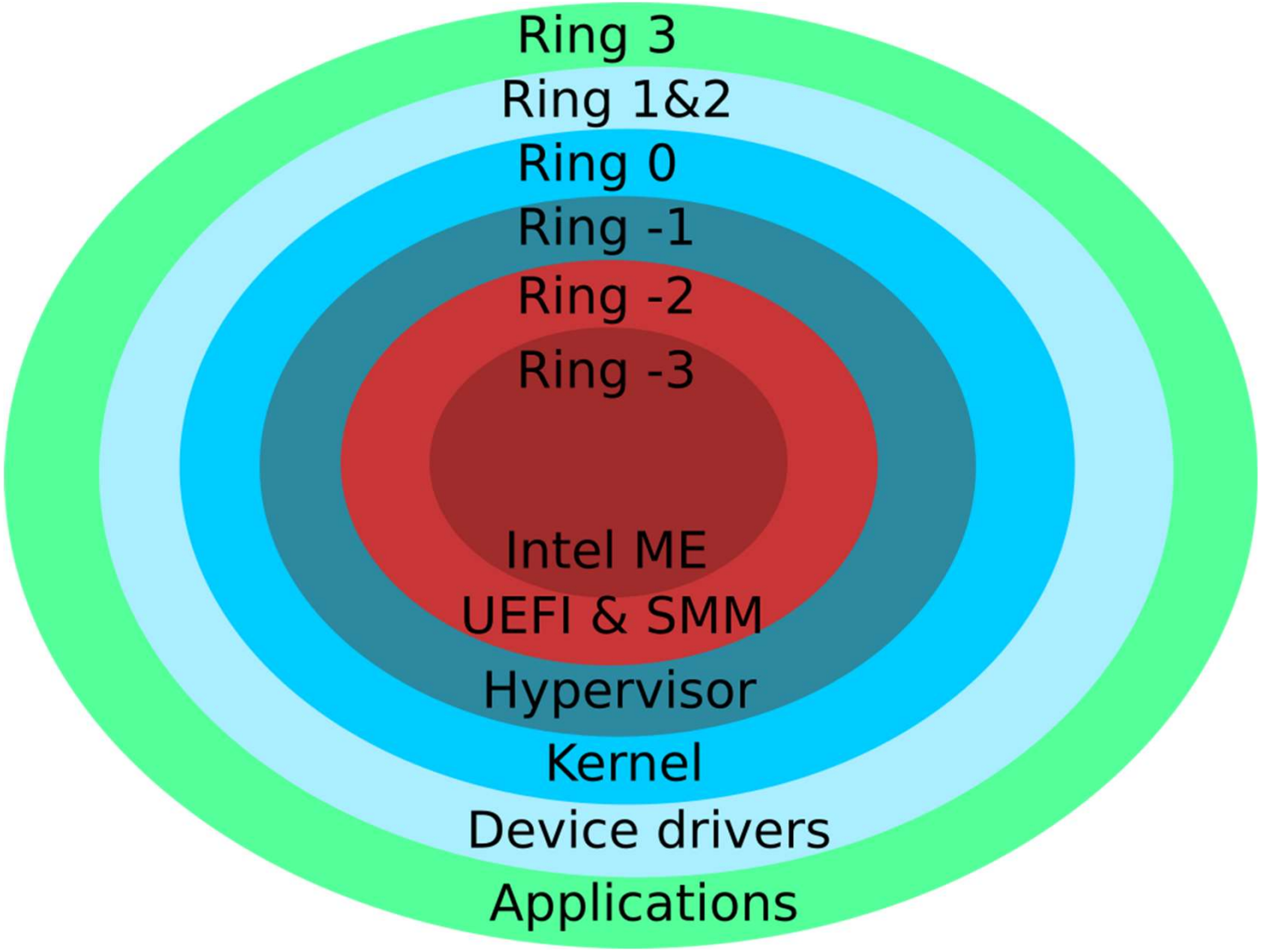
Looking at the discussions and development of sophisticated attack techniques, there is a significant difference between the theory and in-the-wild observations. So what is missing? Here's a list of possible culprits:

- 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.
- Destructive attacks against hardware – the Stuxnet payload is an example, but also BlackEnergy attacks in which UPS 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



תזכורת: היפרוייזור

- 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 הוירטואליות רצות תחת מחשב אחד
    - בחדר השרתים הפקולטי
    - בעל זיכרון גדול ומרובה ליבות
  - וההיפרוייזור זו מ"ה-על
    - שמריצה תחתיה את המכונות הוירטואליות
    - מחליטה מי מהן תרוץ באיזה ליבה ומתי
    - מבצעת context-switching בין המכונות הוירטואליות
    - מקצה להן זיכרון, ופונה עבורן להתקני קלט-פלט

Blue Pill

- Blue Pill זה תוכנה שמסתתרת כהיפרוייזור
  - 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

תזכורת: תהליך עליית מ"ה ב-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

תזכורת: תהליך עליית מ"ה ב-PC

- הוא פונה לדיסק הנבחר, וקורא ממנו את הסקטור הראשון בדיסק
  - נקרא Master Boot Record בדיסקים
    - או Boot Sector בדיסקטים
  - בו יש פרטים נוספים לגבי הדיסק, חלוקתו למחיצות, וכו'
- ומפעיל ממנו קטע קוד קטן
  - שתפקידו להביא את קוד הטעינה של מערכת ההפעלה
  - בגלל קטנותו, בדר"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

תזכורת: תהליך עליית מ"ה ב-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

הדבקת 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

הדבקת ה–Boot Sector

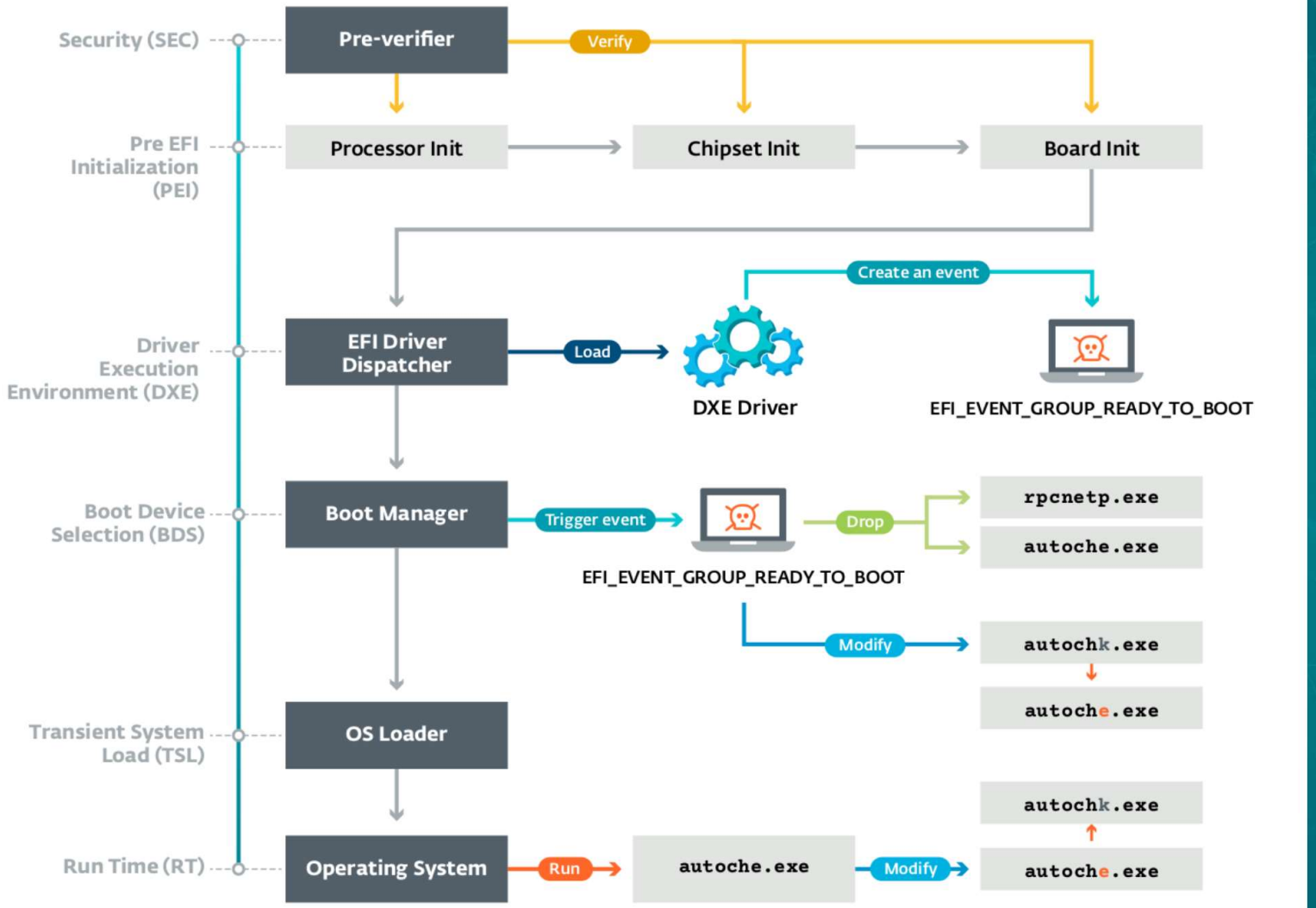
- צורת הדבקה זו הייתה מאד נפוצה בתקופה שמידע דיגיטלי הועבר בדיסקטים
  - בעלת יתרונות רבים
  - אבל קשה לביצוע כיום
    - יש הגנות על כתיבה ל-boot sector של דיסקים קשיחים
    - וכמעט שאין אתחול של מחשבים מהתקנים נתיקים

ENJOY SAFER TECHNOLOGY™

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 בדיסק 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

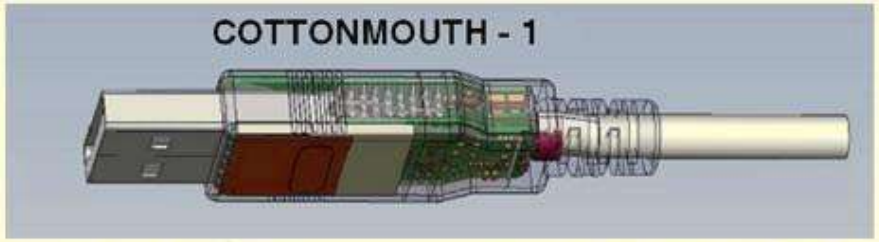
TOP SECRET//COMINT//REL TO USA, FVEY

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



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 גישת 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 מי"ה ברגע שההתקן הזה מעוניין לפעול
  - חלומו של כל וירוס...

ויכוסים מפורסמים

Stuxnet

- וירוס לתקיפת מערכות חלונות המחוברים לבקרים תעשייתיים של סימנס

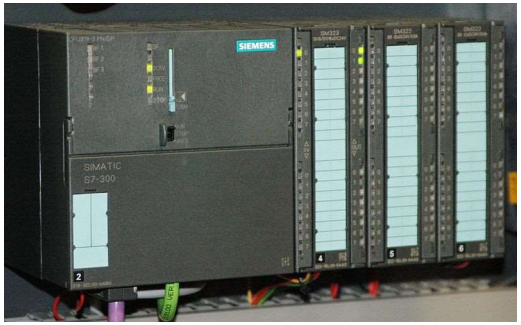
- כנראה יועד כנגד אירן
  - אבל הדביק גם תחנות גרעיניות ברוסיה ובעולם
  - אפילו הדביק את תחנת החלל הבינלאומית
    - כנראה דרך דיסק USB
- חיפש מחשבים עם תוכנת Step7 וחיבור ל-PLC של סימנס
  - מותנה בחיבור לציודים מסוימים, ובסיבוב בתדרים מסוימים
  - כשמצא כאלה – ניסה להשיג גרסה מעודכנת שלו
  - לא הפעיל את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

שלוש מטרות הדבקה

- הדבקת מחשבי חלונות
  - במגוון דרכים
    - מתוכן 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

תוכנת SCADA	בקרי PLC	ציוד חיצוני
• תוכנה להצגת נתוני בקרים • מתקשרת עם הבקרים • ויכולה לשלוט עליהם • רצה בחלונות	• בקרים תעשייתיים • שולטים על ציוד חיצוני • קלט ופלט אנלוגי • כוללים תוכנה • מאפשרים החלפת התוכנה בזמן ריצה • ללא אמצעי אבטחה • יש אפילו במערכות החשמל כאן בבניין	• מנועים • צנטריפוגות • מערכות חשמל • משאבות מים • וכו'

PLC של סימנס – S7-300



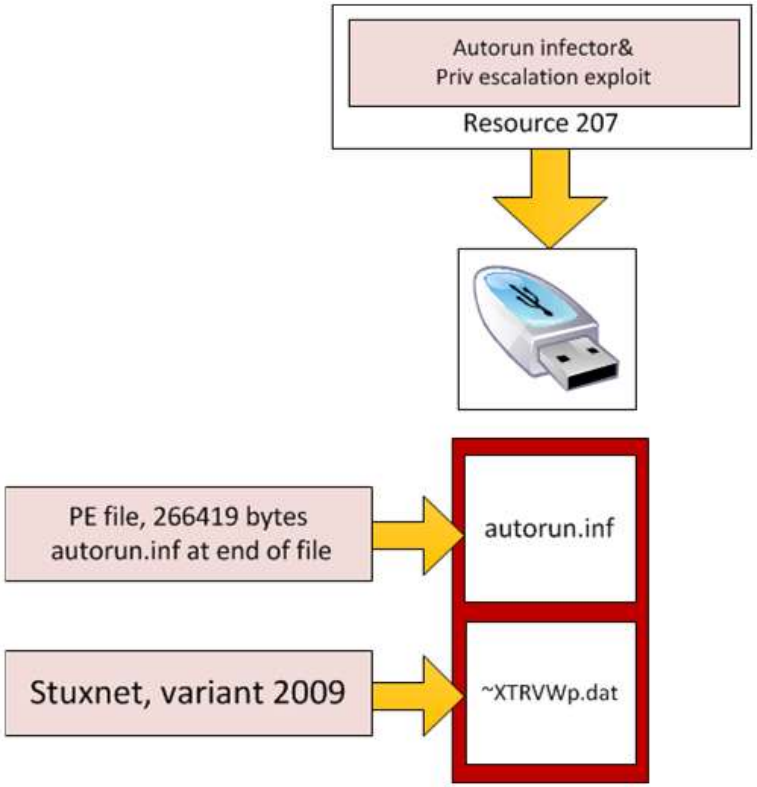
Stuxnet

- 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

שימוש ב-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

הסוואה

- 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

מגוון ספריות תוכנה

- גודלו ענק : 20MB

- כולל מגוון ספריות לתמיכה
  - בחמישה סוגי הצפנה,
  - כמה סוגי דחיסה,
  - SQLite,
  - מכונה וירטואלית,
  - ועוד

- התקנות במי"ה
  - משתמש ב-registry
  - מתקין דרייבר אודיו מזויף
    - משמש כעוגן נגד מחיקתו מהמחשב

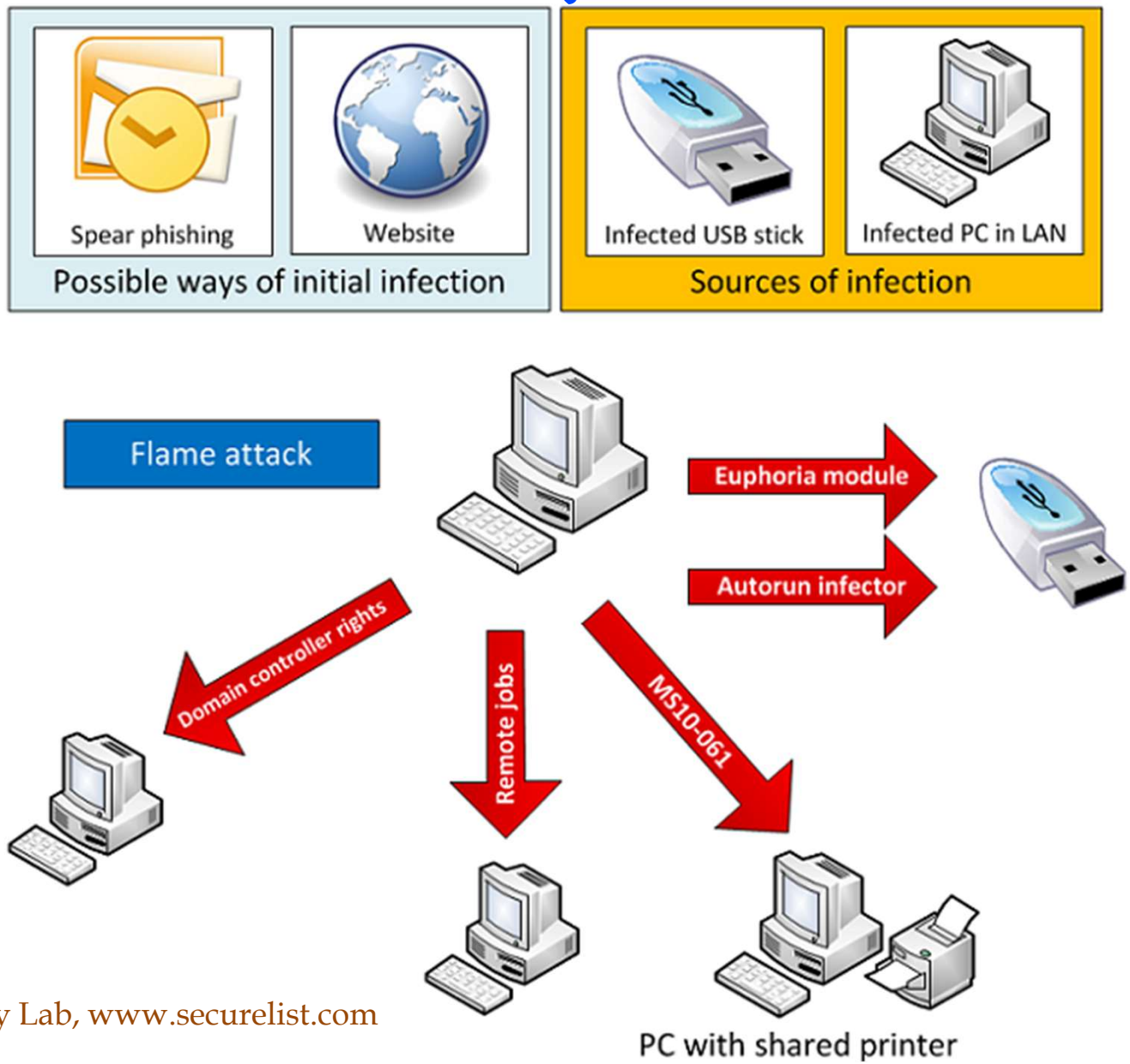
תכונות נוספות

- 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 של 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



מניעת גילוי ע"י זיוף סרטיפיקט

- חלונות מגנה כנגד דרייברים ותוכנות מתחזות ע"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

הסרטיפיקט המנוית – צ"י שבירת MD5

Version	V3
Serial number	3a ab 11 de e5 2f 1b 19 d0 56
Signature algorithm	md5RSA
Signature hash algorithm	md5
Issuer	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
Public key	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)

פרק אחרון

לקראת סיום...

קורס הנדסה לאחור

נושאים מרכזיים

- X86 Assembly
- ניתוח סטטי ודינאמי
- Hooking and Injecting *(and Instrumentation)*
- חולשות, ניצולן ומנגנוני הגנה *(ללא חולשות מתקדמות)*
- נוזקות, הסוואה וכלי נגד להנדסה לאחור
- ערוצי צד וחומרה

© פרופ' אלי ביהם, אביעד כרמל, עמר קדמיאל

מבחן

- מבחן בשני חלקים בני שלוש שעות כ״א
  - כ-4-5 שאלות
  - חלק ראשון : 09:00-12:00
  - חלק שני : 14:00-17:00
  - יש לגשת לשני חלקי המבחן באותו מועד
- תכולה : כל החומר שנלמד במקצוע
  - הרצאות, תרגולים, סדנאות, שקפים, תרגילי בית
- המבחן יתקיים בספריה
- הביאו מחשבים ניידים
  - לסטודנט שאין מחשב נייד מתאים יושאל מחשב נייד
  - המבחן יופץ דרך אתר GR++
  - הגשת הפתרונות גם כן דרך האתר (כהגשת ״תרגיל בית״)
  - אין להביא מסכים

תכורת: יושר אקדמי

- העבודה היא שלך – כתוב אותה בעצמך
  - אין להעתיק מאחרים
  - אל תשתמש בניתוחים קודמים של החומר אותו אתה מנתח
    - אחרת הציון יגיע למי שכתב אותו
  - צטט מקורות
- ואם בכל זאת עשית משהו אסור – אמור זאת בפירוש
- במבחן:
  - אסור לשוחח על נושאי המבחן עם אחרים בזמן המבחן
    - ז"א שגם אסור להשוות פתרונות
  - הפתרונות שלכם מהתחלה ועד הסוף
- העוברים על הכללים יוענשו בחומרה

סוף

בהצלחה!