



הפקולטה למדעי המחשב
הטכניון

הנדסה לאחור

236496

מרצה: עמר קדמיאל

מתרגל: טל שנקר

© Eli Biham, Omer Kadmiel and Aviad Carmel

יקץ רצף

- שפות עיליות
 - C
- אסמבי

- Intel 32-bit IA32 assembly
 - ימד בקורס בתרגול
 - לא נתיחס לגרסאות 64 סיביות
- ידע בסיסי במערכות הפעלה
 - הבנת דרך הפעולה של מערכת הפעלה והמבנה שלה
 - השימוש בחלונות, כולל תכנות ודיבוג
 - ניסיון כמשתמש וירטואלייזציה
- ידע בסיסי באבטחת מידע
 - וחולשות אבטחה



2



קְנָאִם

- קדם חובה
 - מערכות הפעלה 234123 או 046209 או שקול
- צמוד חובה
 - הגנה במערכות מתוכנות (הגנה בראשות) 236350
 - או אבטחת מחשבים 236652 236653 או 236607 או 236350
 - או השלמה עצמית של שיעור חולשות ב-236350
 - תוך נוכחות בהרצאה וบทרגול בנושא (בסביבות השבוע השלישי)
- מומלץ
 - קומפילציה
- כדאי לזכור
 - את'ם



3



הכט איזע אַז האקזא

- הודעות וציוונים יועברו דרך מערכת GR
 - עליהם לודא רישום במערכת GR כדי לקבל הודעות ונוכן להזין להם ציוונים
 - זה אמור להתבצע אוטומטית למי רשום רשמי למקצוע
 - הירשמו ליום של GR, וכן הסדנאות, הבוחן, המבחןים ותרגילי הבית יופיעו להם אוטומטית ביום שלהם
- שקיים הרצאות ותרגולים, פרטי צוות המקצוע ושעה קבלה, הודעות, ציוונים, ופרטים נוספים באתר המקצוע
<http://webcourse.cs.technion.ac.il/236653/>

The screenshot shows the homepage of the "Reverse Engineering and Malware - 236653" course. At the top, there are language options: العربية (Arabic), עברית (Hebrew), Русский (Russian), and English. The title "Reverse Engineering and Malware - 236653" is displayed prominently. Below it, the semester "אביב 2014" (Spring 2014) is mentioned. A large text block describes the course content: "במסגרת הקורס נעסק בהנדסה לאחרור של תוכנה. נלמד את השיטות המקובלות להנדסה לאחרור, ואיך משתמשים בהן בפועל. בישם טכניקות אלה לחקור נזקota, ונלמד גם טכניקות אחרות המאפשרות דיזיין תוכנה או מידע מתוך הנתונים על הדיסק, מתקשות מחשבים והתקפות נוספות." On the right side, there is a sidebar with blue buttons labeled: סילבו (Syllabus), הودעות (Announcements), מידע כללי (General Information), סילבוס (Syllabus), and סגל (Faculty). The top right corner of the page has the Technion logo and the text "הטכניון - מכון טכנולוגי לישראל".



4



סזום

- יתקיימו מספר סדנאות בכיתה
 - בשעות של השיעורים והתרגולים
 - בשאייפה – אחת לשלושה שבועות
- הביאו את המחשבים הניידים שלכם
 - במקרה הצורך אפשר להפעיל את המחשב הווירטואלי הפוליטי מהנייד שלכם



5



אפקט האקזא

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



6



יעך קקאי

- העבודה היא שלך – כתוב אותה בעצמך
 - אין להעתיק מזרים
 - אל תשתמש בניתוחים קודמים של החומר אותו אתה מנתח
 - אחרית הציון הגיע למי שכتب אותו
 - אין לשתמש בדה-קומפיילרים
 - הפתרונות שלכם מהתחלת ועד הסוף
 - צטט מקורות
- ואם בכלל זאת עשית משהו אסור –-Amor זאת בפירוש
- **העוברים על הכללים יונשו בחומרה**



7



המכורמת פאייזר

- **אתם מתבקשים להתכוון לשיעורים**
 - על ידי קריית שקפי השיעור הבא לפני השיעור
- כך נוכל לדון בעקרונות, ולא לעסוק בפרטים משנהים



8



סיכום

- רשימת הספרות נמצאת באתר המקצוע
 - ספרים נמצאים בספרייה
- בנוסף, קיים חומר רב באינטרנט
 - בפרט, תיאור מלא של אסמבלי של אינטל
 - link באתר המקצועי



9



פרק 1

הנתקה מהר



10

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

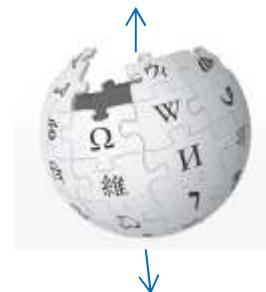
23.01.2020



אפק



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



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 פעולהיה.



הנֶּסֶת מִן אַכְזָרָם

- בעידון התעשייתי, עוד לפני עידון המחשב, הנדסה לאחר היותה נפוצה מאוד
 - עוד לפני חוקי הפטנטים וזכויות היוצרים 
- קל יחסית לבצע הנדסה לאחר של מכונות
 - רכיבים פיסיים גדולים
 - אבל לעיתים מסתמכים על חומרים ייחודיים
 - בעלי תכונות מסוימות
 - למשל בשעונים



12



גזרה מוחור מואה

- בעידן המחשב – מסובך יותר – בגלל הקוטן והמורכבות
 - והקושי של פירוק למרכיבים
- אבל לא בלתי אפשרי
 - יתכן שהצורה של המוצר מדילפה מידע או שנייתן להיות כזו באופן ויזואלי
 - במעבדים מודרניים:
 - מיקרוסקופים ומיקרוסקופים אלקטרוניים
 - מדידת מתח בזמן הפעולה
 - הזרקת מתח למעבדים ומעקב אחרי התגובה
 - גרים להשיגות בחישוב, ומעקב אחרי השינויים
 - קריית זיכרון, או האזנה ל-*bus*, אם אפשר
 - ועוד



13



הנזהה מוכנה

נתרכז במקרים
הלו רוב הזמן



- הבנת קוד תוכנה בשפה עילית
- הבנת קבצי הרצה (כגון EXE או a.out)
 - אנליזה סטטית – דה-קומpileציה
 - אנליזה סטטית – שפת מכונה
 - אנליזה דינמית – דיבגרים
- קוד בינאים ו-JIT – .NET & Java bytecode
- רלונטי גם ל
 - אופטימיזציה של קוד עי'י המהדר
 - בהינתן קוד המקור או ייצוג פנימי של המהדר
 - Post-link optimization
 - קלומר אופטימיזציה בהינתן קבצי הרצה
 - בדיקות אבטחה של קוד
 - תלויות במה הקוד עושה



גזרה מתחם: כלוקט מידע וקבעים

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



15



אריך-ג'ים / Hooking-

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



מה הזרה מתחילה?

- מחקר תוכנות תקיפה לצורך פיתוח אמצעי הגנה כנגד
 - למשל לצורך Antivirus
- חיפוש חולשות – לצורך פיתוח הגנות כנגד
 - 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
- בוחינה אם אחרים העתיקו קוד שלך
 - Forensics
- זיהוי API של תוכנות, מ"ה, וכו'
- וידוא שיטויות מספקים אחרים עומדות בדרישות האבטחה של המוצר



17



מה הזרה מתקינה?

- סיבות לא חוקיות
 - עקיפת הגנה בתוכנה
 - פריצת תוכנה – Crack
 - חיפוש חולשות – מחקר לצורך פיתוח שיטות תקיפה
 - גניבת קוד ממוצר מתחילה
- כמובן שבמקרה זה לא עוסוק בפעולות לא חוקיות



18



אקלים יזום בקורס זה

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



19



...התקן מכך...

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



הפרק הקרוב

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



21



היאוים גראזקוט

- מפתחי נזקיות מחפשים אחרי חולשות לא מוכנות
 - הן צריכות לדעת
 - איך להסתיר את עצמו
 - איך להדביק תוכנות ומערכות אחרות
 - איך לגרום נזק
 - איך להגן בנגד RE
 - RE משמש לגילוי חולשות ולהבנה של התוכנות המותקפות
- מגנים נגד נזקיות צריכים להבין את דרך פעולה הנזקה
 - וחולשותיה, כולל כל ה"איך" לעיל
 - RE משמש למחקר נזקיות, ולפיתוח שיטות זיהוי וניקוי של הונן



22



Blaster – מאיימת?

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



זוכרים?



23



Blaster – מאיימת?

- כיצד התולעת הדרישה מחשבים?
- MS03-026 מתיחס לחולשת אבטחה מטיפוס חריגה מהווצץ במנגנון RPC במערכות חלונות השונות (XP בין היתר)

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
 - שרבו נעשה באמצעות Reverse Engineering



Blaster – מילצת קואט

- כיצד עצרו את התולעת?
- RE של התולעת אפשר לדעת
 - כיצד היא מתפשתת? (אייזו חולשה צריך לסגור)
 - כיצד להסיר אותה מהמחשב?
 - איך לגנות בזמן אמת שמחשב מודבק על מנת למנוע זאת?
 - מה המטרה של התולעת?
 - כיצד היא מקבלת פקודות מהמעביל?
 - וכו'



25



התקפות RE-f סוליד

- תשתיות לאומיות
 - השתלטות על תחנות כוח והפסקת פעילותן
 - או שריפת הגנרטורים על ידי הגברת מהירותם מעבר למותר
 - השתלטות על אספקת המים
 - מערכות רמזוריים, מחלפי רכבות, וכו'
 - מצלמות מהירות, ומצלמות אבטחה
 - נניח שהיא 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.



26

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד קרמל, עמר קדמיאל

23.01.2020



סינרים אפליקטיבים RE-f

- נניח שבמקרים שלנו יש באג שמאפשר להחביא ממנו מטוסים או שנייתו לכבות אותו מרחוק – חייבים לגלוות ולתקון
- על פי סנוודן, ה-NSA גילתה עשרות בעיות אבטחה במערכות הפעלה



חדשנות בעולם

איראן הציגה את המל"ט האמריקאי החמקן שהופל

מפקד חיל האוויר המשמרות המהפקה חשף את המל"ט הסודי שנרגל בשנים האחרונות לאחרי הגרען: "יחידת הלוחמה האלקטרונית שלנו הפילה אותנו"

הארץ | פורסם לראשונה: 08.12.2011 | 18:47 | עדכן ב: 09.12.2011 | 00:07

| 45 | הוסף תגובה

Ewen MacAskill in Washington
Sunday 22 April 2012 10.50 BST

the guardian
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



27

הנדסה לאחור – חורף תשע"ט

© פרופ' אליאב יהם, אביעד כרמל, עמר קדמייאל

23.01.2020



מיצג RE



28

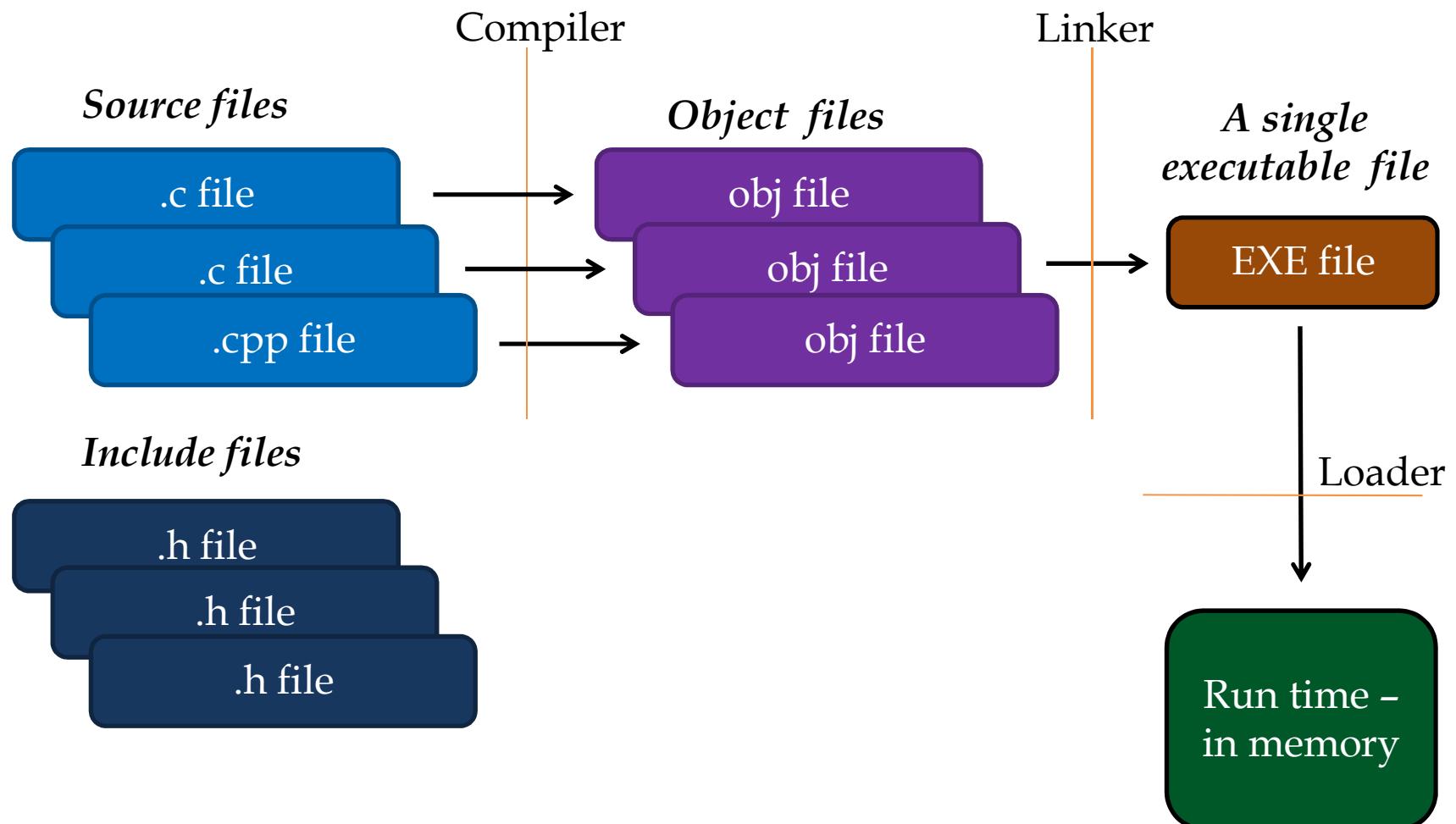
הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020



תפקיד ההיינר וה.nr ב-C++ ו-C



איךagi

- למרות שקובץ הרצאה בחלונות הוא תמיד באותו פורמט
ואוֹתָה שְׁפַת מִכּוֹנָה, כֵּל מַהְדֵּר יֹצֵר קּוֹד שׁוֹנוֹה
 - 3 שורות ב-C יכולות להתדר למספר שורות באסמבלי, בעודו קוד שנכתב ב-VB יכול להתדר למאות שורות
 - ואוֹתוֹ קּוֹד ב-C יִתְוָגֶם עַל יָדֵי מַהְדֵּר אַחֲרָן אַפְּנָן אַחֲרָן
 - וכמו כן, ניתן להוראות למַהְדֵּר לְבָצָע אֲוֹפְּטִימִיזָּצִיּוֹת מִסּוֹגִים שׁוֹנוֹים
וכך ליצור קוד שׁוֹנוֹה
 - (כפי שתראו בתרגיל הבית)

קוד

הידור וلينק

EXE file



30



המהג'יק ההפאך

- כדי להבין קובץ הרצאה, כדאי לדעת מהי השפה שבה נכתב הקוד ואיזה מהדר היה בשימוש
 - בהרבה מקרים קל לשלוֹף את המידע הזה מקובץ הרצאה, על פי הספריות שבשימוש, ומידע אחר הנמצא בקובץ
- לעיתים בקובץ הרצאה יש מידע נוסף המועד לlienker וואו לדיבגר שיכל לכלול שמות פונקציות ושמות משתנים
- ידע זה עוזר לזהות מבנים באסמבלי ולתרגם בקלות הקוד
- כמו כן, משתמשים בכלים שעוזרים לפענח את פקודות המכונה לקוד ברמה גבוהה יותר

קוד או אלגוריתם

RE
←

EXE file



31



קואץ אֲחֵזָה?

```
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
ecx, ecx
```

כמה זמן לוקח להבין מה הקוד הבא
עשוה? (אל תنسו)

האם זה אפשרי למצוא את קטע
הקוד הרלוונטי לנו מתוך מיליון
שורות אסטבלי?

האם כותב הווירוס יכול להוסיף
cosa נוסף במכוען?

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



Java Bytecode & .NET

- ישנו שפות שמהוודרות לקוד בינאים (Bytecode)
 - לכל שפה כזו תיש מכונה וירטואלית
 - ש.compiled בזמן ריצה מkode הבינאים לאסמבלי (JIT)
- אחד היתרונות בשיטה זאת הוא האפשרות להריץ את אותו הקוד במערכות הפעלה שונות וארכיטקטורות שונות בקלות
 - Java רצה בכל מערכות הפעלה לרבות פלאפונים ושעונים
- תכניות.NET. (למשל C#) מהוודרות לקבצי הריצה (EXE)
 - הכוללים קוד אסמבלי וקוד בינאים (CIL, נקרא בעבר MSIL)
 - קוד האסמבלי שנמצא שם אחראי להרצה קוד הבינאים C-JIT
 - זיהוי: קבצי EXE משתמשים ב-.dll או ms.net написו ב-.NET.
- בדרך כלל מחקר Bytecode לא נחשב מסובך
 - המונומידיע נשמר בקובץ
 - ניתן לחסית בקלות להבין מה היה הקוד בשפה העילית
 - יש כלים אוטומטיים שמקלים על כך



כיצד אָהֶן הַגְּזֵסָה בְּמִתְכְּנָת?

- הנדסה לאחרור היא בעצם ניסיון להבין למה המתכנת התכוון
- לצורך כך מאד חשוב להכיר את השפה שבה הקוד נכתב ואת מערכת הפעלה לעומקן
- ואיך מהדר מיציר קוד בשפת מכונה לפקודות שונות בשפה העילית
- ומאז מועיל גם להבין את צורת החשיבה של המתכנת המקורי ואת סגנון כתיבתו, שיעזרו להבין את מה שהוא התכוון לעשות
- ולהשווות לי"יך אני הייתי כותב את זה"



מה עושה הקוד הבא?

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
```



קונטן קידומס – מנגנון גיבוב

הוא שקול לקוד הבא, בהחלפת הgiשות לזיכרונו 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
```



קונסיגנציון – הגדה של גאנטס

בחירה מושכלת של שמות משתנים תקל עליינו 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 בהסכם 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 ניתן מוצר מתחרה המעתיק את המקור
- בחלק מהמדינות חל איסור גורף לפרסם מידע שהושג על ידי RE
- שימוש שקווד תוכנה עשוי להיות מוגן בזכויות יוצרים
 - ולכן צוותים המבצעים RE לשם פיתוח מוצר אינם מתקשרים ישירות עם הגורמים שמשתמשים בתוצרי ה-RE
 - אלא דרך מסמכים בלבד המפרטים את הנדרש
 - ולא פירוט בכתב קוד



38



אכג"ה: חוק ה-DMCA

The Digital Millennium Copyright Act

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



39



אֲכָרְגָן אַקְזִיאָן?

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



40



האם?



41

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020



תלכאות: ארכיטקט הפעלה והלכளון



42

הנדסה לאחור – חורף תשע"ט

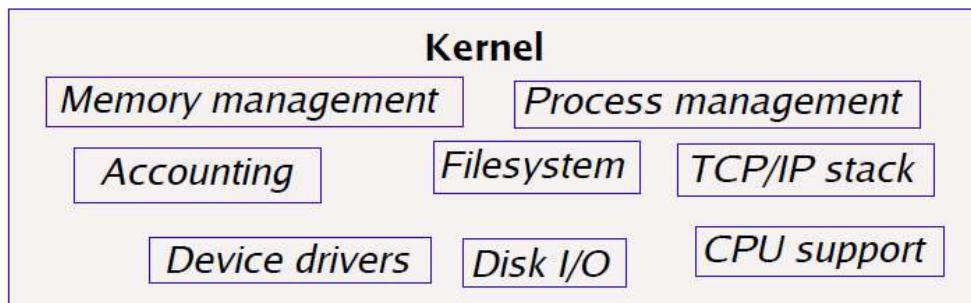
© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020



ארכיטקטורת הפעלה

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



שכל אחת מהן מאפשרת
פקודות אחרות
ומשתמשת במבנה
זכרון אחרים.

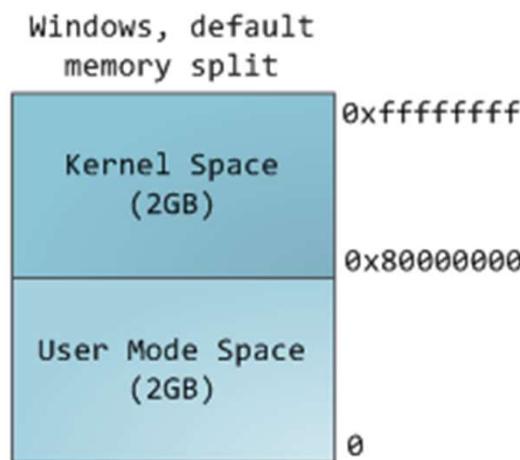


43



ריאיון נכון

- לכל תהליך "יוזר" יש את מרחב הזכרון הווירטואלי שלו.
 - כל קוד הkernel 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 הkernel.
- קוד שרצה באזור היוזר אין גישה למרחב הזכרון של הkernel
 - האם הדבר נכון גם לkernel? למה?

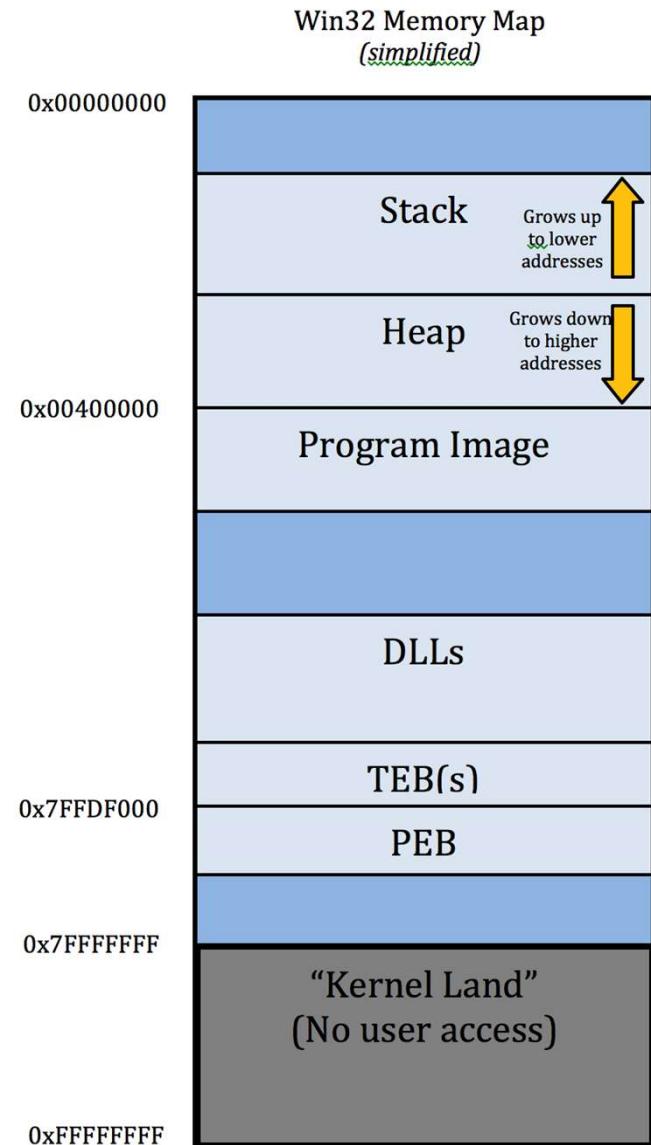


44



ארכיטקטורת מחשב

- **קוד** – Program Image – זהו קוד התהlixir עצמו. מכיל את הקוד (.text) ואת המידע (.data) של התהlixir.
- **DLL-ים** – ספריות הנטענות למרחב הזכרון של התהlixir.
- **מבנה ניהול זכרון Stack** (מחסנית) – מחזיקה משתנים מקומיים, פרמטרים וכתובות חוזרת (עוד בהמשך).
- **Heap** – משמש להקצת זכרון בזמן ריצה, מיועד למידע שנשאר אחריו הפונקציה שיצרה אותו.
- **טבלאות TEB** (thread) ו-**PEB** (process) שמכילות את כל המידע הדרוש לצרכי ריצה.



45



פרק 2

רימוח סטס אקראי



46

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020



רימוח 1660



47

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020



רימוח סטטי

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



48



אֲהֵה קִיס-אַסְמָבֶּל?

- דיס-אסטמבלר הוא תכנית, שקוראתקובץ הרצאה
 - ממיר את תוכן הקובץ משפט מכונה לאסטמבי
 - מנקודת ראות מייה והמעבד הוא סתם תכנית
- דיס-אסטמבלר צריך
 - להכיר את מבנה קובץ הרצאה (למשל PE של חלונות)
 - לפרש פקודות אסטמבי
 - לחבר כתובות לשם פונקציות ושמות משתנים
 - כולל קישורים אל DLL-ים
 - זיהוי מיקום פקודות המכונה
 - במעבדים בהם גודל פקודות מכונה אינו קבוע זה לא תמיד פשוט
 - זהות בין קוד למשתנים, ובין סוגים משתנים שונים
 - למשל להדפס נכוון מחרוזות, שלמים, וכו'



49



ק'יך פואט? ייס-סואג'ר?

- על פניו דיס-אSEMBLER מזזה מיקומי פקודות מכונה באופן דומה לזייהוי על ידי המעבד בזמן ריצה
 - ככלומר, פקודה מתחילה בבית שאחורי סוף הפקודה הקודמת
 - או, אם יש פקודת קומז, בבית שלוו ה-קומז מפנה
- דיס-אSEMBLER בסיסי מפרש פקודות זו אחר זו
 - על פי סדרן בזיכרון
 - ע"י תרגום פקודה אחר פקודה (Linear Sweep) 



50



ק'יך פואף? ייס-סאמבל?

- לאותו רצף של בתים יכולות להיות כמהמשמעות
 - לבן התרגום האוטומטי לא תמיד זיהה את המשמעות הנכונה
 - נראה דוגמאות בהמשך הקורס
 - ישנים אלגוריתמים ושיטות יותר מתקדמות לזהות האסמלבי
 - אחת השיטות היא לזהות קפיצות ולקבוע שיעד הקפיצה חיב להכיל קוד אסמלבי תקין (נקרא **Recursive Traversal**

 - איזה שיטות נוספות אפשריות?

...	...	E8	74	48	66	B8	48	EB	E8	31	C0						
...	call eip+0xb8664874				dec eax	jmp eip-24		xor eax, eax		...									
...		jz eip+72		mov ax, 0xeb48			call ...												
...			dec eax	mov ax, 0xeb48			call ...												



ק'יך פואף? ייס-סמאכט?

- כישיש תערובת של קוד ונתונים באותו section, זה לא תמיד קל
 - אחרת, שם ה-section יכול לעזור
 - מקובל ש-text. מצין קוד, ו-**sh-data**. מצין משתנים
- דיס-אSEMBLER יכול להשתמש במידע נוסף שקיים בקובץ ההרצתה, למשל
 - נתוניים הקשורות קובץ ההרצתה
 - כגון כתובת תחילת התוכנית
 - מידע על הספריות הדינמיות (DLL, **ko**)
 - כתובות הפונקציות מטבלת ה-export
 - מידע שמיועד לדיבגר
 - אם התוכנית הודרכה עם אופציית דיבגר
 - כולל שמות משתנים ומיקומים 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 האחריות בעלות יכולות נוספות, לא חופשיות
 - **אין להשתמש בגרסאות אחרות בקורס זה.**
- מאפשר רישום הערות ומtran שמות לתאי זיכרון
 - למשל משתנים וקוד
 - **יכולת מומלצת ביותר**
- אפשרות לעבוד עם קוד קרייא יותר אחרי שזיהינו חלק מרכיביו
 - וanno נבקש שכך תעשו



53



רימוח ציוני



54

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020



לימוח קייזי

- ניתוח דינמי מפעיל את התוכנה הנחקרת, ועוקב אחרי פעולה
 - לצורך כך, תהליך אחר, הנקרא דיבגר, שולט על התהlixir הנחקר
 - קובע متى יעשה מה
 - יכול לעזר ולחדש את הפעולה
 - עוקב ויכול לשנות את כמעט כל משאב של התהlixir המודובג
- בשונה מניתוח סטטי, הניתוח נעשה בזמן ריצה
 - ולא על קובץ ההערכתה ללא ריצה
 - לכן, ניתן לבחון נתוניים שאינם זמינים בניתוח סטטי



אה לא ? ימcalc?

• Debugger מאפשר למשתמש

- הרצת הקוד
 - תוך כדי בדיקת האוגרים והזיכרון של התהיליך המדובג
- שינוי כל אחד מהערכים הנוכחי
 - כולל של הקוד עצמו
- עצירת ריצת התוכנית (breakpoints) לפי מספר שיטות
 - נרחב בהמשך
- לעיתים גם פירוש מבנים מסוימים בזיכרון
 - למשל ניתוח מבנה קובץ הרצה



56



כגיאת רפואית

mdbgers תהליך ב-user mode בלבד

mdbger הכל, הכל היחידי שmdbger kernel

- כלים נפוצים בחלונות :

- Ollydbg
- Immunity Debugger
- Windbg

- לינוקס : gdb



57



קִימָאַפְּ כְּכָעֵין אַ"ג

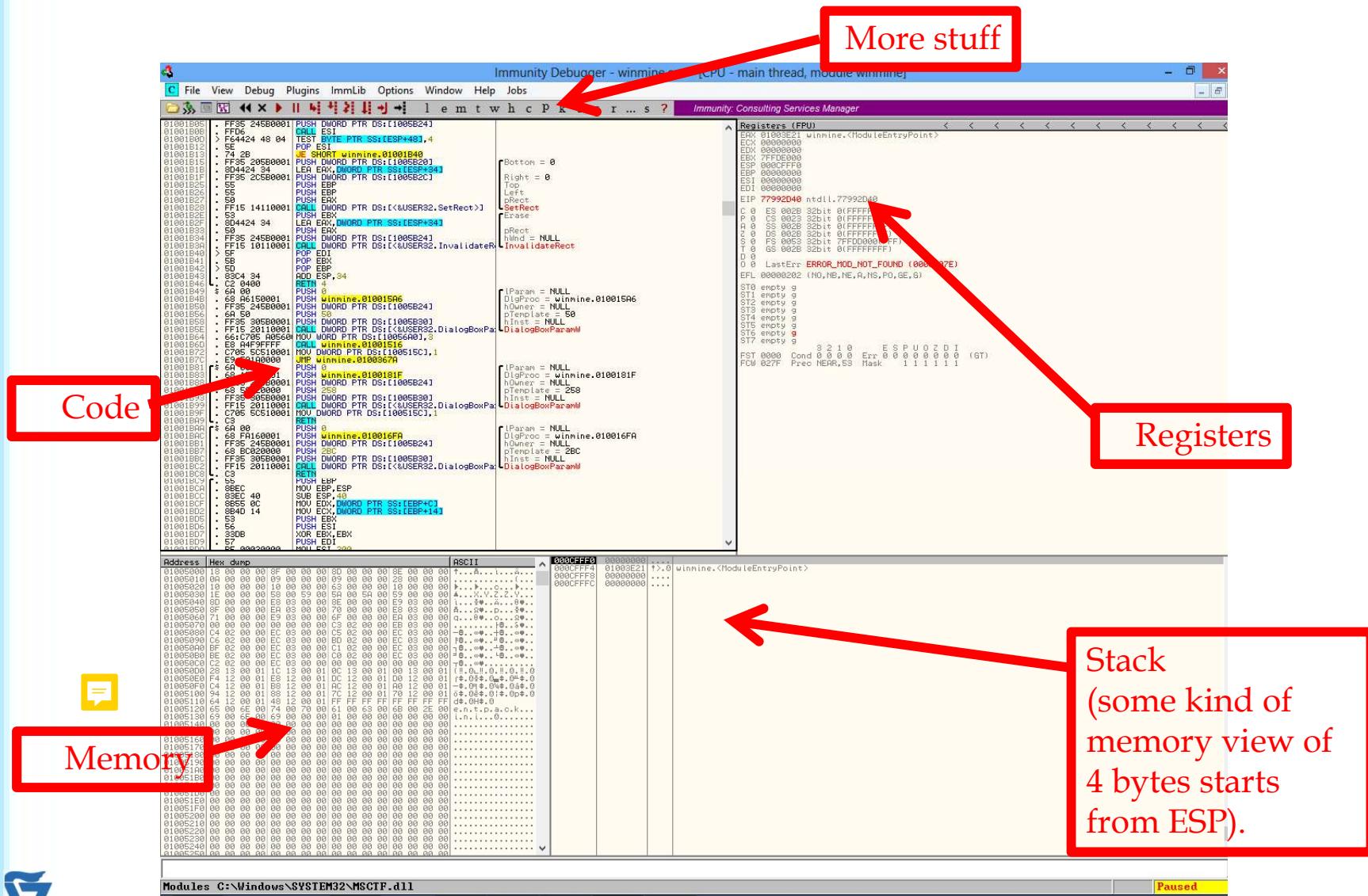
- דיבוג גרעין מייה (kernel) אינו יכול להיעשות כמו דיבוג תhalbיך
 - כי הוא לא תhalbיך
 - אסור לעצור את פעולת הגרעין באמצעות ביצוע
 - ואין משמעות להחלפת הקשר של גרעין מייה
 - הכליםשמייה מספקת לדיבגר לא קיימים במקרה זה
- לכן דיבוג של גרעין מייה דורש כלים ייעודיים
- לא עוסוק בכך בקורס זה



58



Ollydbg/IMM



59

הנדסה לאחור - חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020



איך פועל קיינט?

- דיבגר הוא תהליך, השולט על פעולות תהליך אחר
- מנקודת ראות מייה הוא
 - יכול להריץ אותו
 - לעזר את פועלתו
 - כולל לעזר את פועלתו באופן מתוכנו בפקודה מסוימת
 - לשלוט על הזיכרון שלו
 - לקרוא ולשנות את תוכן הזיכרון שלו
 - כולל טבלת הדפים שלו
 - לנוהל את הטיפול בחיריגות של התהליך המדובג
 - לדעת את מצבו ומצב כל הרגיסטרים בכל עזירה
 - ולשנות אותם לצורך המשך הריצה
- **המעבד ומיה מתוכננים לתמוך בכך**



60



מַאיִכֶת גָּאָזְקָן גְּזִיאָה

- דיבגר משתמש בשירותי המעבד על מנת לדבג תהליכי
- לשם כך יש
 - פקודות מכונה ופסיקת תוכנה מיוחדות
 - int 3
 - מבחר exceptions עבור breakpoints
 - רегистרים מיוחדים
 - צורות הפעלה ייחודיות
 - למשל step single
 - ו שימוש בתכונות אחרות הנתמכות ע"י המעבד
 - למשל זיכרון וירטואלי



תאיון א"ה גזיין

- דיבגר משתמש בשירותי מערכת הפעלה על מנת לדבג תהליך
- מ"ה מתאפיינת שונה לתחליך מדובג, למשל
 - מדוחת ל-Debugger על אירועים שונים (events) שRELONENTIALLYם לתהליכי המדובג
 - בעת חריגה, מ"ה מעבירה את הטיפול ל-Debugger, ועוצרת את פעולה התוכנית
 - אם התוכנית לא מסוגלת לטפל בחריגה, מ"ה תיתן הזדמנויות שנייה ל-Debugger לפני הקרטת התוכנית
- כאשר מדברים תהליכי, הוא עשוי להתנהג בצורה שונה
- שיטות Anti-Debugging מנצלות זאת



אקה הַמְוִימָה שֶׁיְקַפֵּה

- **מבנה הנתונים שמייה שולחת לתהlixir המdbg :**

```
typedef struct _DEBUG_EVENT {
    DWORD dwDebugEventCode; // מציין מה סוג האירוע בಗלו נוצרה 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;
```



ןינכז באנף זיין?

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



ןְּקָדִישׁ וְּנַחֲנָה בְּשֵׁם

// Example based on MSDN

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

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



הנפקה קיינט

```
// 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)  
        {
```



הנפקה קיינסית

```
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;
```



הנפקה ב-*Windows*

```
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 = OnLoadDIIIDebugEvent(DebugEv);  
    break;
```



הנפקה קיינית

```
case UNLOAD_DLL_DEBUG_EVENT:  
    dwContinueStatus = OnUnloadDIIDebugEvent(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);  
}  
}
```



הארכת המריצה בקיעות

כיצד נאככת הטעות

```
if (PreviousMode == KernelMode) { ... }
```

(from reactos.org)

```
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();
```

KiDispatchException

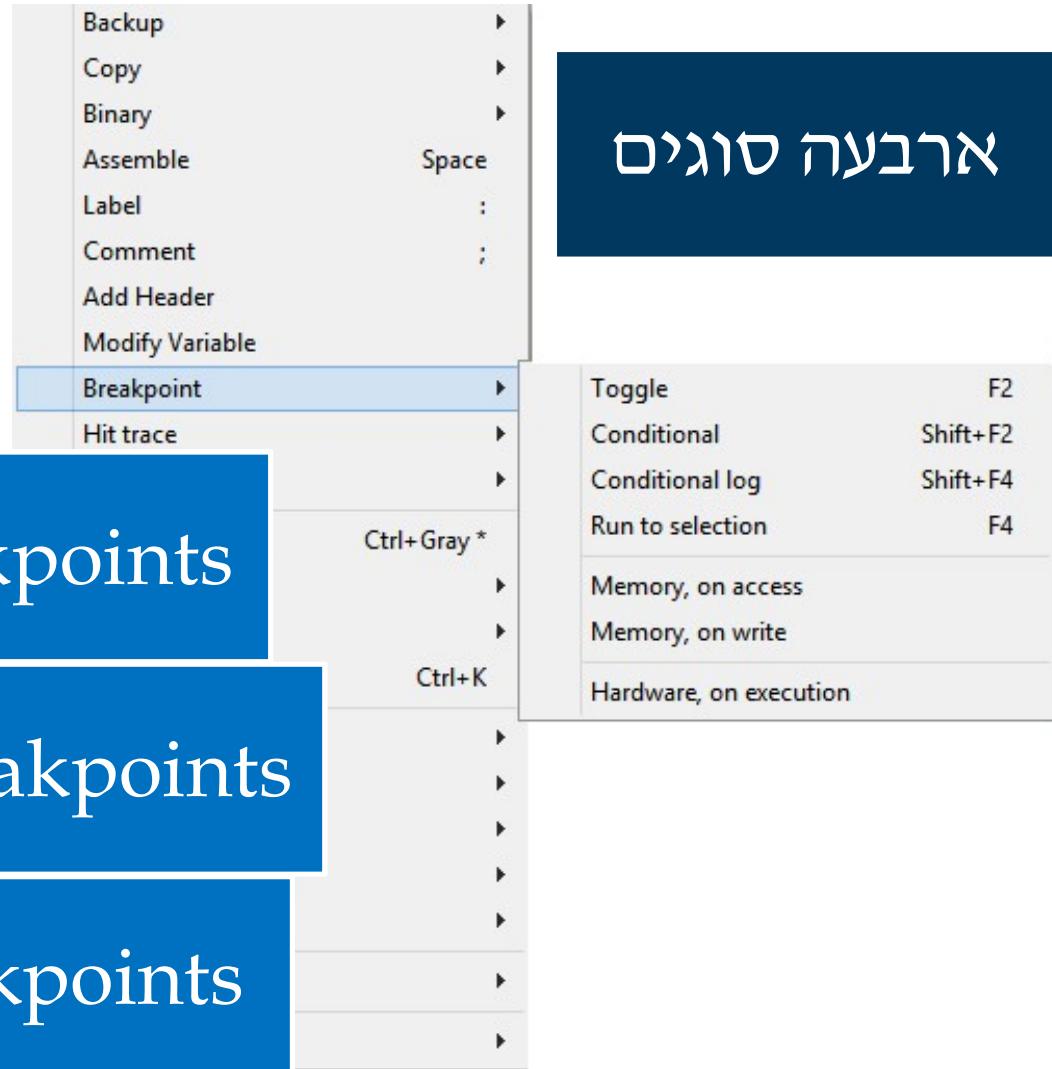
הוותקן ביצועים

Single step

Memory breakpoints

Hardware breakpoints

Software breakpoints



סינון

עיצוב בharצת קוד	עיצוב בכתיבה לזיכרון	עיצוב בקריאה مزיכרון	סוג
<input checked="" type="checkbox"/>			Single Step
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Memory BPT
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Hardware BPT
<input checked="" type="checkbox"/>			Software BPT

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



Single Step

- המעבד תומך בהפעלת פקודת מכונה בודדת בתהיליך מדובג
 - על פי בקשת הדיבגר
- התהיליך מופעל לפקודת מכונה אחת, ועוצר
- התהיליך המדובג חוזר לפעולה ומחליט מה לעשות הלאה
 - הוא גם מקבל מידע על המצב של התהיליך המדובג
 - למשל אם הייתה חריגה
 - וכמוון ערכי הרегистרים, וכו'
- **שימושי לצורן**
 - הרצת פקודה בודדת (פקודת Step של הדיבגר)
 - ולשליטה על הפעולה של שירותים דיבגר אחרים
 - בהם הדיבגר צריך לשנות את סביבת ההרצה לפקודת מכונה אחת
 - למשל BPT-Memory 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 הרשות הקריאה/כתיבה/ריצה של דף הזיכרון
 - באמצעות שינוי בטלת הדפים של התהיליך
 - גורם ל-PAGE_GUARD_EXCEPTION בזמן ריצה
- ה-breakpoint תקף לכל דף הזיכרון
 - כלומר ל-4096 בתים (תלוי בהגדרות כموון)
 - הדיבגר צריך לבדוק לאן פקודת המכונה ניגשה בדף הזיכרון
 - ואם זה לא לאחד הבטים אחוריים עוקבים – אז הוא מתעלם
 - לצורך זה הוא מפרש בעצמו את פקודת המכונה בה קرتה החריגה
 - ולהבין מדוע קرتה ובאיזה נישה לזכרו
 - ✓ מיקום קוד התכנית, קריאה או כתיבה לזכרו, וכו'
- יתרון : אין הגבלה על מספר ה-breakpoints



Memory Breakpoints

- לאחר העצירה, לא ניתן להמשיך ריצה בלי תיקון הרשאות בטבלת הדפים
 - לשם כך מופעל הנוהל הבא
 - תיקון הרשאות הדף
 - הרצת התכנית ב-single Step
 - כולם הפעלת פקודה מכונה בודדת
 - זה נטמא בחומרה של המעבד (TF flag)
 - תמיכת המעבד הכרחית! אחרת אי אפשר להבטיח ריצת כל פקודה עם עצירה אחרת
 - יתכן שהתכנית תיעצר שוב באותו פקודה 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 עיצרנו בגלל גישה לבית בזיכרון באותו הדף
 - שאינו דורש מעקב
 - אבל, בלי שהדיבגר מציג את העצירה זו למשתמש



Hardware Breakpoints

- ניתן להגדיר מספר מוגבל של בתים בזיכרון שיגרמו לחריגה אם
 - היה ניסיון לקרוא אותם או לכתוב אליהם
 - היה ניסיון להריץ קוד שכותב 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 לתחילה המדובג
 - וגם ייעילה יותר מהאחרות

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



76



Software Breakpoints

- מיוושמים על ידי שינוי בקוד התכנית
 - שטיילת פקודת המכונה `int` במקומות בו רוצים לעצור
 - זהה ל-BPT ב-PDP
- כמשמעותם לנקודת ה-breakpoint, פקודת `int` מkapיצה חריגה שמועברת ל-**Debugger**
 - (בහנחה שהתהליך מדובג)
 - כדי המשיך את הריצעה, יש צורך
 - לשחזר את פקודת המכונה המקורי שהיא הייתה שם להריצ' ב-single step
 - ואו להחזיר את פקודת `int`
 - כדי שבפעם הבאה שנגיע שוב תהיה עצירה



77



Software Breakpoints

- למה יש ל-`int 3` אופקود CC בן בית אחד?
 - ועוד ניתן היה להשתמש באופקוד CD 03?
 - כלומר איזה בעיות גורם אופקוד בן שני בתים?
- הדיבגר ממשיך להציג "נכון" את התרגום לאסמבלי
 - למראות 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



78



Software Breakpoints-פְּקַדִּים

```
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-פְּקָדִים

```
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-פְּקַדְיָה

```
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-פְּקָדִים

```
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.



אסכראם Software Breakpoints

- מתערבים בקוד של תהליך
- לא יעבד במקרה שתהליכי לגיטימי ישנה את הקוד
 - נDIR, אבל קורה למשל ב-.NET.
- תהליכי פחות לגיטימיים יכולים לסרוק את קוד התוכנה
ולאטר את ה-breakpoint
 - לכן עדיף להשתמש ב-Hardware BP בניתוח נזקот ותוכנות
חשודות אחרות
- בעת ניתוח קוד שמשנה את עצמו, פקודת ה-`int i = 3` עלולה
להימחק...
 - למשל כאשר וירוס דחוס פותח את עצמו ומשנה את המקום בו
עשינו breakpoint
 - אין בעיה כזו ב-hardware breakpoints – הם יעצרו כשהגיעו
למקום זהה בקוד החדש



האם?



84

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020



פרק 3

התקינה



85

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020



אָהָה גַּוְקִירְכְּ?

"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)



סיביוק' הakinj

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



סינון הפקה

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



88



סיאמי הוקינג

- עברנו הוקינג חשוב משתי סיבות
 - האחת, נרצה להשתמש בהוקינג בזמן ניתוח קוד דינמי
 - להציג ערכי ביוניים וניתור
 - השנייה, לזהות הוקינג בקוד שאנו בוחנים
 - למשל כאשר נזקוט ביצעו הוקינג לצרכיהם
 - כמובן גם לבטל הוקינג כזה



89



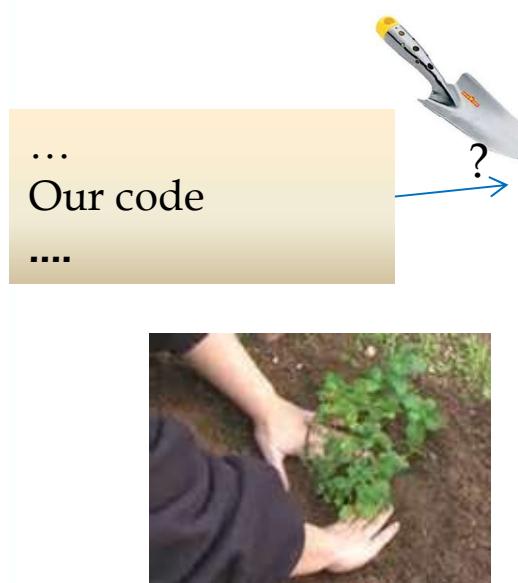
6. כרייקום האקיינט

- ניתן לחלק לשתי שיטות
 - שינוי תוכנה בזמן ריצה – מתייחס לשינוי התנагות התוכנה כאשר היא רצה. כלומר שינוי חלק מסוים בזיכרון
 - שינוי הקובץ עצמו – שינוי הקוד בקובץ עצמוטרם הריצה
 - לעיתים נשלב את שתיהן
 - השינוי בקובץ יפעל שינוי נוסף בזמן ריצה
- דרישות
 - לצורך שינוי קובץ נדרש
 - פונקציות בסיסיות לקריאה/כתיבה לקובץ
 - לצורך שינוי בזמן ריצה נדרש
 - פונקציות לקריאה וכתיבה מתהילך אחר
- נתחיל מהוקינג עיי' שינוי תוכנה בזמן ריצה
 - כשהיא כבר טעונה לזכרו
 - ונניח לרגע שיש ביכולתנו לעשות זאת, בלי הפרט איך



ה乐观ט קואץ

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



someFunction:

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



91



התקפת קוז

- לצורך הזרקת קוד ניתן להשתמש ב-jmp בתחלת 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
...
...
```



```
E9xxxxxxxxx  JMP OUR_CODE
83E4 F8       AND ESP,FFFFFFF8
6A 01         PUSH 1
....          ...
...
```



התקפת קוז

- לצורך הזרקה נשתמש ב-kmpj בתחילת הפונקציה
 - שטקפוֹץ לקוד שלנו במקום פנוּי אחר
 - הפקודה kmpj לוקחת 5 בתים
- אם הפקודה לא מתאימה לקוד מבחינת הגודל, נשתמש ב-dop לריפוד (לא חובה)

someFunction2:

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

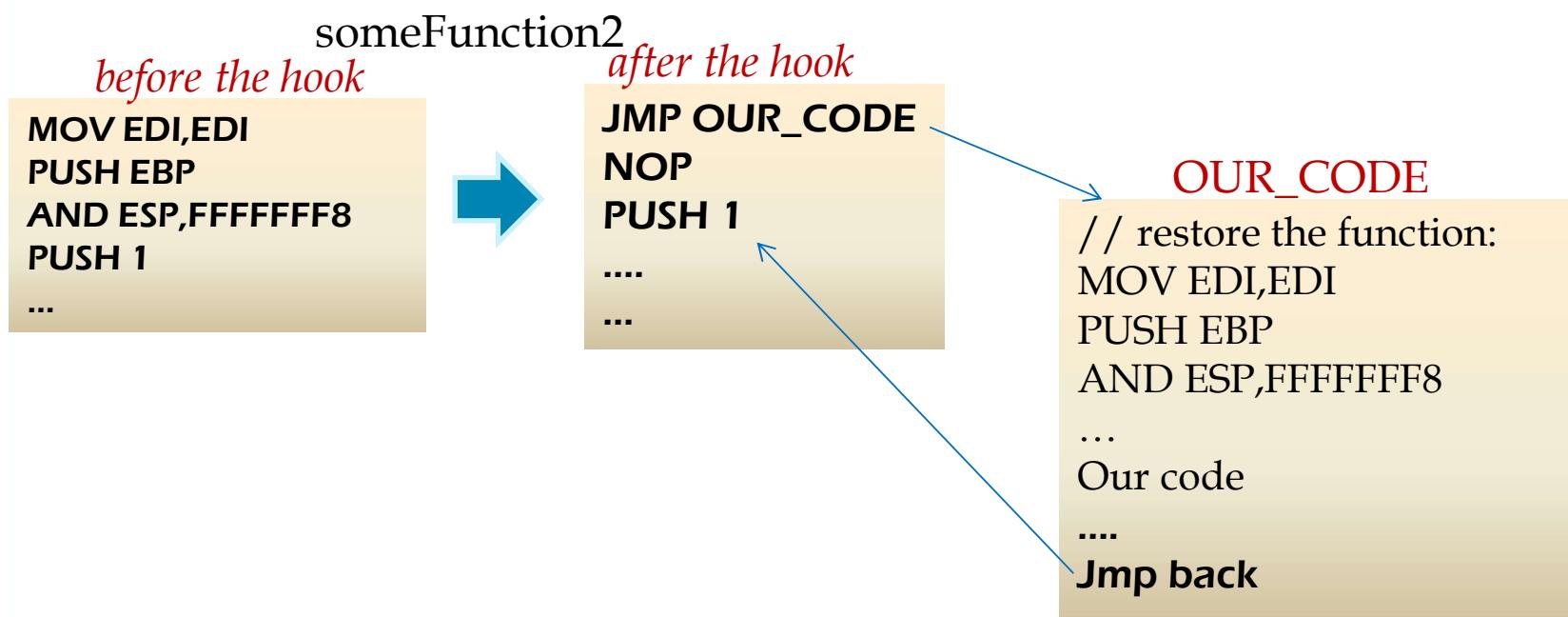


```
E9xxxxxxxxx  JMP OUR_CODE
90          NOP
6A 01        PUSH 1
...
...
```



התקפת קוז

- יש צורך לזכור את הקוד שנמחק ע"י ה-jmp
 - כדי להחזירם למקום אם נרצה אי פעם
 - וכן להריץ את הפקודות שהיו שם, לפני הריצת הקוד שלנו
- כמובן לкопואז בחזרה לפקודת המכוונה הבאה בסוף הקוד שלנו



ה乐观ית קוא

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

$xxxxxxx = \text{OUR_CODE} - (\text{someFunction2} + 5);$

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
```

שימוש לב:

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



התקפת אולצת HOT Patching

- hotpatching תומך בהידור עם הדגל Visual Studio •
 - חלק גדול מה-DLL-ים קומפלו בשיטה זו
 - במצב זה הפונקציות "מכננות" ל-hooking
- במצב Hot patching
 - כל פונקציה מתחילה ב-edi, edi, edi
 - בעצם kop שניtinן לדפוס jmp short
 - גודל של jmp
 - לפניהם יהיו בדיקות חמישת kop'ים
 - גודל של jmp
 - לא מבוצעים ברייצה רגילה של הפונקציה



התקה אולמית HOT Patching

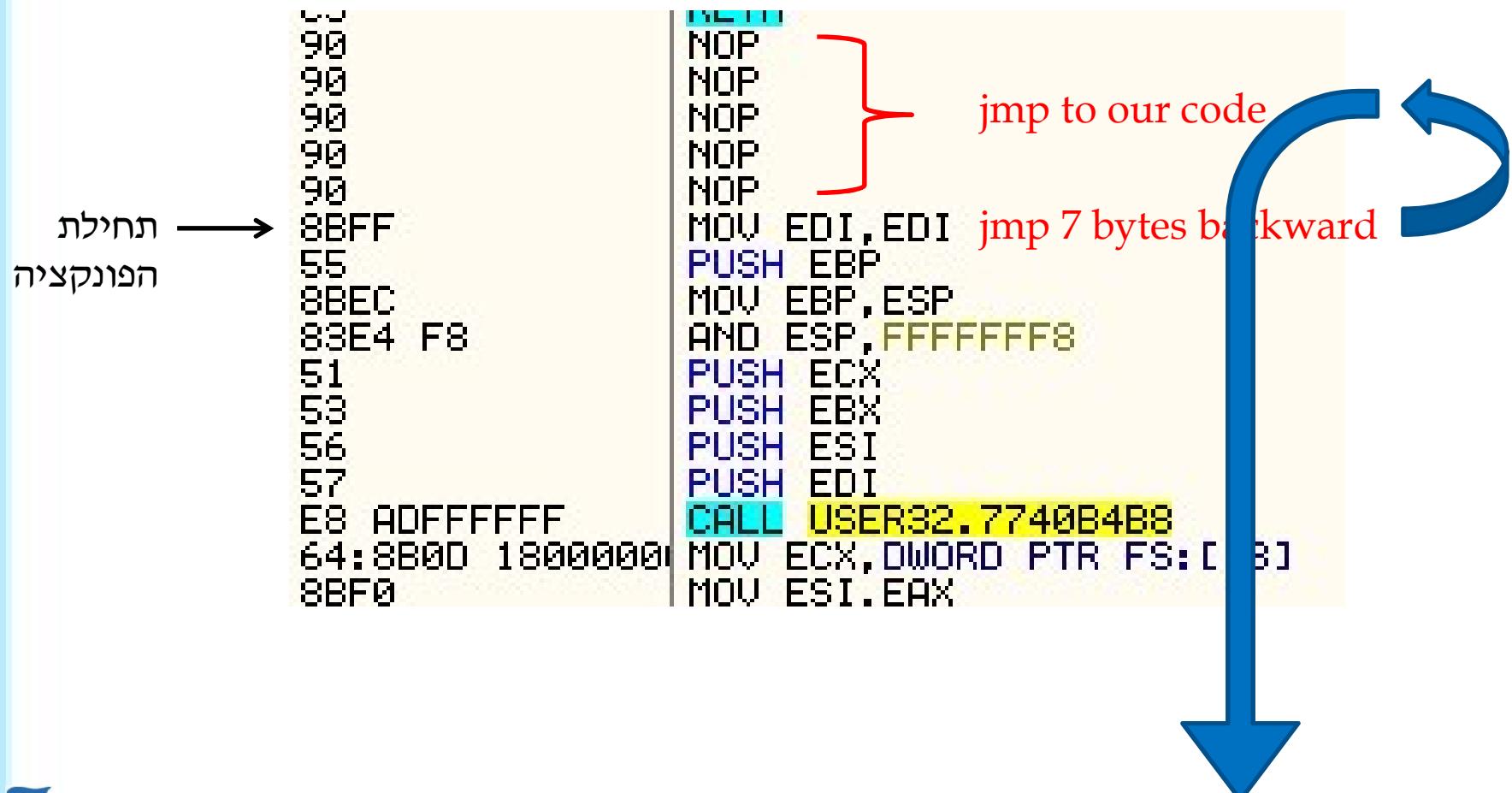
- לצורך הזרקת קוד נחליף את שבעת הבטים הללו בקפיצות

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 אותו לקוד שלנו
- לא דרשו פקודות חשובות – אין צורך לשחרר
- למה שני jmp כשהאפשר היה אחד?
 - כי בריצה רגילה, ללא הוקינג, ביצוע mov edi, edi ייעיל יותר מלהפעיל 5 כסח-ים בתחילת כל פונקציה



התקפה ה-**HOT Patching**



התקנה ה-dll מוצב ב-IAT

- בזמן ריצה, ה-IAT מציביע פונקציות ב-DLLים השונים
 - כל קריאה לפונקציה של DLL עוברת דרך ה-IAT
 - שינוי ה-IAT, כך שיציביע קוד שלנו במקום פונקציה ב-DLL, יפנה את הקריאה לפונקציה הקוד שלנו
 - הקוד שלנו יקרא הקוד המקורי, אם יש צורך בכך
 - כאשר הקוד שלנו נטען כ-DLL, אין צורך לזכור את הכתובת המקורי
 - נקרא לפונקציה ישירות דרך ה-IAT של ה-DLL שלנו
 - כך שהשינוי בזמן ריצה יהיה רק שינוי המציביע ב-IAT
- יתרון נוסף: ה-IAT נשמר באיזור זיכרון שאינו מוגן מכתיבת



99



אינז'ין קודץ הכללה

- בשקפים הקודמים עשוינו בשינוי הקוד בזמן ריצה
 - כלומר שינוי מקומי בכל פונקציה ופונקציה
 - אפשר גם לשנות פונקציות מ-DLL-ים
- לעיתים נעדיף לשנות את הקוד ישירות בקובץ הרצה
 - אפשר לבצע הוקינג באופן דומה למילה שהציגנו בזמן ריצה
 - אבל לא לשנות קוד בספריות נתונות דינמית
 - טכניקות מורכבות יותר מאשר לבצע שינויים גדולים בקוד הנמצא בקובץ הרצה
 - למשל post-link optimizations



מַאֲכָה מִתְאַכֵּל

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



101



סיכום

- דנו איך לשנות קוד על מנת להכניס פונקציה משלנו
 - כשקוראים לפונקציה אחרת, למשל פונקציית מערכת לא הצנו כיצד לבצע זאת באופן מעשי, נדגים בהמשך
- יש מגוון דרכיים שבהם ניתן לעשות זאת
 - הצנו דוגמאות פשוטות
 - ישנו אנטי-וירוסים שמשהים שינויי הוקינג
 - לכן וירוסים פיתחו שיטות "יצירתיות", ומנגנונים שטרם נבדקים ע"י אנטי-וירוסים
- הצעו דרך לבצע הוקינג שלא ניתנת לזיהוי פשוט
- הצעו דרך לבצע הוקינג בזמן ריצה
 - בלי לשנות בכלל את קובץ ההרצה



מִכְאָכָל penne ein'e f haNei



103

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020



אֲהָ לְכָ ?API

Application Programming Interface

- מתייחס לקבוצת פונקציות מוכנות שמסופקות לתוכנת
באופן מובנה
 - בדרך כלל ע"י מערכת הפעלה
 - בחלונות זה נקרא WinApi
 - גם שירות ענן מספקים API לאפליקציות, למשל פייסבוק
- לדוגמה, פונקציות לטיפול בקבצים
 - מיצאות ע"י kernel32.dll
 - שבגרסאות חדשות של חלונות מפנה ל-dll kernelbase.dll
 - CreateFile, ReadFile, WriteFile
 - fopen, fread, fwrite
- בرمת המהדר, המשמשת בפונקציות לעיל



גָזֹאת מְכוּרָאת

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

הודעות =

אירועים של מ"ה שמדווחים לתהlixir

וועוד

אירועי עבר

הקלדות במקלדת



הוֹצָאת כְּחִילָה

```
WNDCLASS wc;  
...  
wc.lpfnWndProc = (WNDPROC) WndProc; → 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  
...  
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);  
    }  
}
```

Message Loop



קונסטנסים מוגדרים

למשל הודעות על שימוש בעבר

```
...
WM_MOUSEFIRST = 0x200
WM_MOUSEMOVE = 0x200
WM_LBUTTONDOWN = 0x201
WM_LBUTTONUP = 0x202
WM_LBUTTONDOWNDBLCLK = 0x203
WM_RBUTTONDOWN = 0x204
WM_RBUTTONUP = 0x205
WM_RBUTTONDOWNDBLCLK = 0x206
WM_MBUTTONDOWN = 0x207
WM_MBUTTONUP = 0x208
WM_MBUTTONDOWNDBLCLK = 0x209
WM_MOUSEWHEEL = 0x20A
WM_MOUSEHWHEEL = 0x20E
...
...
```



פרק 3'ית SetWindowsHookEx

קצת יותר ממה אם כן

- חלונות מאפשרת לבצע Hook בתהליכיים אחרים
 - באמצעות הפונקציה SetWindowsHookEx
- הפונקציה מאפשרת לנטר הודעות בתוכנית
 - כך שההודעה הגיע לקוד שלך בנוסף לקוד הרגיל של התוכנית
 - תומך במגוון סוגי הודעות, חלקן אפשר לפני העברה לתהליך או אחריו
- ניתן גם להזיר DLL באמצעות הפונקציה
 - על הזירות נדון בהרחבה בהמשך
- הפונקציה יכולה לבצע Hook בתוכנית מסוימת או בכל התוכניות
- הפונקציה היא מעתפת לקוד שכתב לזכור של תהליך אחר
- כל המידע נמצא ב-MSDN



Key Logger- f kndz

```
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-סנדי?

```
HRESULT 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);  
}
```



סכימת קייראט – DLL

- ספריות סטטיות משולבות בזמןリンク לתוך קובץ ההרצה
 - הקוד שקורא להן יודע את כתובתן היחסית כפי שהוא יודע לכל פונקציה שכותבה ישירות בקוד
- ספריות דינמיות נתענות ע"י ה-loader
 - וرك אז מקשרות הפונקציות בספריה למקומות בהן קוראים להן
 - ה-loader עשוי לטוען אותן לכותבת אקראית
 - ע"י ASLR
- בלינוקס הן עם סיומת OS
 - למשל
 - /usr/lib64/libc.so ◦
 - /usr/lib64/libcrypto.so ◦
 - /usr/lib64/libX11.so ◦



איך מוגדרת סכירתם ? יייאוות?

"copy-on-write"

- כדי לחסוך בזיכרון סביר שטלאות הדפים של כל התהליכים מפנות ספרייה דינמית לאותו שטח בזיכרון הפיסי
 - כך נשמר רק עותק אחד של הספרייה
- ומה קורה כשהליד רוצה לכתוב בדף זה?
 - ???



הוקיינט אלגוריתם *kNCF*



113

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020



קונסולט הפקים

- מתכנת כתוב 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
- בראותנו הקובץ הבינארי בלבד
 - בלי אפשרות להציג את קוד מקור
- איך נפתרת הבעיה?



היקף, "אם

- 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`



איך רעה?

- kernel32.dll
 - נשמע לא רלוונטי
- user32.dll
 - נשמע המקום הנכון
 - אבל ישנה גם **בתכניות אחרות!**
 - המקום הנכון כשרוצים לעקוב אחריו כל התהליכים
- program.exe
 - מסובך. ואין לנו קוד מקור
 - אבל זה הקובץ היחיד שnitן לשנות ללא הרשות מיוחדות
 - ודרך נוכל לשנות את השאר בזמן ריצה!
 - ללא צורך בשינוי קבצי DLL
 - קלומר ללא השפעה על שאר התהליכים במחשב



איך רעגה?

... Program.exe

User32.dll

Kernel

Call MessageBox

Call MessageBox

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

האם כדאי לשנות את הקוד ב Program.exe ?
מה לגבי User32.dll ?
או kernel32.dll ?
שימוש לב:
User32.dll ממופת ע"י ה Loader לזכור של Program.exe. לכן בזמן ריצה ניתן להתייחס
ל-User32.dll שנטען לזכור קוד בלעדי של Program – כל שינוי יהיה בלעדי לתהליך זהה.
לעומת זאת שינוי הקובץ (User32.dll) עצמו, ישפיע על כל התהליכים במערכת.



איך מEARת?

- אפשרות נוספת
 - מטהליק אחר שאנו מתכוונים בעצמנו
 - שינוי את הפעולה (או תוכן הזיכרון) של התהליק שרצינו
 - תוך שימוש בשירותי מערכת הפעלה שתומכים בכך
 - יתרונות
 - לא צריך לשנות שום קובץ!
 - כן צריך הרשות מתאמיות
 - וכי אפשר לזהות דבר מניתוך קבצי התכנית וה-DLLים
 - כי השינוי לא נעשה דרכם



האקרים הבודקים: איך מתואכרים

- תחיליה נדגים כיצד לשנות את MessageBox כאשר יש לנו גישה לקוד מקור (של Program)
- התהlixir מתמקד בשינוי הפקציה ב-user32.dll



המי הרכסמת MessageBox

7513C0D2	.text	Export	MessageBoxBeep
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 Export table של user32.dll

- מה ההבדל בין MessageBoxA ל-MessageBoxW
 - הראשון נועד לתווים ASCII והשני לתווים Unicode
 - ממיר את הקלט ל-Unicode, ואז קורא בתורה
 - כלומר W MessageBox ייקרא בסוף בכל מקרה
 - מומלץ לעבור על הקוד של הפונקציה ולאיתר את ההמרה



האקיינט MessageBoxW

- הפונקציה מוכנה להוקינט עם

השורה	הכתובת	הפקודה
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
75185F00	9A	NOP



האקייז MessgeBoxW

- את ה- mov edi, edi נושא לкопיצה 7 בתים אחורנית
 - כלומר ל- EB F9 (הערך 7 מוצג עי' F9)
- שם נקבע לפונקציה שלנו עי' שינוי התוכן של חמישת פקודות ה-NOP ל-jmp
- לא נוכל לדעת את כתובות הקפיצה ממש מראש

השורה	הכתובת	הפקודה	ההצגה	ההצגה
75185F87	90	NOP		
75185F88	90	NOP		
75185F89	90	NOP		
75185F8A	90	NOP		
75185F8B	90	NOP		
75185F8C	8BFF	MOV EDI,EDI	EB F9	JMP SHORT USER32.75185F87
75185F8E	55	PUSH EBP	55	PUSH EBP
75185F8F	8BEC	MOV EBP,ESP	8BEC	MOV EBP,ESP
75185F91	6A FF	PUSH -1	6A FF	PUSH -1
75185F93	6A 00	PUSH 0	6A 00	PUSH 0
75185F95	FF75 14	PUSH DWORD PTR SS:[EBP FF75 14]		PUSH DWORD PTR SS:[EBP+14]
75185F98	FF75 10	PUSH DWORD PTR SS:[EBP FF75 10]		PUSH DWORD PTR SS:[EBP+10]
75185F9B	FF75 0C	PUSH DWORD PTR SS:[EBP FF75 0C]		PUSH DWORD PTR SS:[EBP+C]
75185F9E	FF75 08	PUSH DWORD PTR SS:[EBP FF75 08]		PUSH DWORD PTR SS:[EBP+8]
75185FA1	E8 51000000	CALL USER32.MessageBox	E8 51000000	CALL USER32.MessageBoxTimeout
75185FA6	5D	POP EBP	5D	POP EBP
75185FA7	C2 1000	RETN 10	C2 1000	RETN 10
75185F00	90	NOP	90	NOP



122



hook-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 הקומpileר (כמו הקצתה frame בתחילת הפונקציה) הfrmter **IpText** נמצא בכתובת **esp+8** • לא ידוע לנו מהן הרשות של **Page** בו נמצא **VirtualProtect**, لكن נשתמש ב **VirtualProtect** לשינוי הרשות • **VirtualProtect** דואג לשחרור המחסנית בסיוםו, וכן אין צורך לעשות **add esp** לניקוי הfrmtersים •IpText מקודדת ב-**Unicode**, לעומת זאת לכל תוו בסוף הפונקציה יש צורך לחזור ל-**MessageBoxW** (בתוספת שני בתים) • אך כרגע אנחנו לא יודעים את הכתובת • וכך גם במקרה הכללי • וכן נשאיר מקום לפקודת **jmp** אם נדייק, בדוגמא זהו כן ניתן לדעת • רמז : נחשו איך אנחנו יודעים את הכתובת של **VirtualProtect**



MessageboxW ייְהִי מֶלֶךְ כָּתָאת

```
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);
```

הסבר לחלק הראשון:



- שימוש בפונקציות GetModuleHandle ו-GetProcAddress להשגת הכתובת של MessageBoxW בתהליך שלנו
- ניתן לקרוא עליו ב-MSDN
- הנתן מחרוזת מחוץ לjmpopcode (קפיצת חזרה)



MessageboxW עירוי – אפקט מושך

```
void setHook() {
    LPVOID f;
    HMODULE h;
    CHAR JmpOpcode[6] = "\xE9\x90\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
- אפשרים כתיבה לפונקציה ע"י VirtualProtect
- מעדכנים את jmpOpcode
- משנים את MessageBoxW



MessageBoxW 'ije

```
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 d1 lmsg.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
CD	NON

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);



MessageboxW עירוי

hook - גנטה מתקבץ ה-*Hook*

הסבר לחלק השלישי:

```
void setHook() {  
    LPVOID f;  
    HMODULE h;  
    CHAR JmpOpcode[6] = "\xE9\x90\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);
```

```
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);
```

• **VirtualProtect** לפונקציה שלנו

• חישוב "כמה" לקפוץ :

נרצה לקפוץ ל- $f + 2 - \text{offset}$ כאשר
הקפיצה תموוקם בפונקציה שלנו החל
מהבית $0x2e$ (זה מספר שרירוטי, מה
שחשוב זה למצוא מקום עם ס'ופ אחריו
הקוד שלנו)

מכיוון שהמරחק מחושב מהפקודה הבאה

- נסיף 5



Hook-ה זיירק אַלטְהָן פֿעֶן

```
__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
        ...
    }
}
```

```
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 dl\msg.69D1101C
NOP
JMP USER32.75185F8E
NOP
NOP
NOP
NOP
NOP
NOP
```

jmp back to MessageBoxW



Hook-ה מfine

```
INT WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,  
    LPSTR lpCmdLine, int nCmdShow)
```

```
{
```

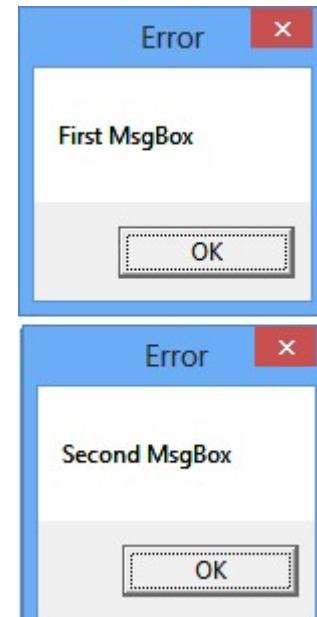
```
    setHook();
```

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

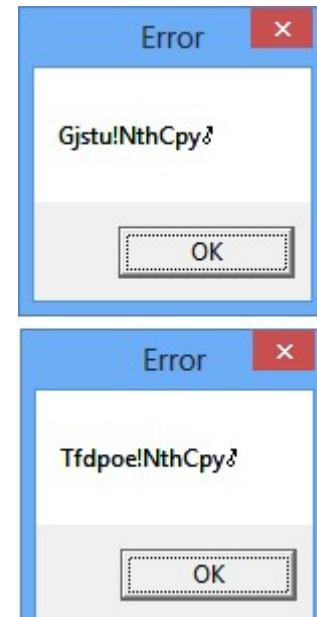
```
    return 0;
```

```
}
```

without Hook:



With Hook:



הטייגר Hook

- ללא קוד מקור, נוכל לשתול את הקוד שלנו בקובץ ההרצה ע"י שינוי כך שיכלול
 - את הפונקציות `setHook` ו-`msgHook`
 - 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"...



130

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמיאל

23.01.2020



המיון ה-Hook-DLL

- נרצה שה Hook שלנו יהיה גנרי, כך שייהיה ניתן להשתמש במודול שלנו בקלות מכל קובץ DLL
- לכן נכתוב את setHook ב-DLL
- מוכנת 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;
}
```



DLL-ה Hook-ה מים

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

- ניתן לקרוא ב-**MSDN** על **DLLs**

- חשוב להכיר – ברגע טעינת **DLL** המערכת קוראת ל-**DllMain**

- **DLL** הוא בפורמט **PE**

- בקוד של המתכנת במקום ל-**LoadLibrary** קוראים ל-**hook()**

```
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;
}
```



{

132



התקפת 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 **זרקת DLL**
 - DLL Injection
- ישנן מספר שיטות לבצע זאת
 - למשל יש משתנה סביבה שמאפשר לטעון DLL בהרצת EXE
 - אנו נתמקד בשיטה הנפוצה המתבססת על `CreateRemoteThread` עם `WriteProcessMemory`
- בשקפים הבאים נראה מימוש דוגמא ל-**DLL Injector**



dll הרכבת

```
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



134

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמיאל

23.01.2020



dll הרכבת

```
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;
}
```



התקפת DLL

חשוב להקים את התהליך עם הרשאות הנכונות.

```
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;
}
```

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



התקמת 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;
}
```

- השתמשנו בפונקציות הניל בשקפים קודמים.
- יש לשים לב שմבקשים את **A** LoadLibrary כלומר פונקציה שמקבלת ASCII כקלט.

remoteLoadLib יחזיק את הכתובת של LoadLibrary בתהליך שלנו.
הכתובת הזאת גמ **לתהליכים אחרים** במחשב
(כגון kernel32.dll **תמיד** נטען **לאוטו** המקום
שנקבע מראש בזמן טעינה מ"ה).



התקפת 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;
}
```

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



התקפת 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
הינו כתבים ? “file.dll”



התקפת 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;
}
```

שימוש ב-`VirtualAllocEx` (הקצת זיכרון בתהיליך מרוחק) ו-`WriteProcessMemory` (כתיבת זיכרון בתהיליך מרוחק). כך נבטיח שהמחוזת "dllmsg.dll" נמצאת ב-`Process` המרוחק.



התקמת DLL

- הצגנו דוגמא לשינוי קוד עייתי הזרקת DLL
 - יש דרכים נוספות לבצע זאת
- הצגנו את הדוגמא על תוכנית ייודית, אך ניתן כМОבן להזrik את ה-DLL לכל תהליך
 - חפשו ב-MSDN כיצד לקבל רשיימה של תהליכיים רצויים
 - רمز : CreateToolhelp32Snapshot



141



הוק בסוףORK3יה

- כל מה שעשינו עד עתה מבצע הוק בתחילת פונקציה
 - ככלומר אפשר לנו להריץ קוד לפני שהפונקציה רצתה
 - כולל אפשרות לשנות את הפרמטרים שהוא מקבלת כקלט
- לעיתים נרצה להחליף את הפונקציה באחרת בלי לקרוא למקורית
 - איך נעשה זאת?
- במקרים אחרים נרצה להריץ קוד בסוף הפונקציה
 - למשל קוד שמשנה את ערך החזורה שלה
 - חישבו על לפחות שלוש דרכים לעשות זאת...
 - חשוב לשים לב לשני דברים:
 - איך לקבל את הבקרה בסוף חישוב הפונקציה?
 - איך נוכל לקרוא לה למרות שתלנו בה הוק?
- במקרים מורכבים יותר נרצה
 - לקרוא לפונקציה המקורית מההוק יותר מפעם אחת



הוק כסואט אורך 3 ית' ג' hot-patching

- במקרה שאין תמיכה ב-hotpatching הפתרון נעשה מסובך יותר
 - מה הביעות הנוספות שנוצרות?
 - איך ניתן לפתור אותן?
-
- רמז 1 : קריאה לפונקציה
 - רמז 2 : ביצוע פקודות המכונה שモחלפות ע"י jmp אחרי פקודת ה-call לפונקציה



התקינה מודולית



אֲהֵה כִּיְהַ גָּרִיז?

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

- אפשר לשנות בקוד המקור
- אפשר לשנות בקובץ הבינארי
- יש פונקציות API המאפשרות hooking במקומות ספציפיים.
- קיימות עוד שיטות לօד hooking on-the-fly



Instrumentation

ההכרה: הרכבת סיג'אלים
להקירט נומרים
מתכתי, הניתן בסעיפים
פנויים



Pin Tutorial
(Robert Cohn, Intel)

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*).

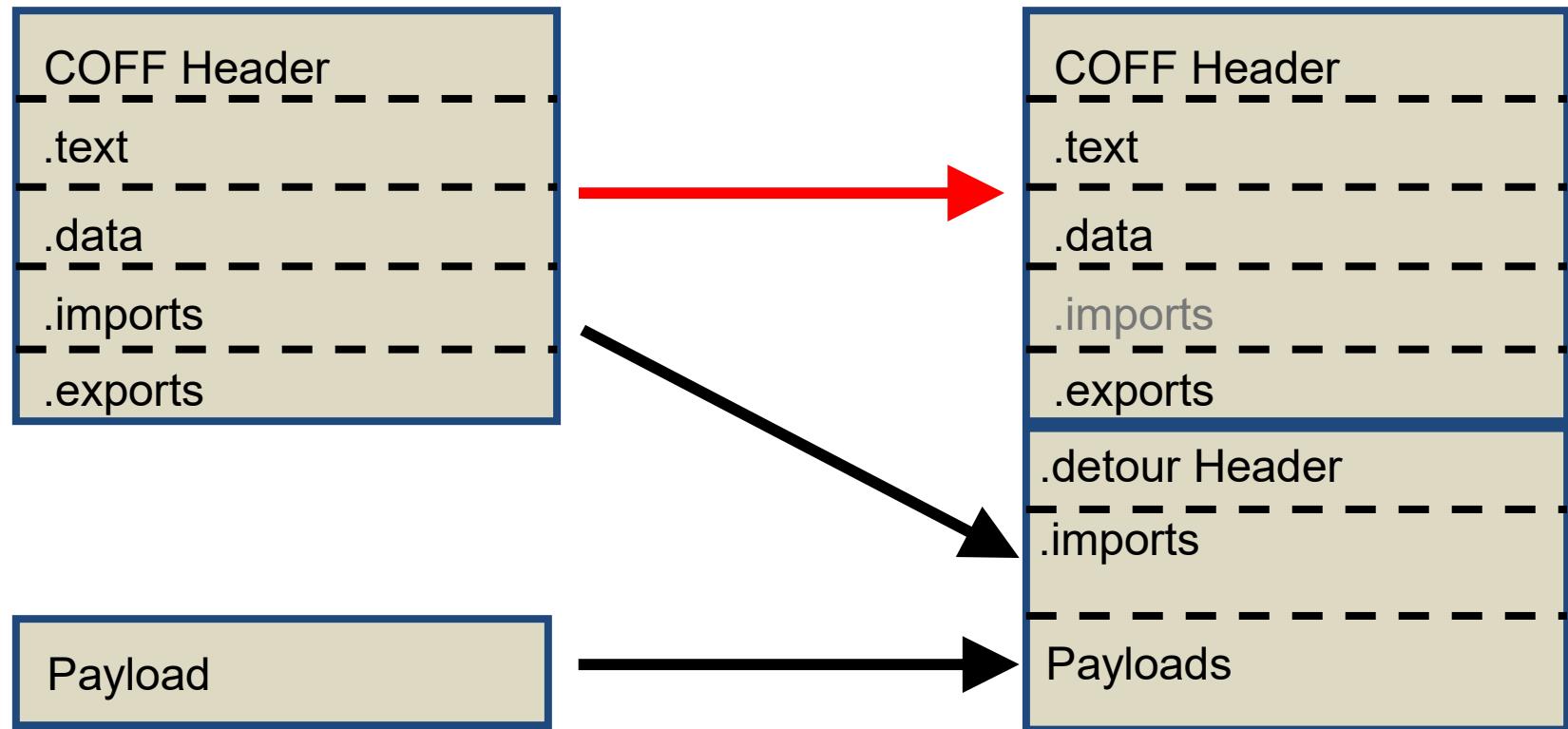


How do you get your code into an application's address space?

- 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          [1 byte]
    .....
; ; Trampoline Function
UntimedSleep:
    jmp    Sleep
; ; Detour Function
TimedSleep:
    .....
```

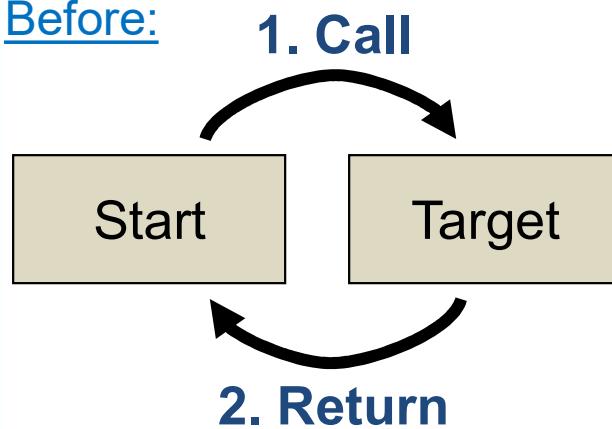
After:

```
; ; 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:
    .....
```

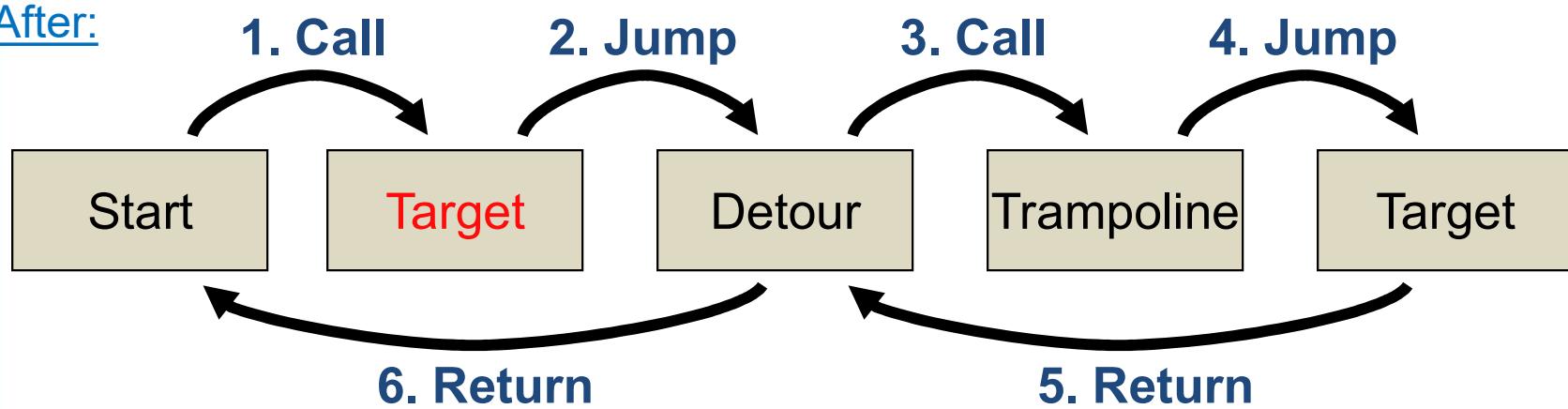


Invoking Your Code:

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

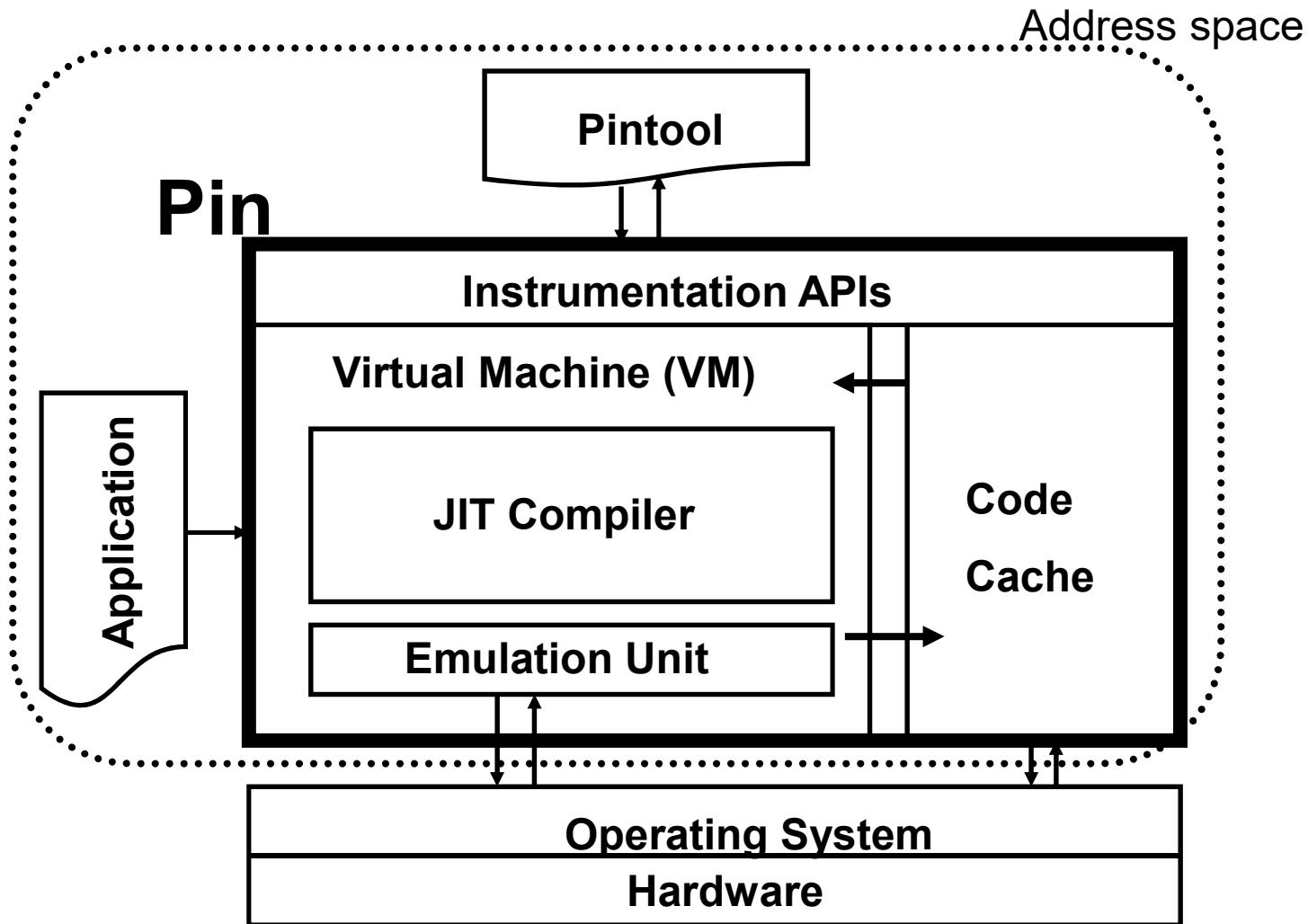
void Instruction(INS ins, void *v) instrumentation routine
{
    INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)docount, IARG_END);
}

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



רכורג'וּטִיט JIT

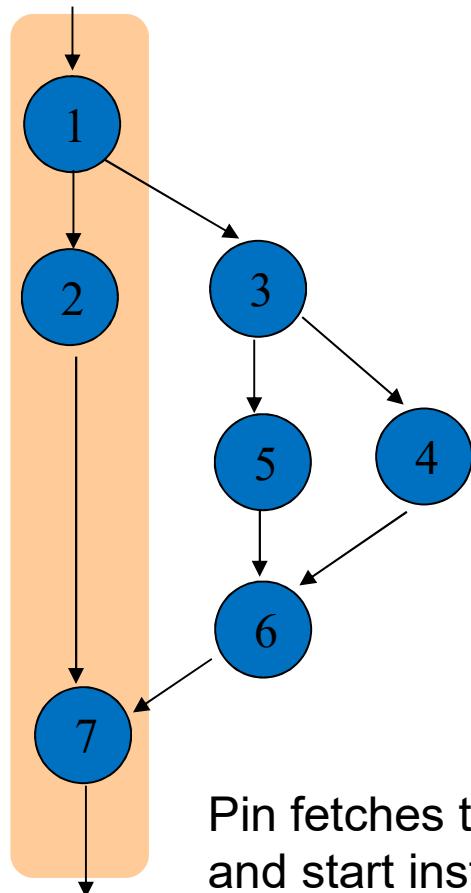
Just In Time compilation

- הידור קוד בזמן ריצה
- יכולה לבצע אינטרפרטציה לקוד, ולבצע הידור רק לקוד בשימוש רב
 - וללמוד מהאינטרפרטציה איך כדאי לבצע אופטימיזציות
- אפשר לוודא שהקוד המהדור אינו מתרכן לגורם נזק
 - קוד הביניים פשוט יותר לבדיקה
 - הקוד המהדור אינו ניתן לטיפול ע"י תוקף
 - ככלומר, נזקה אינה יכולה להשתלת על הקוד...



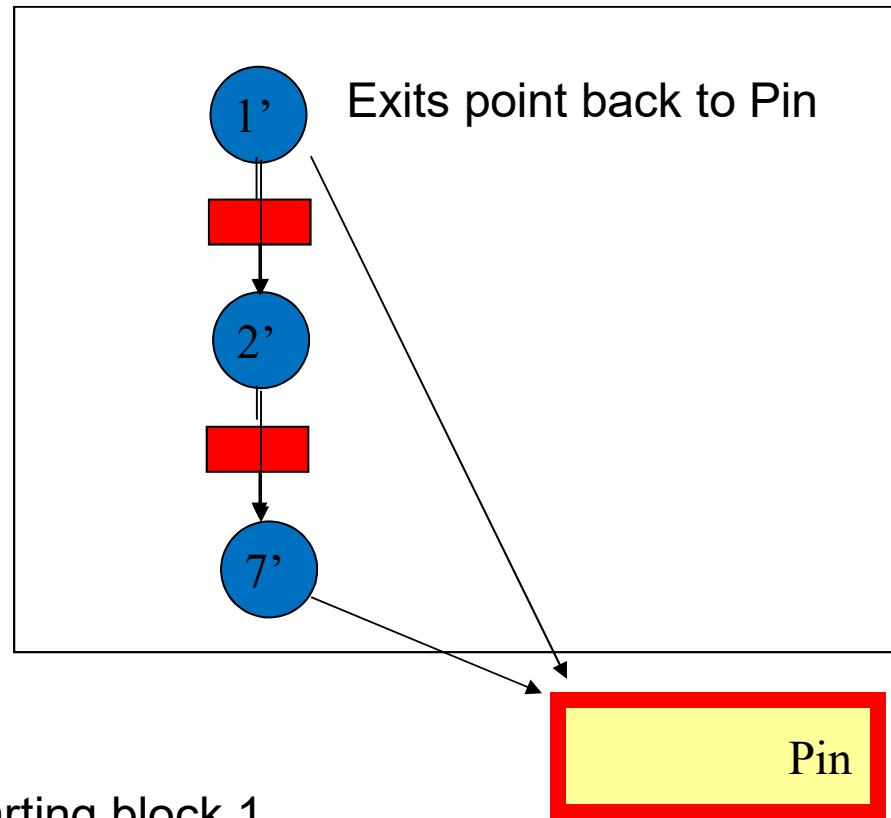
JIT-Mode Instrumentation

Original code



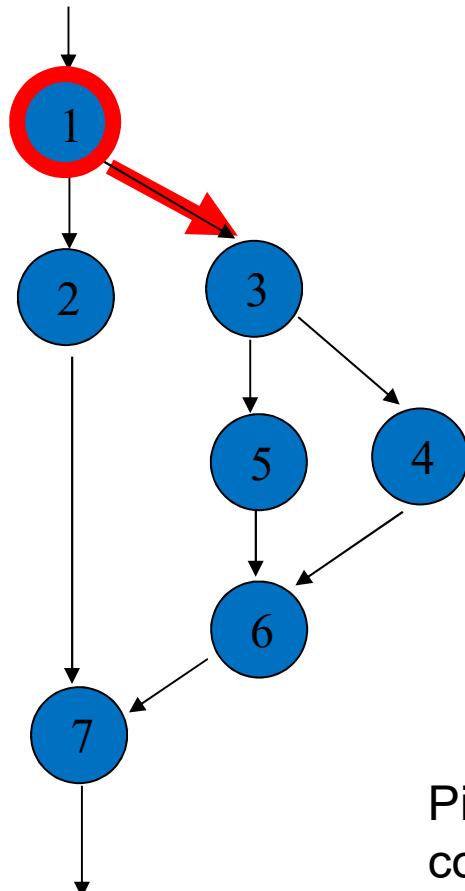
Pin fetches trace starting block 1
and start instrumentation

Code cache

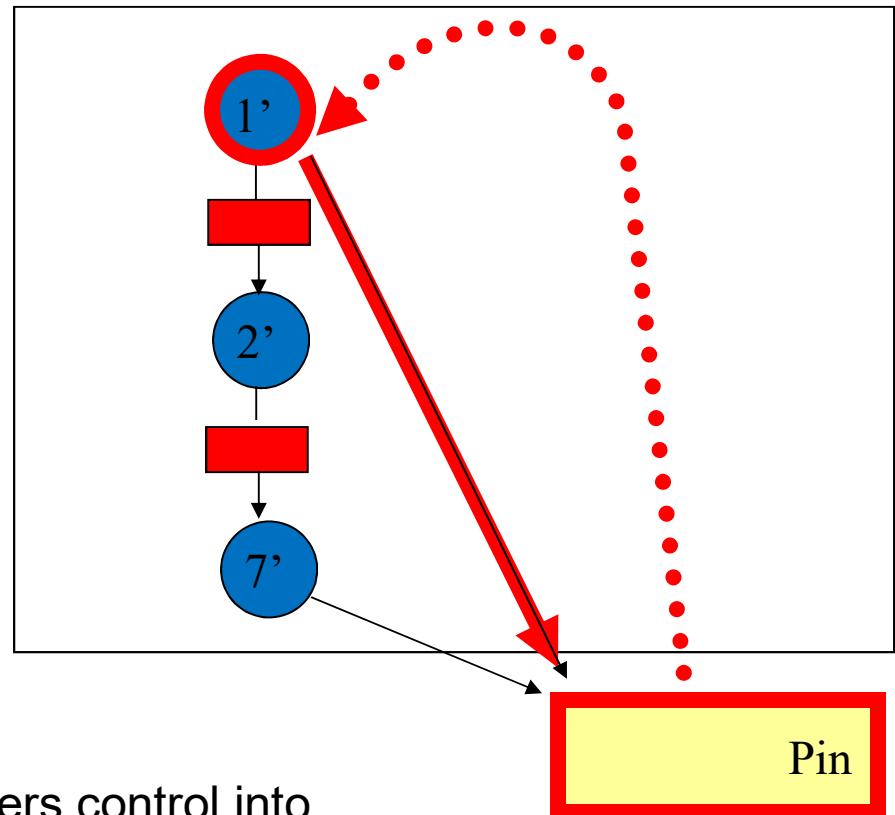


JIT-Mode Instrumentation

Original code



Code cache

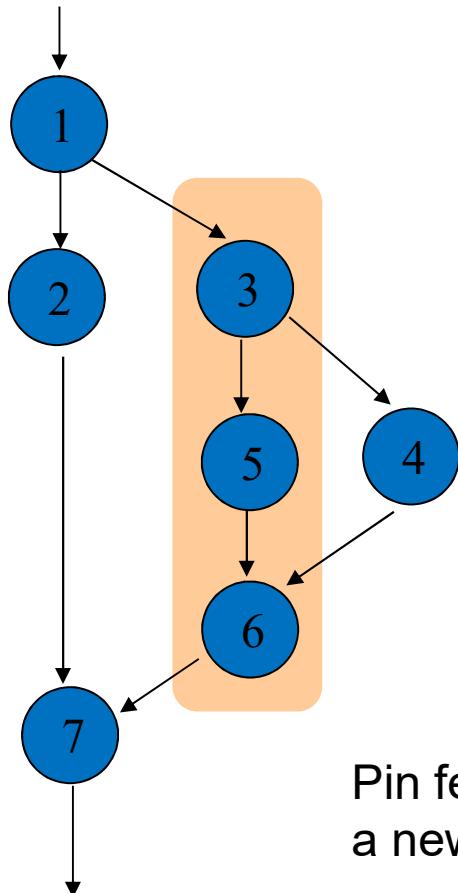


Pin transfers control into
code cache (block 1)

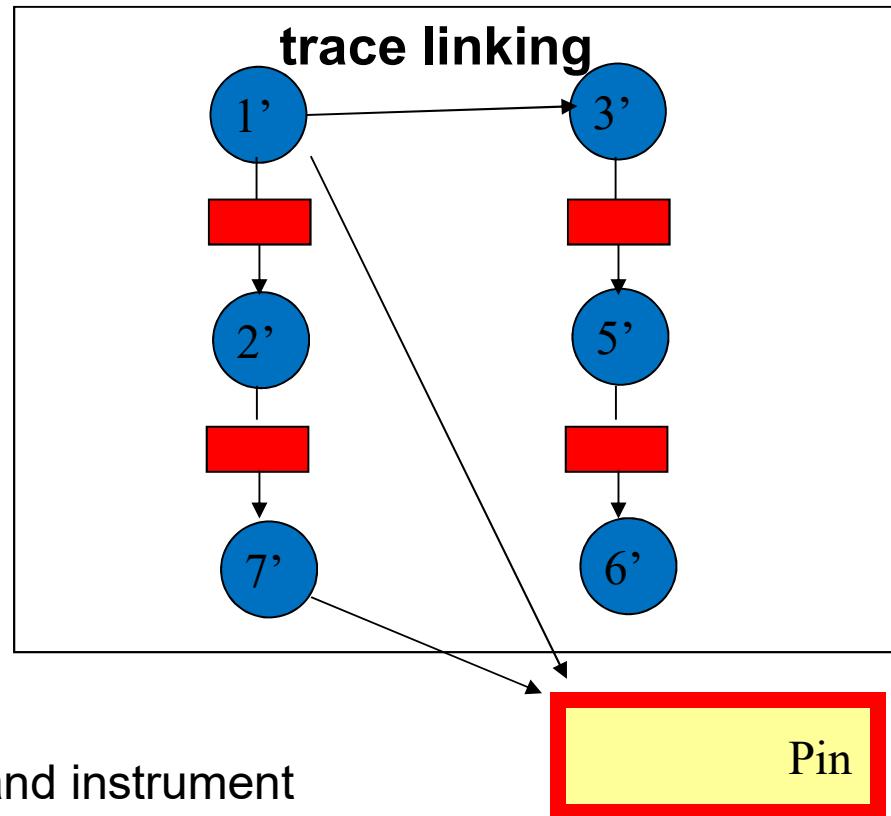


JIT-Mode Instrumentation

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



?האם?



172

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020



הוכחה ליכרין

Buffer Overflow



173

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020



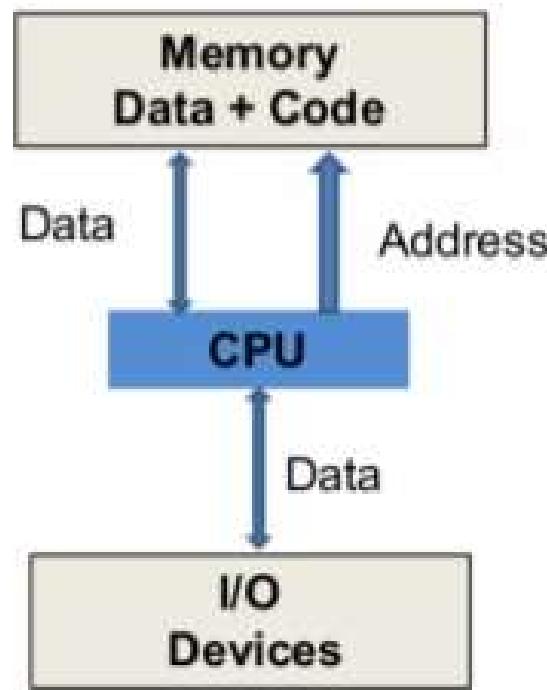
אֲהַתְּקֹפֵן כָּאֶבֶה? מִכְרֹאָזְנָה בְּעַמְּדָקָה

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

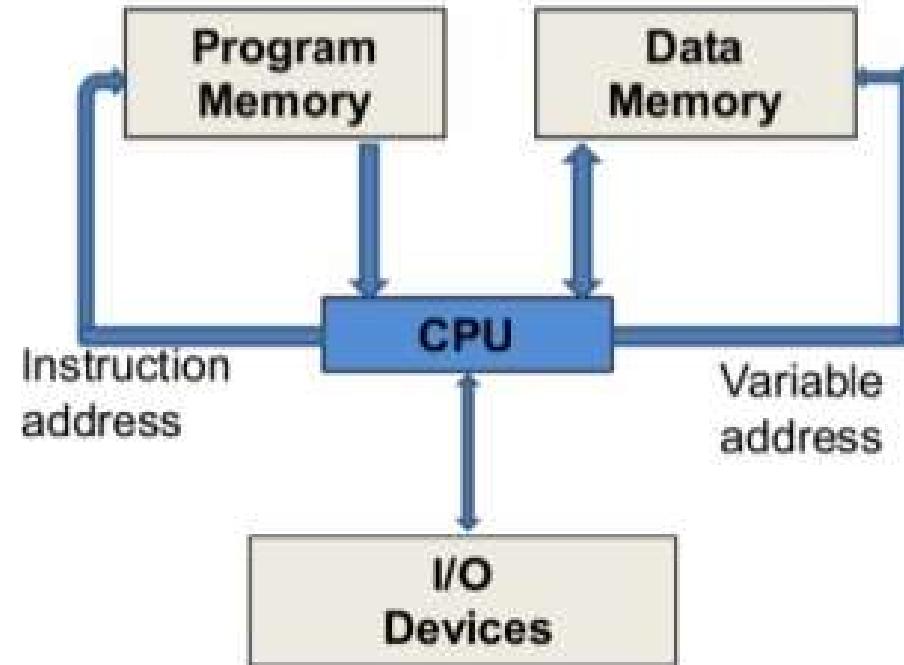


רכזים קומפקטיים

Von Neumann •



Harvard •



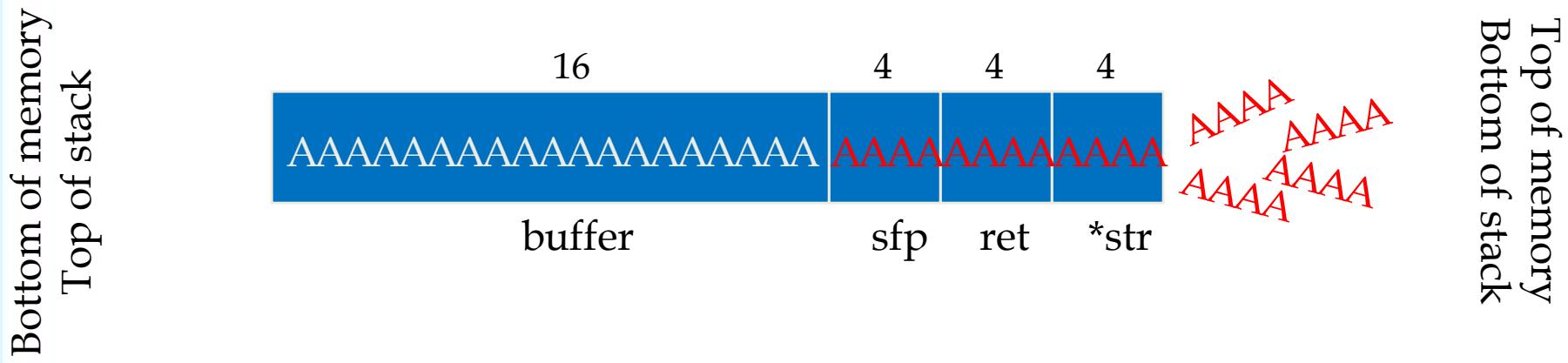
אָה הַגָּאֵחַ?

```
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 Overflow

- כאשר התוכנית רצה נגרמת Seg-Fault

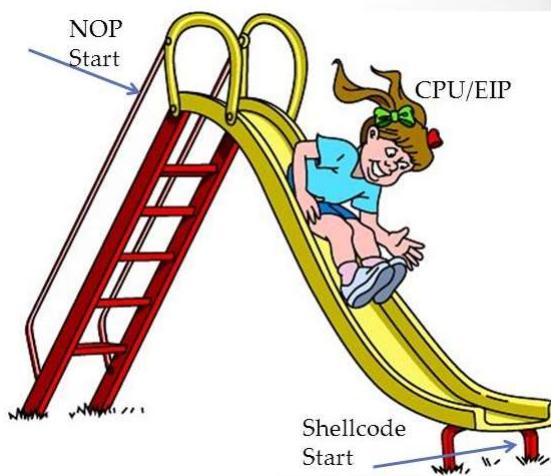


- כתובת החזרה נדרשת על ידי 'AAAA' (0x41414141)
- כשהפונקציה מסיימת היא מנסה לחזור לכתובת 0x41414141



היכן נזרק הקוד?

- דורסים את כתובת החזורה
 - לאן להציבו?
- כותבים קטע קוד שאינו תלוי במיקומו (Shellcode)
 - ושמים אותו על הstack
 - איך נתוכנו לעובדה שאנו לא יודעים תמיד לבדוק מה יש על stack? ומה המרחק הנכון?
NOOOOOOOOOOOOOOOOP
- 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

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Smashing The Stack For Fun And Profit
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

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 lassage, overrun screw.

Introduction

~~~~~

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



# קיך רימן פאנליז מה זה?

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



# Dangerous C system calls

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

## Extreme risk

- `gets`
- `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`



# strcmp(computed\_response, user\_response, response\_length)

```
NETSTACK_CODE:20431F80
NETSTACK_CODE:20431F80 loc_20431F80:
NETSTACK_CODE:20431F80      add    r14, sp, 0x10C+var_F4
NETSTACK_CODE:20431F80      mov    r0, r13
NETSTACK_CODE:20431F82      add    r0, r0, 0x55
NETSTACK_CODE:20431F84      mov    r1, r14
NETSTACK_CODE:20431F86      bl     NETSTACK_CODE_2043218C
NETSTACK_CODE:20431F88      ld    r4, [sp, 0x10C+nc.value_len]
NETSTACK_CODE:20431F8C      ld    r5, [sp, 0x10C+var_74]
NETSTACK_CODE:20431F90      ld    r0, [sp, 0x10C+qop.value_len]
NETSTACK_CODE:20431F94      ld    r7, [sp, 0x10C+qop]
NETSTACK_CODE:20431F98      st    r0, [sp, 0x10C+var_10C]
NETSTACK_CODE:20431F9C      st    r0, [sp, 0x10C+uri]
NETSTACK_CODE:20431F9E      ld    r0, [sp, 0x10C+var_108]
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_108]
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     strcmp
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!**



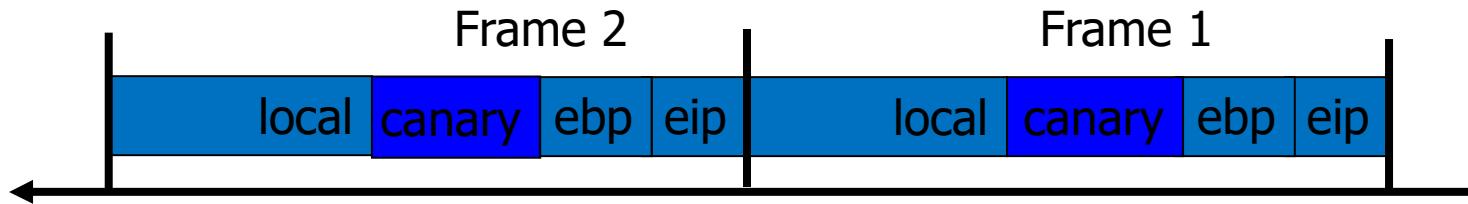
# קיך רימר סגנור מה זה?

- קידוד מאובטח...
  - שימוש בספריות מאובטחות Type-Safe
  - שימוש בשפות שהן 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 ערכיהם כדאי שיהיו למשתנה זהה? מה לא?
- מה המאפיינים שצרכיים להיות למשתנה?
  - מה יכול תוקף ללמידה?
    - מריצה אחרת?
    - מראית הזכרון?
    - מהכרת תהליכיים 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 prologue & epilogue – significant overhead.
- So with /GS adds stack cookie only to functions that contain string buffers or 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.



# אֲכָלָה כְּבָשָׂר יִקְרָא?



188

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020



# וַיְכֹל אֶמְכִיאת

## Exception Handling



# אנו מוכנים לאתגר

- טיפול בחריגות הינו דבר נפוץ שנמצא בשימוש כמעט בכל תוכנית.
- חוקרים אנו נתקל רבות בטיפול בחריגות, ועלינו לזהות ולהבין כיצד זה עובד.
- נתחיל משירותי מ"ה לחריגות ברמת התהילה.
- ונמשיך עם איך השירותים הללו ניתנים ע"י גרעין מ"ה.



190



# 6. וַיְכֹלֶת אָמְרִים כֵּן

C

```
_try
{
    // guarded code
}
_except ( expression )
{
    // exception handler code
}
```

C++

```
try {
    //throw
}
catch (...) {
    // exception handler code
}
```



# עיכומי א"ה סtruсted exception handling (SEH)

## Structured Exception Handling (SEH)

- מערכת הפעלה מאפשרת טיפול בחיריגות ע"י מנגנון בסיסי המשותף לכל המהדרים והתהליכיים.
- כל קומפיילר ממש implements Exception-Handling בצורה שונה.
  - לא נכנס לפרטים לגבי המימושים השונים של הקומפיילרים.
  - קומפיילרים מרחיבים את המנגנון של מערכת הפעלה.



# אפקט SEH

- תוכנית יכולה לקרוס מסיבות שונות (חלוקת ב-0 למשל).
- מערכת הפעלה מיידעת את התוכנית על הקריסה, ואנוטנת לה הזדמנות לתקן את הבעיה.
  - מ"ה קוראת לפונקציה ששייכת לתוכנית שקרסה.
  - הפונקציה תעשה מה שתעשה, ותחזיר למ"ה ערך חזרה.
  - בתחילת ריצת התוכנית או קטע ה-`try..except..` התוכנית מבונצרתrica  - כדי לידע את מ"ה שיש לה פונקציה כזו ומה הכתובת שלה.
- לפונקציה קוראים `.except_handler`.



# הפקזה `except_handler`

- הפקזה שנתקלה בזמן חריגה.
  - בדרך כלל מדובר בפקזה שהקומפイルר מספק.
  - אבל גם התכנית יכולה לספק אותה עצמה.
- קלט:
  - מקבלת מידע לגבי סוג החריגה ומצב התחלתיך בזמן החריגה.
- ערכי חזרה:
  - `:ExceptionContinueExecution`
    - מסמל טיפול מוצלח בחריגה.
  - `:ExceptionContinueSearch`
    - אם הפקזה לא יודעת כיצד לטפל בחריגה.
  - יש עוד שני ערכי חזרה אפשריים עליהם לא נרחיב:
    - `.ExceptionNestedException`
    - `.ExceptionCollidedUnwind`



# הארקזיה except\_handler

```
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 אלףים 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 : בתחילת הפעלה מכיל את מצב האוגרים בעת החיריגת ובהמשך הפונקציה `except_handler` יכולה לשנות אותו. אם הפונקציה `ExceptionContinueExecution` החזירה `except_handler`, אז מ"ה תעדכן את האוגרים בהתאם למבנה CONTEXT.



# ונדי? – 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

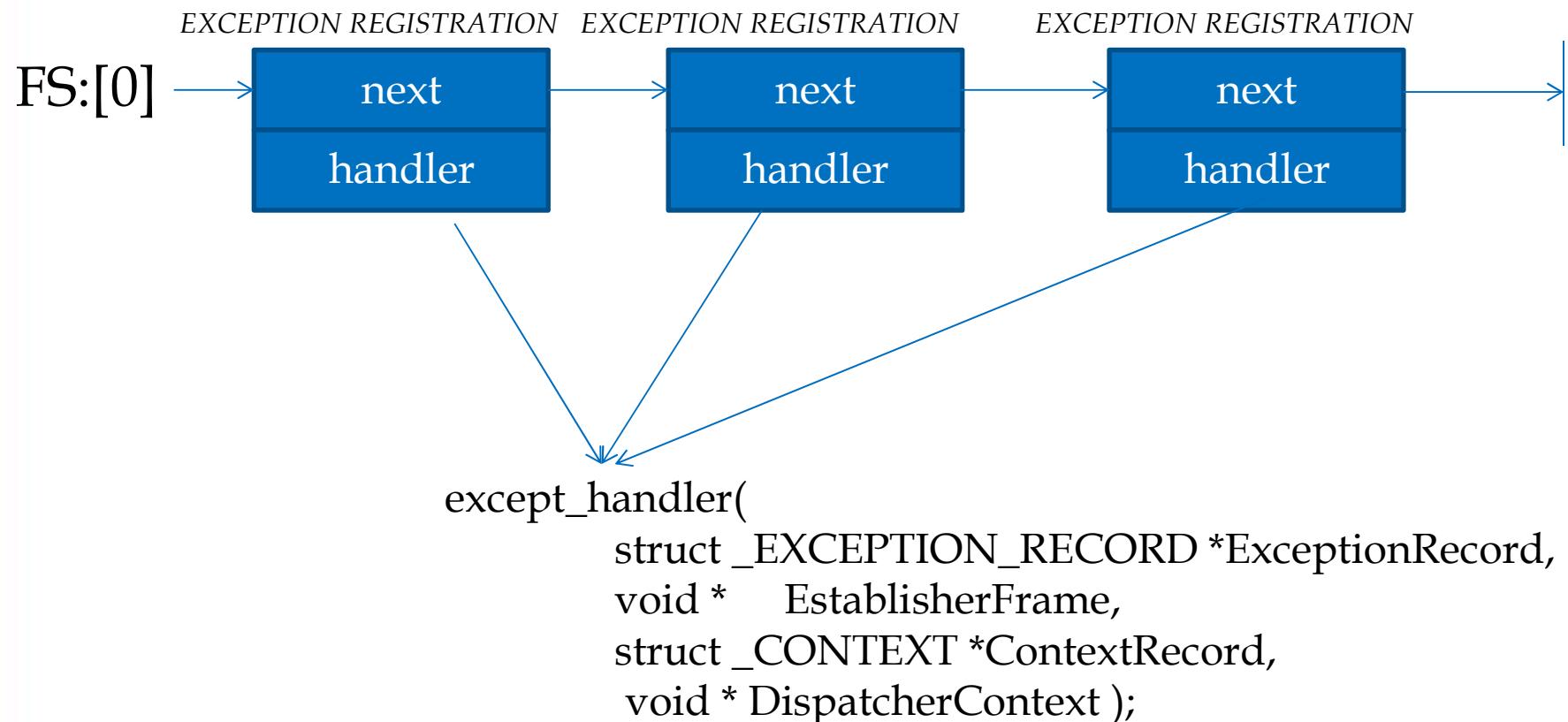
- ראיינו כיצד נראה הפקציה שמייה תקרה לה.
- אבל איך נידע את מייה שיש לנו פונקציה כזו?
  - איך מייה תמצא אותה בזיכרון התוכנית?



199



# EXCEPTION REGISTRATION



# 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 בראשימה מקוشرת.
- מקובל לשים את אברי הרשומות על המחסנית.



# kNCI?

```
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



# kNCI?

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

esp →

FS:[0] →



+



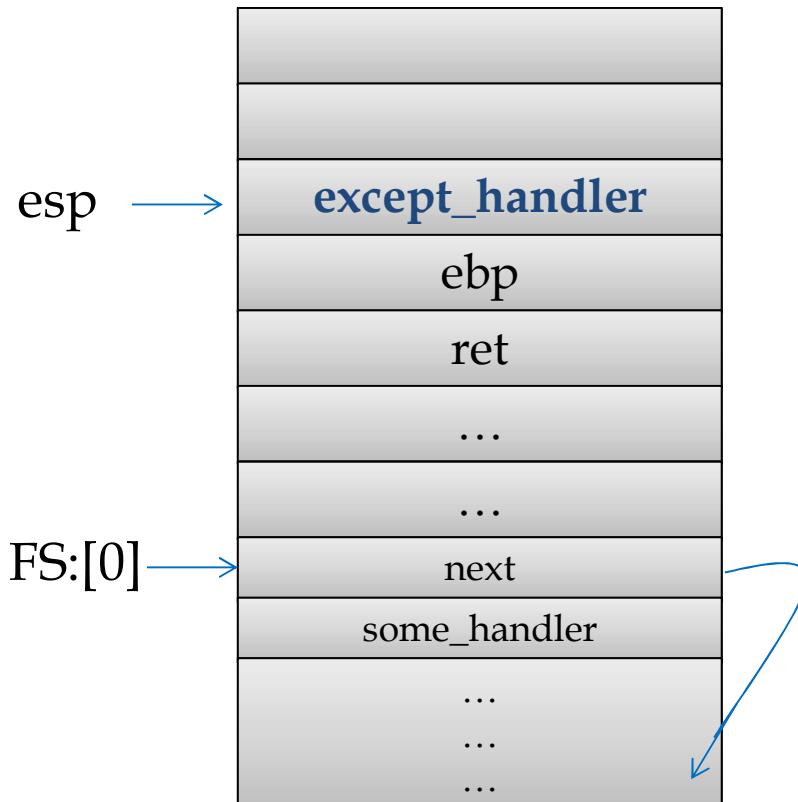
# kNCI?

```
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
```



+



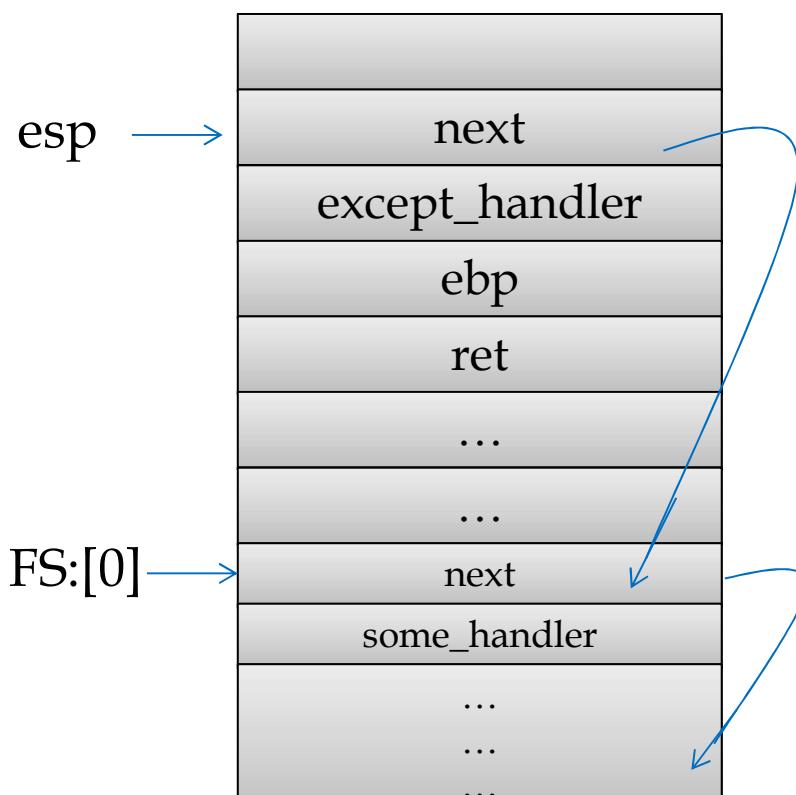
# kNCI?

```
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
```



# kNCI?

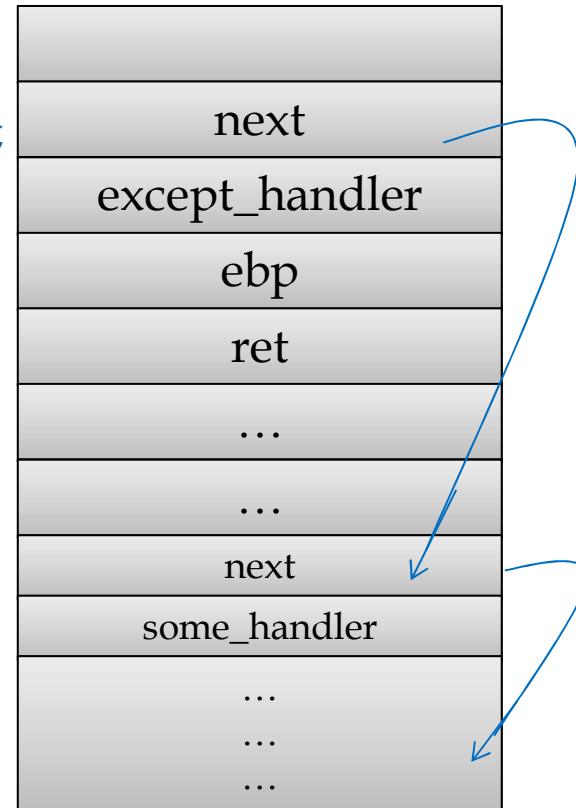
```
push except_handler  
push fs:[0]  
mov fs:[0],esp
```

... some code

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

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

esp →  
FS:[0]



+



# kNCI?

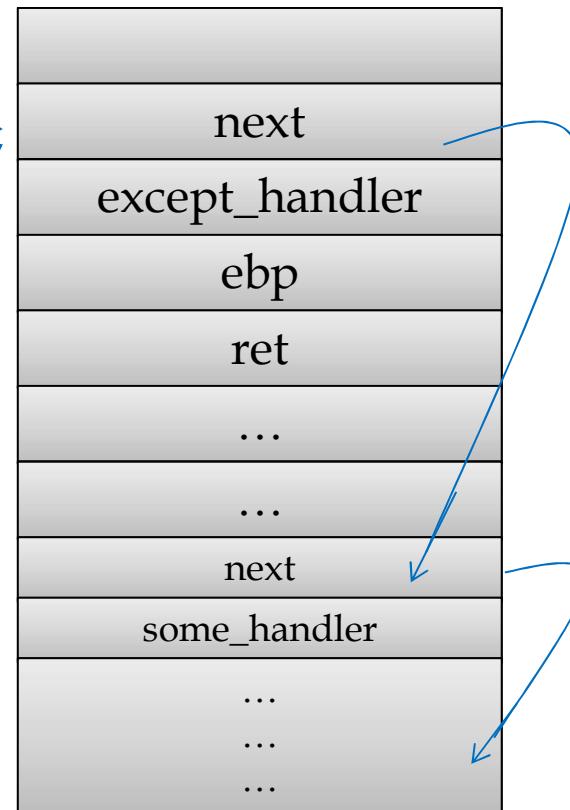
```
push except_handler  
push fs:[0]  
mov fs:[0],esp
```

... some code

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

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

esp →  
FS:[0]



# kNCI?

```
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
```

esp →

FS:[0] →



# kNCI?

```
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
```

esp →

FS:[0] →



+



# המקרה *exception*

- כאמור, בעת חריגה מייה עוברת על הרשימה עד למציאת ה- **handler** שמצויל לטפל בחריגה.
- ה- **handler** שמצויל אחראי למחוק את יתרת המחסנית ואת רשומות ה-**Registration** **Exception** המיותרות.
  - יש לו חופש פעולה למחוק את מה שרלוונטי לאוטו מקרה.
  - בדרך כלל ה- **handlers** שיסירבו לטפל בחריגה, כולל זה שהצליח.
  - אבל לעיתים ללא מחיקה.
    - למשל תיקון חילוק באפס שמחזיר ערך קבוע כתוצאה החילוק.
- במהלך המחיקה, כל **handler** שנמחק נקרא פעמיים נוספת שחרור משאים.
  - למשל שחרור מחלוקת ב-**C++**.
- לא נהגייב לעומק כיצד המנגנון עובד.



# השאלה

- מה קורה אם נזרקת חריגה בקוד מוביל שהשתמשנו ב-try?
- תשובה :

- בתחילת הrinch מ"ה מתחילה את [0]:fs במצבי handler.
- כאשר ה-handler נקרא, הוא מציג הודעה מתאימה.
- הוא נקרא אם כל האחרים לא הצליחו לטפל בחറיגת.



# סיכום מילוי הרכבת המתהפיק

- ה-Thread צריך לספק למערכת הפעלה רשיימה מקוושרת של Exception Registration.
- הקומpileר מממש את המנגנון שלו בשימוש ב-SEH.
  - כל try יוצר ExceptionRegistration.
  - ה-handler יהיה קוד פנימי של הקומpileר שייקרא בסוף לקוד שלכם (מה שתכתבם בתוך ה-except).
  - הקומpileר כאמור יכול להרחיב את המנגנון.
    - למשל באתחול ה-ExceptionRegistration נראה פקודות נוספות שקומpileר יצר.



# SEH Implementation

|          |          |                                              |
|----------|----------|----------------------------------------------|
| 0189FF88 | 0189FF94 | kernel32.764B1174                            |
| 0189FF8C | 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 | FFFFFFF  | End of SEH chain                             |
| 0189FFC8 | 777A0555 | SE handler                                   |
| 0189FFCC | 00013988 |                                              |
| 0189FFD0 | 00000000 |                                              |
| 0189FFD4 | 0189FFEC |                                              |
| 0189FFD8 | 777EB3FC | RETURN to ntdll.777EB3FC from ntdll.777EB402 |
| 0189FFDC | 00401848 | vulnser.00401848                             |
| 0189FFE0 | 00000060 |                                              |
| 0189FFE4 | 00000000 |                                              |
| 0189FFE8 | 00000000 |                                              |
| 0189FFEC | 00000000 |                                              |
| 0189FFF0 | 00000000 |                                              |
| 0189FFF4 | 00401848 | vulnser.00401848                             |
| 0189FFF8 | 00000060 |                                              |
| 0189FFFC | 00000000 |                                              |



217

0. מדריך ל-

התקנת וריאנטים, ביצועים, סולvit ו-

29.01.2020



# Did you say “STACK”?

A A A A A A A A A A A A A A A A A A A A A A  
A A A A A A A A A A A A A A A A A A A A A A  
A A A A A A A A A A A A A A A A A A A A A A  
A A A A A A A A A A A A A A A A A A A A A A  
A A A A A A A A A A A A A A A A A A A A A A  
A A A A A A A A A A A A A A A A A A A A A A  
A A A A A A A A A A A A A A A A A A A A A A  
A A A A A A A A A A A A A A A A A A A A A A  
A A A A A A A A A A A A A A A A A A A A A A  
A A A A A A A A A A A A A A A A A A A A A A  
A A A A A A A A A A A A A A A A A A A A A A  
A A A A A A A A A A A A A A A A A A A A A A  
A A A A A A A A A A A A A A A A A A A A A A  
A A A A A A A A A A A A A A A A A A A A A A  
A A A A A A A A A A A A A A A A A A A A A A  
A A A A A A A A A A A A A A A A A A A A A A  
A A A A A A A A A A A A A A A A A A A A A A  
A A A A A A A A A A A A A A A A A A A A A A  
A A A A A A A A A A A A A A A A A A A A A A



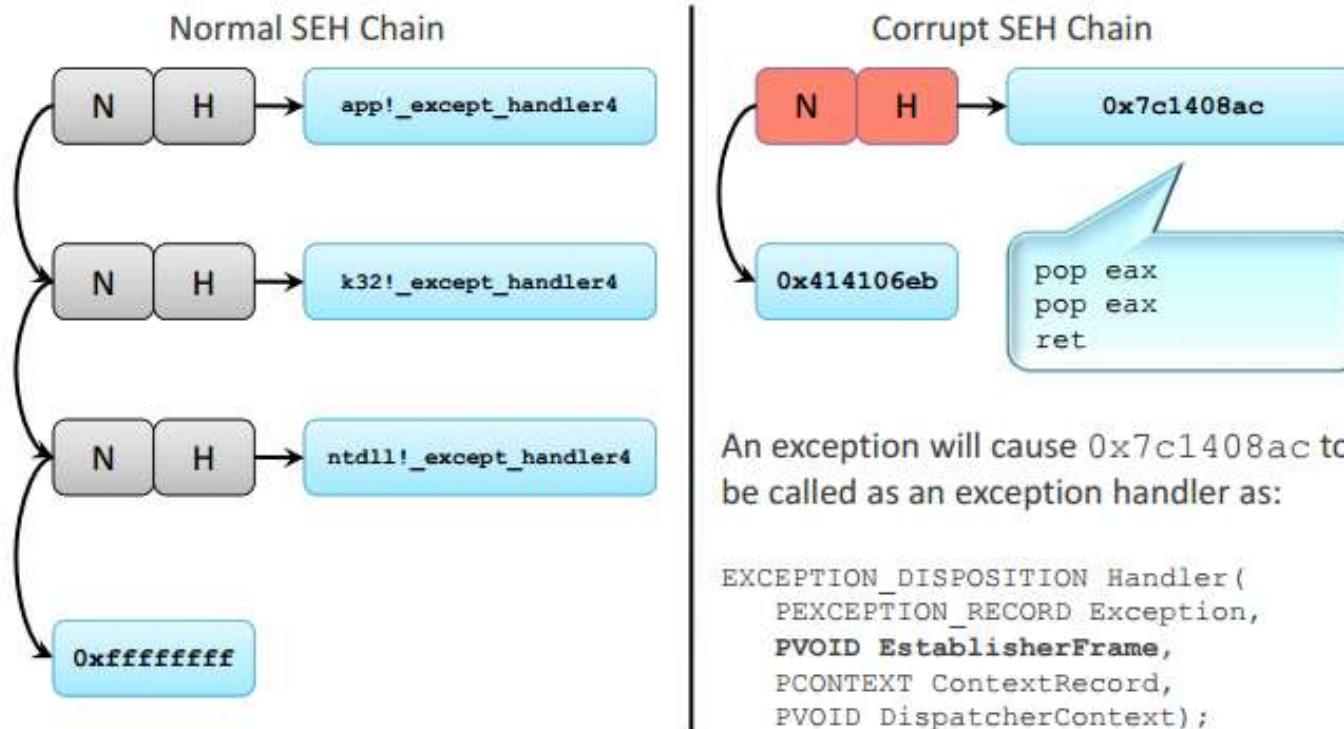
# Buffer Overflow to the Rescue!

|           |          |
|-----------|----------|
| 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 |
| 019FFFC8  | 41414141 |
| 019FFFCC  | 41414141 |
| 019FFFD0  | 41414141 |
| 019FFFD4  | 41414141 |
| 019FFFD8  | 41414141 |
| 019FFFDCC | 41414141 |
| 019FFE0   | 41414141 |
| 019FFE4   | 41414141 |
| 019FFE8   | 41414141 |
| 019FFFECC | 41414141 |
| 019FFFF0  | 41414141 |
| 019FFFF4  | 41414141 |
| 019FFFF8  | 41414141 |
| 019FFFFC  | 41414141 |

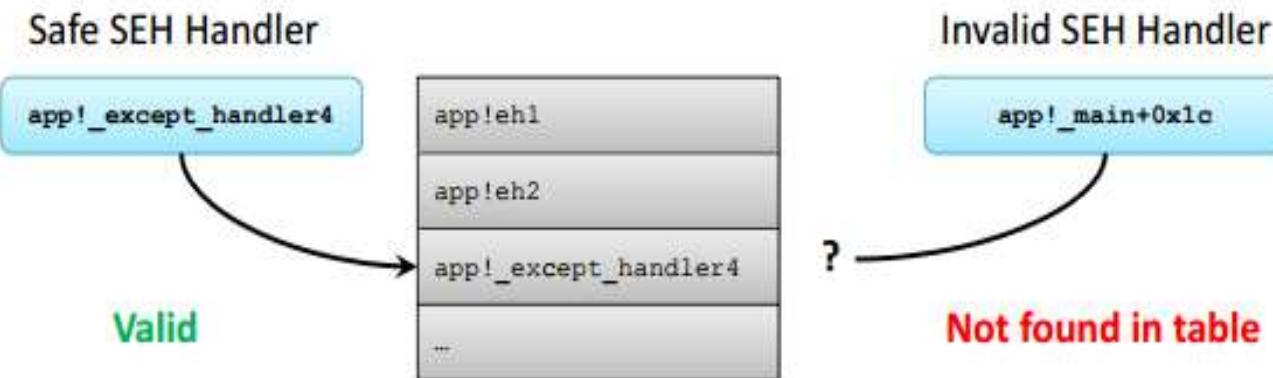
Pointer to next SEH record  
SE handler



# Exploit: SEH Overwrite



# 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?**)

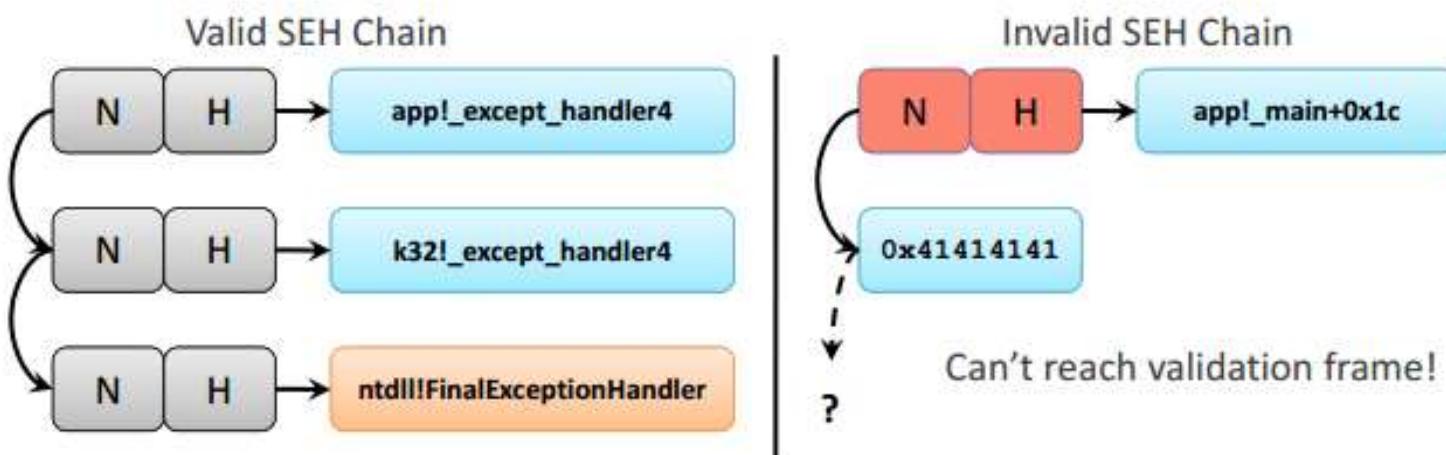


# Sentinel

- 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 and onwards.
  - Checks that the linked list of SEH handlers is valid and -1 terminated.



# It is probably not enough...

איך, כך, מרסמתamt הפתכון הטעזיז?

- האם ניתן להבחין בין איזורי זיכרונות שונים?
- דורך תמיכת חומרה ושל מערכת הפעלה..

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.)

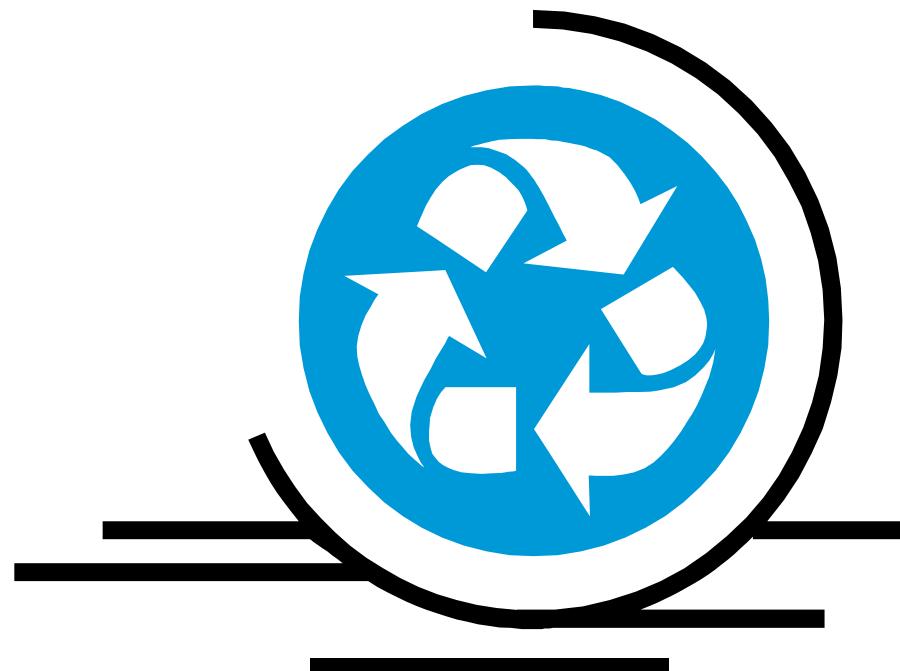


# Breaking DEP

- 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?



227

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020



# 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.



220

UNIVERSITY OF CALIFORNIA SAN DIEGO

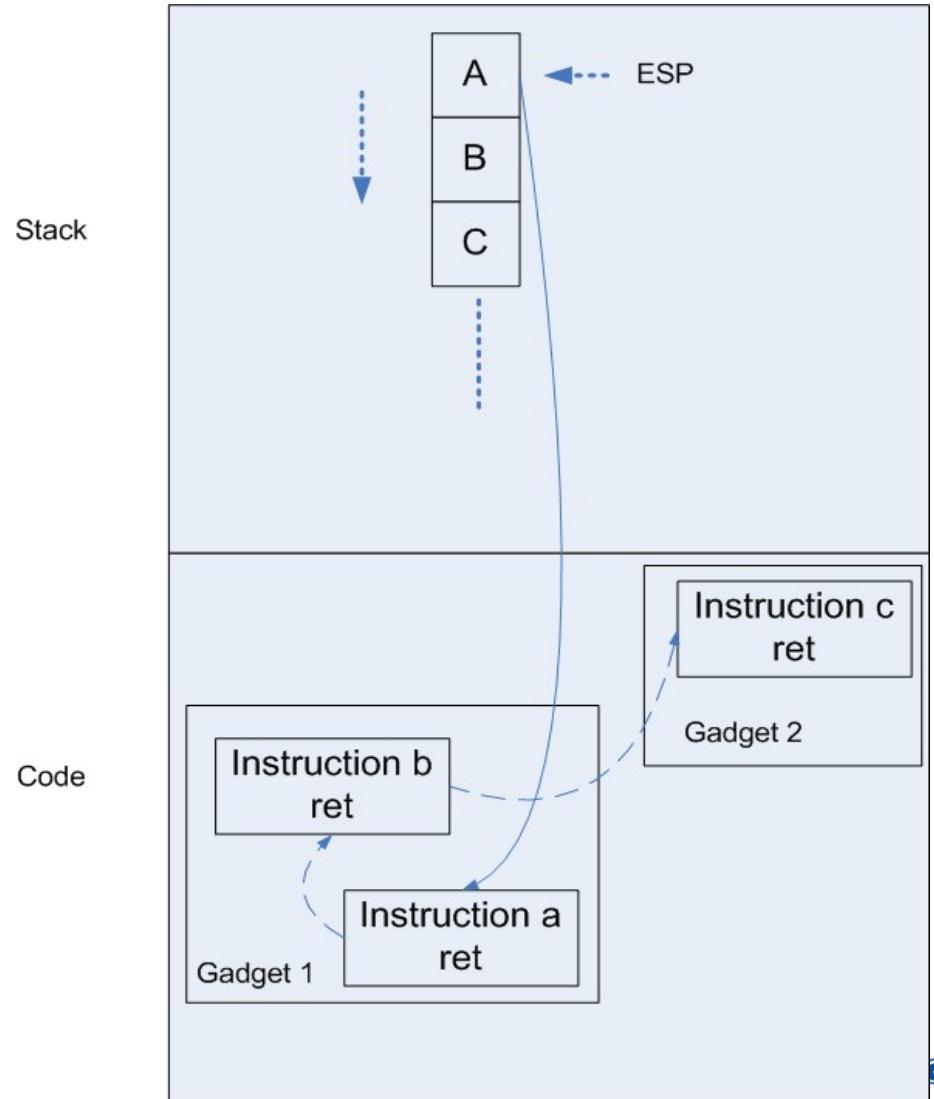
2017-18, 2018-19, 2019-20, 2020-21, 2021-22

2020-21



# 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.

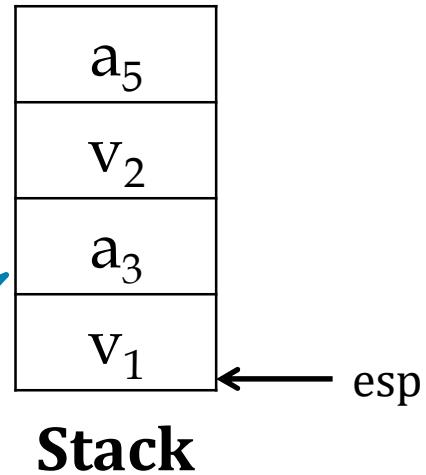


# Gadgets

Mem[v2] = v1

Desired Logic

Suppose a<sub>2</sub> and a<sub>3</sub> on stack



|     |                |
|-----|----------------|
| eax | v <sub>1</sub> |
| ebx |                |
| eip | a <sub>1</sub> |

a<sub>1</sub>: pop eax;  
a<sub>2</sub>: ret  
a<sub>3</sub>: pop ebx;  
a<sub>4</sub>: ret  
a<sub>5</sub>: mov [ebx], eax

Implementation 2

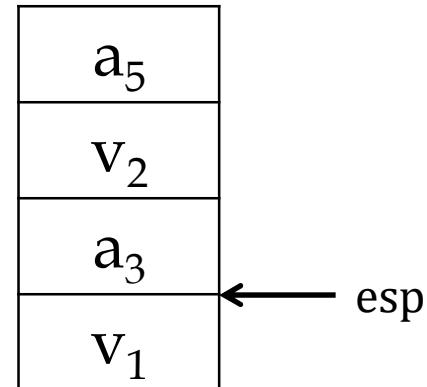


# Gadgets

Mem[v2] = v1

Desired Logic

|     |    |
|-----|----|
| eax | v1 |
| ebx |    |
| eip | a3 |



Stack

a<sub>1</sub>: pop eax;  
a<sub>2</sub>: ret  
a<sub>3</sub>: pop ebx;  
a<sub>4</sub>: ret  
a<sub>5</sub>: mov [ebx], eax

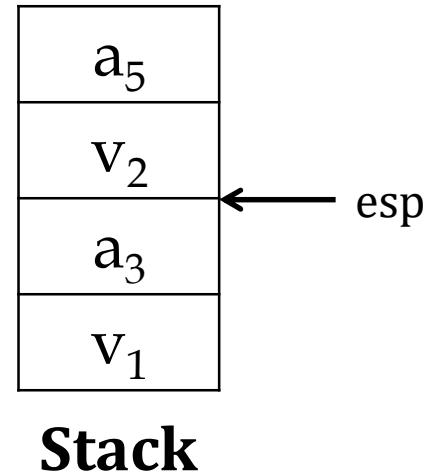
Implementation 2



# Gadgets

Mem[v2] = v1

Desired Logic



Stack

|     |    |
|-----|----|
| eax | v1 |
| ebx | v2 |
| eip | a3 |

$a_1: \text{pop eax};$   
 $a_2: \text{ret}$   
 **$a_3: \text{pop ebx};$**   
 $a_4: \text{ret}$   
 $a_5: \text{mov [ebx], eax}$

Implementation 2

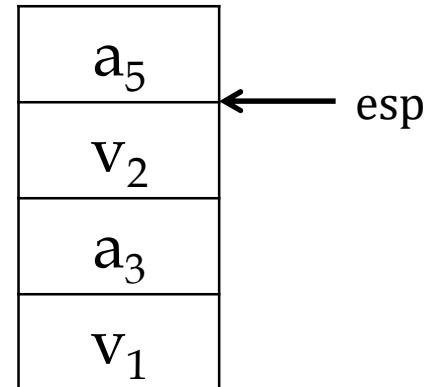


# Gadgets

Mem[v2] = v1

Desired Logic

|     |                |
|-----|----------------|
| eax | v1             |
| ebx | v2             |
| eip | a <sub>4</sub> |



Stack

a<sub>1</sub>: pop eax;  
a<sub>2</sub>: ret  
a<sub>3</sub>: pop ebx;  
a<sub>4</sub>: ret  
a<sub>5</sub>: mov [ebx], eax

Implementation 2

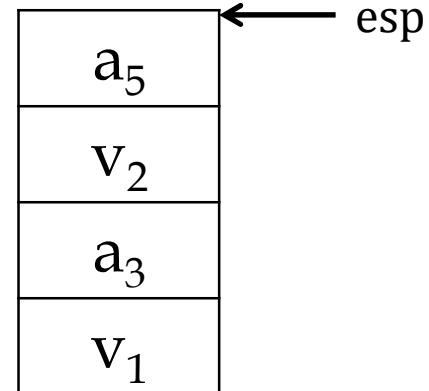


# Gadgets

Mem[v2] = v1

Desired Logic

|     |    |
|-----|----|
| eax | v1 |
| ebx | v2 |
| eip | a5 |



Stack

a<sub>1</sub>: pop eax;  
a<sub>2</sub>: ret  
a<sub>3</sub>: pop ebx;  
a<sub>4</sub>: ret  
a<sub>5</sub>: mov [ebx], eax

Implementation 2

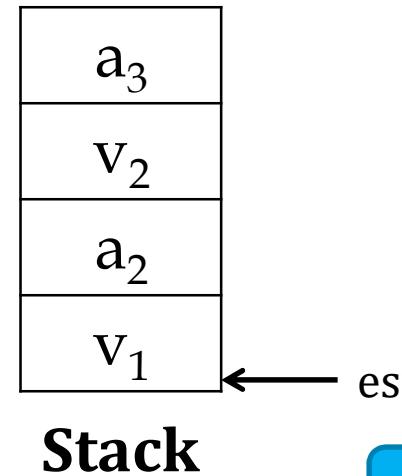


# Equivalence

Mem[v2] = v1

Desired Logic

semantically  
equivalent



Stack

"Gadgets"

- ↔  $a_1$ : pop eax; ret
- ↔  $a_2$ : pop ebx; ret
- ↔  $a_3$ : mov [ebx], eax

Implementation 2



# Equivalence

Mem[v2] = v1

Desired Logic

a<sub>1</sub>: pop eax; ret

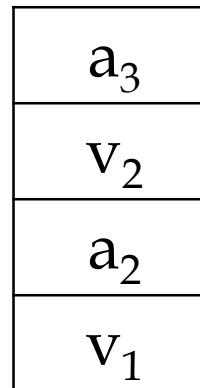
...

a<sub>3</sub>: mov [ebx], eax

...

a<sub>2</sub>: pop ebx; ret

Address  
independent!



Stack

a<sub>1</sub>: pop eax; ret

a<sub>2</sub>: pop ebx; ret

a<sub>3</sub>: 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.



# ?האם?



239

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020



# פרק 7

## ארכיטקטורת כויהן גייליאם

### Java Bytecode & CIL



240

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד קרמל, עמר קדמייאל

23.01.2020



# קוֹד בִּינְיִים

- תרגום שפה עילית לקוד ביניים שאינו שפת מכונה

- קוד הביניים הינו שפה פשוטה

- בדר"כ ממודלת **מכונת מחסנית אבסטרקטית**

- שקל לפרש אותה ולהריץ אותה, ללא צורך במהדר נוסף

- בכל מחשב יש תוכנה ש谋ריצה את קוד הביניים

- באינטפרטציה, ללא צורך בהידור נוסף

- בחלק מהמקרים קוד הביניים מתרגoms בזמן ריצה לשפת מכונה

- (Just In Time Compiler)

- שתי דוגמאות עיקריות

| הערות                                                   | שפת תוכנות    | שפת קוד ביניים       |
|---------------------------------------------------------|---------------|----------------------|
| מתוכנתה במיוחד ל- <b>Java</b>                           | <b>Java</b>   | <b>Java Bytecode</b> |
| תומכת ב מגוון שפות (ידעועה גם <b>C#</b> ו <b>.NET</b> ) | <b>(MSIL)</b> | <b>CIL</b>           |

- נתמקד ב-**Java Bytecode** – בקיצור **JB**



# ארכ' קאש ג'לצ'ה Java

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



# אֱלָה קַאֲס Class

- קובץ class כולל

- תיאור התוכן הפומבי של המחלקה
- רשימת קבועים (Constant pool)
  - כוללת את כל הקבועים וסוגם
  - קלומר
- הגדרות המשתנים (בפרט הפומביים של המחלקה), עם הפניות לтиיפוסים
  - ✓ ההפניה היא לאיבר אחר ברשימה
- מחרוזות
  - ✓ מחרוזות שמופיעות בתכנית
  - ✓ מחרוזות של שמות מחלקות ושמות פונקציות שבשימוש
- כל הティיפוסים שיש בתכנית
  - מבנה של ענפים של עץ
- הקוד (bytecode) של כל הֆונקציות של המחלקה
  - בשפה דמוית אסמבלי



# וַיְכֹסֵם גָּסִיסִים גָּמֶת Java

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

| טיפוס     | קיצור |
|-----------|-------|
| integer   | I     |
| long      | L     |
| short     | S     |
| byte      | B     |
| character | C     |
| float     | F     |
| double    | D     |
| boolean   | Z     |
| reference | A     |

signed      {

Unicode (16 bit)      {

- לערך יש תוספה "]" לפני האות ...unsigned int/long/byte
- אין טיפוסי



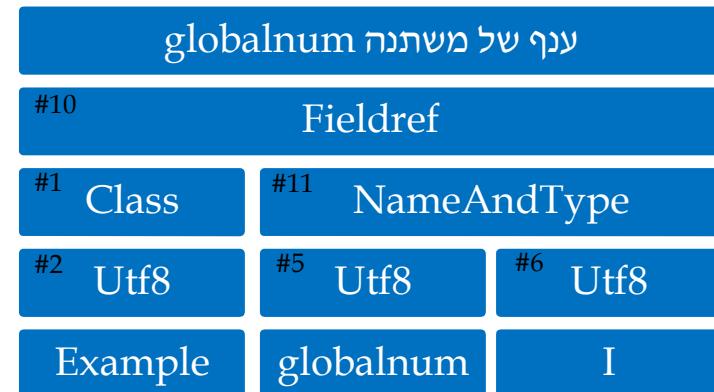
# אגה קבועה קבועה

- 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 שונים מבתקן
    - String : מחרוזת, בצירוף האינדקס של הטקסט UTF-8 שלו
    - Class : מחלוקת, בצירוף האינדקס של הטקסט UTF-8 של השם שלו



# אגה כויאט הקהosit

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



# קָדָם מִכְרִימַת

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

ה משתנה  
MOVFNה ע"י #10

"The sum is"  
MOVFNה ע"י #26

פלט של הרצת javap על קובץ המחלקה



אינדקס

טיפוס

עדכ או הפניה

הערות



# אנה כסיס' fe מט Java Bytecode

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



# אנה כסיס' fe סט Java Bytecode

- JB רץ בתוך מכונה וירטואלית
  - מתוכנת במיוחד להגנה נגד חולשות
  - למשל, המכונה בודקת חריגה מערכיים
    - מייצרת exception במקרה של חריגה
  - המכונה הווירטואלית מודאת שקיימות הן תמיד לתחילה פקודות
    - נבדק על ידי ה-Verifier בתחילת הרצה
- כל כניסה על המחסנית היא בת 32 סיביות
  - שלם, מצביע, וכו'
  - ייצוג של long הוא כשתי כניסה רצופות (64 סיביות)



# המוכרין fe JB הנדרן

```
static {};
```

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 // Method java/lang/StringBuilder.append:(I)Ljava/lang/StringBuilder;  
30: invokevirtual #31 // Method java/lang/StringBuilder.toString():Ljava/lang/String;  
33: invokevirtual #35 // Method java/io/PrintStream.println:(Ljava/lang/String;)V  
36: invokevirtual #39  
39: return

כתובות

opcode

אופרנדים

הערוות



# רכורג'וּת JIT

## Just In Time compilation

- הידור קוד בזמן ריצה
- יכולה לבצע אינטראפטציה לקוד, ולבצע הידור רק לקוד בשימוש רב
  - וללמוד מהאינטראפטציה איך כדאי לבצע אופטימיזציות
- אפשר לוודא שהקוד המהודר אינו מתרכן לגורם נזק
  - קוד הביניים פשוט יותר לבדיקה
  - הקוד המהודר אינו ניתן לטיפול ע"י תוקף
  - ככלומר, נזקה אינה יכולה להשולט על הקוד...



# אוצר ה-**JAVA**

- תמיינה בrogram מבנים מאובטחים ע"י שפת התכנות
  - **Strong data typing**
  - ניהול זיכרון אוטומטי (ושחרור אוטומטי)
  - אין cast ללא בדיקה
  - אין אריתמטיקה של מצביעים
  - מערכיים שמוררים בזיכרון הדינמי
    - מוקצים בזמן ריצה (ע"י new), לא על המחסנית
- מכונה וירטואלית שבודקת כל מה שאפשר
  - **rogram בדיקות בזמן טעינת מחלקה**
    - בפרט, Bytecode verification
    - למשל, כל הקפיצות למקומות חוקיים (לא באמצע פקודות BJ)
  - **בדיקות בזמן ריצה**
    - למשל בבדיקות חריגת ממך
    - בדיקות שימוש במצב בעל ערך NULL



# אפקט המה-Java

- sandbox
  - מאפשר קביעת הרשות לקוד שרצה
  - ככלומר, לא כל ההרשאות של מרייצ' הקוד מאפשרות
    - למשל, מניעת גישה לקבצים, גישה לרשות או גישה לשרתים מסוימים בראשת, יציאה מהמכונה הוירטואלית, יצרת תהליך חדש, וכו'
- JIT
  - תוקף לא יכול לשתול קוד כרצונו
- לסייע, מניעת בעיות אבטחה מקוד עווין על ידי מניעת היכולת של הקוד לבצע פעולות בעיתיות
  - ברמת שפת התכנות, המכונה הוירטואלית, וקריאה לפעולות מסוימות
  - אולם, ההגנות לא מושלמות
    - בשנים האחרונות התגלו מגוון בעיות אבטחה ב-Java שגרמו להפסקת השימוש ב-Java במספר ארכיטקטורות



# אפקט ה-overflow ב-Java

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



# קִיס-אַסְמְבָלִים וּזְה-קִוםְפִּילִים

- קיימים מגוון של דיס-אסמלרים וזה-קומפיילים
  - הлокחים קוד BJ ומתרגמים לקוד קריא
- למשל
  - דיס-אסמלרים
    - javap
    - גם גרסאות מסוימות של IDA
  - זה-קומפיילים
    - JD
    - JAD
    - Krakatau
- בדרכ' מיצרים קוד די טוב



# ב-קלאסים ג'י-די אקספלורר

```
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 & Obfuscation

- העלמת מחרוזות מרשימה הקבועים
  - למשל יוצרים בקוד
- שינוי שמות משתנים וمتודות
- ניקוי נתוני דיבוג
- וכמובן, סיבוך הקוד
- יש מגוון תוכנות obfuscation ל-Java



# קץ של האנה Class fe קאנס

- רשימת הקבועים

- הרשומות מסודרות לפי סדר

- לא אינדקס מפורש בפנים

- כל רשומה מתחילה בבית המטא טיפוס

- ומידע באורך קבוע

- במקורה 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 באמצעות

- JB

- קפיצות נשות באופן ייחסי למיקום תחילת הפקודה

- שינוי קוד אינו דורש תיקון בקפיצות שאין קופצות מעל השינוי

- עדין עדיף לשנות בלי לשנות אורך



# האקיינס של קבץ Class

- דוגמאות לשינויים ב-JN
  - דוגמא לשינוי מחרוזת ישירות ב-class
    - כולל שינוי אורץ
  - דוגמא לשינוי opcode ישירות ב-class
    - למשל שינוי חיבור לכפל
  - שינויים יותר גדולים
- ה-verifier bytecode מונע מאיתנו לשנות בדרךים לא קבילות
  - למשל קפיצה לאמצע פקודה
- לא ניתן לבצע הוקינס לקוד בזיכרון בזמן ריצה
  - אין לתוכנית גישה לקוד...



# CIL-פJB איקרייט פין

- CIL תוכנה על סמך העקרונות של JB

- JB תוכנה במיוחד עבור Java

- CIL תוכנה לתמוך במגוון שפות

- לכן כוללת מגוון גדול יותר של טיפוסים ומבנהים

- המבנה העקרוני של השפות דומה

- שתיהן מבוססות מחסנית

- ואפילו עם פקודות דומות רבות

- אבל CIL במידה מהניסיון של JB

- עם שיפורים

- קוד הביניים תוכנן ל-TIJ

- JB תוכנה לתמוך באינטפרטציה וב-TIJ

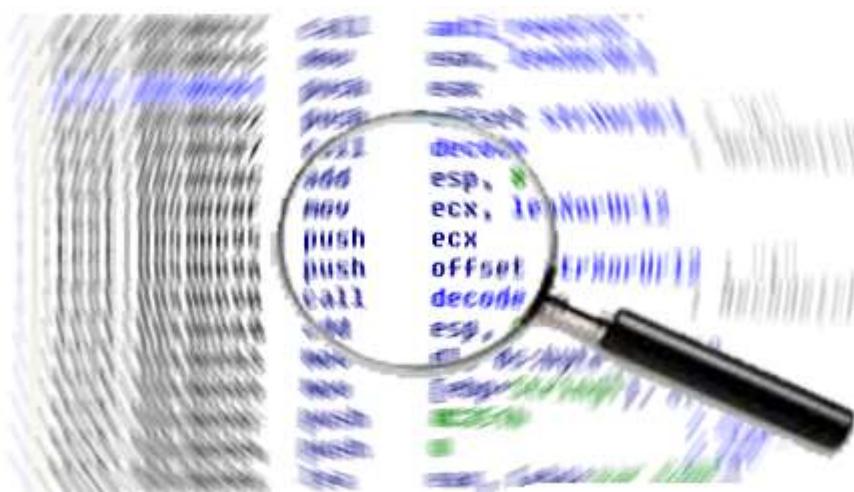


# הארץ Java

- אנדרואיד הפעלת Java מורכבת יותר
  - כדי לאפשר את יתרונות Java מצד אחד ופולה עיליה 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 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 ההרצתה, ללא צורך בשום הידור נוספת
- אנדרואיד 6 – ART משולב JIT
  - כדי לחסוך בזמן התקנה ובזמן שדרוג מ"ה
  - ההמרה לקובץ הרצתה נעשית בזמן פינוי של המערכת
  - עד אז מבוצע JIT

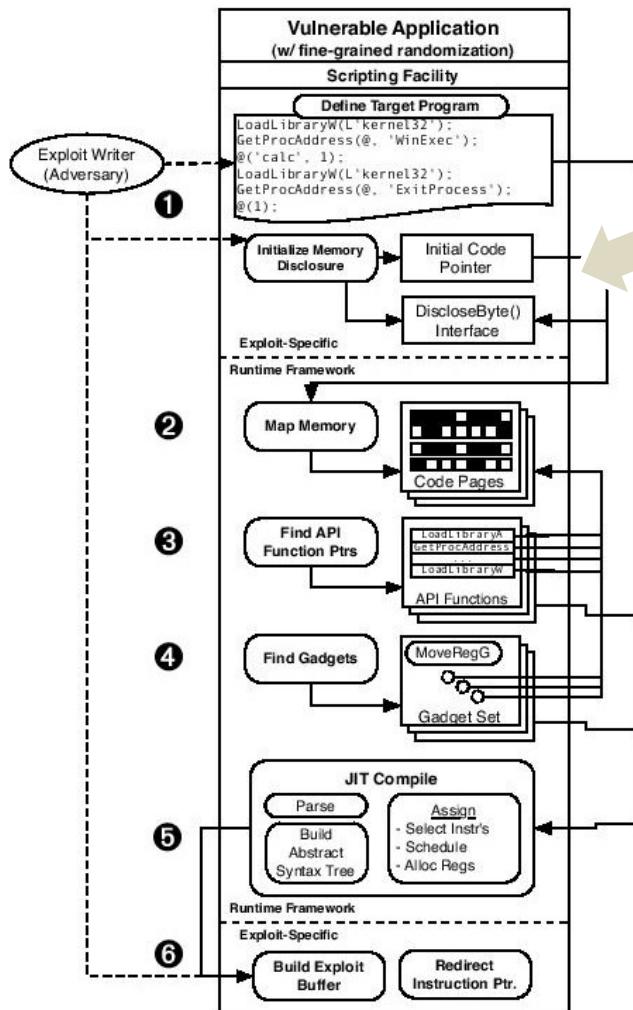


# JIT Reuse



# Just-in-Time Code Reuse (1)

[Snow et al., Oakland 2013]



Find one code pointer  
(using any disclosure vulnerability)

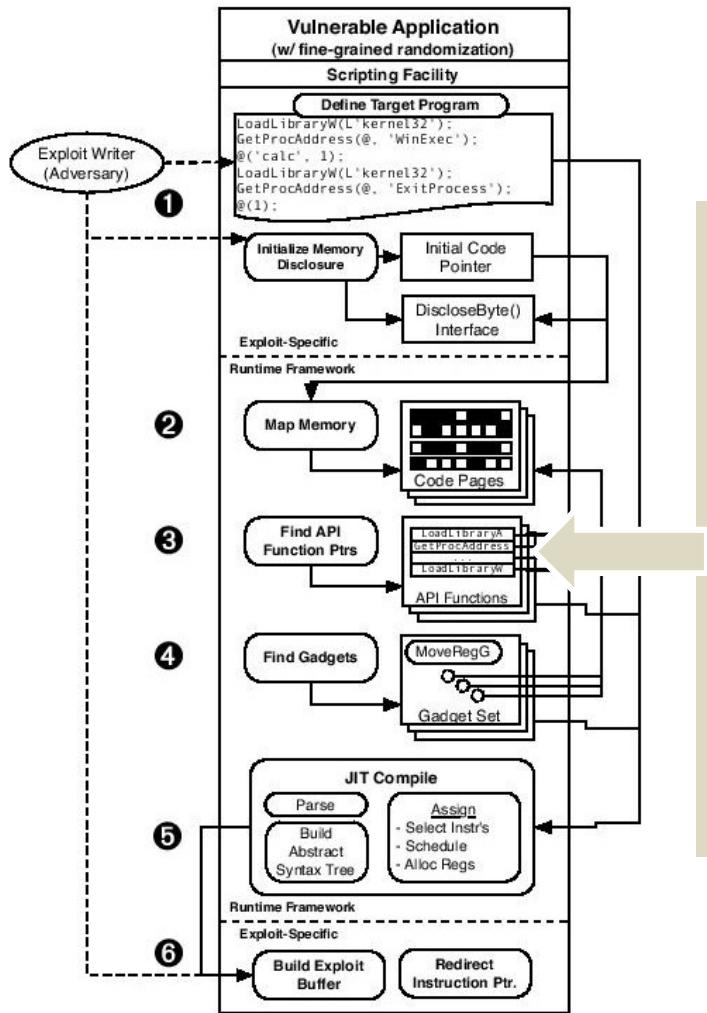
The entire page must be code...  
Analyze the instructions to find  
jumps and calls to other code pages...

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



# Just-in-Time Code Reuse (2)

[Snow et al., Oakland 2013]



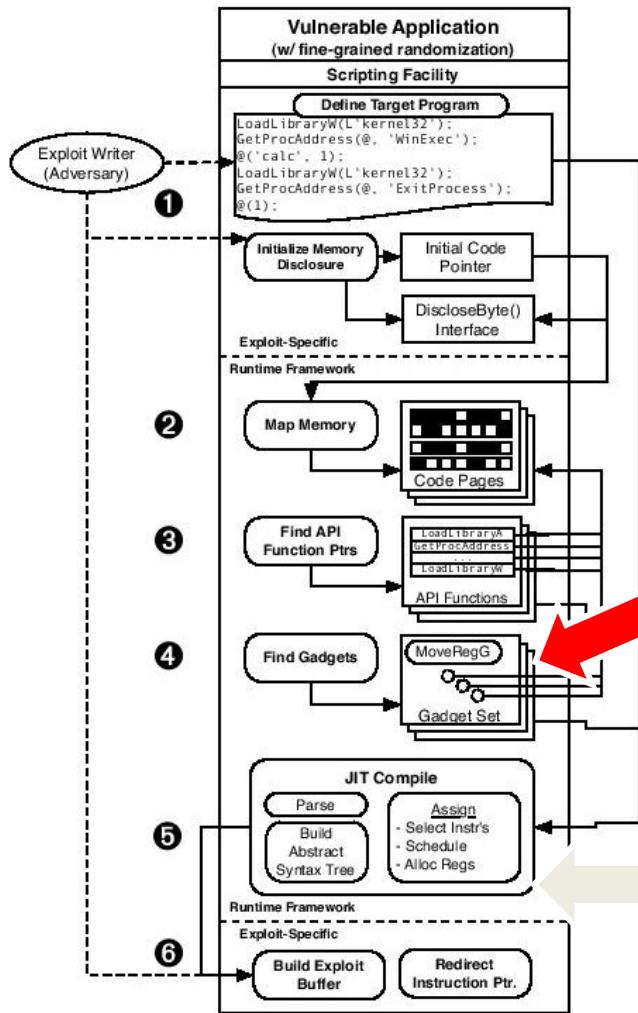
Use typical opcode sequences to find calls to `LoadLibrary()` and `GetProcAddress()`...

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

# 6 כנראה מה RE

## Anti-Reversing



267

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד קרמל, עמר קדמייאל

23.01.2020



# Anti-Reversing

- לתוכנות מסויימות, כגון נזקות, יש צורך בהשתרת הפעולות מפני גורמים מסויימים, ביניהם
  - כלים סריקה אוטומטיים, כדוגמת אנטי-וירוסים, מסוגלים לגלוות קוד עוין ע"י חתימות
    - או ע"י זיהוי דפוס התנהגות מסוים שנחשב שלא טריוניAli
  - 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 שמטרתו להקשות על גילוי התוכנה ועל ביצוע RE
  - אין אפשרות מוחלטת למנוע RE
  - אך אפשר לבלב את החוקר, לבזבז את זמןו, ולהפריע לו
  - שיטות הגנה מתחכਮות דורשות חוקרים יותר מנוסים והמון זמן מחקר
- **אנו נדונם בטכניקות אלה, ונתינח גם להתמודדות מולן**



# רכז'יקאט Anti-Reversing

- ניתן לחלק את השיטות לכמה סוגים
    - Obfuscation
      - טכניות להקשנות על הקורא להבין את הקוד, או לעכב אותו
    - Anti-Disassembly
      - טכניות למנוע את הצגת הקוד האמתי לחוקר או סורק
    - Anti-Debugging
      - טכניות לזיהוי סביבת ריצה עוינית
    - ולשינוי (או שיבוש או עצירה) פעילות התוכנית כתוצאה מכד
  - דחיסת והצפנת הקוד בקובץ ההרצה
- 
- נציג כמה רעיונות בסיסיים על מנת להדגים את הנושא
    - כל רעיון בסיסי ניתן להרחבה לשיטה מתקדמת יותר



# הנלה? רקיקת מהו נאה?

תשובה : 3.141



# הקיא ? הלא ?

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

**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, brainsck, 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 بصورة קשה להבנה
- אפשר לבצע obfuscation גם בשפה עילית וגם באסמבלי
- דוגמאות
  - כתיבה מסובכת של קוד פשוט
    - כל מי שבודק תרגילים נתקל בקוד כזה ☺
  - משפטים תנאי וולואות בהן לא ברור לקורא איזה קוד יבוצע
  - הוספת "garbage code" – קוד שאינו חלק רלוונטי מפעולות התוכנה, אך עשוי לבלום שעת לחוקר
  - הצפנה/דחיסה של הקוד



# Obfuscation-פְּנִימָה כְּנִזְקִית

```
#include <Windows.h>

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

void main()
{
    _asm
    {
        pushad

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

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

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

        popad
    }
}
```

From “leetMatrix” blog



דוגמא פשוטה ב-C:

```
static int two=1;

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



# תלכארט: DIS-ASMBLER

- כזכור, DIS-ASMBLER 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++;  
    }  
}
```



# jk-קיס-סמאגי

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



275

הנדסה לאחרור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמיאל

23.01.2020



# jk - קיס-נאנס – כינוי נאנו-וירטואלי

- מה עושה הקוד הבא?

```
00401328 E8 C7042424 CALL 246417F4
0040132D 3040 00 XOR BYTE PTR DS:[EAX],AL
00401330 E8 A7060000 CALL <JMP.&msvort.printf>
00401335 807424 1E LEA ESI,DWORD PTR SS:[ESP+1E]
```

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

- והקוד הזה?

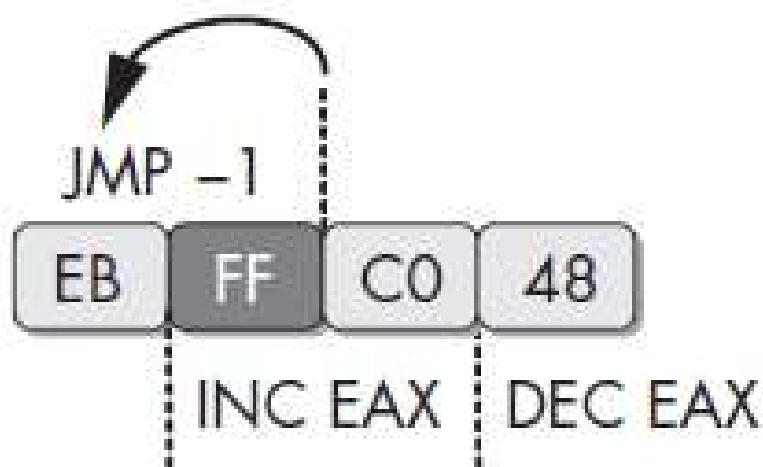
```
00401329 C70424 24304000 MOV DWORD PTR SS:[ESP],crackV2.00403024 ASCII "Enter the password:"
00401330 E8 H76666666 CALL <JMP.&msvort.printf>
00401335 807424 1E LEA ESI,DWORD PTR SS:[ESP+1E]
```

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



# jk - ?ok-SAnfa - מפיהם קיז

- מה עושה הקוד הבא?



- זה למעשה (כמעט) פקודת סוף בת 4 בתים

- קפוץ אחריה בית אחד, לקוד חופף (2 בתים : EBFF )
- הגדיל את EAX באחד ( 2 בתים : FFC0 )
- הקטן את EAX באחד ( בית אחד : 48 )
  - זה אינו סוף בגלל ההשפעה על הדגמים

from Practical Malware Analysis



277

הנדסה לאחור – חורף תשע"ט

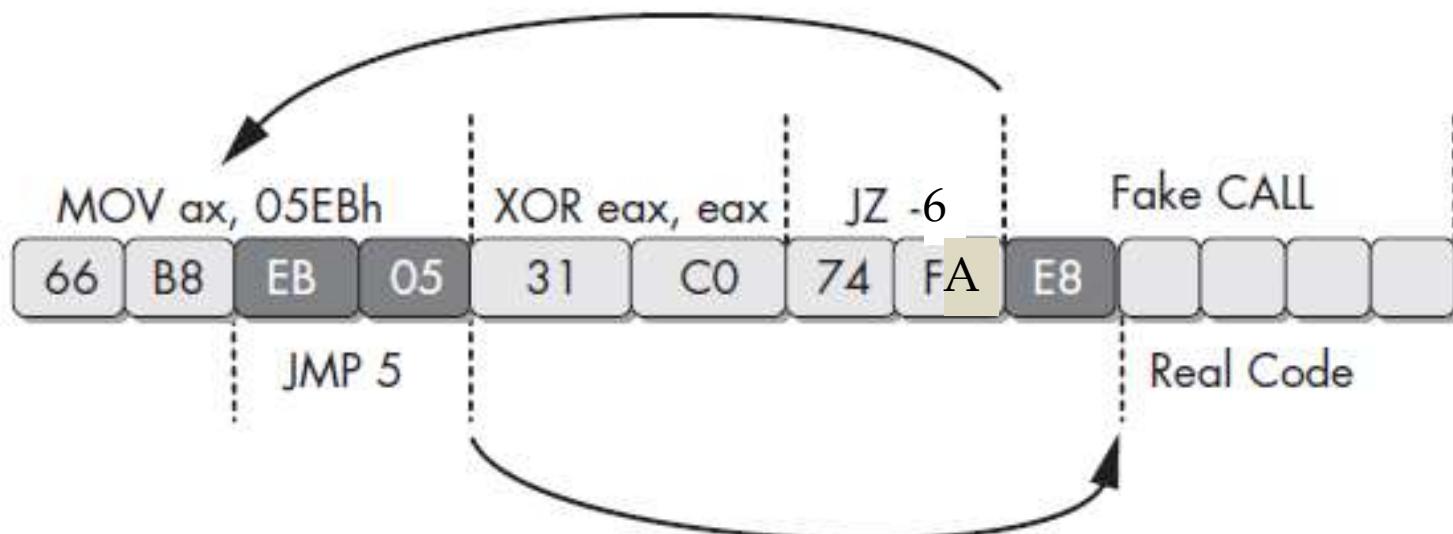
© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020



# גְּרֹזִי-קַיס-סָנוּסָה - מֵפִיכָת קָרֶז

- ומה עושה הקוד הזה?



- ה-*disassembler* לא ידע לתרגם את הפקודות נכון

from Practical Malware Analysis  
עם תיקון



278

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020



# תזכורת: ויכוח אמצעים

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

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



# kr6i-קיס-סואנס, געינא ממיינט

- ניתן לתכנן את התכנית כך שמהלכה יועבר לקוד במקום אחר על ידי חריגות
  - כגון חילוק באפס
  - קלומר, ללא פקودת `jmp` כלשהו



# הנדסה לאחור – חורף תשע"ט

## kncljk

from Practical Malware Analysis

```
00401050      ②mov    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      ③div    ecx
0040106B
0040106B loc_40106B:           ; DATA XREF: sub_4010500
0040106B          call   near ptr Sleep
00401070          retn
00401070 sub_401050          endp ; sp-analysis failed
00401070
00401070 ;
00401071          align 10h
00401080          mov    esp, [esp+8]
00401084          mov    eax, large fs:0
0040108A          mov    eax, [eax]
0040108C          mov    eax, [eax]
0040108E          mov    large fs:0, eax
00401094          add    esp, 8
00401097          push   offset aMysteryCode ; "Mystery Code"
0040109C          call   printf
```



# איך קוראים סיסמה?

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



# Anti-Debugging

- כאשר תוכנה מזזה שהיא רצה תחת כלי דיבוג או בוירטוואלייזציה, היא לעיתים תבחר
  - לעצור את הפעולות
  - לשנות אותה לחלוטין
  - להפעיל קוד לא רלוונטי
  - במטרה לבלב את החוקר ולבזבזו לו זמן יקר
- ישנו מספר הבדלים בין ריצה טبيعית לבין ריצה בסביבת מחקר מסויימת
- Anti-Debugging מתייחס לניצול הבדלים אלו לטובות הזיהוי
  - נציג מספר שיטות בסיסיות וכי怎ד להתמודד מולם
  - ניתן למצוא את הטכניקות בחלק גדול מהנוזקות



# בכריית אפקט אוורט זיהוי

- קראיה ל-API :

- isDebuggerPresent

- CheckRemoteDebuggerPresent

- NtQueryInformationProcess

- זיהוי לפי מדידת זמן

- חיפוש 3 int

- outputDebugString

- בדיקה מי הטליך 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
```

פתרונות : תיקון ה .Flag

```
typedef struct _PEB {
    BYTE             Reserved1[2];
    BYTE             BeingDebugged;
    BYTE             Reserved2[1];
    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;
```



# 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 יישירות 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  
}
```

שולחת מחרוזת להציג ע"י הדיבגר, אם קיים כזה.

אם אין Debugger משוחזרת שגיאה (משנה את ה-`.lastError`)



# ליחוי ספי איזיקט נאן

- ניתן לנצל את הפעולה שריצה ב-debugger איתית יותר,

```
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
```

לדוגמה

- ניתן להשתמש גם ב-API למדידת זמנים
  - כמו `getLocalTime` / `getSystemTime` / `getTickCount` וכיו'



# ליחוי קיימט int 3

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

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



# גזיקה באמצעות int 3

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

```
push handler  
push fs:[0]  
mov fs:[0],esp  
mov eax,debh  
int 3  
pop fs:[0]  
add esp,4  
cmp eax,debh  
je Debugger  
jmp noDebugger  
Handler:  
    mov eax, [esp+0x0c] // get the context record  
    mov [eax+0xb0],100h // eax in context record  
    inc [eax + 0xb8] // eip in context record  
xor eax,eax  
ret
```

- קלומר int 3 זה יוצר חריגה רגילה
- אם אין Debugger אז



# 6.1.1 פונקציית `decode`



291

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020

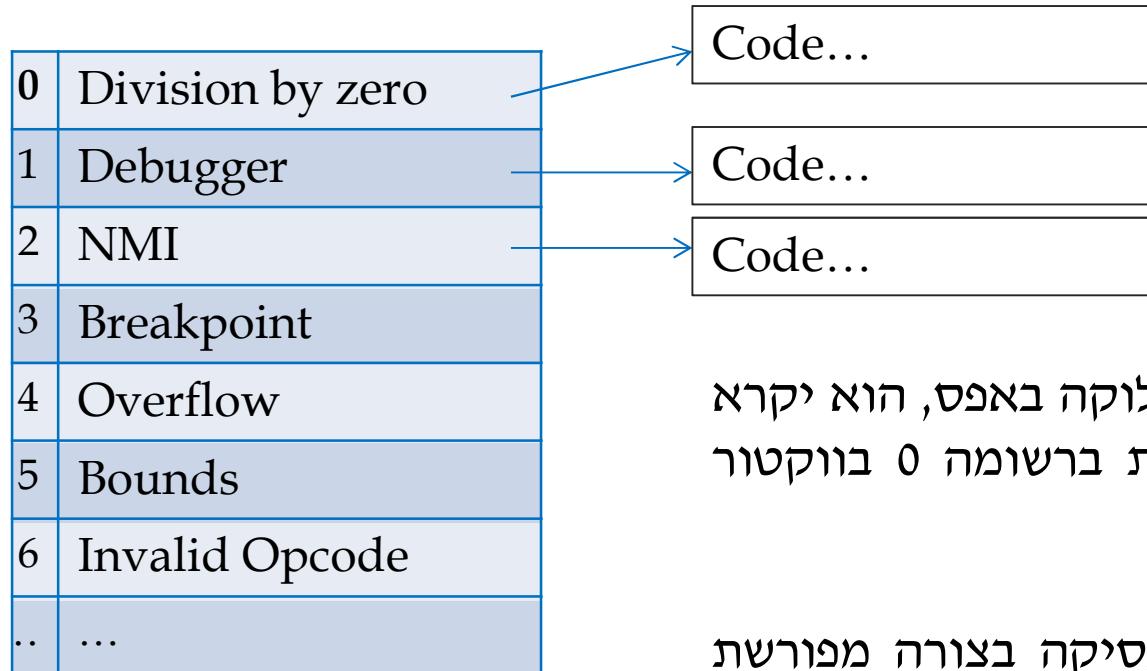


# איך הפסיקו Interrupt Descriptor Table (IDT)

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



# אקורד הפסיקות



- אם המעבד ביצע חלוקה באפס, הוא יקרא לכתובת אשר נמצאת ברשומה 0 בוקטור הפסיקות.
- ניתן גם לקרוא לפסיקה בצורה מפורשת ע"י פקודת האסמבלי `.int val`.
- לדוגמה `.int 0`.



# אפקט הפסיקות

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

(שימוש 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 בפסקה.
  - לכל סוג של חריגה יש רשומה מתאימה בטבלת ה-IDT (ראינו שחלוקת ב-0 זה אינדקס 0 בטבלה).
  - כל שגרה כזו תקורה ל-`kiDispatchException`.



# KiDispatchException

- מדובר בפונקציה המרכזית של מערכת הפעלה לטיפול בחיריגות.
- הפונקציה מ Chapman את ה-`handler` המתאים כפי שמצוין בthread שזרק את החיריגה.
- אם התהיליך נמצא תחת דיבוג הפונקציה קרא קודם ל-`KiDispatchException` הדיבאגר.
- ניתן למצוא מידע נוסף על הפונקציה באתר [.reactos.org](https://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](http://reactos.org)



297

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמיאל

23.01.2020



# קורס ריאסאת

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



# הוֹמֶט רָאֵסְפּוֹט מִלְיכָהָי קַיְסָר

- ניתן לעבור על כל התהליכים באמצעות CreateToolhelp32Snapshot 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 זהה, יש סיבה לחשוד שדייבגר יצר אותו
  - שיטה זאת תגרום לעיתים קרובות ל-false-positive
    - לדוגמה הפעלה דרך cmd



# הוּאָמָת רַוְסֶפֹּאָת בְּלִיְהָאֵן

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

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

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



# ליכוי אזהה ויכוח

- רבים מניטוחי RE של נזקות מבוצעים על מכונות וירטואליות
- לכן רבות מהנזקות משנות את התנהגותן תחת וירטואלייזציה
- תרגיל
  - מצאו שני דרכי לאיתור VMware
    - באמצעות פקודות פשוטות ב-cmd



# ליכוי אזהה ויכוח

- דוגמאות לשוני במחשב וירטואלי

- מחסור בדרייברים בסיסיים
  - כגון כרטיס קול
- קיומם דרייברים ייחודיים
- חומרה מסויימת
- תוכן וקטור הפסיכות
- ערכאים ב-*Registry*
- התנהגות שונה בפקודות מסויימות
  - בעיקר פקודות מורצות על המעבד
- תהליכיים של מערכת הווירטואלייזציה
- שינוי מבנים מסויימים במערכת הפעלה
- זמני ריצה שונים



# ליכוי אזהה ויכוח

- קל לזהות את VMware
  - תהליכי ניהול המחשב הווירטואלי VMwareService, VMwareTray, VMware User
  - כתובת MAC ייחודית
- כדי להקשות על זיהוי, אפשר ליצור patch עם שמות אחרים וכותבות אחרות
  - ניתן להקשות על זיהוי, אך לא למנוע
- ישן תוכנות וירטואלייזציה שיוטר קשה לאתרן
- במידה שרוצים למנוע לחוטין אפשרות לזיהוי, צריך להריץ בסביבה טבעית במחשב ייעודי
  - אבל אז צריך להשקיע יותר בהתקנה חדשה כל פעם שנגרם נזק



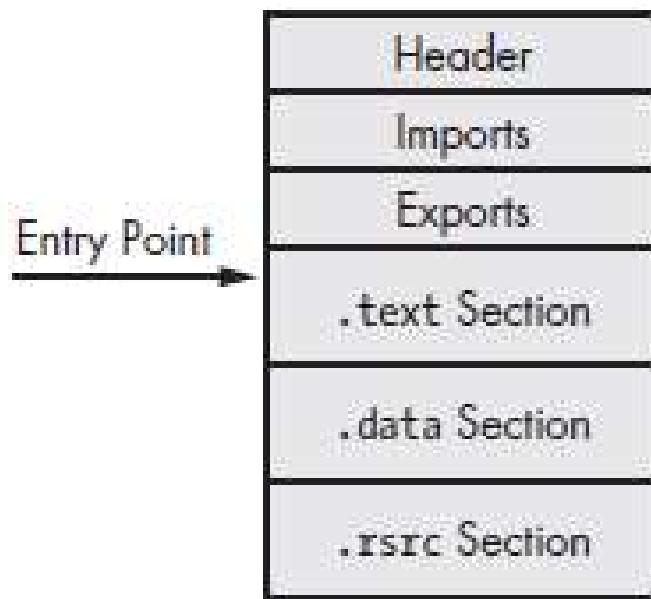
# צמיסת קאף הגדלה

- ישנים כלים שמאפשרים דחיסת תוכנה
  - תוכנה דחוסה, בנוסף לקוד הרגיל, מכיל מותוודה שאחראית על חילוץ הקוד והרצתו
  - במקרה כזה לא נוכל לנטר את התוכנה באופן סטטי
    - שכן הקוד לא חשוף לנו
    - ככלומר לא ניתן להריץ IDA
  - הצפנה היא אחד הכלים המשמשים בזמן דחיסת קובץ ההרצה
    - אבל קוד הפענוח/מפתח נמצא בתוכנה, אחרת לא תוכל לפעול
    - לשם פשטות, השתמש 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**

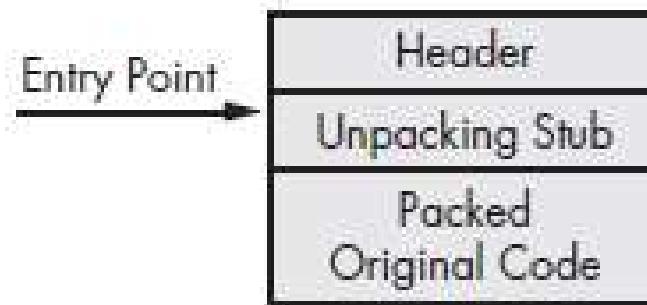


# זמן איסת קאף ההכנתה

תוכנה רגילה



תוכנה דחוסה



# צמיסת קאף הגלגה

## Packing

- הטכנית המכנייה נפוצה כיום למניעת Debugging
- הקוד האמיתי של התוכנה יכול להיות מכוזץ או מוצפן באופן שלא ניתן לפתיחה ללא המתודה הרלוונטית
  - המתודה שאחרראית על הפתיחה תוכל ככל כמה שיותר דרכי שתפקידם להטעות על החוקר
- בתוכנות כאלו ניתן למצוא את מרבית שיטות ה-Anti-debugging הנפוצות (שחלקן אינם בהרצאה)
- ישנים Packers שדריכי החילוץ שלהם מתועדות בצורה נרחבת
  - ויש כאלה ייעודיים שמכללים טרייקים פחות ידועים
  - אין זמן מוגדר להתמודדות
    - ישן נזקoot שדרשו מספר שבועות מחקר עד להשגת הקוד



# קחיסת קאף הרכבה

## UPX מונט

Code?:

|    |       |  |          |
|----|-------|--|----------|
| C0 | DB C0 |  | CHAR 'd' |
| 64 | DB 64 |  | CHAR 'v' |
| B0 | DB B0 |  | CHAR 'r' |
| 2F | DB 2F |  | CHAR 't' |
| D8 | DB D8 |  | CHAR '0' |
| 2E | DB 2E |  | CHAR '3' |
| 72 | DB 72 |  | CHAR 'n' |
| 28 | DB 28 |  | CHAR 'o' |
| 1F | DB 1F |  | CHAR '9' |
| 30 | DB 30 |  | CHAR '8' |
| 27 | DB 27 |  | CHAR '7' |
| 12 | DB 12 |  | CHAR '=' |
| F2 | DB F2 |  | CHAR '4' |
| 3D | DB 3D |  | CHAR '5' |
| C7 | DB C7 |  | CHAR '6' |
| 8F | DB 8F |  | CHAR 'B' |
| 60 | DB 60 |  | CHAR 'E' |
| 40 | DB 40 |  | CHAR '@' |
| 2E | DB 2E |  | CHAR 'A' |
| 62 | DB 62 |  | CHAR 'B' |
| 62 | DB 62 |  | CHAR 'B' |
| 5B | DB 5B |  | CHAR 'C' |
| 9B | DB 9B |  | CHAR 'D' |
| 40 | DB 40 |  | CHAR 'E' |
| 30 | DB 30 |  | CHAR 'O' |
| D9 | DB D9 |  | CHAR 'P' |
| DA | DB DA |  | CHAR 'W' |
| 05 | DB 05 |  | CHAR '4' |
| 80 | DB 80 |  | CHAR 'B' |
| 18 | DB 18 |  | CHAR 'S' |
| 4F | DB 4F |  | CHAR 'H' |
| 69 | DB 69 |  |          |
| 60 | DB 60 |  |          |
| 65 | DB 65 |  |          |
| 83 | DB 83 |  |          |
| 8D | DB 8D |  |          |
| 1B | DB 1B |  |          |
| 50 | DB 50 |  |          |
| 9B | DB 9B |  |          |
| 14 | DB 14 |  |          |
| 77 | DB 77 |  |          |
| 00 | DB 00 |  |          |
| 34 | DB 34 |  |          |
| 62 | DB 62 |  |          |
| B0 | DB B0 |  |          |
| 53 | DB 53 |  |          |
| 48 | DB 48 |  |          |
| 07 | DB 07 |  |          |

No strings:

|          |                  |
|----------|------------------|
| 004076C2 | ASCII "cv"       |
| 004076F5 | ASCII "siz"      |
| 004077FA | ASCII "4JT\fp",0 |
| 00407810 | ASCII "O6.",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 |

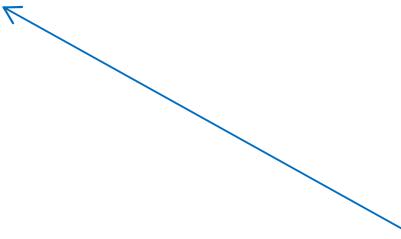


# 谦謙的UPX解壓

```

004079B0: 5 60 PUSHAD
004079B1: . BE 15704000 MOV ESI,crackV2.00407015
004079B6: . 8DBE EB9FFFFF LEH EDI,DWORD PTR DS:[ESI+FFFF9FEB]
004079C0: . 57 PUSH EDI
004079D0: . EB 0B JMP SHORT crackV2.004079CA
004079E0: . 90 NOP
004079F0: > 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
004079C0: . 11DB ADC EBX,EBX
004079D1: >> 72 ED JB SHORT crackV2.004079C0
004079D3: . B8 01000000 MOV EAX,1
004079D8: > 01DB ADD EBX,EBX
004079D9: . 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 00 JB SHORT crackV2.00407A08
004079FB: . C1E9 08 SHL EAX,8
004079FE: . 8A06 MOV AL,BYTE PTR DS:[ESI]
00407A00: . 46 INC ESI
00407A01: . 83F0 FF XOR EAX,FFFFFF
00407A04: . 74 74 JE SHORT crackV2.00407A7A
00407A06: . 89C5 MOU 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
00407A36: . 75 00 JNZ SHORT crackV2.00407A41

```



הקוד הראשון שרצ :

ניתן לראות בבירור שלא מדובר במשהו  
טריומיאלי שנרצה לחקור.



# 谦卑的UPX解密

```
00407B0C : 53      PUSH EBP
00407B0D .: FFD5    CALL EBP
00407B0F .: 8D87 9F010000 LEA EAX,DWORD PTR DS:[EBP+19F]
00407B15 .: 8020 7F AND BYTE PTR DS:[EAX],?F
00407B18 .: 8060 28 ?F AND BYTE PTR DS:[EAX+28],?F
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.

קפייה לקוד (לאחר החילוץ).



# זמן אמת קואץ הגדלה

## מיינוף הקואץ הפתוח

- כאשר יש לנו כתובת (בזיכרונו) 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
  - תזכורות : ל-Import table יש שני מצבים
    1. לפני פתיחת הקובץ – יכול טבלה עם שמות DLLs ופונקציות של מערכת הפעלה לטוען
    2. לאחר פתיחת הקובץ – יכול טבלה עם כתובות. לא חייב להכיל שמות (לעתים Packers) ימחקו את השמות כחלק מתהליך ה-Anti-Debugging
  - יש צורך לעבור ממצב 2 למצב 1, למשל לעשות "Reverse" ל- Import table
    - במקרים מסוימים נצטרך לאתר בצורה ידנית את שמות הפונקציות ולתקן את ה-IAT בהתאם
    - במקרים אחרים ישן תוכנות אוטומטיות שעושות עבורנו את העבודה



# חומר קואץ הצלחה

## סיכום

- ראשית נבדוק שאכן הקובץ דחוס/מוצפן
  - בדרך כלל מחסור ב-**Import Table/Strings** מרמז על כך
- ננסה לẤתר את ה-**Packer**
  - האיתור יחשוף את הדרכן הנכונה להתחמיזות מולו
- במידה שה-**Packer** לא מוכר, נשתמש בכלים שלנו
  - נẤתר שימוש ב-**Anti-Debugging** וננסה למנוע את מרבית השיטות
    - ע"י Hook, דיבוג איטי או דרכי אחרות
    - המטרה שלנו היא לתת למתקה המחלצת לרווח
    - 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



# גְּפָרָת אַקְזֹאת אֲכֹאֶה

## fərət əkzəət əkəəh

- אינטל מאפשרים במסגרת SGX להציג קוד בסביבה בה
  - הזיכרון מוצפן כולו
    - כלומר כל פקודות המכונה והמשתנים מגיעים לمعالג מוצפנים
  - התקשורת פנימית והחוצה מוצפנת
- כך שגם האזנה על העורוץ הפנימי של המחשב לא תסגיר דבר על מה שמתבצע
- זה לא מיועד לחישוב כללי, אלא לנסיבות ייעודיות
  - בהן נדרש קריפטוגרפיה להגנה על מידע על בעליים מרוחק
  - למשל הפעלה אינטראקטיבית של שירותי בעלי זכויות יוצרים
- ל-AMD יש ארכיטקטורה משלهم שעשוה דבר דומה



# סיכום

- ראיינו בהרצאה מספר טכניות נפוצות להתמודדות נגד RE
- אין נוסחה או דרך אחרת להתמודד נגד
  - יש להיות יצירתיים ועקשנים
- כאשר נתקלים במכשול, כדאי לבדוק האם מישחו נתקל בזה לנו
  - למשל בגוגל
- כמשמעותי שעות רבות על הבנת קוד, כדאי לשאול האם להמשיך, או למצוא דרך אחרת
  - לא נרצה למשל לחקור Packer במקומם לחקור את החלק שמעוניין אותנו
- הבנת חלקים שונים במערכת הפעלה (Internals ו-API) הינה הכרחית
- מדובר על תחום מאד רחב שהווצג רק בחלוקת
  - מאד מועיל להכיר טכניות נוספות וכליים אוטומטיים



# פרק 9

## הסוללה כאלקטרומ

פיזי ונוירולוג פיזי



# רָוְקָאָת: אִיכָּסֵיָת וְחַיָּת מְכֹאָת

פצצות  
לוגיות

סוסים  
טרויאניים

ביצי  
פסחא

קופר

ボוטנטים

רוגלות

תוכנות  
פרטים

Rootkits

דלטות  
סתרים

תולעים

וירוסים

כיום מקובל לקראה לכולם וירוסים  
או נזקota



315

הנדסה לאחור – חורף תשע"ט

© פרופ' אלי ביהם, אביעד כרמל, עמר קדמיאל

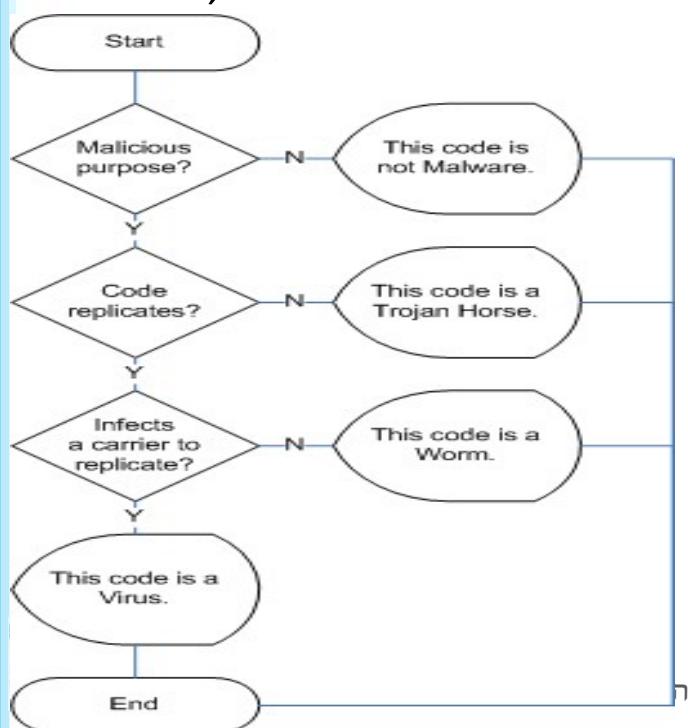
23.01.2020



# נוֹזָקָה

- נוֹזָקָה הינה מונח כוללני המתיחס למגוון שונים של תוכנה עוינת או מזיקה, ובכלל זאת וירוסים, תולעים, סוסים טרояניים, סוגים פרסומות, נזקות כופר, נזקות שמטרתן להפחיד את המשתמש ועוד.

- נוֹזָקָה יכולה להיות קוד בינהי, סкриיפט, active content ועוד. **הנוֹזָקָה מוגדרת על ידי כוונתה העוינית,** והתנагותה כנגד המשתמש. לפיכך, המונח נזק אינו כולל תוכנה אשר גורמת נזק ללא כוונה.



- נוֹזָקָה יכולה להיות מכוונת לתפוצה רחבה, או להיות ממוקדת ומכוונת למטרה ספציפית...

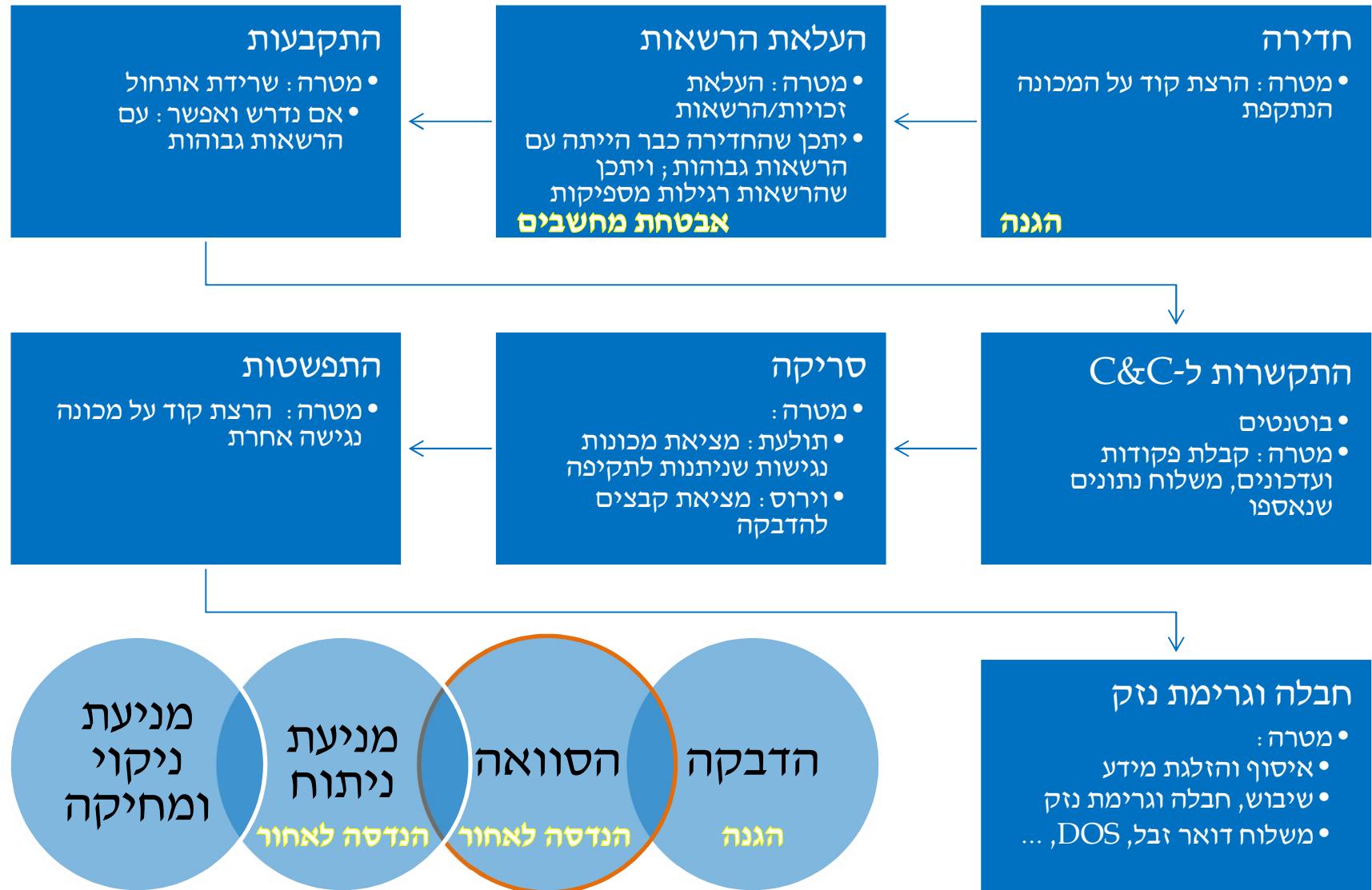


# כאה הפקה

- **חדרה**
  - הפעלה ראשונית של קוד זדוני במחשב
  - נועה באמצעות ווקטורית תקיפה
- **התקבעות**
  - הבטחת המשך הפעלה העתידית של הנזקה
    - גם אחרי אתחול
  - הפיכה לשווה
- **הדבקה**
  - שינוי קבצים או מידע אחר
    - שמאפשר בעקבותיו להדביך מחשבים נוספים
    - או שמאפשר לנזקה להתקבע
    - או להיות מורצת בהמשך
  - המונח בשימוש גם במובן של חדרה
- **הסואה**
  - הסתרות באופן שלא מאפשר לזהות את קיומם של הנזקה



# אקרה כפוי של אבטחת רוזקה



# הקלים לאן כי

- נזקה יכולה לרוץ במחשב במגוון הקשרים
  - כתהlixir
  - כדריבר
    - או DLL
    - או ברכיב של מ"ה
- קובץ הרצה
  - או ע"י הדבקת קובץ הרצה קיים
  - בתקופה שימושה יריץ אותו
- ב-DOS : TSR (terminate and stay resident)
  - תוך כדי השתלטות על וקטור הפסיקות
- כ-plug-in של תוכנה אחרת
  - או סקריפט שמורץ על ידי תוכנה אחרת
- בשילד של תהליך אחר
  - אם פרץ אליו, למשל ע"י חירגה מחווצץ
- ואפילו כ-hypervisor
  - ולהריץ את מ"ה תחתיה
- וגם שילובים שלהם



# הצקת קובץ ה-CODE

התקנה

• הדבקת קובץ הרצה כוללת כמה חלקים:

▪ הוספת קוד לקובץ ההרצתה.

○ ע"י הוספת section נוסף.

○ או ע"י הגדלת section קיימים.

○ או שימוש במקום פנווי ב-section קיימים.

▪ חלק מהוירוסים דוחסים את תוכן הקובץ.

○ כדי לשמור על גודלו ללא שינוי.

▪ הוירוס צריך לפתח 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.

▪ הפנית החישוב לקוד שנוסף.

○ על ידי שינוי כתובות תחילת הריצה בקובץ ההרצתה.

○ או על ידי ביצוע הוקינג מתחילת התכנית המקורי.

▪ הבתחת המשך פעולה הוירוס בזמן ריצת התכנית.

○ אם הוירוס מעוניין בכך.

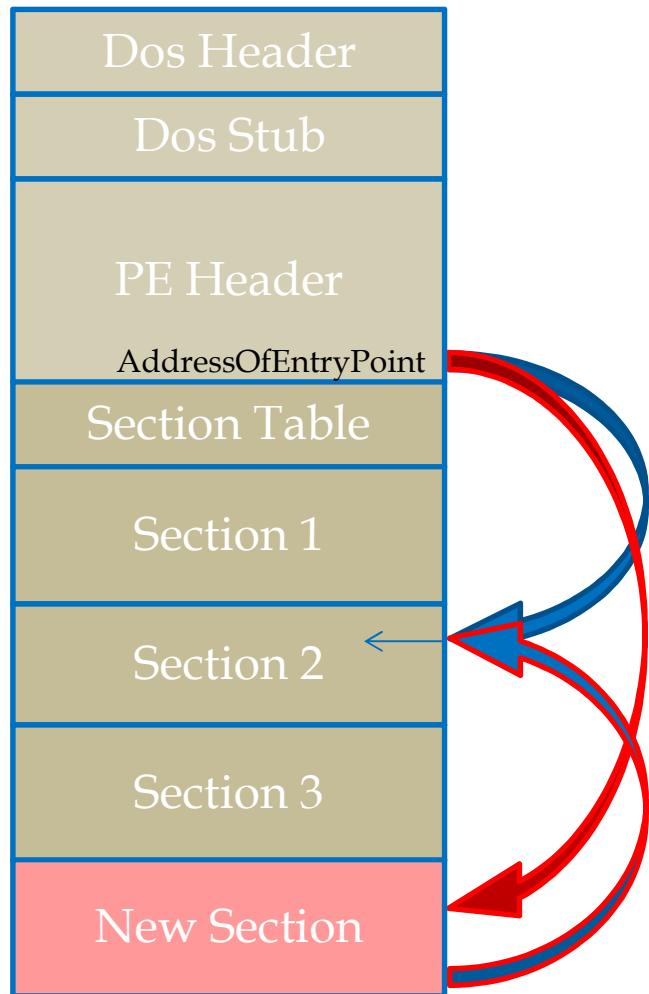
○ על ידי השתלטות על וקטור הפסיקות.

○ או הוקינג לקריאות מ"ה.

○ או הוקינג לקטעי קוד מרכזיים בתכנית.



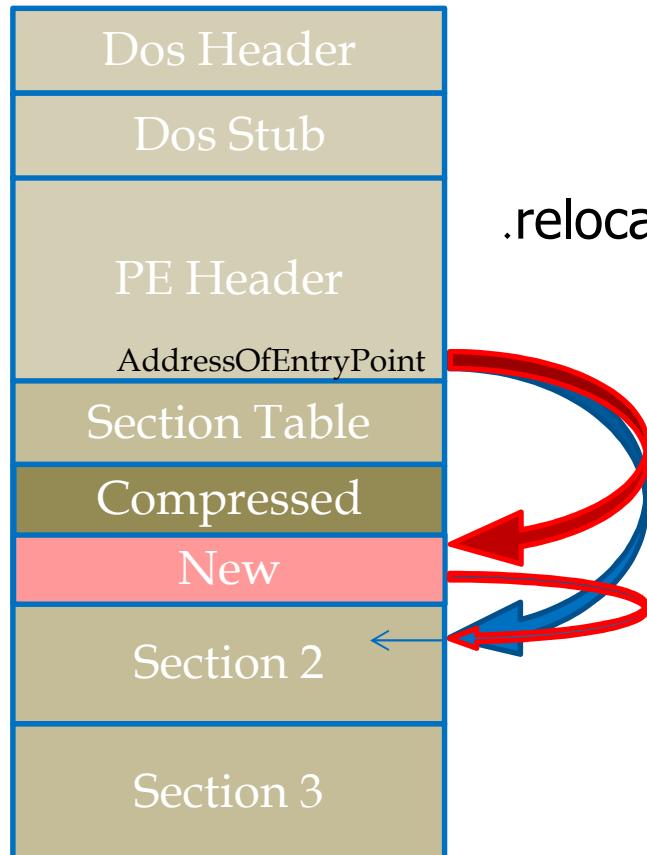
# התקנת קאדי הכללה



- קובץ הרצה מקורי.
- הוספת Section חדש.
  - שכולל את הקוד המזיך.
  - קוד SMBTICH המשך פעילות.
    - כאשר עוברים לתוכנית המקורית.
- הפניות כתובות תחילת הריצה.
- וקפיצה לקוד המקורי.



# הפקת קאדי הלה אם ? מיסה



- 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 אותו גודל שהוא.
  - כולל קוד פתיחת הדחיסה.
  - כולל טיפול ב-relocation, אם צריך.
  - ואת הקוד המזijk.
  - וכן שmbטיח המשך פעילות.
  - כאשר עוברים לתוכנית המקורית.
- הפניות כתובות תחילת הריצה.
- וקפיצה לקוד המקורי.
- בזמן ריצה:
  - העתקת קוד הוירוס למקום פנוי, ופתיחה הדחיסה.

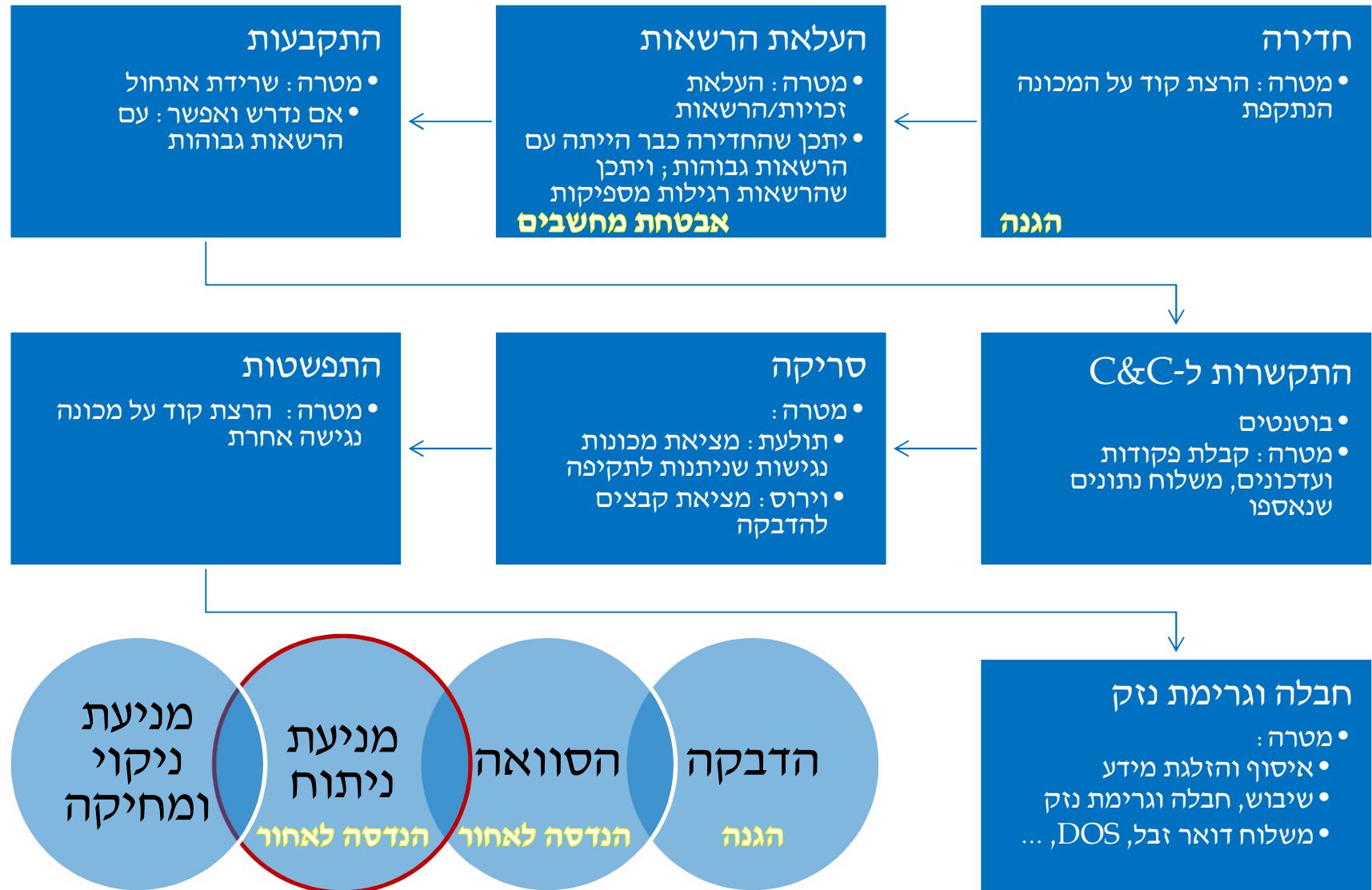


# RootKits

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



# אקרה כפוי של אבטחת רוזקה



# Anti-Reversing

- לתוכנות מסויימות, כגון נזקות, יש צורך בהשתרת הפעולות מפני גורמים מסויימים, ביניהם
  - כלים סריקה אוטומטיים, כדוגמת אנטי-וירוסים, מסוגלים לגלוות קוד עוין ע"י חתימות
    - או ע"י זיהוי דפוס התנהגות מסוים שנחשב שלא טריוניAli
  - חוקרים שתפקידם למנוע את פעילות התוכנה
    - בדרך כלל מדובר בכאלו שעוסקים בפיתוח אמצעי נגד
  - חברות מתחרות שמנסחות לגנוב קוד
- **Anti-Reversing** הוא אוסף של טכניקות שמטרתו להקשות על גילוי התוכנה ועל ביצוע RE
  - אין אפשרות מוחלטת למנוע RE
  - אך אפשר לבלב את החוקר, לבזבז את זמןו, ולהפריע לו
  - שיטות הגנה מתחכਮות דורשות חוקרים יותר מנוסים והמון זמן מחקר
- **אנו נדונן בטכניקות אלה, ונתינחס גם להתמודדות מולן**



# רכז'יקאט Anti-Reversing

- ניתן לחלק את השיטות לכמה סוגים
  - Obfuscation
    - טכניות להקשנות על הקורא להבין את הקוד, או לעכב אותו
  - Anti-Disassembly
    - טכניות למנוע את הצגת הקוד האמיתי לחוקר או סורק
  - דחיסת והצפנה הקוד בקובץ הרצה
- ניתן כמה רעיונות בסיסיים על מנת להציג את הנושא
  - כל רעיון בסיסי ניתן להרחבה לשיטה מתקדמת יותר



# הנלה? רקיקת מהו נאה?

3.141 תשובה :



# הקיא ? הלא ?

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

**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, brainsck, 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 بصورة קשה להבנה
- אפשר לבצע obfuscation גם בשפה עילית וגם באסמבלי
- דוגמאות
  - כתיבה מסובכת של קוד פשוט
    - כל מי שבודק תרגילים נתקל בקוד כזה ☺
  - משפטים תנאי וולאوت בהן לא ברור לקורא איזה קוד יבוצע
  - הוספת "garbage code" – קוד שאינו חלק רלוונטי מפעולות התוכנה, אך עשוי לבלום שעת לחוקר
  - הצפנה/דחיסה של הקוד



# Obfuscation-פְּנִימָה כַּאֲמֵת?

```
#include <Windows.h>

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

void main()
{
    _asm
    {
        pushad

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

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

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

        popad
    }
}
```

From “leetMatrix” blog



דוגמא פשוטה ב-C:

```
static int two=1;

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



# תלכארט: DIS-ASMBLER

- כזכור, DIS-ASMBLER 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++;  
    }  
}
```



# jk-קיס-סמאגי

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



332



# jk - קיס-נאנס – כינוי נאנו-וירטואלי

- מה עושה הקוד הבא?

```
00401328 E8 C7042424 CALL 246417F4
0040132D 3040 00 XOR BYTE PTR DS:[EAX],AL
00401330 E8 A7060000 CALL <JMP.&msvort.printf>
00401335 807424 1E LEA ESI,DWORD PTR SS:[ESP+1E]
```

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

- והקוד הזה?

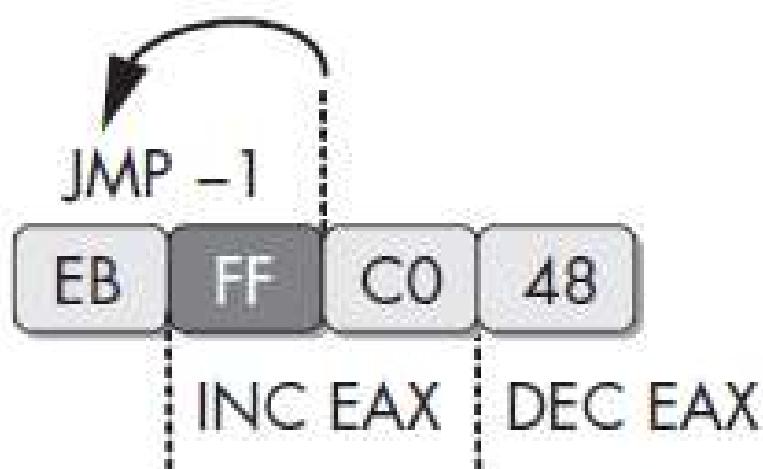
```
00401329 C70424 24304000 MOV DWORD PTR SS:[ESP],crackV2.00403024 ASCII "Enter the password:"
00401330 E8 H76666666 CALL <JMP.&msvort.printf>
00401335 807424 1E LEA ESI,DWORD PTR SS:[ESP+1E]
```

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



# jk - ?ok-SANdja - מפיהם קיז

- מה עושה הקוד הבא?



- זה למעשה (כמעט) פקודת סוף בת 4 בתים

- קפוץ אחורה בית אחד, לקוד חופף (2 בתים : EBFF )
- הגדיל את EAX באחד ( 2 בתים : FFC0 )
- הקטן את EAX באחד (בית אחד : 48 )
  - זה אינו סוף בגלל ההשפעה על הדגמים

from Practical Malware Analysis



334

הנדסה לאחור – חורף תשע"ט

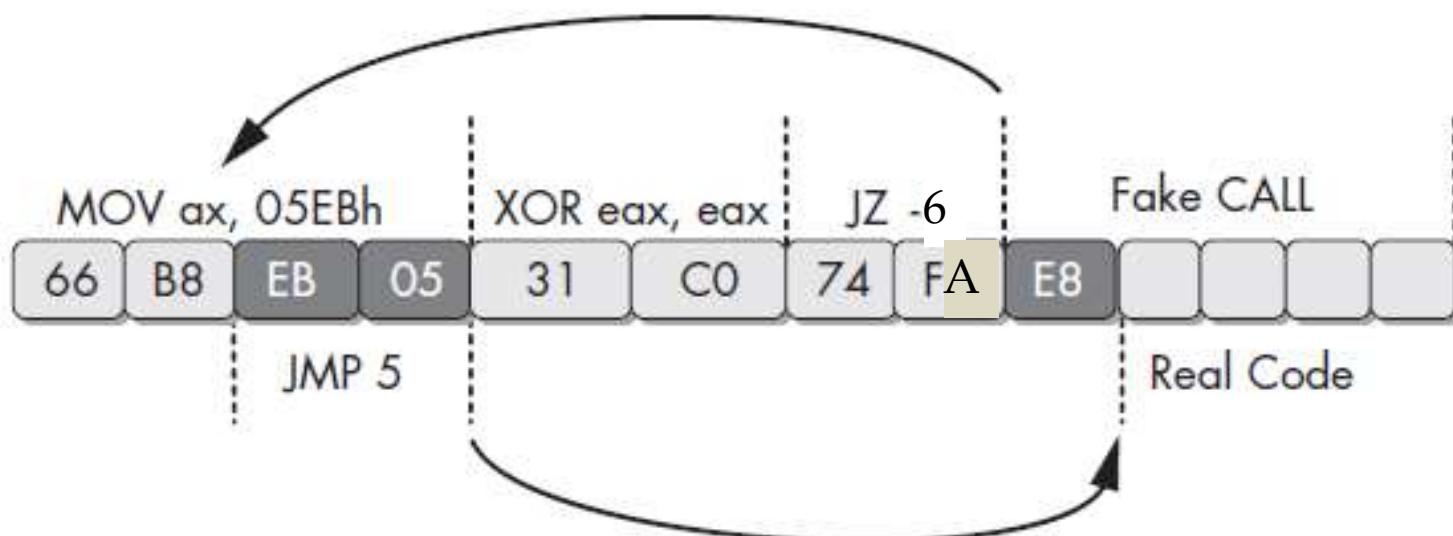
© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020



# גְּרֹזִי-קַיס-סָנוּסָה - מֵפִיכָת קָאָז

- ומה עושה הקוד זהה?



- ה-*disassembler* לא ידע לתרגם את הפקודות נכון

from Practical Malware Analysis  
עם תיקון



335

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020



# תזכורת: ויכוח אמצעים

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

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



# kr6i-קיס-סואנס, געינא ממיינט

- ניתן לתכנן את התכנית כך שמהלכה יועבר לקוד במקום אחר על ידי חריגות
  - כגון חילוק באפס
  - קלומר, ללא פקودת `jmp` כלשהו



337

הנדסה לאחר – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020



# הנדסה לאחור – חורף תשע"ט

## kncljk

from Practical Malware Analysis

```
00401050      ②mov    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      ③div    ecx
0040106B
0040106B loc_40106B:           ; DATA XREF: sub_4010500
0040106B          call   near ptr Sleep
00401070          retn
00401070 sub_401050          endp ; sp-analysis failed
00401070
00401070 ;
00401071          align 10h
00401080          mov    esp, [esp+8]
00401084          mov    eax, large fs:0
0040108A          mov    eax, [eax]
0040108C          mov    eax, [eax]
0040108E          mov    large fs:0, eax
00401094          add    esp, 8
00401097          push   offset aMysteryCode ; "Mystery Code"
0040109C          call   printf
```



# איך קוראים סיס-סאק?

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



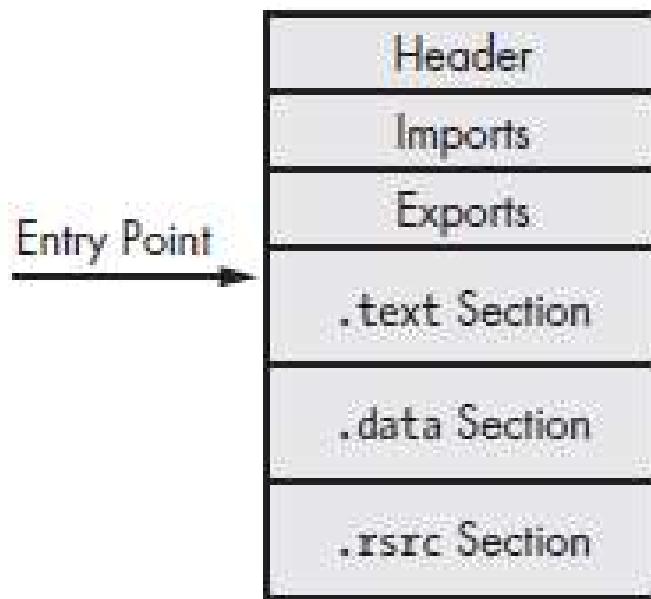
# צמיסת קאף הגדלה

- ישנים כלים שמאפשרים דחיסת תוכנה
  - תוכנה דחוסה, בנוסף לקוד הרגיל, מכיל מותוודה שאחראית על חילוץ הקוד והרצתו
  - במקרה כזה לא נוכל לנטר את התוכנה באופן סטטי
    - שכן הקוד לא חשוף לנו
    - ככלומר לא ניתן להריץ IDA
  - הצפנה היא אחד הכלים המשמשים בזמן דחיסת קובץ ההרצה
    - אבל קוד הפענוח/מפתח נמצא בתוכנה, אחרת לא תוכל לפענה
    - לשם פשטות, השתמש 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**

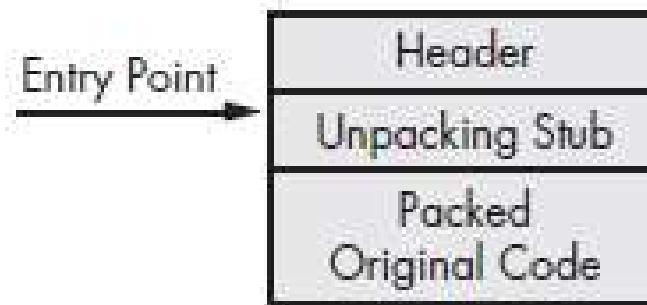


# זמן איסת קאף הרכבה

תוכנה רגילה



תוכנה דחוסה



# צמיסת קאף הילגה

## Packing

- הטכנית הבי נפוצה כיום למניעת Debugging
- הקוד האמיתי של התוכנה יכול להיות מכוזץ או מוצפן באופן שלא ניתן לפתיחה ללא המתודה הרלוונטית
  - המתודה שאחרראית על הפתיחה תכיל כמה שיותר דרכי שתפקידם להטעות על החוקר
- בתוכנות כאלו ניתן למצוא את מרבית שיטות ה-Anti-debugging הנפוצות (שחלקן אינם בהרצאה)
- ישנים Packers שדריכי החילוץ שלהם מתועדות בצורה נרחבת
  - ויש כאלה ייעודיים שמכללים טרייקים פחות ידועים
  - אין זמן מוגדר להתמודדות
    - ישן נזקوت שדרשו מספר שבועות מחקר עד להשגת הקוד



# קחיסת קאף הרכבה

## UPX מונט

Code?:

|    |       |  |          |
|----|-------|--|----------|
| C0 | DB C0 |  | CHAR 'd' |
| 64 | DB 64 |  | CHAR 'v' |
| B0 | DB B0 |  | CHAR 'r' |
| 2F | DB 2F |  | CHAR 't' |
| D8 | DB D8 |  | CHAR '0' |
| 2E | DB 2E |  | CHAR '3' |
| 72 | DB 72 |  | CHAR 'f' |
| 28 | DB 28 |  | CHAR 'l' |
| 1F | DB 1F |  | CHAR 'o' |
| 30 | DB 30 |  | CHAR '9' |
| 27 | DB 27 |  | CHAR '6' |
| 12 | DB 12 |  | CHAR '=' |
| F2 | DB F2 |  | CHAR '4' |
| 3D | DB 3D |  | CHAR '@' |
| C7 | DB C7 |  | CHAR '1' |
| 8F | DB 8F |  | CHAR 'b' |
| 60 | DB 60 |  | CHAR 'b' |
| 40 | DB 40 |  | CHAR 'b' |
| 2E | DB 2E |  | CHAR 'b' |
| 62 | DB 62 |  | CHAR 'c' |
| 62 | DB 62 |  | CHAR 'e' |
| 5B | DB 5B |  | CHAR 'e' |
| 9B | DB 9B |  | CHAR 'e' |
| 40 | DB 40 |  | CHAR 'e' |
| 30 | DB 30 |  | CHAR '0' |
| D9 | DB D9 |  |          |
| DA | DB DA |  |          |
| 05 | DB 05 |  |          |
| 80 | DB 80 |  |          |
| 18 | DB 18 |  |          |
| 4F | DB 4F |  |          |
| 69 | DB 69 |  |          |
| 60 | DB 60 |  |          |
| 65 | DB 65 |  |          |
| 83 | DB 83 |  |          |
| 8D | DB 8D |  |          |
| 1B | DB 1B |  |          |
| 50 | DB 50 |  |          |
| 9B | DB 9B |  |          |
| 14 | DB 14 |  |          |
| 77 | DB 77 |  |          |
| 00 | DB 00 |  |          |
| 34 | DB 34 |  |          |
| 62 | DB 62 |  |          |
| B0 | DB B0 |  |          |
| 53 | DB 53 |  |          |
| 48 | DB 48 |  |          |
| 07 | DB 07 |  |          |

No strings:

|          |                  |
|----------|------------------|
| 004076C2 | ASCII "cv"       |
| 004076F5 | ASCII "siz"      |
| 004077FA | ASCII "4JT\fp",0 |
| 00407810 | ASCII "O6.",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 |



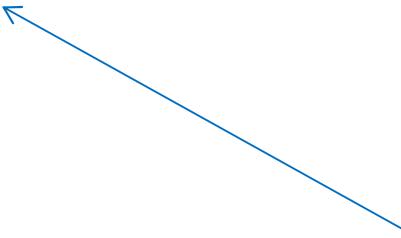
# 谦謙的解密

## UPX 加密

```

004079B0: 5 60           PUSHAD
004079B1: . BE 15704000  MOV ESI,crackV2.00407015
004079B2: . 8DBE EB9FFFFF LEA EDI,DWORD PTR DS:[ESI+FFFF9FEB]
004079B3: . 57             PUSH EDI
004079B4: . EB 0B          JMP SHORT crackV2.004079CA
004079B5: . 90             NOP
004079B6: > 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
004079C0: . 11DB          ADC EBX,EBX
004079D1: >> 72 ED        JB SHORT crackV2.004079C0
004079D3: . B8 01000000  MOV EAX,1
004079D8: > 01DB          ADD EBX,EBX
004079D9: . 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 00          JB SHORT crackV2.00407A08
004079F8: . C1E9 08        SHL EAX,8
004079F6: . 8A06          MOV AL,BYTE PTR DS:[ESI]
00407A00: . 46             INC ESI
00407A01: . 83F0 FF        XOR EAX,FFFFFF
00407A04: . 74 74          JE SHORT crackV2.00407A7A
00407A06: . 89C5          MOU 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
00407A34: . 75 00          JNZ SHORT crackV2.00407A41

```



הקוד הראשון שרצ :

ניתן לראות בבירור שלא מדובר במשהו  
טריומיאלי שנרצה לחקור.



# 谦卑的UPX解密

```
00407B0C : 53      PUSH EBP
00407B0D .: FFD5    CALL EBP
00407B0F .: 8D87 9F010000 LEA EAX,DWORD PTR DS:[EBP+19F]
00407B15 .: 8020 7F AND BYTE PTR DS:[EAX],?F
00407B18 .: 8060 28 ?F AND BYTE PTR DS:[EAX+28],?F
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.

קפייה לקוד (לאחר החילוץ).



# חישות קבוצה הרכבת מיון הקבוצות

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



# חומר קואץ הצלחה

## סיכום

- ראשית נבדוק שאכן הקובץ דחוס/מוצפן
  - בדרך כלל מחסור ב-**Import Table/Strings** מרמז על כך
- ננסה לẤתר את ה-**Packer**
  - האיתור יחשוף את הדרכן הנכונה להתחמיזות מולו
- במידה שה-**Packer** לא מוכר, נשתמש בכלים שלנו
  - נẤתר שימוש ב-**Anti-Debugging** וננסה למנוע את מרבית השיטות
    - ע"י Hook, דיבוג איטי או דרכי אחרות
    - המטרה שלנו היא לתת למетодה המחלצת לרווח
    - 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



?nifke



# Anti-Debugging

- כאשר תוכנה מזזה שהיא רצה תחת כלי דיבוג או בוירטוואלייזציה, היא לעיתים תבחר
  - לעצור את הפעולות
  - לשנות אותה לחלוטין
  - להפעיל קוד לא רלוונטי
  - במטרה לבלב את החוקר ולבזבזו לו זמן יקר
- ישנו מספר הבדלים בין ריצה טبيعית לבין ריצה בסביבת מחקר מסויימת
- Anti-Debugging מתייחס לניצול הבדלים אלו לטובות הזיהוי
  - נציג מספר שיטות בסיסיות וכי怎ד להתמודד מולם
  - ניתן למצוא את הטכניקות בחלק גדול מהנוזקות



# בכריית אפקט זיהוי קידום

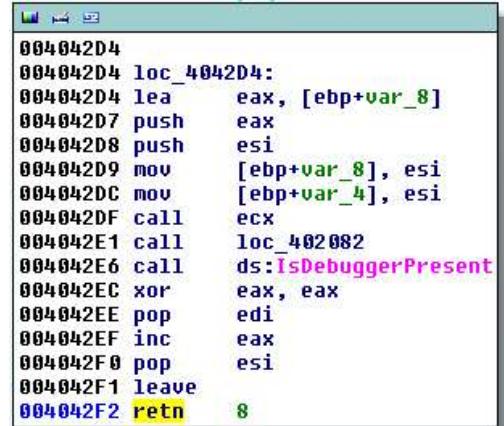
- קראיה ל-API:
  - isDebuggerPresent
  - CheckRemoteDebuggerPresent
  - NtQueryInformationProcess
- זיהוי לפי מדידת זמן
- חיפוש 3 int
- outputDebugString
- בדיקה מי התחליק האב (Parent Process)
- בדיקת קיום דיבגר
- זיהוי breakpoints



# 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
```

הפתרון : תיקון ה .Flag

```
typedef struct _PEB {
    BYTE             Reserved1[2];
    BYTE             BeingDebugged;
    BYTE             Reserved2[1];
    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;
```



# 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 יישירות 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  
}
```

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

אם אין Debugger משוחזרת שגיאה (משנה את ה-`.lastError`)



# ליחוי ספי אקייזט נאן

- ניתן לנצל את הפעולה שריצה ב-debugger איתית יותר,

```
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
```

לדוגמה

- ניתן להשתמש גם ב-API למדידת זמנים
  - כמו `getLocalTime` / `getSystemTime` / `getTickCount` וכיו'



# ליחוי קיימט int 3

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

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



# גזיקה באמצעות int 3

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

```
push handler  
push fs:[0]  
mov fs:[0],esp  
mov eax,debh  
int 3  
pop fs:[0]  
add esp,4  
cmp eax,debh  
je Debugger  
jmp noDebugger  
Handler:  
    mov eax, [esp+0x0c] // get the context record  
    mov [eax+0xb0],100h // eax in context record  
    inc [eax + 0xb8] // eip in context record  
xor eax,eax  
ret
```

- קלומר 3 זה שהוא软件 breakpoint
- אם אין Debugger אז שימוש ב-3 יוצר חריגה רגילה



# הוֹמֶט רָיסְפּוֹט מִלְיכָהָי קַיְסָר

- ניתן לעבור על כל התהליכים באמצעות CreateToolhelp32Snapshot 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 זהה, יש סיבה לחשוד שדייבגר יצר אותו
  - שיטה זאת תגרום לעיתים קרובות ל-false-positive
    - לדוגמה הפעלה דרך cmd



# הוּאָמָת רַוְסֶפֹּאָת בְּלִיְהָוֵי קַיְמָאָן

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

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

- כמובן שלא מדובר בשיטות מדויקות
- השיטות הללו נפוצות בעיקר לאיתור תוכנות שונות שאינן חלק מהדייבגר
  - שכן אין דרך אחרת לאתרו
  - לעומת זאת Debugger שראינו רק חלק קטן מהשיטות לאתרו



# ליכוי אזהה ויכוח

- רבים מניטוחי RE של נזקות מבוצעים על מכונות וירטואליות
- לכן רבות מהנזקות משנות את התנהגותן תחת וירטואלייזציה
- תרגיל
  - מצאו שני דרכי לאיתור VMware
    - באמצעות פקודות פשוטות ב-cmd



359



# ליכוי אזהה ויכוח

- דוגמאות לשוני במחשב וירטואלי

- מחסור בדרייברים בסיסיים
  - כגון כרטיס קול
- קיומם דרייברים ייחודיים
- חומרה מסויימת
- תוכן וקטור הפסיכות
- ערכאים ב-*Registry*
- התנהגות שונה בפקודות מסויימות
  - בעיקר פקודות מורצות על המעבד
- תהליכיים של מערכת הווירטואלייזציה
- שינוי מבנים מסויימים במערכת הפעלה
- זמני ריצה שונים

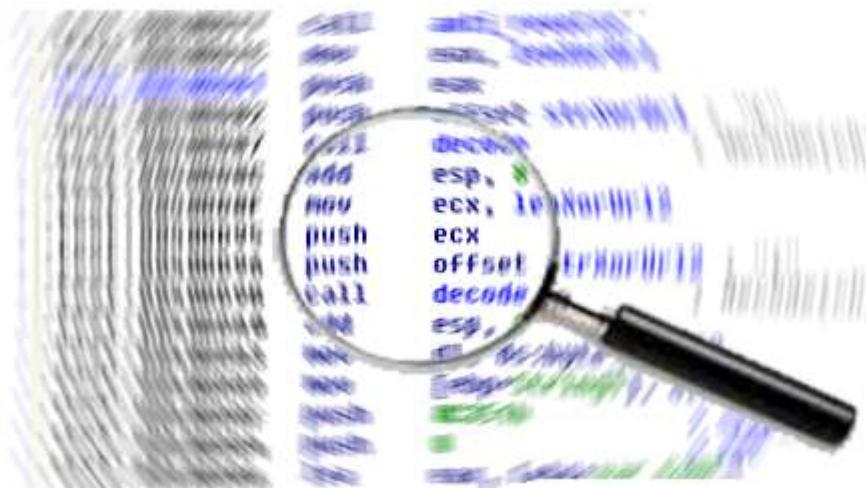


# ליכוי אזהה ויכוח

- קל לזהות את VMware
  - תהליכי ניהול המחשב הווירטואלי VMwareService, VMwareTray, VMware User
  - כתובת MAC ייחודית
- כדי להקשות על זיהוי, אפשר ליצור patch עם שמות אחרים וכותבות אחרות
  - ניתן להקשות על זיהוי, אך לא למנוע
- ישן תוכנות וירטואלייזציה שיוטר קשה לאתרן
- במידה שרוצים למנוע לחוטין אפשרות לזיהוי, צריך להריץ בסביבה טבעית במחשב ייעודי
  - אבל אז צריך להשקיע יותר בהתקנה חדשה כל פעם שנגרם נזק



# ?היא?



362

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020



# הסוללה המתואמת ל-KILOG-6 (1988)

- כיצד התולעת הגנה על עצמה מפני מפעילי מחשב?
  - שינוי שם התהיליך ל-sh ומחיקת הfrmטורים
    - בפלט של zk התהיליך נראה כמו shell תמים
  - קריית כל הקבצים לזכרו 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 תהיליך נשאר קטן
  - מניעת הקטנת עדיפות (כ噫 תהיליך שraz הרבה זמן מאבד עדיפות)



# היבואת הסטטוסים ליסכום

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

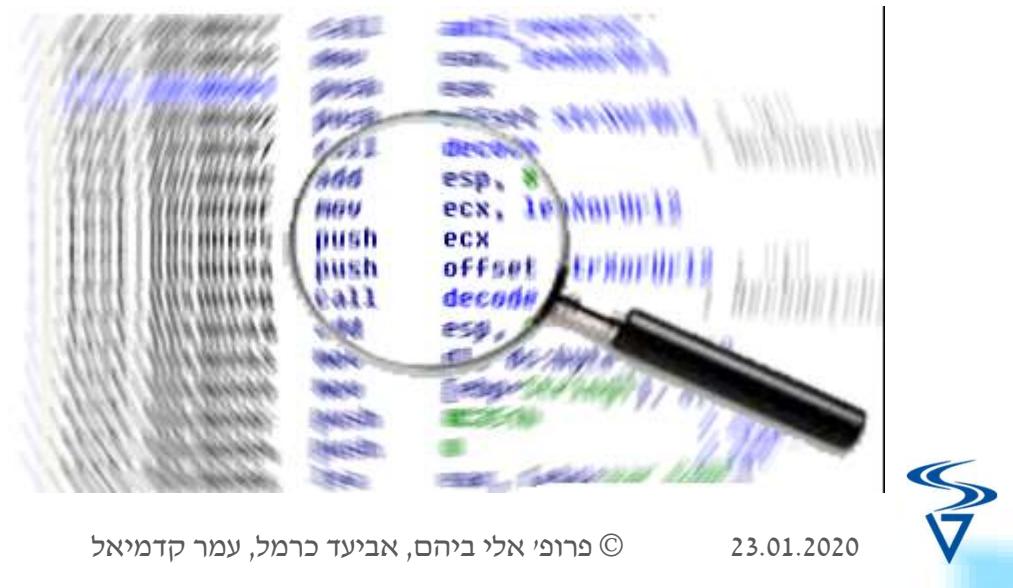


364



# פרק 10

## ליכוי ואפקט רולקאם



# ארכאום אוטו-וירוס

- מערכות אנטי-וירוס מגנות נגד מגוון נזקיות נפוצות
  - וירוסים, תולעים
  - רוגלות, תוכנות פרסום
  - סוסים טロיאניים
  - 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 נזקיות לא מוכחות
  - ובפרט נגד נזקיות ייעודיות שנכתבו במיוחד למטרה מסויימת



# היגוי רזקאות א"י מכך את הארץ-וירוס

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

- איסוף דוגמאות של וירוסים ונזקנות
- איסוף "מודיעין"

איסוף

- תשתיות ניתוח נזקנות
- ויצירת חתימות
- ופיתוח אלגוריתמי זיהוי וניקוי

ניתוח

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

פרסום

- בסה"כ השקעה ענקית...



# כינור רזקאות א"י מכרות הארץ-ויקס

- חיפוש בראשת
- קבצים חשודים שנשלחים מלוקחות מודאים
- מלכודות דבש
- מידע שנאסף מתוכנות אנטי-ווירוס שהותקנו ע"י לkokות
  - לקספרסקי יש מיליון מחשבים מהם הם מקבלים מידע
- איסוף "מודיעין"  
[virustotal.com](http://virustotal.com) •



368



# ליפוי רזקאות

## • זיהוי נזקות נעשה במגוון של אלגוריתמים היוריסטיים

- זיהוי שינויים בקובץ
- בדיקת תוכן קובץ
- זיהוי התנהגות
- בדיקת שינויים במערכת הקבצים (white list)
- וכו'

## זיהוי חיובי ושלילי

- זיהוי מקרים נקיים
- ע"י השוואת תוכן ידוע
- white list

## זיהוי חיובי

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

## זיהוי שלילי

## שני מקרים עקרוניים

- **זיהוי סטטי – בדיקת קובץ ההרצה**
  - למשל אחרי הורדת קובץ מהרשת USB
  - או בהנחת דיסק USB

## זיהוי דינמי

- בזמן ריצה

## וכמוון אפשר לשלב

- למשל זיהוי חסד באופן סטטי, ווידוא דינמי

▪ ואז מציאת חתימה לוירוס שהתגלה, וזיהויו באופן סטטי

◦ או קבצי הרצה באמצעות חתימות, וקבצי מערכת באמצעות white list



# אפקט Black Box – גלגול

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



# כאי ארכיטקט – כאי רקח

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



# ריתוך אירויים קאוח

## SysInternals •

- סט כלים ותוכנות לניטוח וניהול מחשבי windows:
  - Process Explorer
  - Process Monitor
  - Autoruns
  - RootkitRevealer
  - ועוד כ-60 כלים.

## ▪ חברת שנקננה על ידי MS

## RegShot •

- הקלטה מצב המערכת registry וגם קבועים והשוואה למול מצב הבא.

## WINDBG או כלי מעקב אחרים על מערכת הפעלה

- בדיקה אלו פונקציות מערכת הפעלה הופעלו..



# קאנדי: קידום

|                      |          |
|----------------------|----------|
| TextOutA             | GDI32    |
| GetStockObject       | GDI32    |
| VirtualAlloc         | KERNEL32 |
| GetCurrentProcess    | KERNEL32 |
| GetStartupInfoA      | KERNEL32 |
| GlobalAlloc          | KERNEL32 |
| GetWindowsDirectoryA | KERNEL32 |
| GetWindowsDirectoryW | KERNEL32 |
| IstrcatW             | 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   |

- הפקציות אליהן הקובץ עושה Import

(נראה כי מדובר בתהליך עם ממashing שותם. לא אופייני לנוזקה)



# Process Explorer

## ▪ נסתמך ב- Process Explorer ▪

|                                      |  | 0,000 K  | 1,000 K  | 10,000 K                           | 100,000 K                           | 1,000,000 K | 10,000,000 K | 100,000,000 K |
|--------------------------------------|--|----------|----------|------------------------------------|-------------------------------------|-------------|--------------|---------------|
| explorer.exe                         |  | 33,840 K | 30,100 K | 1856 Windows Explorer              |                                     |             |              |               |
| vmtoolsd.exe                         |  | 9,960 K  | 7,920 K  | 1980 VMware Tools Core Service     |                                     |             |              |               |
| PWRISOVM.EXE                         |  | 904 K    | 544 K    | 2000 PowerISO Virtual Drive Man... |                                     |             |              |               |
| ctfmon.exe                           |  | 1,056 K  | 1,632 K  | 160 CTF Loader                     |                                     |             |              |               |
| cmd.exe                              |  | 2,280 K  | 72 K     | 2084 Windows Command Processor     |                                     |             |              |               |
| procexp.exe                          |  | 2.94     | 10,004 K | 16,452 K                           | 5940 Sysinternals Process Explorer  |             |              |               |
| Procmon.exe                          |  | 16,092 K | 1,372 K  | 2068 Process Monitor               |                                     |             |              |               |
| 71aad9ed4a2e68d77a0d32ca2917e39f.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... | Micro       |              |               |
| TPAutoConnect.exe                    |  | 12,392 K | 4,240 K  | 2496 ThinPrint AutoConnect comp... | Cort                                |             |              |               |
| IEXPLORE.EXE                         |  | 7,416 K  | 17,252 K | 4732 Internet Explorer             |                                     |             |              |               |
| IEXPLORE.EXE                         |  | 67,924 K | 81,116 K | 3220 Internet Explorer             |                                     |             |              |               |

- התרהlixir מפעיל את svchost.exe ונסגר.
  - חלק מהנוזקות מתחזות ל-svchost.exe.
  - אבל במקרה זהה הנזקה מפעילה את ה-svchost המקורי שבא עם הווינדוס.



# הצאת sysinternals

- נשתמש ב- Process Monitor :

| svchost.exe | 4788 CreateFile           | C:\Documents and Settings\Administrator\Desktop\sql\7faad9ed4a2e68d77a0d32ca2917e39\7faad9ed4a2e68d77a0d: |
|-------------|---------------------------|-----------------------------------------------------------------------------------------------------------|
| svchost.exe | 4788 QueryInformatio...   | C:\Documents and Settings\Administrator\Desktop\sql\7faad9ed4a2e68d77a0d32ca2917e39\7faad9ed4a2e68d77a0d: |
| svchost.exe | 4788 QueryAllInformati... | C:\Documents and Settings\Administrator\Desktop\sql\7faad9ed4a2e68d77a0d32ca2917e39\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 | 4788                      | 4788                                                                                                      |

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



# הציגות sysinternals

## ▪ נשתמש ב-Autoruns :

| Autorun Entry                                                            | Description | Publisher | Image Path                                                                           | Timestamp          |
|--------------------------------------------------------------------------|-------------|-----------|--------------------------------------------------------------------------------------|--------------------|
| HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run                       |             |           |                                                                                      |                    |
| 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\pwrisolvm.exe                                              | 7/20/2013 3:18 PM  |
| vm 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                |             |           |                                                                                      |                    |
| 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                       |             |           |                                                                                      |                    |
| nvjfrpja                                                                 |             |           | c:\documents and settings\administrator\local settings\application data\gapvfbsv.exe | 2/6/2014 4:51 PM   |
| HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce                   |             |           |                                                                                      |                    |
| 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             |             |           |                                                                                      |                    |
| 0                                                                        |             |           | File not found: About:Home                                                           | 8/4/2013 1:46 PM   |
| HKLM\Software\Classes\*\ShellEx\ContextMenuHandlers                      |             |           |                                                                                      |                    |
| ANotepad++ ShellHandler for Notepad++                                    |             |           | c:\program files\notepad++\nppshell_05.dll                                           | 6/18/2012 6:24 PM  |

▪ רץ עם אתחול המערכת gapvfbsv.exe



# סיכום ה-ANSWER

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



# רימוח מקוֹאכט

- כל Malware חייב לתקשר עם האינטרנט.
  - בש سبيل לשולח מידע לתוקף.
  - בש سبيل להתפשט ברשת.
  - וسلح סיבות אחרות...
- בדרך כלל ההתנהגות של נזקיה שונה מההתנהגות של תוכנה רגילה. דוגמאות:
  - נסיוון להתחבר לעשרות מחשבים בראשת 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 הנזקיה.
  - ויתר מזה - לגלוות קיום של נזקיות.
- Wireshark היא התוכנה המוביילית כיומן לניטור תעבורת הרשת שעוברת במחשב מסוים.
  - פורמט הקובץ - PCAP
- האם ניתן לנטר תעבורת באופן בטוח? מה הסימנים המעידים?



# זיהוי IP

- דוגמה לגילישה באתר אינטרנט (Wireshark)

| Index | Source IP     | Destination IP | Protocol | Time                                 | Length | Information |
|-------|---------------|----------------|----------|--------------------------------------|--------|-------------|
| 128   | 192.168.2.102 | 62.219.78.115  | HTTP     | 582 GET / HTTP/1.1                   | 582    |             |
| 134   | 19.4334850    | 62.219.78.115  | HTTP     | 627 GET /style.css HTTP/1.1          | 627    |             |
| 145   | 19.4584920    | 62.219.78.115  | HTTP     | 638 GET /images/logo_01.gif HTTP/1.1 | 638    |             |
| 148   | 19.4588960    | 62.219.78.115  | HTTP     | 638 GET /images/logo_02.gif HTTP/1.1 | 638    |             |
| 151   | 19.4606560    | 62.219.78.115  | HTTP     | 638 GET /images/logo_03.gif HTTP/1.1 | 638    |             |
| 159   | 19.4643110    | 62.219.78.115  | HTTP     | 633 GET /images/bg.png HTTP/1.1      | 633    |             |
| 181   | 19.5093640    | 192.168.2.102  | HTTP     | 257 HTTP/1.1 200 OK (text/css)       | 257    |             |
| 183   | 19.5097570    | 62.219.78.115  | HTTP     | 638 GET /images/logo_05.gif HTTP/1.1 | 638    |             |
| 233   | 19.5677730    | 192.168.2.102  | HTTP     | 162 HTTP/1.1 200 OK (GIF89a)         | 162    |             |
| 235   | 19.5687120    | 62.219.78.115  | HTTP     | 638 GET /images/logo_06.gif HTTP/1.1 | 638    |             |
| 242   | 19.5687120    | 62.219.78.115  | HTTP     | 221 HTTP/1.1 200 OK (text/html)      | 221    |             |

```
+ 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: 521
+ 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=187449709.1396732414\r\n
```



# רַוְקָה כְּסֵת

- נזקה כאמור, מתנהגת שונה מתוכנה רגילה.
- דוגמאות לפעולות חשודות של נזקה:
  - סריקת מחשבים – אם הסקייה היא בתוך הרשת אנחנו נראה המון בקשות ARP יוצאות מהמחשב (הדוגמה בהמשך).
  - סריקת פורטים – נראה התחברות לאותו IP אך עם פורטים שונים.
    - בדרך כלל חיבורים שנכשלים מרמזים על פעולה חשודה. שכן תוכנה רגילה לא מנסה לאיילו פורטים להתחבר.
  - בקשות HTTP מזוירות – למשל שליחת כינויים מסויימים של מידע לשרת C&C.
  - שימוש ב프וטוקול מסויים בצורה חריגה.
    - יש צורך להכיר מאות פרוטוקלים בשביל להגדיר מה זה חריג, אבל ההיכרות הכרחית.
  - ועוד



# ליחוי המתקאה ב-Sandbox

- מערכת זיהוי התנהגות (Sandbox) אוספת מגוון נתונים ואירועים במהלך ריצה
  - ובודקת אם הם צפויים או לא
  - כלומר אם התוכנה שרצה מבצעת סדרת אירועים כזו או שתוכנה שאינה נזקנית צפוייה גם כן לבצע סדרה כזו
- אירועים שנאספים יכולים לכלול
  - קריאה לקריאות ממכשיר חשודות
    - כמו כתיבה לזכרון של תהליך אחר
  - פתיחת קבצים
    - למשל כתיבה לקובץ הרצה אחר
  - ייצור תהליכי חדשים
  - שינוי בוקטור הפסיקות
  - ועוד – דוגמאות?
- דוגמא - Cuckoo



# ויריה ונקמי ויכום – אה אכאי?

הה אס גיאו ויכום lk  
הה אס גיאו ויכום lk



# חתימת fe וירוס

- חתימה של וירוס: מהרוזת ייחודית המופיעה בו, ולא מופיעה בתוכנות אחרות
  - השיטה הקלאסית לזיהוי וירוסים ונזקот היא על ידי זיהוי ה"חתימות" שלהם
  - יוצר החתימות דורש השוואה לכל התוכנות המוכרות!
    - היו מקרים שיצרני אnty-ווירוס 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 רצתה (למשל, אחרי שווירוס פענח את הקוד שלו)



# מתיאום fe וירוסים

- אנטיבירוסים מקבלים מידי פעם קובץ חתימות חדש
  - שישמש מאותו רגע להשוואה עם כל הקבצים במערכת
- יצרן האנטיבירוס מחשב אותו על סמך
  - כל הווירוסים המוכרים
  - וכל התוכנות החוקיות המוכרכות
- הבדיקה נעשית מול התוכן הנוכחי בלבד
  - אין השוואה לתוכן ההיסטורי
    - **למשל גרסה מקורית של הקבצים**
  - בדיקה מאד יעילה (מעבר אחד מהיר על כל קובץ)



384



# אפקט איטמיים היוצרים מכך

- מסוגלים לזהות וירוסים לא מוכרים
  - אבל בלי לדעת מה הם מבצעים
- טכנולוגיה – חיפוש היריסטי בקוד
  - חיפוש פועלות מחשידות בקוד
  - חיפוש מחרוזות קוד מחשידות
    - מחרוזות שוירוסים נוטים להפעיל
  - ביצוע RE של קוד באופן אוטומטי
    - כדי לזהות פועלות חשודות
- עלולים לטעות...
  - תוכנה חוקית עשויה להפעיל קוד מחשיד



# חסימת המתהpid אסורה

- תוכנות אנטי-ווירוס יכולות לחסום קריאה לקריאות מייה
  - או פונקציות ספריה מסוימות
  - כצפוי שתוכנה חוקית לא תקרא להן
- בזמן קריאה לקריאות מייה כאלה
  - או אם קריאות מסוימות מופיעות ב-table Import
- האנטי-ווירוס יעצור את התוכנית
  - או יפתח חלון עם שאלת למשתמש
  - או ישלח דוגמא "הביתה"



# White List

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



# IPS/IDS

- *Intrusion prevention/detection systems*.
- רכיבים שמטרתם לנטר את התעבורה ולהיפש פועלות שנחשות זדוניות.
- יכולים לפקד כרכיבים נפרדים או בתוך רכיבים קיימים (למשל ראותרים).
- IPS מנסה להסום את הפעולות האלו בעוד IDS תפקידו רק לדוח.
- איך זה עובד?
- זיהוי סריקות או מתkopות רשות (מתkopות על פרוטוקלים).
- חתימות על חbillות שמוגדרות כחוודות.
- למשל חתימה על אקספלויט מסוים.
- ניתוח פעולות חריגות ( anomalיות ).
- קלומר ללמידה איך הרשות צריכה להתנהג במצב נורמלי ולזהות פעולות שחורגות מן המצב הנורמלי. כל רשות מתנהגת בצורה שונה.
- ועוד שיטות



# ארכו-אלרום - סיכום

## РЕЗЮМЕ: АКИКА ИГИРЫ

- ניתוח : בדיקה וגילוי
  - על ידי ניתוח אנושי
  - מערכות אוטומטיות הבודקות פרמטרים חשודים
    - שינוי קבצים, הדבקת קבצים אחרים, פעילות לא מקובלת
- טכנולוגיות בשימוש
  - ניתוח סטטי ודינמי
  - הרצה במכונות וירטואליות
    - מונע גרימת נזק, קל לבטל הנזקים
  - אמלציה
    - תוך חיפוש פועלות שתוכנות רגילים לא נוהגות לבצע
- מטרה
  - מציאת מאפיינים ייחודיים על פייהם תוכנות ה-AV יפעלו
  - העדפה מובנית למניעת false-positives מלמציאת יותר נזקיות



# ?האם?



390

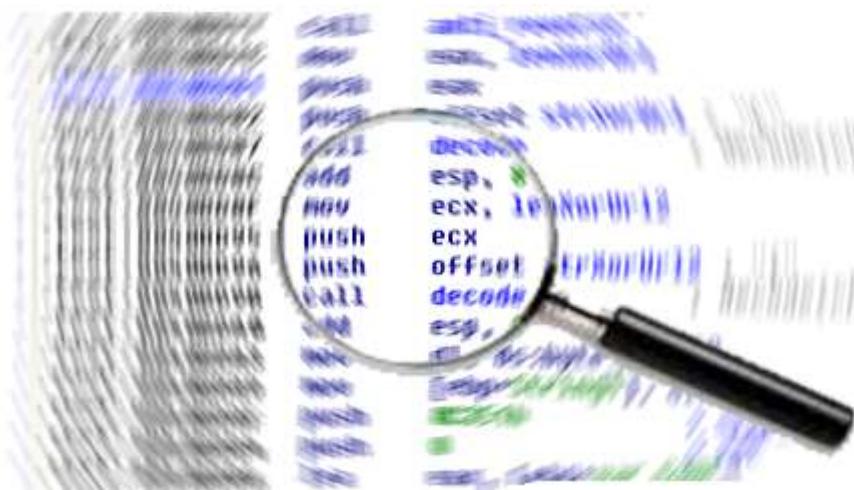
הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020



# APT



391

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020



# 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?

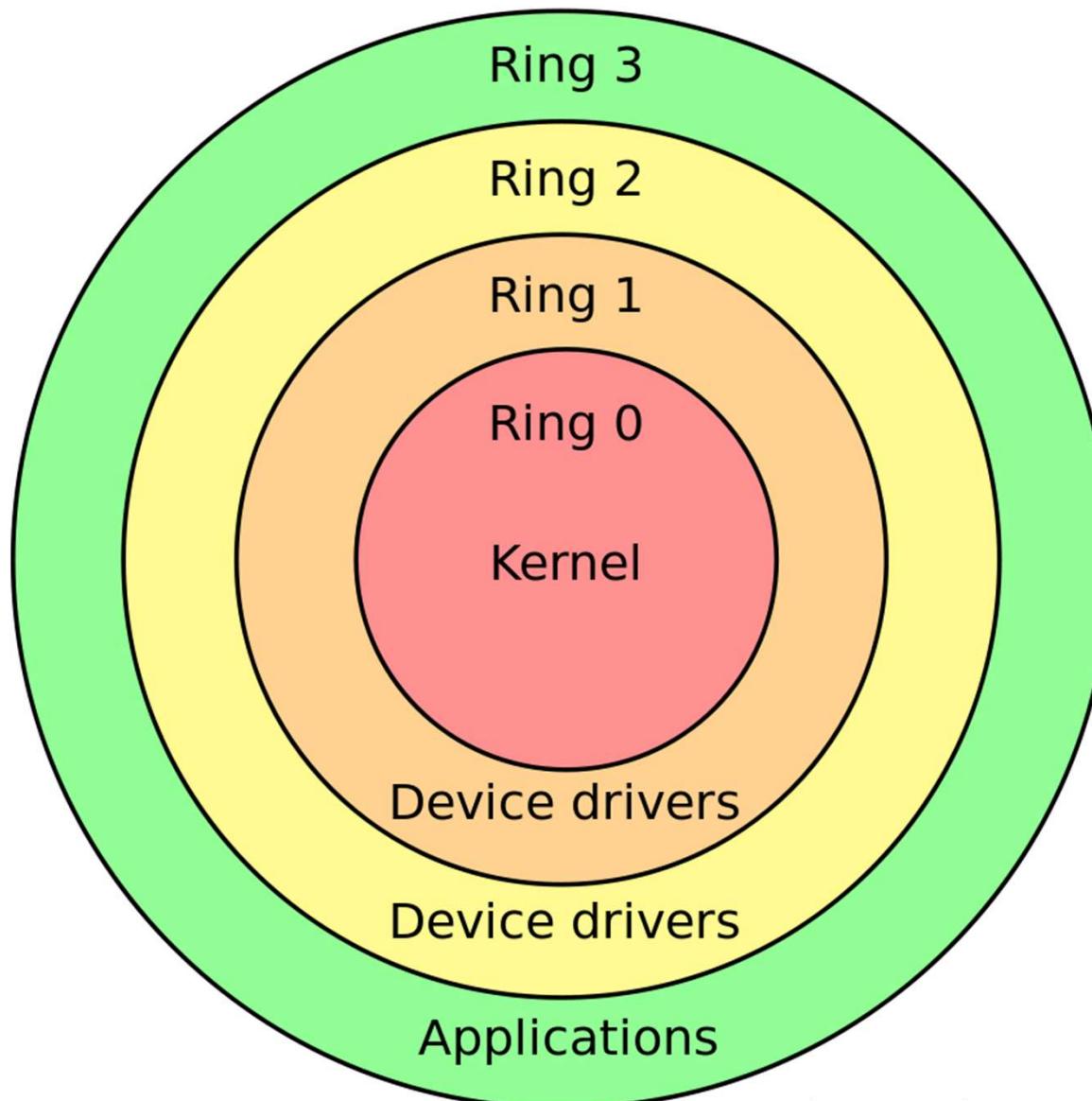
Looking at the discussions and development of sophisticated attack techniques, there is a significant difference between the theory and in-the-wild observations. So what is missing? Here's a list of possible culprits:

- 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.

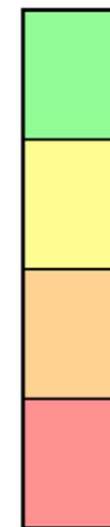
firmware was overwritten with trash.

- 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



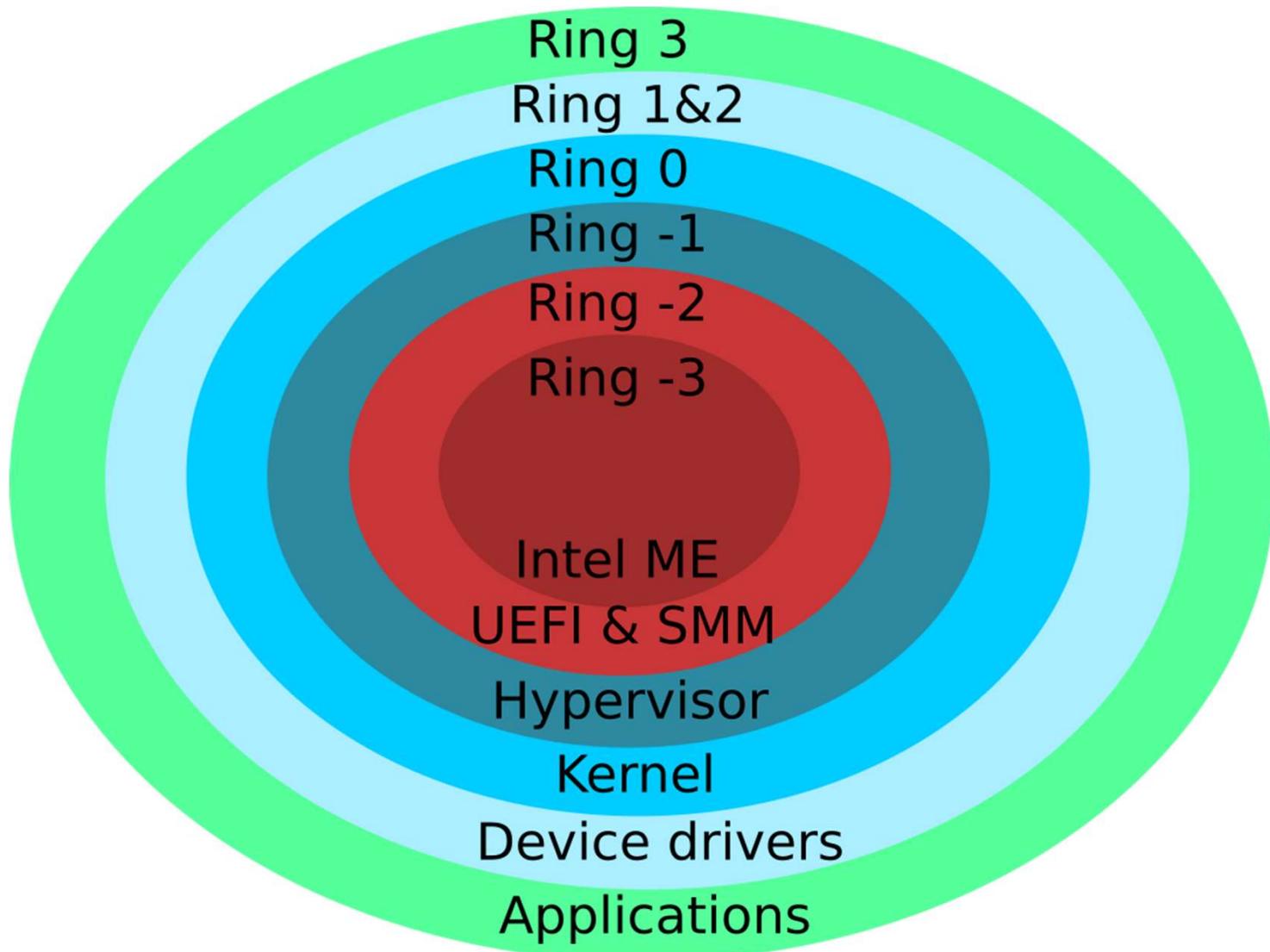
Least privileged



Most privileged



# Real Protection Rings



# תלכאות: היפרוייזר

- היפרוייזר (hypervisor) הוא מערכת שMRIיצה תחתית את המחשבים הווירטואליים (ומערכות הפעלה שלהם) במערכות וירטואליות
  - למשל VMWARE
    - מקום מעולה לנזקה להסתתר בו
- למשל, כאשרם מרים מרכזים בחוות המחשבים,
  - לכל אחד מהם יש מכונה וירטואלית משלו
    - "מתחזה" למחשב אמיתי עם מ"מיה וכל הציוד הנדרש
  - כל המכונות הווירטואליות רצות תחת מחשב אחד
    - בחדר השירותים הפוקולטי
    - בעל זיכרון גדול ומרובה ליבנות
  - וההיפרוייזר זו מ"מיה-על
    - שMRIיצה תחתית את המכונות הווירטואליות
    - מחליטה מי מהן תרצו באיזה ליבה ומתי
    - מבצעת context-switching בין המכונות הווירטואליות
    - מקצת להן זיכרון, ופונה עבורה להתקני קלט-פלט



# Blue Pill

- Blue Pill זה תוכנה שמסתתרת כהיפרוייזר
  - ולכון מ"ה ואנטי-וירוסים שרוצים תחתיו לא רואים אותו
- כש-Hypervisor מופעל הוא הופך ל-
  - כלומר משטלט על המחשב
  - משטלט על וקטור הפסיכות, והגישה להתקנים
  - מפעיל את שירותי הווירטואלייזציה של המעבד
  - וMRIIZ תחתיו בוירטואלייזציה את מערכת ההפעלה המקורית
  - תוך בקרה ושליטה על כל פעולותיה



# Blue Pill

- מערכת הפעלה שומרת על הגישה הקיימת להתקנים ולביצים
  - אבל הגישה בפועל מבוקרת ע"י ההיפרוייזור
  - כל גישה חדשה נעשית באמצעות ההיפרוייזור
    - כולל פסיקות חומרה
    - בקשנות לנתונים
    - כולל גם שעון המערכת
- וכמובו ההיפרוייזור מזיין לפני מייה כאילו הוא החומרה המקורית



# Blue Pill

- קשה מאד לזיהות את קיומם Blue Pill מתוך מייה שרצה תחתיו
  - אם היישום בוצע בזרירות המתבקשת
    - כי Blue Pill שולט על התשובות שם"ה מקבלת בגישה לחומרה
  - אבל נראה לא בלתי אפשרי



398



# תזכורת: מהו BIOS ב-PC

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



# תזכורת: מהו יקע א"ה ב-PC

- הוא פונה לדיסק הנבחר, וקורא ממנו את הSKUטור 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
  - שຕפקידו להביא את קוד הטעינה של מערכת הפעלה
  - בغالל קטענותו, בדר"כ אינו מכיר את מבנה מערכת הקבצים
  - זה הקוד הראשון שמופעל, שנייתן לשינוי על ידי מ"ה
    - ובתלות בהרשאות, על ידי תוכנות אחרות, למשל נזקאות
  - קוד זה קורא ממוקם קבוע בדיסק את קוד הטעינה של מ"ה
    - ומפעיל אותו



# תזכורת: מהagic א"ה PC

- קוד הטעינה של מ"ה ניגש למערכת הקבצים בדיסק
  - וטוען את מ"ה
  - יתכן שכמה 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



401



# ג茲קם Boot Sector-ה

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



# הsector-ה- גְּזַקְט

- **היות שהוא נתען לפני מייה**
  - **היא לא יכולה למנוע את עלייתו**
    - **ולא יודעת על קיומו**
  - **הוא יכול לשנות את הטיעינה שלה, ואת מה שהיא עושה**
    - **היות שכל הקריאה שלה מהדייסק נעשה דרך שירותים שהוא יכול להשתלט עליהם לפני שהיא עלתה**
    - **והיא לא יכולה לבדוק אם הוא שינה, בלי עזרתו**
- **"בעיות"**:
  - **אם הוירוס מסתתר בתוך הזיכרון הנגיש למ"ה**
  - **ואם היא תחשוד בקיומו, היא תוכל לזהות אותו**



# גזקם Boot Sector-ה

- צורת הדבקה זו הייתה מאד נפוצה בתקופה שמיידע דיגיטלי הועבר בדיסקטים
  - בעלת יתרונות רבים
  - אבל קשה לביצוע כיום
    - יש הגנות על כתיבה ל-boot sector של דיסקים קשוחים
    - ומעט שאין אתחול של מחשבים מהתקנים נטיקים





# 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



405

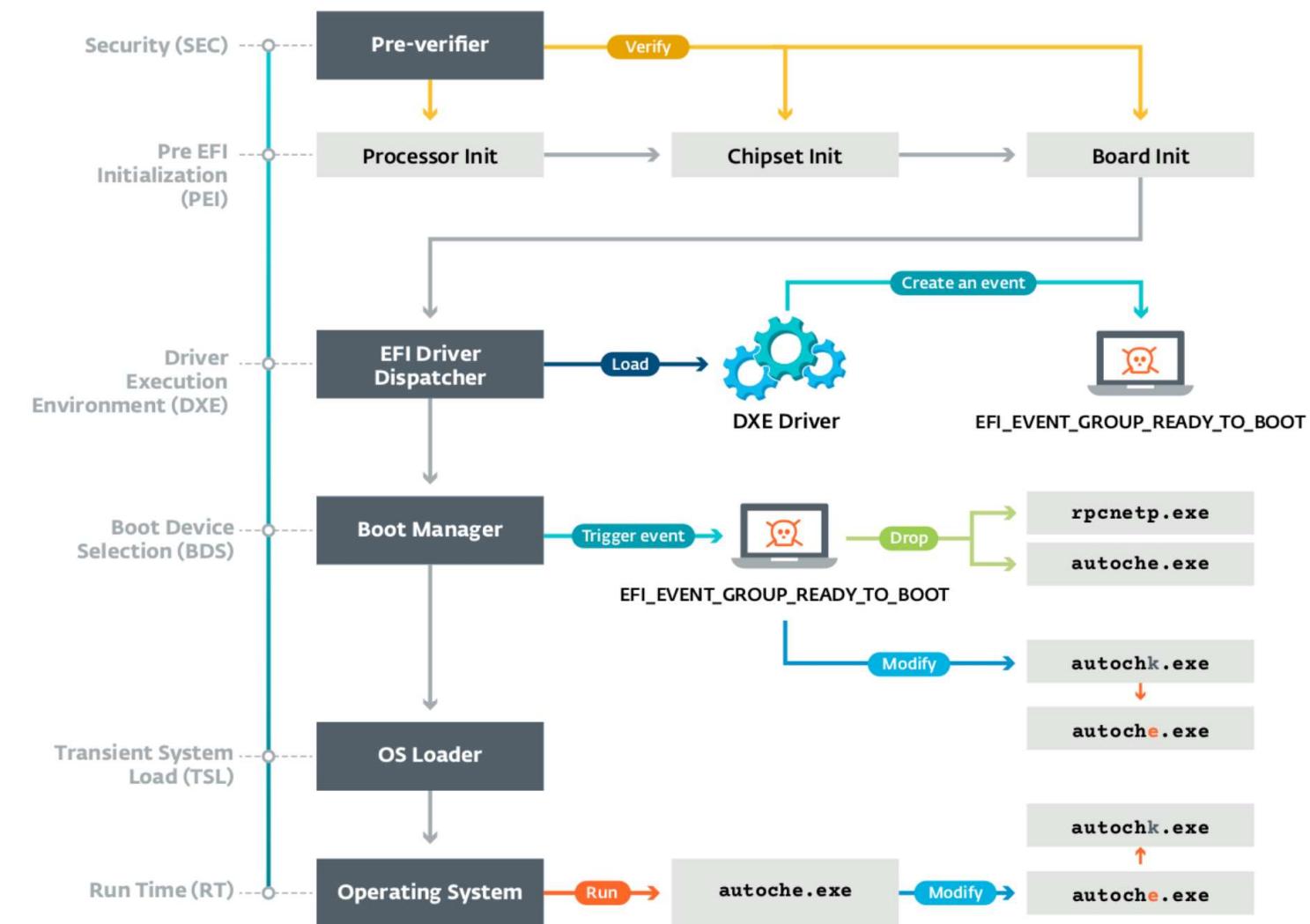
הנדסה לאחור – חורף תשע"ט

© פרופ' אלי ביהם, אביעד קרמל, עמר קדמייאל

23.01.2020



# UEFI Rootkit Workflow



# הסתמכת קזיסק קריין

- הסתרות בקורסחה של דיסק קשה
  - 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
  - והשתלטה בדיסק
  - תוך עקיפת אמצעי ההגנה על הקושחה (אם יש כאלה)
  - לא קל, אבל אפשרי



# גסמתכאות מחיצות USB

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

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



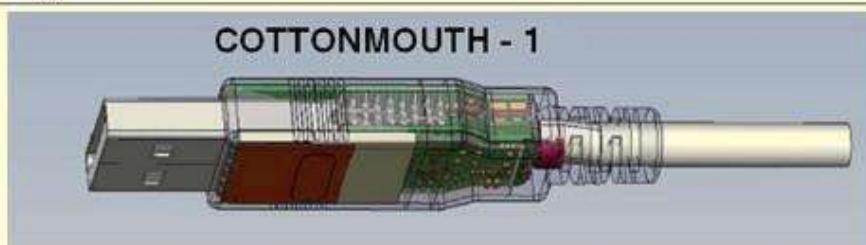
## COTTONMOUTH-I

### ANT Product Data

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

08/05/08

**COTTONMOUTH - 1**



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

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

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

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



408

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020



# גסמתכאות USB אמיאור

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



# ויליאם נאלסן



410

הנדסה לאחור – חורף תשע"ט

© פרופ' אלי ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020



# Stuxnet

- וירוס לתקיפת מערכות חלונות המוחברים לבקרים תעשייתיים של סימנס
- נראה יועד נגד אירן
  - אבל הדביק גם תחנות גרעיניות ברוסיה ובעולם
  - אפילו הדביק את תחנת החול היבנלאומית
    - נראה דרך דיסק USB
- חיפש מחשבים עם תוכנת Step7 וחיבור ל-PLC של סימנס
  - מותנה בחיבור לציודים מסויימים, ובסיבוב בתדרים מסויימים
  - כשמצא כאלה – ניסה להציג גרסה מעודכנת שלו
  - לא הפעיל את עצמו על מחשבים אחרים



# התקפת אבטחה גזעית

- הבדיקה מחשבי חלונות
  - במנועו דרכים
    - מתוכן שתאים דרך הבדיקה מדייסק USB
  - מטרה : לאפשר את שתי התקיפות הנוספות
- הבדיקה מערכות ה-SCADA (תוכנה של סימנס)
  - כדי להציג כאיilo הציוד פועל תקין, ושלוט על הבקרים
  - על ידי התקנת DLL, ושימוש בחולשת 0-day של תוכנת סימנס
- שכתב תוכנת בקרים ה-PLC
  - כדי לשלוט ישירות על הציוד, ולגרום נזק לציוד



S7-300 של סימנס – PLC



© פרופ' אליא ביהם, אביעד קרמל, עמר קדמיאל

23.01.2020



412



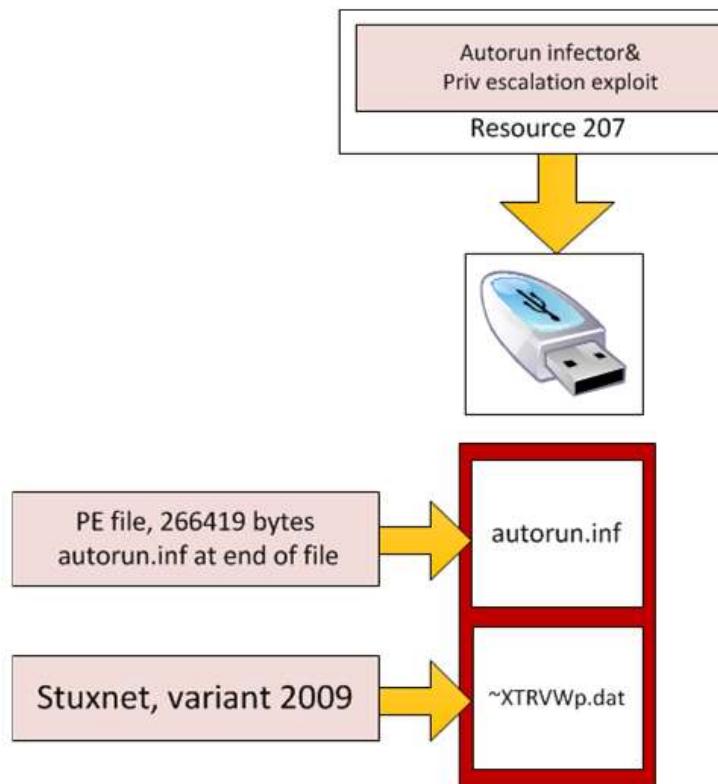
# Stuxnet

- מחולק על פני מספר רב של קבצים במחשב!!!
- משלט על מערכת הקבצים
  - ומסתיר קבצי LNK בגודל 1714 בתים
  - ע"י שימוש בחולשה שמאפשרת להעלים אותם מסieur חלונות
- נחתם על ידי סרטייפיקט שנגנוב מ-Verisign
- מפעיל בקרים תעשייתיים של סימנס
  - ושולט על צנטריפוגות
  - מסובב אותן מהר מידי לזמן קצר
    - באופן שלא יזוהה מה גורם לביעיה הזו
  - עד שהן מתקללות



# התקפה א"י Autorun.inf

- קובץ Autorun.inf
  - מכיל קובץ PE
  - שבשימוש יש נתוני inf
  - באופן שהוא מורץ בעצמו כקובץ הריצה בזמן חיבור ההתקן
  - אnty-וירוס חיפש נתוני בקרה של autorun רק בהתחלה
    - ולכן לא זיהה...
- בנוסף קובץ שנראה קובץ זמני
  - שם יש את התוכנה עצמה



# P2P-ein ג

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



# Flame

- וירוס שמטרתו לאסוף חומר ממוחשבים, ולשלוח אותו לבעל הוירוס
  - דרך שירותי Command & Control
- אוסף חומר מסווגים שונים
  - קבצים
  - צילומי מסך
  - הקלטות אודיו מהמיקרופון
  - ועוד
- התפשת במדיניות בمزarah התיכון



416



# הסואם

- Flame מחולק על פני כמה קבצי DLL
  - מקשה על זיהוי, ומקשה על דיבוג
- מסתיר את עצמו בדפי זיכרון של תהליכיים אחרים
  - עם הזרקת קוד
  - והרשאות דפים שמונעות מאפליקציות ב-user mode לגשת אליהם או להריץ מהם
  - המודולים אינם מופיעים בlijow 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 סימות שמות הקבצים



# אפליקציית מזג האוויר

- גודלו ענק : 20MB
- כולל מגוון ספריות לתמיכה
  - בחמישה סוגי הצפנה,
  - כמה סוגי דחיסה,
  - SQLite,
  - מכונה וירטואלית,
  - ועוד
- התקנות במיה
  - משתמש ב-registry
  - מתokin דרייבר אודיו מזוייף
    - משמש כעוגן נגד מחיקתו מהמחשב



# מכשור ווסכום

- Flame מסוגל לזהות התקני בלוטות פועלים בסביבתו
  - ולאסוף עליהם נתוניים
  - להתחבר אליהם ולקבל מידע
    - כולל גרסאות מעודכנות
    - כולל מכמה קילומטרים, אם משתמשים באנטנה כיוונית מיוחדת
- מסוגל להפוך את הבלוטות שלו למזווהה
  - ולקודד מידע על הסטטוס של עצמו בפרטיו היזחי של הבלוטות
- צילומי המסך נעשים
  - מיידי פעם בפעולה רגילה
  - וכן כאשר פועלות תוכנות "מעניינות"
    - למשל תוכנות מסרים מיידיים
- מסוגל למחוק עצמו בבקשת בעליו

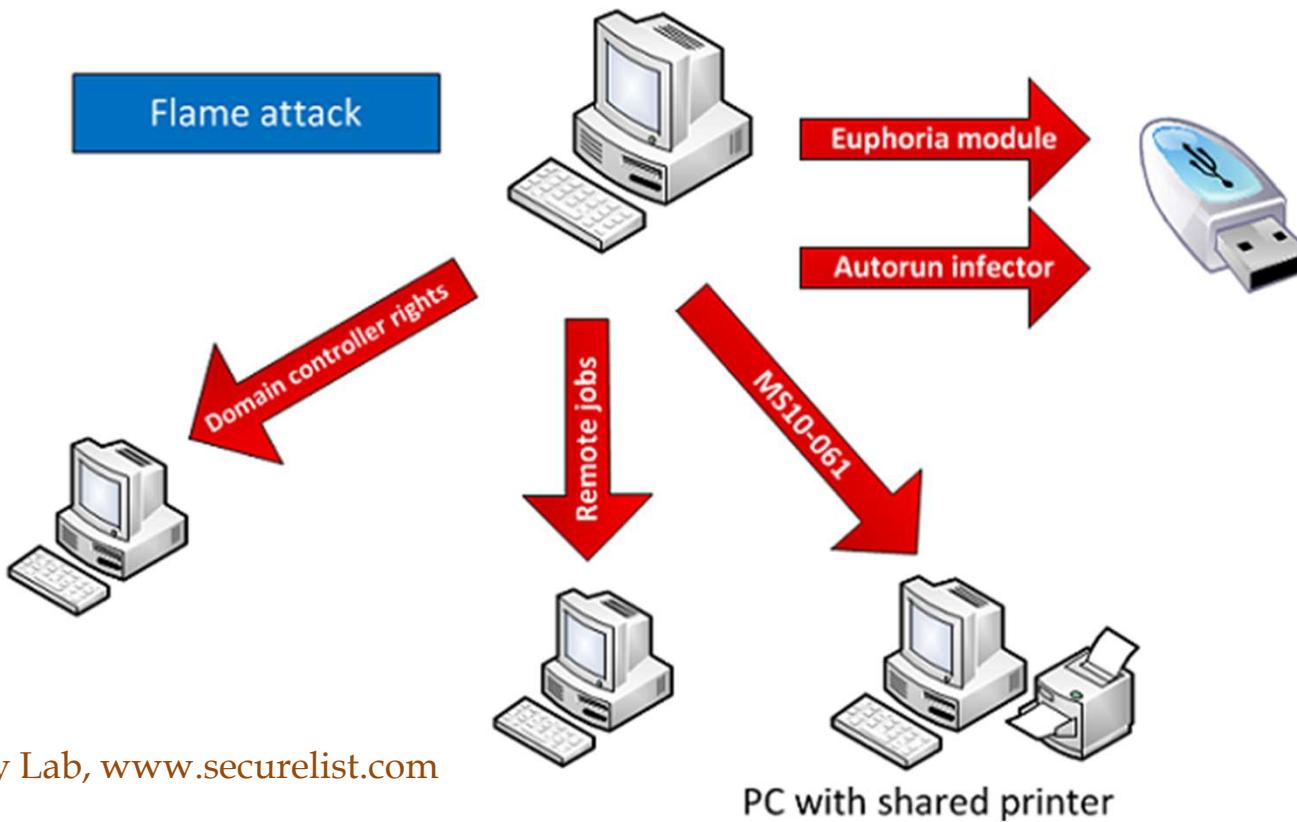


# קלכי הגזקה Flame fe

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



# Flam fe התקפה ?



Kaspersky Lab, [www.securelist.com](http://www.securelist.com)

421

23.01.2020



# איימת ליגו ציוף סטיפיקט

- חלונות מגנה נגד דרייברים ותוכנות מתחזות עיי' חתימה דיגיטלית
  - לכן, כדי לא להציג, צריך חתימה עם סטרטיפיקט מורשה
    - אבל שאינו מזזה 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
  - דרך הצליח לזייף סטרטיפיקט לחתימה על תוכנה
    - בשימוש בהתקפה קריפטוגרפית
- לשם כך, זיין סטרטיפיקט
  - בשימוש בפונקציית התמצאות MD5 השבורה
  - בגרסת שבירה מיוחדת של MD5 לצורך זיין סטרטיפיקטים
  - הרעיון פורסם במקור על ידי חוקרים מהולנד
    - וכותבי Flame השתמשו בו לצרכיהם, עם שיפורים משליהם

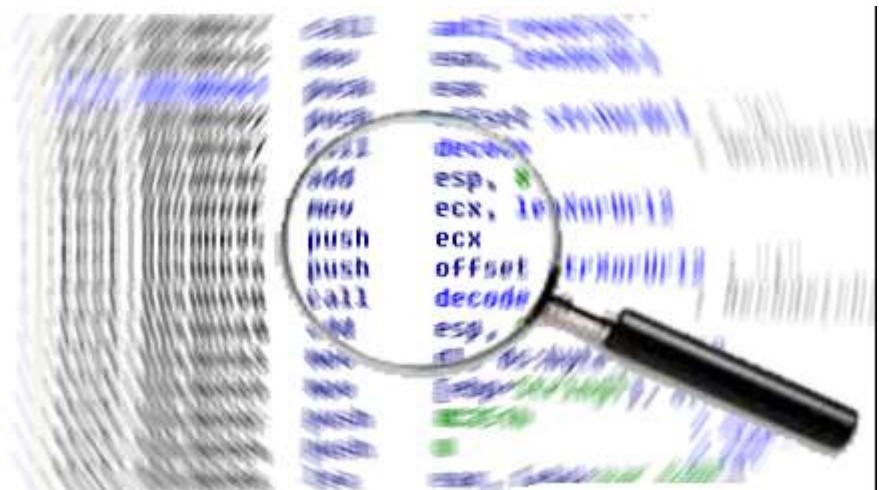


# הסכטינקן אַזְיִינָה – גַּזְרָת MD5

|                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Version                                  | V3                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Serial number                            | 3a ab 11 de e5 2f 1b 19 d0 56                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Signature algorithm                      | md5RSA                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Signature <a href="#">hash algorithm</a> | md5                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <u>Issuer</u>                            | 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                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <u>Public key</u>                        | 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 2e 89 8b d5 f3 cd 0e 27 14 49 67 94 ce b9 25 02 03 01 00 01 |
| Enhance key usage                        | <b>Code Signing (1.3.6.1.5.5.7.3.3)</b><br><b>Key Pack Licenses (1.3.6.1.4.1.311.10.6.1)</b><br><b>License Server Verification (1.3.6.1.4.1.311.10.6.2)</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 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                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

# פרק חמוץ

## מקרה סיאם...



# קורס הרצאה של מוכך

## רילאיימ אסמייימ

- X86 Assembly
- ניתוח סטטי ודינامي
- Hooking and Injecting (*and Instrumentation*)
- חולשות, ניצולן ומנגנוני הגנה ללא חולשות מתקדמות
- נזקות, הסואה וכלי נגד להנדסה לאחר
- ערוצי צד וחומרה



# אמן

- מבחן שני חלקים בני שלוש שעות כ"א
  - כ-5-4 שאלות
  - חלק ראשון : 09:00-12:00
  - חלק שני : 14:00-17:00
  - יש לגשת לשני חלקים המבחן באותו מועד
- תכולה : כל החומר שנלמד במקצוץ
  - הרצאות, תרגולים, סדנאות, שקיים, תרגילי בית
- המבחן יתקיים בספרייה
- הביאו מחשבים ניידים
  - לסטודנט שאינו מחשב נייד מתאים יושאל מחשב נייד
  - המבחן יופץ דרך אתר GR++
  - הגשת הפתרונות גם כוון דרך האתר (כהגשה "תרגיל בית")
  - אין להביא מסכים



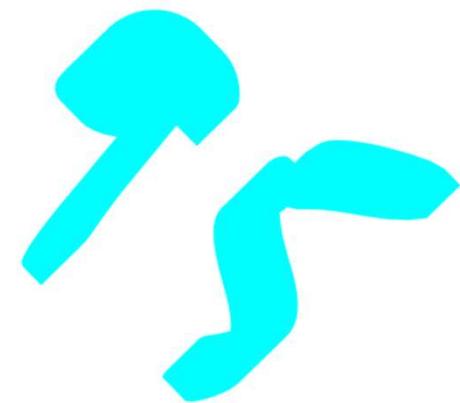
# תזכורת: יאור קזאי

- העבודה היא שלך – כתוב אותה בעצמך
  - אין להעתיק מזרים
  - אל תשתמש בניתוחים קודמים של החומר אותו אתה מנתח
    - אחרית הציון הגיע למי שכتب אותו
  - צטט מקורות
- ואם בכלל זאת עשית משהו אסור –-Amor זאת בפירוש
- ב מבחן:
  - אסור לשוחח על נושא המבחן עם אחרים בזמן המבחן
    - ז"א שגם אסור להשווות פתרונות
  - הפתרונות שלכם מההתחלה ועד הסוף
- **העוררים על הכללים יונשו בחומרה**



100

9/10



ההצפנה  
ההצפנה



428

הנדסה לאחור – חורף תשע"ט

© פרופ' אליא ביהם, אביעד כרמל, עמר קדמייאל

23.01.2020

