הפקולטה למדעי המחשב
הטכניון

הנדסה לאחור

מרצה: פרופ' אלי ביהם

מתרגל: אביעד כרמל

© Eli Biham and Aviad Carmel

ידע נדרש

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

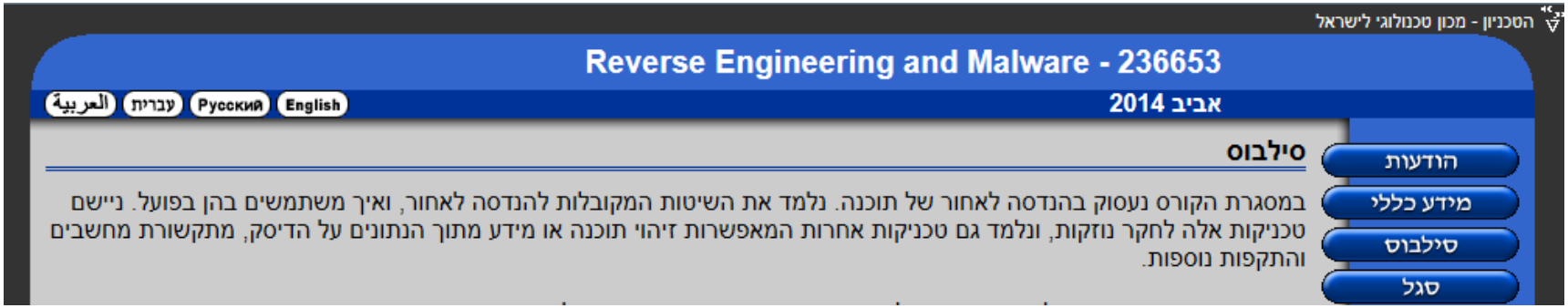
קדמים

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

הפצת מידע על המקצוע

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

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



סדנאות בחוות המחשבים

- יתקיימו מספר סדנאות בכתה שבחוות המחשבים
  - בדר״כ בשעות של השיעורים והתרגולים

- הביאו את המחשבים הניידים שלכם
  - במקרה הצורך אפשר להפעיל את המחשוב הווירטואלי הפקולטי מהנייד שלכם
  - במקרה הצורך תוכלו להשתמש במחשבים בחווה

מטלות המקצוע

- המטלות במקצוע כוללות כ-5-6 תרגילי בית
  - כולם או רובם רטובים
  - כולל תרגילים שדורשים השקעה רבה
  - משקלים לא זהים
  - בזוגות
  - ציון תקף
  - אין הגשה באיחור
    - במקרים מוצדקים (מילואים וכו') חובה לבקש אישור מהמתרגל האחראי לפחות 24 שעות מראש
- בוחן אמצע – תקף
- מבחן

בפקולטה – מול מחשב
שלוש שעות כל אחד

- ציון סופי
  - מבחן 40%, בוחן 30%, תרגילים 30%
  - ציון מבחן+בוחן (משוקלל) נמוך מ-50 לא ישוקלל עם ציוני התרגילים

יושר אקדמי

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

- העוברים על הכללים יוענשו בחומרה

התכוננות לשיעורים

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

ספרות

- רשימת הספרות נמצאת באתר המקצוע
  - ספרים נמצאים בספריה

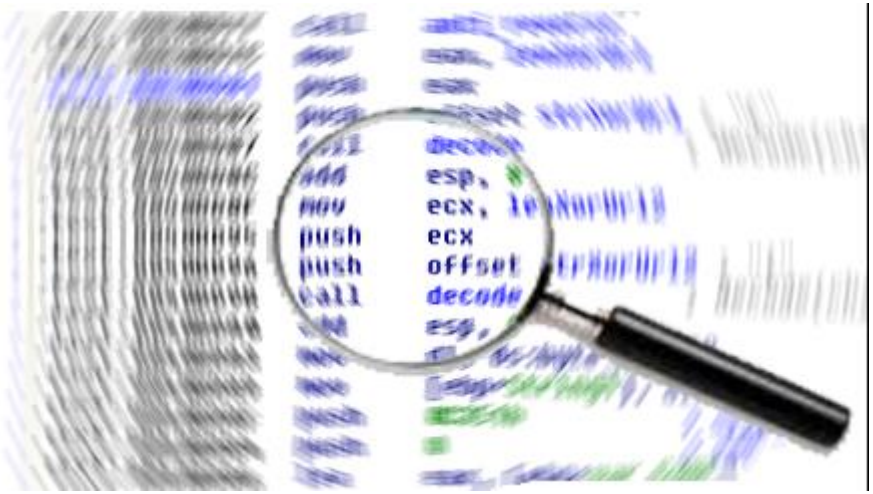
- בנוסף, קיים חומר רב באינטרנט
  - בפרט, תיאור מלא של אסמבלי של אינטל
    - לינק באתר המקצוע

המקצוע עדיין בשלבי בנייה

- לא הכל זהה לשנה שעברה
- יהיו שינויים, הורדות, ותוספות

פרק 1

מבוא להנדסה לאחור



מבוא

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

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

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

הנדסה לאחור של מכונות

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

הנדסה לאחור של חומרה

- בעידן המחשב – מסובך יותר – בגלל הקוטן והמורכבות
  - והקושי של פירוק למרכיבים
- אבל לא בלתי אפשרי

- יתכן שהצורה של המוצר מדליפה מידע או שניתן לזהות כזה באופן ויזואלי
- במעבדים מודרניים :
  - מיקרוסקופים ומיקרוסקופים אלקטרוניים
- מדידת מתח בזמן הפעולה
- הזרקת מתח למעבדים ומעקב אחרי התגובה
- גרימה לשגיאות בחישוב, ומעקב אחרי השינויים
- קריאת זיכרון, או האזנה ל-bus, אם אפשר
- ועוד

הנדסה לאחור של תוכנה

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

נתרכז במקרים הללו רוב הזמן

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

הנדסה לאחור: פרוטוקולים וקבצים

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

מניפולציות ו–Hooking

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

למה הנדסה לאחור?

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

למה הנדסה לאחור?

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

  - כמובן שבמקצוע זה לא נעסוק בפעולות לא חוקיות

הגנה כנגד הנדסה לאחור

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

שימושים לנוקות

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

מקרים יבואים של הנדסה לאחור

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

Blaster תולעת – קונאי

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

זוכרים?

Blaster תולעת – קונה

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

Microsoft Security Bulletin MS03-026

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

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

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

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

Blaster תולעת – קונאי

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

שימושים אפשריים ל-RE

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

US researchers find 25 security vulnerabilities in SCADA systems

Warwick Ashford
Friday 18 October 2013 15:24

שימושים אפשריים ל-RE

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

חדשות בעולם

איראן הציגה את המל"ט האמריקאי החמקן שהופל

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

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

45 | הוסף תגובה

Ewen MacAskill in Washington

Sunday 22 April 2012 10.50 BST

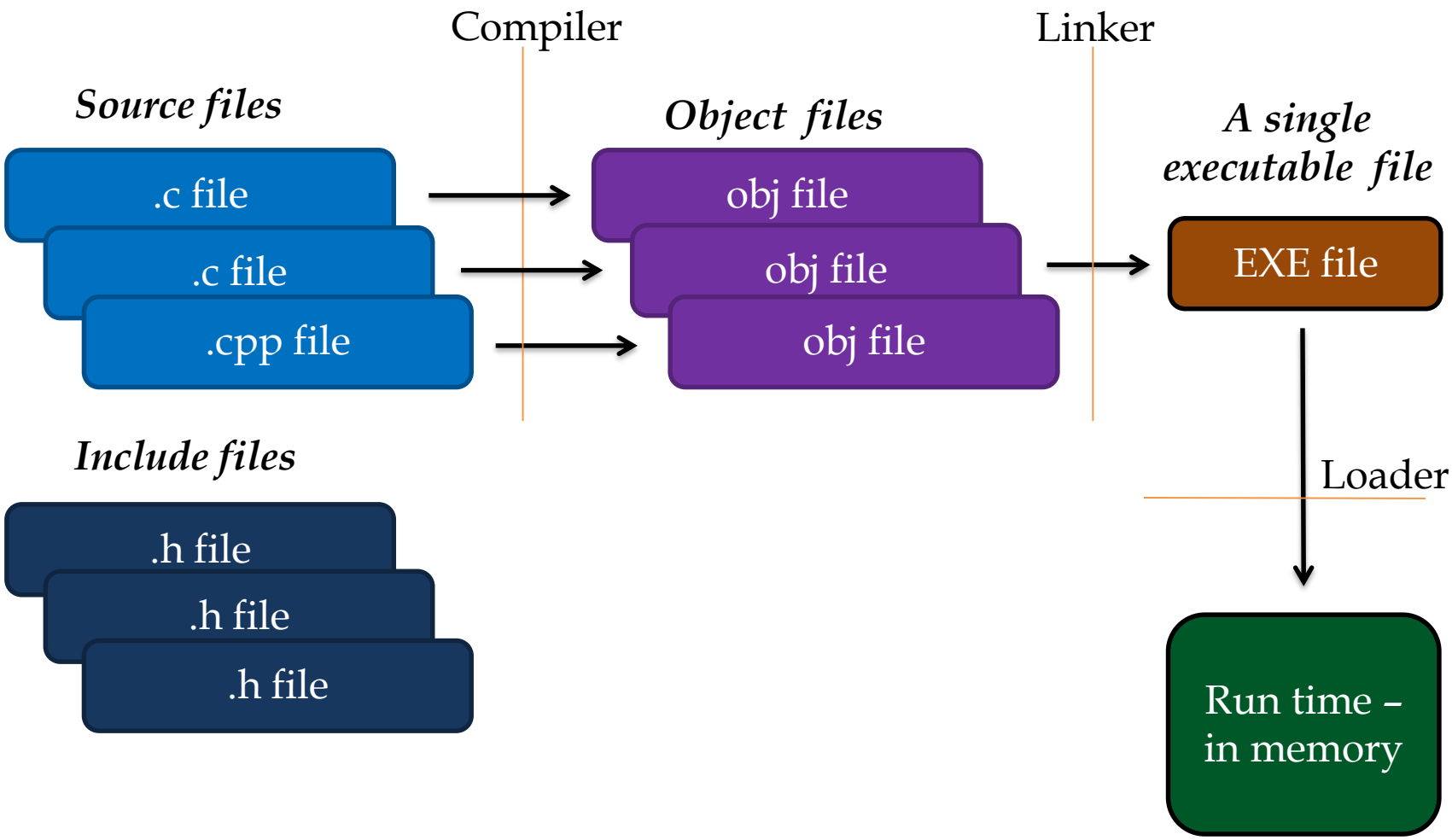
theguardian

Winner of the Pulitzer prize

Iran claims to have reverse-engineered US spy drone

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

תהליך ההידור וההרצה ב-C ו-C++



עוד על הידור

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

התהליך ההפוך

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

דוגמא: מה קוד זה מבצע?

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

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

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

האם כותב הוירוס יכול להוסיף קושי נוסף במכוון?

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

Java Bytecode & .NET

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

בעצם מה נה הנדסה לאחור?

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

דוגמא פשוטה להנדסה לאחור

מה עושה הקוד הבא?

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

דוגמא פשוטה להנדסה לאחור

הוא שקול לקוד הבא, בהחלפת הגישות לזיכרון ברגיסטרים:

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

דוגמא פשוטה להנדסה לאחור

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

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

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

חוק

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

ארה"ב: חוק ה–DMCA

The Digital Millennium Copyright Act

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

מה נלמד במקצוע?

- המקצוע יקנה כלים בסיסיים ל-RE של תוכנות ונוזקות
  - בעיקר בהקשר של תוכנה שקובץ ההרצה שלה זמין
  - בחלקו הראשון של המקצוע נציג עקרונות וכלים
  - בחלקו השני ניישם את הידע ונחקור נוזקות פשוטות
  - בחלקו השלישי נרחיב גם למקרי RE בעלי טכנולוגיות אחרות

- המקצוע מתמקד בסביבת חלונות 32 סיביות
  - אך העקרונות זהים למערכות הפעלה ומעבדים אחרים

- נתייחס גם למניפולציות של קוד, ניתוח וירוסים, סביבות תוכנה שונות, ו-RE למקרים ייחודיים אחרים
  - וגם נתייחס ל-RE של חומרה והתקפות ערוץ צד