שולים מוקשים – על שליטה בזמן ריצה

מאת vbCrLf (אורי להב)

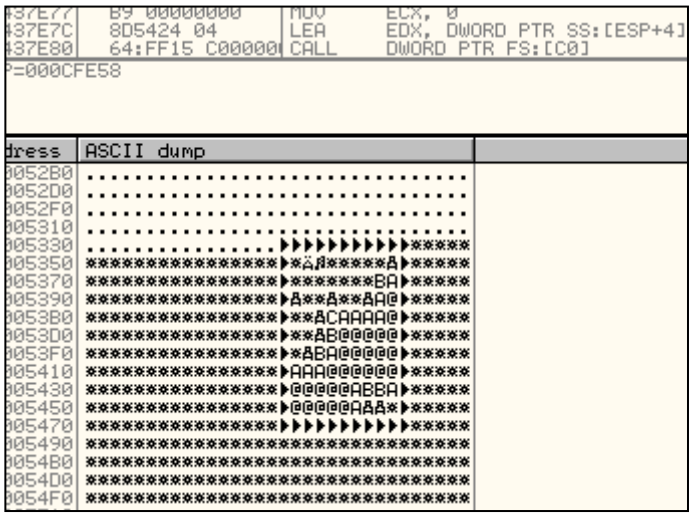
הקדמה

במאמר זה אני רוצה להציג טכניקות מעניינות בשליטה על תהליך רץ ב-Windows, או במקרה שלנו- רמאות בשולה המוקשים. התוכנה שניצור תעצור את הזמן ותגרום לשולה המוקשים לגלות לנו איפה מוחבא כל מוקש, והכל בעזרת עריכות זיכרון פשוטות. כדי לעשות את זה נשתמש בטכניקה הנקראת **הזרקת DLL**. אנחנו נשתמש ב-Visual C++ (וכמובן שאפשר להשתמש בכל IDE ומהדר אחר) עבור בניית מזרק ו-DLL, נשתמש ב-OllyDbg וב-LordPE כדי לאסוף מידע ולבצע קצת עריכות בקובץ. וכמובן את שולה המוקשים של Windows XP. נדרש ידע בסיסי ב-OllyDbg, רצוי ידע ב-C++ (כדי להבין את הקוד).

איסוף המידע

אנו רוצים לעשות שני דברים: גילוי המוקשים ו-הקפאת הזמן.

בשלב הראשון, כדי לגלות איך ואיפה מתכנתי Microsoftהחליטו לשמור את המצב של הלוח (מיקום המוקשים, דגלים, סימני שאלה, וכו') נפתח את התוכנה ב-OllyDbg, נריץ אותה, נשהה (Pause) אותה ונסתכל ב-Dump (החלון התחתון). נגרור קצת למטה ונראה בלוק גדול (בגודל הלוח המקסימלי שאפשר לבחור) שרובו מלא ב-0x0F, חלק ב-0x10 (חץ) וחלק ב-0x8F. כמו שאתם רואים, 0x8F הם המוקשים!



(מפת המוקשים ב-Dump של התהליך)

ככה נוכל למצוא בדיוק איפה הם.

0x10 זה הגבולות של המשחק, ו-0x0F זה ריבוע לא 'לחוץ'. המשיכו את המשחק ולחצו על מוקש. בדקו מהו הערך שמסמן מוקש גלוי, דגל, וכל דבר שמעניין אתכם. ומה שהכי מעניין:

- **מוקש גלוי - 0x8a**
- **מוקש נסתר - 0x8f**

בנוסף, נזכור שהטבלה מתחילה ב-**0x01005340** ומסתיימת ב-**0x0100569f** (הערכים עלולים להיות שונים אצלכם בגלל גרסאות שונות).

נעבור לשלב הבא - המידע שאנחנו צריכים כדי לבצע את הקפאת הזמן. נצטרך למצוא באיזו כתובת בזיכרון winmine.exe שומר את השניות שעברו מתחילת המשחק. OllyDbg מאפשר לנו לחפש בתוך הזיכרון של התוכנה, אז פשוט נחפש את מספר השניות שעברו בזיכרון:
נריץ את המשחק תחת OllyDbg, נתחיל משחק ונחכה שיגיע ל-3 שניות ונעשה Pause. גללו קצת למטה (או שתריצו חיפוש) ונראה כתובת (או כמה) שמכילות 3. נריץ עוד פעם עד 5, ונבדוק איזו אחת מהמשבצות הפכה ל-5. ליתר בטחון אפשר לוודא עוד פעם. הכתובת שקיבלתי היא **0x0100579c** שזו כתובת המשתנה שמכיל את מספר השניות.

הזרקת DLL

לאחר שאספנו את המידע הדרוש נבנה תוכנה שתערוך את הכתובות האלו לערכים שנקבע (לדוגמא, לאפס את המשתנה של הזמן). כדי לעשות את זה נשתמש בטכניקה הנקראת **הזרקת DLL**.
כל תוכנה רצה טוענת למרחב הזיכרון שלה מספר קבצי DLL (קבצים המשמשים כספרייה או מאגר פונקציות, חלקם הכרחיים עבור כל תהליך כדוגמת Kernel32.dll) ותוך כדי ההרצה קוראת לפונקציות מתוך DLL-ים אלו. אנו ננצל אופציה זו ונטען DLL שנכתוב (שיכיל את קוד הרמאות שלנו) לתוך שולה המוקשים. DLL זה ידאג לשמור את המוקשים גלויים ואת הזמן קפוא.

נפתח פרוייקט חדש ב- Microsoft Visual C++ (גרסה 2010 במקרה שלי) ונבחר ב- Win32 Console Application (בחרו באופציה של פרוייקט ריק). נעתיק לתוכו את הקוד שנמצא ב-main.cpp המצורף ל-PDF זה.
השלב הראשון הוא מציאת ה-PID (מזהה ייחודי לכל תהליך תחת Windows):

```
DWORD pid = procNameToPID("winmine.exe");
```

ע"י קריאה לפונקציה procNameToPID. זהו שלב פחות חשוב, אז לא אפרט עליו (אפשר לקרוא על התהליך ב-MSDN). השלב הבא הוא:

```
dllInjection(pid, "DLL.dll");
```

זה החלק החשוב. נסתכל בתוכן של הפונקציה dllInjection. כדי לטעון את ה-DLL אנו משתמשים ב-LoadLibrary שנמצא בתוך Kernel32:

```
HMODULE kernel32 = GetModuleHandle("Kernel32");
FARPROC loadLibrary = GetProcAddress(kernel32, "LoadLibraryA");
```

בשורה הראשונה אנו מקבלים Handle למודול Kernel32, ובשורה השנייה אנו מקבלים מצביע (Pointer) לפונקציה LoadLibraryA (גרסת ASCII של LoadLibrary).

השלב הבא הוא שינוי הרשאות. לא לכל תהליך יש הרשאה לטעון ולשנות זיכרון של תהליך אחר, ולכן אנו מבקשים הרשאות DEBUG. לא נכנס במאמר זה לתהליך עצמו (מידע ב-MSDN).

השלב הבא הוא בקשת Handle שנוכל לבצע עליה (ז"א על התהליך) פעולות:

```
HANDLE processHandle = OpenProcess(PROCESS_ALL_ACCESS, false, pid);
```

כאן מגיע החלק המעניין. אנו נצטרך לקרוא ל-LoadLibrary על התהליך של שולה המוקשים (כדי לטעון את ה-DLL שלנו). הבעיה היא שה- LoadLibrary מקבלת את מצביע לשם ה-DLL כפרמטר, אבל שם ה-DLL נמצא במרחב הזיכרון של התוכנה שלנו ולא של שולה המוקשים (כדרך של הגנה ואבטחה, כל תהליך ב-Windows רץ במרחב כתובות משלו, מה שאומר שלדוגמא הכתובת 0x101010 בתהליך מסוים, תצביע לכתובת פיזית אחרת בתהליך אחר). ולכן נקצה זיכרון מספיק גדול כדי להכיל את שם ה-DLL **בתהליך השני** של שולה המוקשים (כמו שימוש ב-memalloc או new, אבל מתהליך אחד לאחר), נעתיק לשם את שם ה-DLL ונקרא ל-LoadLibrary בתהליך השני עם מיקום שם ה-DLL בתהליך ההוא.

הקצאת מקום מרוחקת:

```
LPVOID remoteDllName = VirtualAllocEx(processHandle, NULL, dll.size()+1,
MEM_COMMIT, PAGE_READWRITE);
```

אנו מקצים זיכרון בתהליך המרוחק (בעזרת ה-processHandle שקיבלנו מ-OpenProcess) בגודל שם ה-DLL (עוד לא מעתיקים אותו). אנו מקבלים מצביע לשם ה-DLL במרחב הכתובות של התהליך השני (אי-אפשר לגשת אליו מהתהליך שלנו ישירות). שימו לב שהוספנו 1 לגודל שם ה-DLL בגלל ה-NULL שיתווסף בסוף (C-style String). וכאן אנו כותבים את שם ה-DLL לכתובת שקיבלנו (remoteDllName):

```
WriteProcessMemory(processHandle, remoteDllName, dll.c_str(),
dll.size()+1, &dummy);
```

וסוף סוף אנו קוראים לפונקציה LoadLibrary (בעזרת הכתובת שקיבלנו בהתחלה, אם אתם זוכרים...) על ידי יצירת Thread חדש בתוך התהליך, ושולחים לה את remoteDllName כפרמטר:

```
HANDLE remoteThread = CreateRemoteThread(processHandle, NULL, 0,
(LPTHREAD_START_ROUTINE)loadLibrary, remoteDllName, 0, NULL);
```

ברגע זה ה-DLL נטען לתוך שולה המוקשים, מה שאומר שנוכל להריץ בתוכו קוד!

בשורות הבאות התוכנה מחכה שה-Thread יסתיים, משחררת את המשאבים ויוצאת:

```
bool finished = WaitForSingleObject(remoteThread, 10000) !=
WAIT_TIMEOUT;
VirtualFreeEx(processHandle, remoteDllName, dll.size(), MEM_RELEASE);
CloseHandle(processHandle);
```

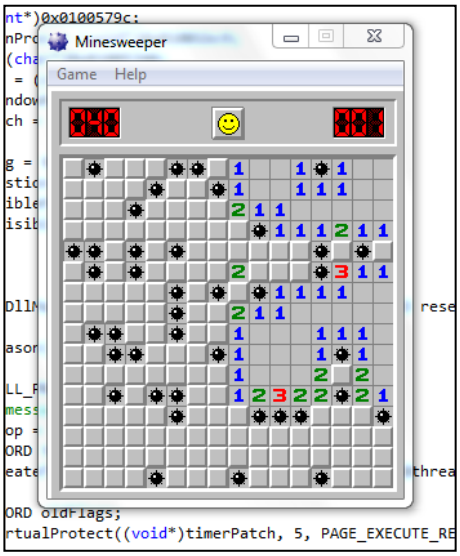
לאחר שכתבנו את קוד טעינת ה-DLL נעבור לכתיבת ה-DLL עצמו.

כתיבת ה-DLL

המטרה של קבצי DLL בד"כ היא לשמש מעין ספריה המכילה פונקציות. כל תוכנה שרוצה להשתמש בפונקציות בספריה זו טוענת את הספריה (קובץ ה-DLL) ולאחר מכן קוראת לפונקציות מתוכה.
אנו נשתמש ב-DLL בצורה קצת אחרת. כידוע, לכל תוכנה (בשפת תכנות פרוצדורלית) יש פרוצדורת **main** שנקראת ברגע שהתוכנה **רצה**. כך לכל DLL, יש פונקצית **DllMain** שנקראת ברגע **שטוענים** את ה-DLL (ובעוד כמה ארועים). בתוך פונקציה זו אנו נכתוב קוד שיאפס את הזמן ויגלה לנו את המוקשים (או כל דבר שנרצה), וקוד זה ירוץ ברגע שנטען את ה-DLL בעזרת התוכנה שכתבנו קודם.

נפתח פרוייקט חדש מסוג Win32 Console Application ונבחר DLL Project. העתיקו את הקוד המצורף שב-dll.h ו-dll.cpp למקומות המתאימים. הדרו את התוכנה וה-DLL שבנינו ושנו את שם ה-DLL ל-"DLL.dll" (או שהתאימו את שמו בתוכנה בקריאה ל-dllInjection). הניחו את שולה המוקשים וה-DLL באותה התיקיה, הריצו את שולה המוקשים.

לאחר מכן הריצו את המזרק כמנהל (נצרך ב-Vista ומעלה אאל"ט), ואם לא יהיו שגיאות תוך שניה כל המוקשים יהיו גלויים והזמן יפסיק לרוץ. מגניב, לא?



אז איך זה פועל? נציץ בקוד:

```
int *time = (int*)0x0100579c;
char *table = (char*)0x01005340;
char *tableEnd = (char*)0x0100569f;
const HWND *windowHandle = (HWND*)0x01005b24;

const char flag = 0x8e;
const char question = 0x0d;
const char visibleMine = 0x8a;
const char invisibleMine = 0x8f;
```

קודם כל מוגדרות הכתובות והערכים של מה שמצאנו בתחילת המאמר. הוספתי גם את המיקום שבו המשתנה שמכיל את ה-HWND של החלון (מצאתי אותו ע"י מציאת ערך ה-HWND דרך החלון Windows שב-OllyDbg, וחיפוש הערך ב-Dump). אנו נצטרך את ה-HWND הזה עוד מעט.

בהמשך הקוד אנו רואים ה-**DllMain** שדיברנו עליו. פונקציה זו מקבלת מספר פרמטרים שהחשוב לנו הוא reason. פרמטר זה אומר לנו למה הפונקציה נקראה. DLL_PROCESS_ATTACH כאשר טוענים את ה-DLL לתוכנה, DLL_PROCESS_DETACH כאשר 'מנתקים' אותו (כאשר קוראים ל-FreeLibrary או שהתהליך נסגר), DLL_THREAD_ATTACH ו-DLL_THREAD_DETACH כאשר התוכנה (שולה המוקשים) יוצר או סוגר Threads.

השניים שמעניינים אותנו הם DLL_PROCESS_ATTACH ו-DLL_PROCESS_DETACH. ברגע שה-DLL נטען (ATTACH) אנו ניצור Thread חדש שירוץ על הזיכרון 'ויתקן' את הזמן והמוקשים. ברגע שה-DLL יוצא מהתוכנה (התוכנה תסגר) אנו נבקש מה-Thread שלנו להסגר:

```
case DLL_PROCESS_ATTACH:
   stop = false;
   DWORD threadId;
   CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)&threadProc, NULL, 0,
&threadId);
   break;

case DLL_PROCESS_DETACH:
   stop = true;
   break;
```

קוד ה-Thread נמצא ב-threadProc:

```
DWORD threadProc(LPVOID lpdwThreadParam)
{
   bool found;
   char *cur;
   while (!stop)
   {
       *time = 0;
```

```
        found = false;
        for (cur = table; cur != tableEnd; cur++)
            if (*cur == invisibleMine)
                *cur = visibleMine, found = true;

        if (found)
            RedrawWindow(*windowHandle, NULL, NULL, RDW_INVALIDATE |
RDW_ERASE | RDW_UPDATENOW);

        Sleep(1000);
    }

    return 0;
}
```

כל עוד המשתנה stop הוא לא true (מה שיקרה כאשר נקבל DLL_PROCESS_DETACH) בצע:

- אפס את time שמצביע לכתובת שקיבלנו בתחילת המאמר.
- עבור על כל טבלת המוקשים, אם מצאת מוקש בלתי נראה, הפוך אותו לנראה.
- אחרי שעברת על כל הטבלה, אם הפכת מוקשים יש צורך לפקוד על התוכנה לצייר מחדש את הלוח.
- שינה של שנייה (1000 מילישניות) כדי שלא להעמיס.

זה הכל! כתבתם תוכנה ש'מתעלקת' על תהליך ועורכת לו את הזיכרון, או במילים אחרות - 'טריינר'.

עצירת הטיימר: דרך שניה

בנוסף לעריכת זיכרון, אפשר אפילו לשנות את קוד ה-Assembly של התוכנה. לדוגמא, במקום לאפס את הטיימר כל שנייה, נמחק את הקוד שמקדם אותו כל שנייה וככה נעצור אותו!

כדי לעשות את זה נצטרך למצוא את השורה שעושה את זה. נפתח את שולה המוקשים ב-OllyDbg, נתחיל משחק ונשהה אותו דרך OllyDbg. נלחץ על חלון ה-Dump (למטה), נלחץ Ctrl+G ונכניס **0100579c** (מיקום המשתנה של מספר השניות שמצאנו) כדי למצוא את מספר השניות. נלחץ לחצן ימני על המשבצת המדויקת, נבחר Breakpoints ואז Memory ובחלון נבחר רק ב-Write access. ונריץ...

OllyDbg יעצר כמעט מיד בשורה:

```
01002FF5  |.  FF05 9C570001 INC DWORD PTR DS:[100579C]
```

שורה זו, היא השורה ששינתה מיקום זה בזיכרון. כמו שאתם רואים, השורה הזו היא בעצם INC 100579C - הוספת אחד לכתובת שאנו כבר מכירים. מה שאנו רוצים לעשות זה להפוך אותה מפקודת INC לפקודת NOP (שבעצם לא עושה כלום).

נצטרך לשים לב לשני דברים - הראשון, **הכתובת**. הפקודה יושבת ב-0x01002FF5. השני, **מספר הבתים**. כל פקודת Assembly תופסת מספר שונה של בתים. כמו שאתם רואים קוד המכונה של פקודה זו בהקס הוא: FF05 9C570001, שהם 6 בתים (כל 2 תווי הקס הם בית אחד). לעומת זאת קוד המכונה של הפקודה NOP הוא 90, רק בית אחד. לכן, נחליף את כל ששת הבתים (כל פקודת ה-INC והפרמטרים שלה) ב-90 (מה שיהפוך את זה לשש פקודות NOP רצופות). וככה אין פקודה שמורה לזמן להתקדם.

לפני שאנו עושים את זה נצטרך להתגבר על בעיה אחרונה. לכל קטע זיכרון יש דגלים המסמנים האם המידע שבתוכו ניתן להרצה, ניתן לשינוי, וכו'. מה שאנו מנסים לשנות נמצא בתוך מקטע הקוד, שמסומן כלא ניתן לכתיבה - ובצדק, כי בשימוש רגיל לא אמור להיות שום שינוי בזמן ריצה בקוד התוכנית. ולכן נצטרך להשתמש בפקודה VirtualProtect כדי להוריד הגנה זו.

נמחק את השורה:

```
*time = 0;
```

מכיוון שהיא כבר לא נצרכת, ונכניס את הקוד הבא לאחר יצירת ה-Thread (בעזרת CreateThread):

```
DWORD oldFlags;
VirtualProtect((void*)timerPatch, 6, PAGE_EXECUTE_READWRITE, &oldFlags);

char *cur;
int i;
for (cur = timerPatch, i = 0; i < 6; i++, cur++)
   *cur = 0x90;

VirtualProtect((void*)timerPatch, 6, oldFlags, &oldFlags);
```

בשלב הראשון אנו משנים את ההרשאות ל-PAGE_EXECUTE_READWRITE (הרשאות ריצה, קריאה וכתיבה) ושומרים את ההרשאות הישנות ב-oldFlags.

בשלב השני אנו רצים מתחילת timerPatch (ששווה ל-0x01002FF5 שזה מיקום פקודת ה-INC) שישה בתים והופכים את כולם ל-90 הקס - הופכים את כולם ל-NOP.

ובשלב השלישי - החזרת ההרשאות למצב הקודם ששמרנו ב-oldFlags (מוגן לכתיבה).

קמפלו והריצו. השעון לא יזוז, מכיוון שהפקודה שמורה לו להתקדם מחוקה.

עכשיו, כשהכל עובד כמו שצריך, אני רוצה להציג טכניקה אחרונה. הטכניקה נקראת Code Cave. כאשר קובץ התוכנה נוצר ע"י המהדר והלינקר נוספים קטעים ארוכים של NULL (ערכים של 0) בסוף כל section כדי לעשות padding (מילוי כדי להגיע לגודל מסוים).
מה שאומר שיש לנו קטעים ארוכים, ריקים, שהם לא בשימוש התוכנה. בקטעים אלו אפשר להכניס כל קוד Assembly או מידע שנרצה (ואח"כ גם לעשות קפיצה מקוד התוכנה אל הקוד שכתבנו). מכאן בא השם **Code Cave** - בתוך כל האפסים יש **'מערת' קוד**.

חזרה לשולה המוקשים. עד עכשיו לא נגענו בקובץ שולה המוקשים עצמו, אבל בשביל השלב הזה אנו נערוך אותו בעזרת OllyDbg. כדי להפטר מהתוכנה שטוענת (מזריקה) את ה-DLL נכתוב קוד בתוך מערת קוד בשולה המוקשים שיעשה בדיוק את מה שהתוכנה המזריקה עשתה- טעינת ה-DLL בעזרת LoadLibrary.

אחרי שכותבים את הקוד במערה צריך להוסיף קפיצה לקוד שבמערה מקוד התוכנית. נצטרך להגיע ל-Entry Point (נקודת 'הכניסה' - הפקודה הראשונה שרצה), **להחליף** אותה בפקודת JMP למערה, במערה לקרוא ל-LoadLibrary, להריץ את הקוד שהיה ב-Entry Point שהחלפנו ב-JMP ולחזור לשורה שאחרי ה-JMP שב-Entry Point.

לעבודה. ה-EP (קיצור ל-Entry Point) של שולה המוקשים יושב ב-01003e21, וקוד האסמבלי שם הוא:

```
01003E21   |.  6A 70          PUSH 70
01003E23   |.  68 90130001    PUSH 01001390
01003E28   |.  E8 DF010000    CALL 0100400C
01003E2D   |.  33DB           XOR EBX,EBX
01003E2F   |.  53             PUSH EBX
...
```

אנו רוצים להחליף את השורה או שתיים הראשונות ב-JMP. פקודת ה-JMP שלנו לוקחת חמישה בתים, שזה אומר כל הפקודה הראשונה (PUSH 70) ושלושה בתים מהפקודה השנייה (PUSH 01001390) יוחלפו בפקודה JMP, ולכן נמחק את שתיהן (אבל נזכור אותן!) ונכתוב במקומן (בחרו אותן ולחצו רווח):

```
JMP 01004ac8
```

זו הקפיצה למערה שלנו. איך אני יודע שזה 01004ac8? התהליך לפניכם:

נמצא קודם כל מערה - שזה לא קשה בכלל, פשוט תגללו עד שתגיעו לקטע שכולו אפס. אני בחרתי את הכתובת 01004ac0. בחרו מכתובת זו ועוד כמה כתובות קדימה, לחצו לחצן ימני, Edit, ואז Binary edit. כיתבו את שם ה-DLL (במקרה שלנו DLL.dll) וודאו שיש NULL בסוף השם. לחצו Ok ואח"כ Ctrl+A כדי לעשות אנליזה מחדש של הקוד. בחרו את השורה שמיד אחרי שם ה-DLL (במקרה שלי 1004ac8), לחצו רווח וכתבו את הקוד הבא (שורה, אנטר, ...):

```
PUSH 01004ac0
CALL LoadLibraryA
PUSH 70
PUSH 01001390
JMP  01003E28
```

השורות הראשונה והשניה: השורה הראשונה מכניסה למחסנית את כתובת שם ה-DLL (כפרמטר) והשורה השנייה קוראת לטעינת ה-DLL שלנו.

השורות השלישית, הרביעית והחמישית: הן השורות שאותן החלפנו ב-JMP ב-EP (זוכרים?), והשורה האחרונה היא חזרה ל-EP.

לחצו על Copy to Executable ואז על All modifications שמרו את הקובץ (כדאי בשם אחר) באותה תיקיה של ה-DLL.

לאחר שהכנסנו את הקוד למערה נשאר שלב אחרון. זוכרים את ההגנה על הזיכרון מהשלב הקודם? גם כאן יש לנו בעיה דומה. את מערת הקוד הזו כתבנו במקטע (section) השייך למידע (data section) ולא לקוד. מכיוון שהוא מיועד למידע הלינקר סימן אותו כלא ניתן להרצה, ולכן כאשר מריצים את התוכנה, הקוד שנמצא במערה שלנו לא יוכל לרוץ והתוכנה תקרוס. נאלץ לשנות את הגדרות המקטע.

נפתח את LordPE, נבחר ב-PE Editor ונבחר בקובץ הערוך שלנו. נלחץ על Sections. שם נראה רשימה, נלחץ לחצן ימני על המקטע שבו שמנו את הקוד - '.data', ונבחר edit section header. נלחץ על הכפתור שליד ה-Flags ונפעיל את הדגלים שקשורים להרצה ("Executable as code" ו-" Contains executable code"). נלחץ Ok, Ok אח"כ Save ונסגור את התוכנית.

זהו, סיימנו. נריץ את התוכנה. ה-DLL אמור להיטען לבד, ושולה המוקשים יציג מיד את כל המוקשים ולא ייתן לטיימר להתקדם.

לסיכום

- התחלנו באיסוף מידע איך שולה המוקשים עובד ואיפה הוא שומר את המידע בעזרת OllyDbg.
- כתבנו תוכנה שמזריקה DLL.
- כתבנו DLL שמשתמש במה שמצאנו בשלב הראשון כדי לרמות בשולה המוקשים, והזרקנו אותו בעזרת התוכנה שבנינו.
- שינינו את הדרך שבה הפסקת הטיימר פועל מאיפוס כל שניה, לדרך טובה יותר - החלפת פקודת ה-INC ב-NOP.
- בעזרת Code Cave ('מערת קוד') גרמנו לתוכנה לטעון את ה-DLL לבד (בלי הזרקה).

בעזרת שימוש בטכניקות האלו אפשר לבנות טריינרים למשחקים, להוסיף פונקצינאליות לתוכנות קוד סגור, לבנות כלים אוניברסליים (לדוגמא, DLL שעובר על כל תיבת סיסמא בתוכנה והופך אותה לתיבה רגילה), וכו'. אין סוף ליישומים האפשריים.

אשמח לקבל תגובות ושאלות לכתובת הדוא"ל: vbCrLf@GMail.com

בימים אלה, אורי ביחד עם אחיו שוקדים על פתיחת בלוג בנושאי מחשבים - תכנות, מערכות הפעלה, Reverse Engeneering ונושאים מעניינים אחרים - מרכז הכפר:

http://www.MerkazHaKfar.co.cc

מצורפים הקבצים:

- main.cpp - קוד ה-Injector שמזריק את ה-DLL
- dll.h, dll.cpp - קוד ה-DLL
- dll_final.cpp - קוד ה-DLL לאחר ה'שכלול' האחרון (הופך את INC ל-NOP)

ניתן להורידם מהקישור הבא:

http://www.digitalwhisper.co.il/files/Zines/0x12/MineSweeping.zip