הסכנה שבקבצי HTML

מאת ישראל חורז'בסקי (Sro)

הקדמה

יש קבצים שכולנו מסכימים שהרצה שלהם היא דבר מסוכן. כאלה הם קבצי exe, reg, bat, cmd או בקיצור - קבצי הרצה. בהתאם, כאשר אתרי אינטרנט ינסו להוריד לנו קבצים אלה נקבל התראה שאלה קבצים מסוכנים ממקור לא ידוע. דוגמא נוספת - שליחת קבצים במייל. קבצים עם סיומת מסוכנת לא ניתן לשלוח במייל, לעיתים אף לא מכווץ בזיפ.

כאן במאמר נחקור קבצי HTML, נראה איך שכולם מתייחסים אליהם כקבצים לא מסוכנים, ננסה לנתח למה, ואז נראה גם כמה שהם מסוכנים, עד כדי קריאת כל הקבצים מהמחשב של ה-Victim. במשפט אחרון של ההקדמה אוסיף שהבדיקות בוצעו על דפדפני Firefox[22] , Chrome[28], IE[10] עדכניים לזמן כתיבת המאמר.

האם גוגל ומייקרוסופט חושבים שקבצי HTML מסוכנים

שיטות ההפצה הקלות ביותר של קבצים הינם הורדה מאתר אינטרנט או שליחה במייל. בדוגמא הבאה שמתי את קובץ ה-PoC (final.htm) יחד עם קובץ downloader.php בתיקיה, והקוד של downloader.php הוא:

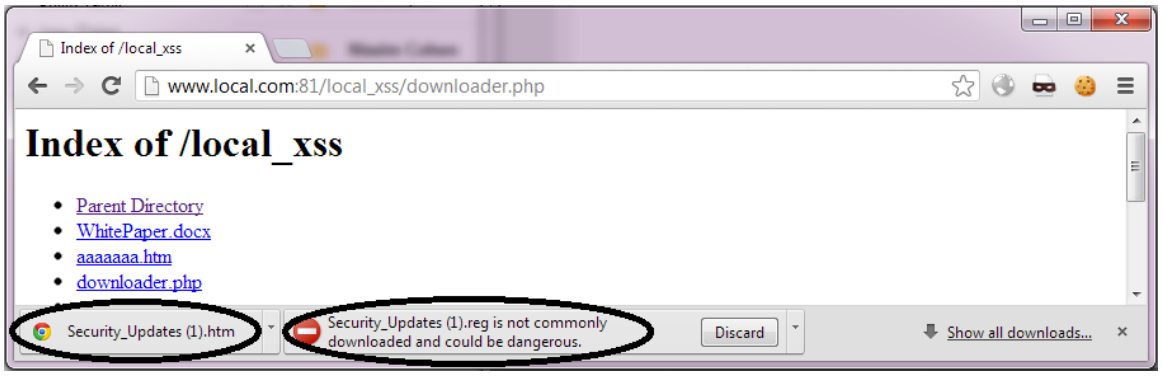
```
<?php
 header('Content-Type: text/html');
 header('Content-Disposition: attachment; filename=Security_Updates.htm');
 echo file_get_contents('final.htm');
?>
```

הקוד שולח לדפדפן 2 כותרים (Headers) על כך שזה קובץ להורדה, שם הקובץ נקבע ל-Seucirty_Updates.htm כדי לבצע Social engineering למשתמש, שיריץ אותו. זה הכל, ב-3 שורות גרמנו לדפדפן להוריד קובץ HTML, בעת ריצה שלו שום דפדפן לא מתריע על כך שזה קובץ מסוכן ממקור לא ידוע.

גולש שגולש לדף downloader.php יקבל את התוצאות הבאות:

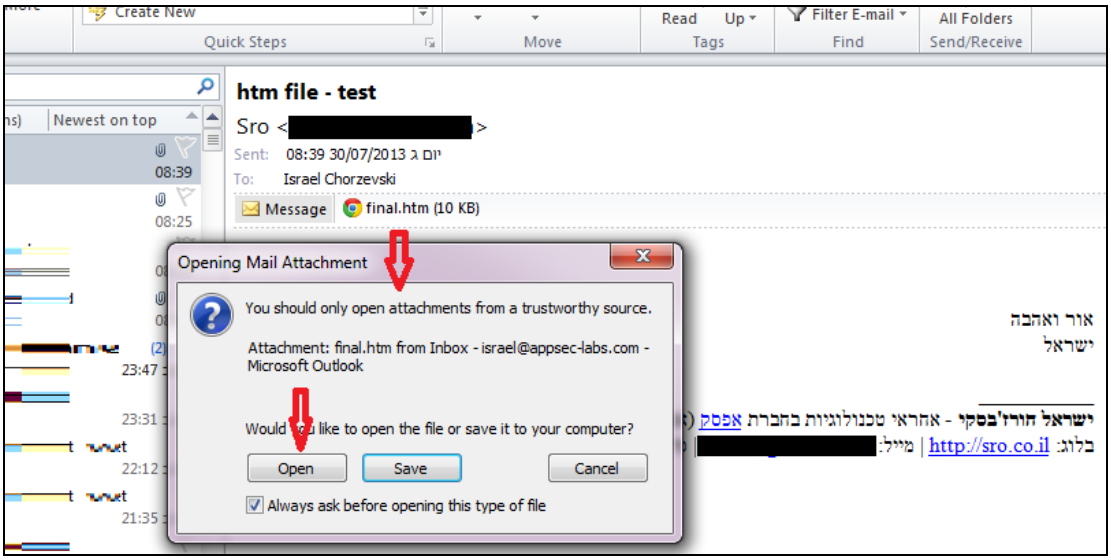
דפדפן	תוצאה
פיירפוקס	שואל אם להוריד או להריץ
כרום	מוריד מיידית
אינטרנט אקספלורר	שואל אם להוריד או להריץ

השוואה גרפית ניתן לראות בתמונה הבאה:

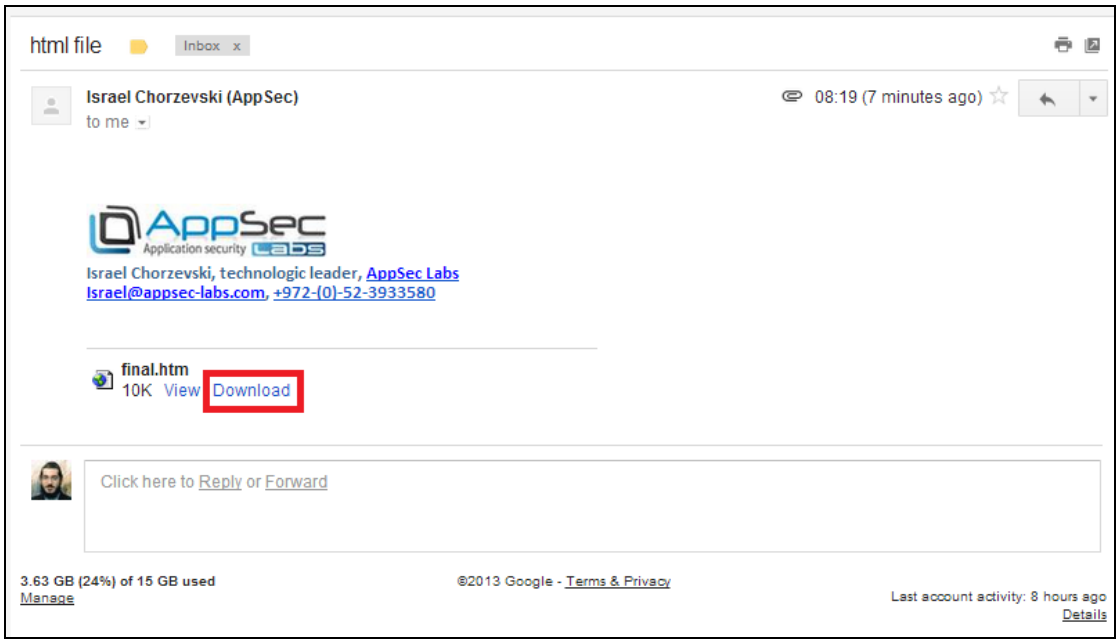


ניתן לראות איך קובץ reg מציג אזהרה בדפדפן כרום, לעומת קובץ htm שיורד מיידית.

שיטת הפצה נוספת, כאמור, היא באמצעות אימייל. כידוע, מערכות אימייל חוסמות אימיילים שמכילים קבצים בעלי סיומות מסוכנות. בתמונות הבאות ניתן לראות איך גם gmail וגם outlook לא חסמו הורדה והרצה של קובץ htm שנשלח במייל. באוטלוק:



:Gmail



מדוע קבצי HTML לא נחסמים? ככל הנראה החשבון הוא כזה. אם יש לי (כתוקף) 0-day (בעיית אבטחה לא ידועה / לא מתוקנת) על אחד הדפדפנים שאותו אני רוצה לנצל עם java script, אוכל להפעיל אותו על המשתמש באמצעות שליחת לינק במייל, או כשהוא גולש לאתר שלי - במקום לגרום להורדת קובץ. אז אם כן, מה מפריע לנו שהוא יפעיל את הקובץ מהמחשב המקומי שלו? בהמשך נראה שהדפדפנים מנסים להתייחס לקובץ שרץ לוקאלי, באופן זהה לקובץ שרץ מרחוק, אולם נגלה שכל שיטת חישוב ה-origin שלהם שאמור לבצע את הפרדת ההרשאות לוקה בחסר.

לקרוא את נתיב ההורדה (FPD - Full Path Disclosure)

זה ה-"Exploit" הכי בסיסי לקובץ HTML שרץ לוקאלית. רק לקרוא את ה-document.location שמציג ה-URL. בקובץ לוקאלי ה-URL הוא הנתיב המלא של הקובץ. בעיה מסוג זה מכונה Full Path Disclosure, זה רק חשיפת מידע, אבל מידע שמאפשר לנו לבצע מתקפות שונות של קריאת קבצים (מתקפות שכאלה מנצלות חולשות בדפדפנים, פלאגינים והרחבות של דפדפנים שכתובים בצורה לא מאובטחת).

מידע נחמד נוסף שבדר"כ נחשף, זה שם המשתמש של הגולש, כי דיפולט ההורדה יהיה פעמים רבות התיקיה Downloads שב-My Documents שבנתיב המלא זה משהו כמו:

```
C:\users\VICTIM\Downloads
```

עם שם המשתמש אפשר לבצע Brute Force ל-RDP ועוד. אם עדיין לא השתכנעתם למה זה שימושי, חכו לסעיף הבא, רק נסכם קודם את הדפדפנים הפגיעים: כולם.

דפדפן	תוצאה
פיירפוקס	מציג נתיב לוקאלי מלא
כרום	מציג נתיב לוקאלי מלא
אינטרנט אקספלורר	מציג נתיב לוקאלי מלא

לקרוא קבצים מהכונן הקשיח (LFD - Local File Disclosure)

אני חושב שזה הממצא הכי חמור בסיפור הזה, אנחנו מדברים על קריאת קבצים מהארד-דיסק. כאן מן הראוי להרחיב מעט על מודל ה-SOP (Same Origin Policy שמונע גישה לקובץ של מערכת/דומיין אחר) בקבצים לוקאלים. יש לנו שלוש גישות:

- כל קובץ הוא "ממלכה" משלו, אין לאפשר זליגת מידע בין קבצים, אפילו באותה תיקיה.
- כל תיקיה היא אפליקציה, "סביר" שקובץ בתיקיה ירצה לגשת לקבצים שנמצאים לידו / בתיקיות משנה.
- אין דומיינים כי זה קובץ לוקאלי, לקובץ יש גישה לכל יתר הקבצים במערכת הקבצים, אפילו בכונן אחר.

לגבי קריאת קבצים (AJAX, או אם תרצו - CORS), כרום ואינטרנט אקספלורר הולכים בגישה של "כל קובץ לחוד" ואף קובץ לא יכול לקרוא קובץ אחר. פיירפורקס לעומתם תומך בגישה ליברלית והוא מאפשר לכל קובץ לגשת לקבצים אחרים בתיקיה / בתיקיות משנה.

כך שכשאני מוריד קובץ HTML לשולחן העבודה ופותח אותו, הוא יכול לקרוא את כל הקבצים שיש לי על שולחן העבודה, ואת הקבצים שבתיקיות שבשולחן העבודה. הוא רק צריך לדעת (או לנחש) את שמותיהם. לקרוא את הקבצים ולשלוח אותם לשרת של התוקף.

כעת נותר לי להודות על כך שאני לא מוריד קבצים ישירות על כונן C עצמו. זה אומר קריאה של כל הקבצים בכונן C., קבצים בעלי כל סיומת שהיא exe, txt, הכל. כעת אפשר להבין למה וכמה לדעת את נתיב ההורדה עוזר לנו, כי כדי לקרוא קובץ אנחנו צריכים לדעת את שמו ומיקומו.

דפדפן	תוצאה
פיירפוקס	מאפשר לקרוא כל קובץ שהוא מהתיקיה הנוכחית ומתיקיות המשנה שלה
כרום	חוסם קריאה של קבצים אחרים
אינטרנט אקספלורר	חוסם קריאה של קבצים אחרים

סריקת כוננים קשיחים

יכול להיות שימושי בכל מיני מקרים (אפי' עבור הדבר הפשוט - זיהוי המשתמש. למרבית המשתמשים אין כונן Q, אז אם פעמיים נתקלנו במשתמש עם כונן Q, כנראה שזה אותו אחד). בכל אופן, אנחנו יכולים לבצע סריקה ולדעת איזה כוננים זמינים במחשב.

בגדול הטכניקה היא פשוטה, כדי לבדוק אם כונן Q קיים, ניצור Iframe, נגדיר לו ארוע onload שיקרא לפונקציה שתבצע רישום שה-Iframe נטען. בנוסף, נגדיר ל-Iframe את ה-src ל:

```
file:///Q:/
```

כעת זה מתחלק בין הדפדפנים. כרום מגיב זהה בין מקרה שהכונן קיים למקרה שהוא לא קיים. פיירפוקס מריץ את onload רק אם הכונן קיים, IE מריץ את onload רק אם הכונן **לא** קיים.

זוג פונקציות Java Script שכתבתי (אחת ל-FF, אחת ל-IE) מבצעות סריקה על כל הכוננים ומחזירות רשימה בתוך פחות משניה. פונקציות אלה ובאופן כללי PoC לכל המוזכר במאמר ניתן להוריד בלינק שיופיע בסוף.

באמצעות אותה שיטה ניתן ב-Firefox וב-Internet Explorer לבדוק אם קובץ כלשהו קיים. במקום להגדיר את ה-src ל-file:///Q:/ נגדיר אותו ל-file:///c:/windows/win.ini ונבדוק האם ארוע onload רץ על ה-Iframe.

סריקת פורטים (או: הוכחה שכרום טוב פי 4 מ-FF)

אציין שאפשר לסרוק גם עם JS שרץ מאתר בעת גלישה רגילה לאתרי אינטרנט, ולא רק מדף לוקאלי. אבל מהבדיקות שביצעתי, מתברר שסריקה מדף לוקאלי עובדת הרבה יותר טוב ובאופן יותר מהיר.

איך מבצעים סריקת פורטים? הצורה המהירה היא באמצעות ניסיון קריאת דף עם AJAX (ליתר דיוק, כאן זה כבר CORS, אנחנו מנסים לקרוא דף מדומיין אחר). אפשר לנצל את זה בשביל סריקת פורטים לוקאלית, ואפשר להשתמש בזה בשביל לסרוק פורטים של מכונות אחרות ברשת/בעולם.

מנסים לקרוא את הדף http://127.0.0.1:8080/. בכל מקרה לא נוכל לקרוא אותו (אלא אם כן הוא מוגדר ומציג כותרים שאומרים לדפדפן שמותר לקרוא אותו גם מדומיין אחר, שלא סביר שזה יקרה), אבל מה שכן נוכל למדוד זה את הזמן שלוקח עד שהדפדפן מסיים לטפל בבקשה.

כעת יש שלוש אפשרויות:

1. בפורט 8080 קיים שרת HTTP, במקרה כזה הדפדפן מסיים את הבקשה (=קורא את התשובה. למרות שהוא לא מציג לנו אותה) בתוך פחות מחצי שניה.
2. בפורט 8080 קיים סרוויס שהוא לא HTTP, במקרה כזה הדפדפן מצפה ל-Response, אבל השרת לא יחזיר לו כזה (בטח לא אחד תקין), מה שאומר שהדפדפן יחזיק קונקשן עד ל-Timeout.
3. אין שום סרוויס/שרת שמאזין בפורט 8080, אורך הטיפול בבקשה של הדפדפן הוא בסביבות שניה בודדת.

אז כעת ננסה "לקרוא" את כל הפורטים, נגדיר טיימאאוט של 3 שניות ל-AJAX ונמדוד את אורך ה-Response. אם האורך הוא פחות מחצי שניה, או 3 שניות (טיימאאוט שהגדרנו), סימן שיש איזשהו סרוויס שמאזין על הפורט הזה. אחרת כנראה שאין מישהו שמאזין על הפורט.

סריקת הפורטים הזו היא לא ממש מדוייקת, אבל כמה שהיא חושפת זה עדיין הרבה ! נזכור שאם המשתמש שלנו נמצא מאחורי NAT (או אפי' כל FW אפליקטיבי, הרבה תוכנות מאזינות רק על ה-loopback), אין לנו אפשרות לסרוק אותו רגיל מבחוץ עם Nmap.

ואיפה מה שהבטחנו לגבי ביצועי דפדפנים?

בהתחלה כתבתי קוד שמבצע סריקה, וכל פעם סורק פורט בודד. התוצאה היא - כדי לסרוק 10 פורטים, צריך 10 שניות.

מי שחושב על workers, אז לא. לא ניתן להוציא בקשות AJAX מ-worker. לכן החלטתי לנצל את מנגנון התורים שקיים בדפדפנים מודרנים. ה"טיפול" של הדפדפן בבקשת AJAX מתבצע א-סינכרונית, כך שניתן להריץ במקביל ניסיון קריאה של הרבה פורטים, והדפדפן יטפל בכולם במקביל.

לדפדפנים ול-JS יש נטייה לא לתת תמיכה מלאה של ביצועים, והם לא מבצעים את מה שהם מבטיחים (נתקלתי בזה גם ב-setInterval שאם הגדרתם לו שייגש ל-DOM, לא תוכלו באמת להריץ אותו אפי' לא כל עשירית שניה, הוא ירוץ מתי שבא לו). מה שקורה זה שנפתחים X ת'רדים, וכל היתר מחכים. מבחינתנו זה שהם מחכים יוצר בעיה - כי פתאום בקשה לפורט סגור שאמורה לקחת שניה, תארך 4 שניות רק כי היא "חכתה". ואנחנו נתקלים ב-False positive (זיהוי שגוי, חושבים שמצאנו משהו). לכן גם העליתי את הטיימאאוט משניה וחצי לשלוש שניות, למקרה ומשהו יעכב את הדפדפן לרגע.

אבל בכל זאת, עדיין הדפדפנים יכולים להריץ כמה וכמה בקשות במקביל. זה מאוד משתנה, כי זה תלוי בעומסים שעל הדפדפן, כנראה שזה גם תלוי בכח המחשוב של המחשב שמריץ אותו, וזה גם תלוי בפורטים הפתוחים במחשב. אם יש פורט ומאזין מאחוריו שרת HTTP, הדפדפן יסיים לטפל בפורט הזה במהירות והוא יתפנה לבא בתור, אם יש פורט שמאחוריו יש סתם מישהו שמאזין, הדפדפן ייתקע איתו שלוש שניות שלמות.

בממוצע גיליתי שכרום מאפשר סריקה של 200 פורטים באופן יחסית יציב, פיירפוקס 50 פורטים, ואינטרנט אקספלורר 5 פורטים. אני מדבר על סריקה מקבילית כמובן. ניתן תמיד להריץ אותם אחד אחרי השני.

החלק האופטימי בסיפור, זה שאם הדפדפן התעכב מכל סיבה שהיא, סך הכל זה אומר שנקבל את הפורט כ-False positive, אז מה שאנחנו נבצע זה סריקה נוספת על כל הפורטים שמצאנו ונאמת שאכן מישהו מאזין אליהם. לא ביג דיל. הרבה יותר טוב ממצב הפוך שבו מתפספסים פורטים. לשמחתנו, אנחנו לא שם.

דפדפנים פגיעים? כהרגלנו בקודש - כולם.

דפדפן	תוצאה
פיירפוקס	מאפשר לסרוק - מהירות בינונית
כרום	מאפשר לסרוק - מהירות גבוהה
אינטרנט אקספלורר	מאפשר לסרוק - מהירות נמוכה

יצויין שהדפדפנים חוסמים פורטים מסויימים ולכן נמצא אותם שהם כאילו תמיד "סגורים" (זה משתנה מעט בין הדפדפנים).

דוגמא לרשימה שמצאתי ברשת על כרום:

1, 7, 9, 11, 13, 15, 17, 19, 20, 21, 22, 23, 25, 37, 42, 43, 53, 70, 77, 79, 80, 87, 95, 101, 102, 103, 104, 109, 110, 111, 113, 115, 117, 119, 123, 135, 139, 143, 179, 194, 210, 389, 443, 465, 512, 513, 514, 515, 526, 530, 531, 532, 540, 556, 563, 587, 601, 636, 993, 995, 2049, 4045, 6000, 6667.

הפרדה בין אתרים לוקאלית במחשב (local storage, web SQL)

כעת אנחנו מגיעים לישורת האחרונה. בדרך כלל אנחנו גולשים לאתרי אינטרנט בשרתים אחרים, וגם אם לוקאלית זה כשאנחנו מריצים שרת HTTP על המחשב שלנו.

אולם, כיוון ש-Java script ו-HTML5 קיבלו יכולות כל כך חזקות, אין זה מן הנמנע שפשוט נלחץ "דאבל קליק" על קובץ HTML ראשי והוא יהווה אפליקציה עצמאית (מזכיר לכם Windows 8?).

זה לא נפוץ כ"כ היום, אבל קיים. נתקלתי בזה באפליקציה שפותחה באפסק (iNalyzer - מערכת לשליטה באפליקציות אייפון, ניתן איתה לראות את המבנה הלוגי של המסכים ולהפעיל אותם ישירות תוך כדי עקיפת לוגיקה שממומשת ב-GUI, ועוד ועוד). נתבקשתי לכתוב שם קטע קצר בקובץ HTML שמשתמש ב-Local storage, וגיליתי שקובץ אחד יכול לגשת למידע ששמר קובץ אחר.

אז בואו נראה מה יש לנו - אפליקציה (= קובץ HTML) שמיועדת לרוץ לוקאלית, שומרת מידע ב-Storages של HTML5 (כמו Local storage, Web SQL). כל קובץ HTML אחר שרץ (וכבר ראינו בהתחלה שלגרום למשתמש להוריד ולהריץ קובץ זה סיפור די קל) יכול לגשת למידע שמאוחסן שם. גם אם אין מידע רגיש עדיין לפי שמות המשתנים / טבלאות ניתן ללמוד שהיוזר משתמש באפליקציה פלונית, מידע שיכול לשמש אותנו בשביל מתקפות שונות (או בשביל לא לבצע מתקפות - כי נתקלנו בחוקר שאנחנו לא רוצים שיגלה אותנו).

דפדפן	תוצאה
פיירפוקס	Local storage - ניתן לקרוא את המידע שנשמר ע"י קבצים מאותה תיקייה Web SQL - המנגנון כלל לא נתמך בפיירפוקס
כרום	Local storage & Web SQL - ניתן לקרוא את המידע שנשמר ע"י קבצים מכל ההארדיסקים, אפי' מכוננים אחרים
אינטרנט אקספלורר	לא פגיע (רק הקובץ עצמו יכול לגשת למידע)

מעקב פיזי - GEO location

זה פשוט באג מעניין. לפחות מייקרוסופט לא יוכלו להגיד שזה פיצ'ר.
כשאתר מנסה לקבל מהדפדפן נתוני מיקום (GEO location), הדפדפן מבקש הרשאה מהמשתמש. למשתמש יש שתי אפשרויות בדרך כלל (תלוי בדפדפן ובגרסה), או אישור חד פעמי, או אישור רב פעמי. אישור חד פעמי זה אומר לדף עצמו עד שהוא נסגר, כך שהדף יכול לבקש אותו שוב ושוב. בכל מקרה, האישור אמור להיות פר אתר (Origin) ולא לזלוג.
מתברר ש-IE שומר את ההרשאה של בקשת נתוני GEO Location לקבצים לוקאלים (קרי: ה-Origin שלהם), ככל הנראה, לפי שם הקובץ ! לא לפי כונן / תיקיה / נתיב מלא.

התוצאה? אם המשתמש גלש לקובץ:

```
C:\big-deal-application\index.html
```

ואישר לדף בדפדפן לקבל נתוני GEO Location. ואני מדגיש - אישור חד פעמי (Allow once). לאחר מכן אם המשתמש גולש באותו טאב (= כרטיסיה) לדף:

```
D:\my_downloads\index.html
```

הדף יוכל לקבל את מיקום ה-GEO Location ללא בקשת אישור מהמשתמש. כי מבחינת הדפדפן זה אותו דף שמנסה לקבל שוב את הפרטים.

דפדפן	תוצאה
פיירפוקס	כל קובץ מבקש אישור לחוד
כרום	קובץ לוקאלי לא יכול לקבל נתוני GEO Location
אינטרנט אקספלורר	באותו טאב, כל קובץ בעל אותו השם יוכל לקבל את נתוני ה-GEO location ללא בקשת אישור נוספת, גם אם הקובץ נמצא בכלל בתיקיה/כונן אחרים

לסיכום

מה שראינו כאן, זה:

1. מערכות מתייחסות לקבצי HTML כאל קבצים לא מסוכנים, בעוד הצלחנו בפועל לקרוא קבצים אחרים מה-Hard disk.
2. דפדפנים לא מבצעים ניהול נכון של Origin לקבצים לוקאלים, וכך קובץ אחד יכול לגשת ל-Storages ששמרו קבצים אחרים.

מה שראינו כאן זה ממצאים שמצאתי בזמן כתיבת המאמר, בטוחני שיש פונקציות נוספות שהדפדפנים לא מטפלים בהם כראוי / לא מטפלים בהם באופן זהה ויש כאן כר למחקר נוסף.

כמשתמש אחשוב פעמיים לפני שאריץ קובץ HTML ולא אסמוך על כך ש"אם האתר היה רוצה, הוא כבר היה מריץ את הסקריפטים / 0-days שלו מ-Remote.

קישור לקובץ PoC למתקפות שהוזכרו במאמר:

http://www.digitalwhisper.co.il/files/Zines/0x2C/the_dangers_of_html_file.zip

על המחבר

ישראל חורז'בסקי, אחראי טכנולוגיות (Tech leader) בחברת AppSec Labs. בנוסף, יועץ אבטחה, פנטסטר, מרצה לבדיקות אבטחה ו-Secure coding ועוד. אשמח לקבל פידבקים / שאלות / הערות / הצעות במייל:

israel@appsec-labs.com