digital whisper
Technologic Papers

One push too far - Exploiting Web Push Notifications

מאת זהר שחר

הקדמה

Push Notifications (להלן "התראות בדחיפה") אינן עניין חדש. למעשה, אנו מכירים את הנושא היטב בהקשר של טלפונים ניידים, כאשר כמעט כל אפליקציה מייצרת עבורנו התראות. עם זאת, בשנים האחרונות היכולת לשלוח התראות בדחיפה התווספה גם אל עולם ה-Web, ועתה ניתן לקבל התראות ישירות אל הדפדפן. לאחרונה, ובעקבות מבול האתרים המבקשים הרשאות לשלוח התראות, החלטתי לבחון את המנגנון בהיבטי אבטחת מידע כדי לנסות ולהבין אילו הרשאות ניתנות לאתרים המשתמשים בהתראות, ואילו סיכונים (אם בכלל) מנגנון זה חושף.

מספיק לשים לב לנקודות הבאות בשביל לגרות את המחשבה של כל תוקף:

- התראות בדחיפה מאפשרות גישה רציפה לדפדפן, גם לאחר שדפדפן נסגר.
- הטכנולוגיה חדשה למדי, חלקים ממנה עדיין בגרסאות "ניסוי", ואין אחידות בין הדפדפנים השונים.
- לא התפרסם מחקר בנושא אבטחה של התראות בדחיפה בעבר (או לפחות לא הצלחתי למצוא מחקר כזה).

ואכן, מחקר קצר יחסית הוביל אותי מספר לתגליות מעניינות:

1. חולשה חדשה ("Zero Day") ב-Firefox המאפשרת ניצול מנגנון ההתראות ליצירת Adware.
2. וקטור התקפה חדש לניצול Cross Site Scripting (XSS) המאפשר גישה רציפה לדפדפן של הקורבן.
3. טכניקה מעניינת לבניית Botnet המשתמש ב-Google כשרת השליטה ובקרה (CNC).

במאמר זה אסקור את מנגנון ההתראות בדחיפה, וכן אציג את הממצאים שזיהיתי בו

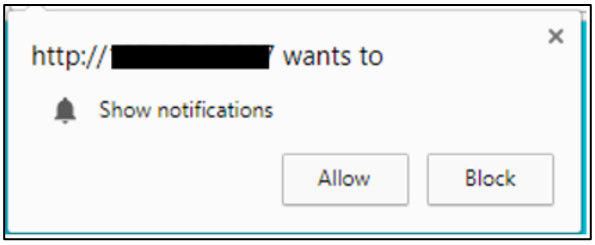
(המאמר פורסם לראשונה באנגלית בבלוג של חברת קומודו)

אז איך עובד מנגנון ההתראות בדחיפה?

קיימים מאמרים רבים באינטרנט אשר מפרטים אודות המנגנון (המאמר הזה חביב עלי), אז אסתפק ברקע הטכנולוגי הבסיסי הדרוש לנו לצורכי מאמר זה. מנגנון ההתראות מורכב משלושה רכיבים - לקוח, שרת וספק השירות.

1. צד הלקוח

צד הלקוח ממומש בדפדפנים תומכים (Chrome ו-Firefox). כאשר משתמש ניגש באמצעות דפדפן נתמך אל אתר המשתמש בהתראות בדחיפה, תוצג בפני המשתמש בקשה למתן הרשאות לשליחת התראות. ההרשאות ניתנות עבור כל אתר בנפרד, והייצוג הגרפי של בקשת ההרשאות גנרי ואינו בשליטת המפתח. כך למשל, נראית בקשה זו ב-Chrome:



במידה והמשתמש מאשר, נוצר ע"י הדפדפן אובייקט JSON הנקרא "Subscription". אובייקט זה מכיל בתוכו כתובת URL ייחודית אשר נוצרה עבור האתר הספציפי והדפדפן הספציפי, וכן ייצוג של מפתח ה**פומבי** של האתר. כך למשל נראה אובייקט Subscription ב-Chrome:

```
{"endpoint":"https://fcm.googleapis.com/fcm/send/cm115VoAuUg:APA91bFOAKnL6zgMiZKGadoH-
voJ5oH5T9gWH4NnWq4FPqyk8QabNbGILlS8ZkZIfbC4RZsXt-
N2COLhjEW6yB2XQRNiyMAxoZ_I5JJxjjj4Q_e0OynN1imUO0rxZZxUDKuM6zo047um","expirationTime":null,
"keys":{"p256dh":"BBClqnjH8hAtzL_iEmdmMTEiPKu2SC-
AZMBJt4Y1bVCACz0hGVUtSVa2xHCTBeN3ke0vmGdZ2ZyZDeDWdyIyh9k=","auth":"O3kgvkoJcV2nU36s_TUSoQ=
="}}
```

בעל המפתח **הפרטי** (התואם למפתח הפומבי שנזכר לעיל) יכול עכשיו לשלוח התראות בדחיפה אל כתובת ה-URL שנוצרה, ע"י פניה אל ספק השירות (כמוסבר בהמשך).

תהליך נוסף המתרחש במקביל לבקשת ההרשאות הוא רישום של Service Worker אל דפדפן המשתמש (Service Workers הם קבצי JavaScript הנשמרים באופן מקומי על תחנת המשתמש, אשר אחראיים על טיפול באירועי צד-לקוח). ה-Service Worker יהיה אחראי על טיפול בהתראות נכנסות. חשוב להדגיש שה-Service Worker רץ תחת Sandbox נפרד משאר קבצי ה-JavaScript באפליקציה (למשל, אין לו יכולת לגשת אל ה-DOM), והוא חייב להגיע מה-Domain של האתר עליו הוא פועל.

2. צד השרת

כצפוי, צד השרת אחראי לשלוח את ההתראות. האתר נדרש להחזיק רשימה של כתובות URL אליהן הוא רשאי לשלוח התראות (כלומר כל אותן כתובות שהתקבלו כתוצאה מתהליך ה-Subscription), ויכול לפנות אליהן ע"י גישה אל ה-API של ספק השירות בצירוף המפתח פרטי. האתר הלימודי הזה יכול לשמש לשליחת התראות בדחיפה אל כתובות URL רלוונטיות.

3. ספק השירות

זוהי ה"רשות" האחראית על הפצת התראות למשתמשים הרלוונטיים (Google בעת שימוש ב-Chrome, Mozilla בעת שימוש ב-Firefox). ספק השירות אחראי גם על ווידוא תקינות המפתחות.

עתה, לאחר שביססנו את הידע הדרוש כדי להשתמש במנגנון ההתראות בדחיפה, נעבור לניצולו!

ממצא מספר 1: חולשה ב FireFox מאפשרת בניית Adware מבוסס התראות

במסגרת המחקר, מצאתי שאם משתמש Firefox מאשר לאתר אינטרנט לשלוח לו התראות בדחיפה, אותו אתר יכול - מתי שהוא רוצה - לפתוח מספר אינסופי של חלונות חדשים (Tabs) בדפדפן של המשתמש הפונים לאתרים לבחירתו של התוקף. הסרטון הבא מדגים את הבעיה.

אז מה בעצם קורה פה?

כמו שאמרנו, ה-Service Worker הוא שאחראי לטפל בהתראות נכנסות, והוא עובד בתוך Sandbox אשר מגביל את הפעולות שהוא רשאי לבצע. הבא נסתכל ב(חלק הרלוונטי של)קוד ה-JavaScript של ה-Service Worker בו השתמשנו לניצול החולשה:

```
self.addEventListener('notificationclick', function(event) {
  event.waitUntil(
    clients.openWindow('https://www.komodosec.com')
  );});

self.addEventListener('notificationclose', function(event) {
  event.waitUntil(
    clients.openWindow('https://www.komodosec.com')
  );});
```

כפי שניתן לראות, הקוד מאזין לשני אירועים שונים (NotificationClose ו-NotificationClick), שכפי שניתן לנחש מתרחשים כאשר המשתמש לוחץ על התראה, או סוגר אותה. עבור כל אחד מהאירועים, הקוד מנסה לפתוח טאב חדש, המוביל אל האתר https://www.komodosec.com. פתיחת טאב בעקבות לחיצה על התראה הינה מצב רצוי, אך ה-Sandbox **אמור** למנוע מה-Service Worker לפתוח טאב חדש כאשר המשתמש מנסה **לסגור** את ההתראה. עם זאת, במסגרת המחקר מצאתי שש-Firefox מאפשר פתיחת טאב גם בזמן סגירת ההתראה, וכך למעשה ניתן להכריח את המשתמש לפתוח טאב (גם אם הוא מעוניין בכך ולוחץ על ההתראה, וגם אם הוא לא מעוניין בכך ומנסה לסגור אותה).

אך רגע, בסרטון ראינו פתיחה של מספר רב של טאבים, ללא כל פעילות מטעם המשתמש? ובכן, זאת תודות להתנהגות חריגה נוספת של Firefox, אשר מזניק בעצמו (ללא התערבות המשתמש) את אירוע NotificationClose עבור התראה קיימת ברגע שמתקבלת התראה חדשה. כלומר, בעת שליחת שתי התראות, אחת אחרי השניה, נוכל להזניק את אירוע הסגירה של ההתראה הראשונה, ולפתוח טאב בדפדפן המשתמש ללא התערבות המשתמש. נחזור על התהליך שוב ושוב ונוכל לפתוח עשרות טאבים (למעשה ניתן לנצל חולשה זו לצורכי השבתת דפדפן המשתמש ע"י פתיחת מספר גדול של טאבים וצריכת משאבים בלתי מוגבלת).

לאחר איתור החולשה, דיווחתי עליה ל-Mozilla במסגרת תכנית ה-Bug Bounty שלהם. Mozilla הכירו בחולשה ותיקנו אותה במסגרת גרסה 59 של Firefox. החולשה קיבלה את הקוד CVE-2018-5141.

ממצא 2: חטיפת מנגנון הרשאות (Notification Hijack)

עתה, לאחר שהבנו שתוקף יכול לנצל הרשאות לשליחת התראות בדחיפה לרעה, נשארנו עם שאלה מרכזית: כיצד יצליח תוקף לקבל הרשאות כאלו מלכתחילה? תשובה אחת, ברורה ומידית, היא באמצעות שימוש באתר זדוני. אבל ישנה דרך נוספת - חטיפת מנגנון הרשאות של אתר לגיטימי ע"י ניצול של Cross Site Scripting (XSS). זהו למעשה וקטור תקיפה חדש למתקפת XSS שאני מכנה Notification Hijack.

כפי שהוסבר קודם לכן, לאחר שהמשתמש העניק לאתר ההרשאות לשליחת הרשאות בדחיפה, נוצר עבור המשתמש אובייקט JSON הנקרא Subscription, המבוסס על המפתח הפומבי של האתר. אובייקט זה משמש לשליחת ההתראות אל המשתמש. כפי שניתן לנחש, אתר יכול לחדש את אובייקט זה עבור המשתמשים שלו (כלומר, להחליף את המפתח הפומבי בו הוא משתמש), באמצעות קוד JavaScript. יכולות להיות סיבות רבות בגינן אתר יבחר לעשות זאת, למשל במקרה של החלפת מפתחות פומביים. עם זאת, במהלך המחקר הופתעתי לגלות שניתן להחליף את אובייקט ה-Subscription ללא "הוכחת בעלות" על האובייקט הקיים (כלומר ללא שימוש במפתח הפרטי המקורי) וללא בקשת הרשאות מחודשת מהמשתמש. במילים אחרות, ברגע שהמשתמש נתן הרשאות לאתר לשלוח לו התראות, קוד JavaScript המגיע מהאתר יכול להחליף את המפתחות המשמשים לשליחת התראות ללא יידוע המשתמש ובלי שום אמצעי הגנה נוסף. מנקודת מבט של תוקף - זו בדיוק רמת הגישה (JS המגיע מהאתר המותקף) שתהיה לנו אל דפדפן המשתמש בעת ניצול XSS.

כך ננצל Reflected XSS לקבלת גישה רציפה (Persistent) אל דפדפן המשתמש

אנו מניחים פה שאתר המותקף משתמש במנגנון התראות, וכי המשתמש כבר העניק לאתר את ההרשאות המתאימות. במקרה זה, קוד ה-JavaScript שלנו יידרש ראשית "לבטל" את אובייקט ה-Subscription הקיים (כלומר האובייקט הלגיטימי המשמש את האתר). ניתן לבצע זאת בקלות, למשל באמצעות הקוד הבא:

```
function unsubscribe(){
    navigator.serviceWorker.ready.then(function(reg) {
      reg.pushManager.getSubscription().then(function(subscription) {
        subscription.unsubscribe().then(function(successful) {
          // You've successfully unsubscribed
        }).catch(function(e) {
          // Unsubscription failed
        }) })});
```

הקוד פשוט ולעניין - פונה אל ה-API בדפדפן, מבקש את אובייקט ה-Subscription הקיים כרגע, ומשחרר אותו (בשלב זה בעל האתר הלגיטימי איבד את היכולת לשלוח התראות אל המשתמש). עתה, כל שנותר לנו לעשות בתור תוקפים הוא לייצר אובייקט חדש, באמצעות המפתח שלנו:

```
function reSubscribe(){

navigator.serviceWorker.getRegistrations().then(function(registrations)
{
        for(let registration of registrations) {
            registration.pushManager.subscribe({
            userVisibleOnly: true,
            applicationServerKey: applicationServerKey
            }).then(function(subscription) {
                console.log(subscription.toJSON());
            });}});

    }
```

ו...זהו. בשלב זה אנחנו יכולים לשלוח אל המשתמש התראות באמצעות שימוש באובייקט החדש שנוצר. למעשה, ע"י ניצול חולשת XSS קיבלנו הרשאה רציפה לדפדפן המשתמש - מעצם טבען של התראות בדחיפה, השליטה שלנו תמשיך גם לאחר שהמשתמש יסגור את הדפדפן (כפי שראינו בסרטון לעיל, למשל).

רגע, אבל מה עם ה-Service Worker?

באמצעות השיטה שהדגמנו זה עתה, קיבלנו יכולת לשלוח למשתמש התראות, אך לא קיבלנו שליטה על מה יקרה עם ההתראות האלו ברגע שיגיעו. כאמור, זה תפקידו של ה-Service Worker. הבעיה שאנו ניצבים בפני היא שכדי לרשום Service Worker חדש, אנו נדרשים לאכסן קובץ JavaScript על האתר המותקף (כפי שהזכרתי למעלה, תחת עקרון ה-Same Origin Policy קובץ ה-Service Worker חייב להגיע מאתר עליו הוא פועל). מכאן, במקרה כללי בו זיהנו חולשת XSS בלבד, לא נוכל להחליף את ה-Service

Worker והשליטה שלנו בדפדפן של הקורבן תהיה מוגבלת - נאלץ להשתמש באותה הלוגיקה שמימש בעל האתר ב-Service Worker המקורי שלו. זה לא בהכרח נורא כל כך - במרבית המקרים, למשל, אתרים יפנו משתמשים אל לינקים אשר ישלחו אליהם באמצעות התראות. במקרה כזה נוכל לנצל את האמון של המשתמש באתר המקורי כדי להפנות אותו (באמצעות התראות דיוג - Phishing Notifications) אל אתרים זדוניים.

אם לעומת זאת זיהנו בעיה נוספת שתאפשר לנו לאכסן קובץ על האתר המותקף, אפילו באופן זמני, נוכל לרשום Service Worker משלנו וההשתלטות על מנגנון ההתראות תהיה מוחלטת.

זה נשמע כמו חולשה... מה אומרים על זה בגוגל?

מנקודת המבט שלי, הבעיה נעוצה בכך שהמשתמש אינו יכול לוודא את זהות האתר על בסיס המפתח הפומבי - המפתח יכול פשוט להשתנות. פניתי אל גוגל בנושא, והצעתי פתרון במסגרתו ניהול המפתחות יתבצע באופן דומה לניהול המפתחות בתקשורת TLS:

- המפתח הפומבי הראשוני יתקבל ישירות מספק השירות (Google / Mozilla), אחרי הוכחת בעלות על האתר הרלוונטי.
- כל מפתח פומבי עתידי עבור האתר יוכל להתקבל רק לאחר הוכחת בעלות על המפתח המקורי.
- לצורכי פיתוח ובדיקות, ניתן יהיה להשתמש במפתח "חתום עצמאית" (Self Signed) שלא ניתן ע"י ספק השירות, אך במקרה זה תוצג למשתמש הערת אזהרה (בדומה לתעודות דיגיטליות שאינן חתומות).

גוגל דחו את הרעיון כיוון שלטענתם הסיכון הפוטנציאלי אינו גדול מספיק בשביל להצדיק את כמות העובדה והניהול שתידרש הן מטעם המפתחים והן מטעת ספק השירות, ביחוד כיוון שהמנגנון גם ככה סבוך. אני לא משוכנע שההסבר שלהם מספק אותי, אך מה שבטוח הוא שלעת עתה זו היא האחריות של בעל אתר המשתמש בהתראות לוודא כי המשתמשים שלו אינם מותקפים (למשל, ניתן לוודא את תקפות אובייקט ה Subscription בכל פעם שהמשתמש ניגש אל האתר).

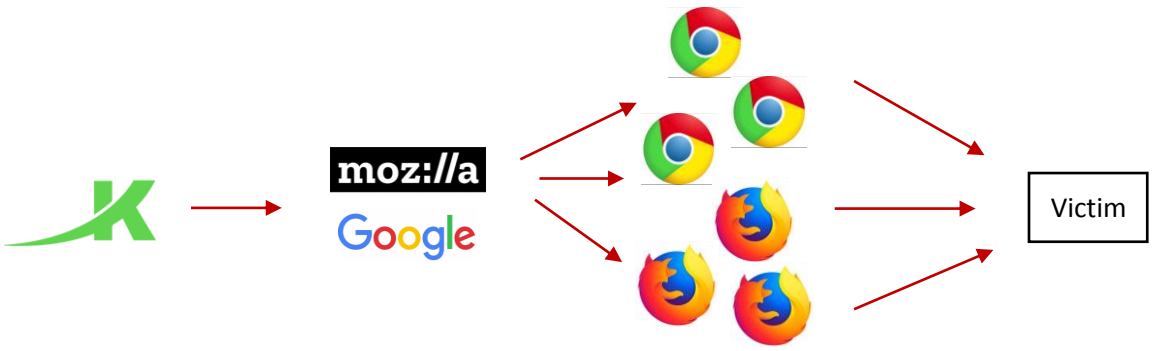
ממצא 3: Botnet מבוסס התראות

כשחושבים לעומק על מנגנון ההתראות בדחיפה כפי שהצגתי אותו לעיל, יצירת Botnet באמצעותו היא היסק לוגי כמעט מידי - המנגנון מאפשר גישה רציפה להרצת קוד JavaScript על דפדפנים רבים. בעוד אין ביכולתנו להריץ פקודות מערכת הפעלה על כל הזומבים ברשת שלנו, אנחנו בהחלט יכולים לשגר מתקפת DDoS.

ניקח כדוגמא את קוד ה-Service Worker הבא:

```
self.addEventListener('push', function(event) {
 event.waitUntil(
    getEndpoint()
    .then(function(endpoint) {
      return fetch("https://www.victim.com", {
    method: 'POST', //
body:JSON.stringify('["very very long","data!!!]')
    }); })
    .then(function(response) {
      return response.text();
    });});
```

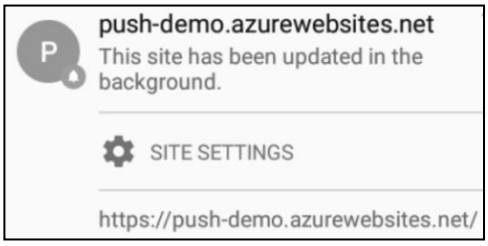
כל מה שאנו עושים פה הוא לצרוך את אירוע ה-"push" (אירוע המוזנק ברגע שמתקבלת התראה בדפדפן), ובתגובה שולחים בקשת HTTP POST אל הקורבן שלנו (כמובן, במקרה אמיתי ה-Body של ה-Request יהיה גדול בהרבה). עתה, כל מה שאנו צריכים זה "להדביק" מספר גדול של קורבנות (כלומר לקבל הרשאות לשלוח אליהם התראות בדחיפה). כאשר נהיה מוכנים להזניק את מתקפת ה-DDoS שלנו, נפנה אל שרת ה-CNC שלנו - דהיינו ספק השירות, גוגל/מוזילה, אשר בתורו יפנה את הבקשות אל הזומבים שלנו, שבתורם יתקפו את הקורבן:



התראות בלתי נראות?

ייתכן ששמת לב שה-Service Worker שהצענו לעיל לא מציג התראות בפועל למשתמשים, כלומר הפעילות אמורה להיות "בלתי נראית". עם זאת, שליחת התראות סמויות לא בהכרח אפשרית:

- ב-Chrome, שליחת התראות סמויות אינה מותרת, אלא במקרים בהם המשתמש עובד מול האתר עצמו בזמן שליחת ההתראה. במידה והמשתמש אינו מחובר בעת קבלת התראה "בלתי נראית", הדפדפן יציג הודעת אזהרה למשתמש אודות הפעילות:



קיימים פתרונות חלקיים (ובראשם מנגנון ה-Budget AP החדש), אך בכולם קיימת הגבלה על מספר מצומצם של התראות סמויות. יש להדגיש, עם זאת, שגם אם מוצגת הודעה אזהרה למשתמש אודות התראה סמויה - שליחת ה-POST תתבצע , ומתקפת ה-DDoS שלנו תצא אל הפועל.

- ב-Firefox, שליחת התראות סמויות דווקא מותרת, עם הגבלה מסוימת על קצב השליחה (כלומר לאחר מספר גדול מספיק של התראות סמויות, Mozilla עשויים לחסום את האתר). התנהגות זו הפתיעה אותי ואפילו דיווחתי על כך למוזילה, אך מסתבר שזו התנהגות רצויה ☺

כשמחברים את כל זה יחד, להערכתי בעת בניית Botnet מבוסס התראות בדחיפה עמודות בפני התוקף שתי אפשרויות - האחת היא להישאר במצב "זהיר" ולהשתמש רק בזומבים מבוססי Firefox או משתמשי Chrome המחוברים בעת המתקפה (וכך רוב הזומבים יהיו עיוורים למתקפה), והשנייה היא לעבור למצב "מתקפה כוללת" בו משתמשים בכל הזומבים, ומסתכנים בחשיפת ה-Botnet.

איך נלחמים ב-Botnet מבוסס התראות?

נקודה מעניינת ששווה להתעכב עליה בנוגע ל-Bonnet שהצגנו היא העמידות של הרשת. בניגוד למודל הקלאסי, לא קיים שרת CNC שניתן פשוט להוריד מהרשת - פקודות נשלחות אל הזומבים ישירות מספק השירות (Google/Mozilla). כלומר, הורדת הרשת בכללותה אפשרית רק ע"י ספק השירות, למשל ע"י דחיית המפתח של התוקף. עם זאת, זיהוי Botnet ע"י ספק השירות אינו עניין פשוט.

לסיכום

גם לאחר ביצוע מחקר אודות מנגנון ההתראות, אני עדיין לא מבין מדוע משתמש כלשהו יבחר לאשר לאתר לשלוח לו sהתראות. זה פשוט מעצבן מדי, חודרני מדי ולא מאובטח מספיק. על כל פנים, מנקודת המבט של תוקף נראה כי יש הרבה כיווני תקיפה מעניינים המנצלים את המנגנון, וסביר שמחקר נוסף יוביל אל תוצאות נוספות. אני, על כל פנים, לא מאשר לאתרים לשלוח לי התראות, וכנראה שאתמיד בסירוב הזה.