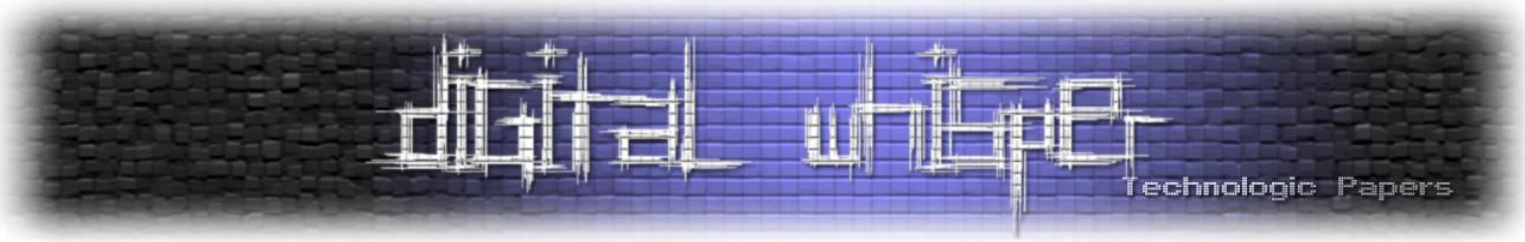

XSHM - Cross-Site History Manipulation

מאמר מאת: אלכס רויכמן, ארכיטקט ראשי ומנהל מעבדת המחקר בצ'קמרקס

אפליקציות ווביות טיפוסיות מורכבות משני רכיבים – רכיב השרת (server-side) ורכיב הלקוח (client-side). היסטורית, רכיב השרת תמיד משך את תשומת לבם של החבר'ה הרעים, אך בשנים אחרונות המגמה הזאת התהפכה ואנו רואים יותר ויותר התקפות על רכיבי הלקוח (כגון XSS, CSRF, JSON Hijacking Clickhacking). ההסבר להיפוך המגמה הוא כנראה ש"יותר קל" להגן על רכיב השרת – יש רק **אחד** כזה ואם כותבים קוד בטוח, מקנפגים את האפליקציה כמו שצריך, משתמשים ב- WAF וכו' אז מקבלים שרת יחסית מוגן. לא כך עם רכיב הלקוח – יש **הרבה** כאלה ולהגן על כמות גדולה קשה יותר מאשר על אחד: תמיד הפורץ יכול למצוא מישהו שמריץ דפדפן בגרסא לא מעודכנת ותמיד יש משתמשים תמימים שלוחצים על לינקים מאתרים מפוקפקים.

אחד המנגנונים הכי חשובים בהגנה על רכיבי הלקוח הוא מנגנון המכונה Same Origin Policy (SOP). במשפט אחד, המנגנון הזה מונע מאפליקציות ממקורות (origins) שונים לשתף ביניהן את התוכן שלהן. כך, למשל, אם משתמש גולש לאתר הבנק שלו ובו זמנית פותח אתר זדוני – האתר הזדוני אינו מסוגל לגנוב מידע מאתר הבנק. לכן SOP מונע מאתרים זדוניים להתייחס לתכנים מאתרים אחרים ובכך מגן על האתרים "טובים" מהתקפות של האתרים ה"רעים", כאשר ההגנה הזאת ממומשת בתוך הדפדפנים של המשתמשים, משמע – ברכיב הלקוח.

כיום, קיימות מספר פרצות ידועות במימוש של מנגנון SOP בדפדפנים הקיימים, פרצות אלה משתמשות באלמנטים הקיימים בצד הלקוח המשותפים לאפליקציות שונות – כמו מנגנון זיכרון המטמון של דפדפנים, או בערוצים נסתרים אחרים כמו timing. במאמר זה נציג פרצה נוספת במנגנון SOP שמנצלת את נקודת התורפה שהתגלתה בניהול אובייקט ההיסטוריה בדפדפים השונים.

ישנם שני סוגים שונים של רכיבי היסטוריה: הסוג הראשון מתייחס לשמירה מקומית של עותקים מדפים שמשתמש גלש אליהם על מנת לזרז גישות עתידיות לדפים האלה – סוג של רכיב cache. זו ההיסטוריה הקבועה (persistent) של הדפדפנים. הפרצה הידועה במנגנון ניהול של היסטוריה הקבועה נקראת CSS history hacking והתגלתה לפני כשלוש שנים על ידי חוקר אבטחה בשם ג'רמי גרוסמן, שמצא כי על ידי הצגת קישורים לאתרים שונים מתוך אתר הפורץ ודגימת צבע של קישורים אלה , ניתן לדעת מה הם האתרים שהמשתמש גלש אליהם בעבר.

הסוג השני של היסטוריה מתייחס לרשימת כל אותם הדפים שמשתמש גלש אליהם מתוך החלון הנוכחי של הדפדפן. לא כמו עבור ההיסטוריה הקבועה, ההיסטוריה הזאת נמחקת כל פעם שחלון הדפדפן נסגר, לכן זוהי היסטוריה זמנית. למעשה, ההיסטוריה הזאת מיוצגת על ידי אובייקט תכנותי שזמין מסקריפטים (JS/VBS) ומנוהל כמערך של הלינקים שהמשתמש פתח מהחלון הנוכחי.

אובייקט ההיסטוריה הינו אובייקט גלובאלי, יצרני דפדפנים הבינו זאת ומנעו גישה לתוכן האיברים שבתוך המערך של היסטוריה, על מנת לא להפר את מנגנון SOP. עם זאת, במחקר שערכנו, התגלה כי מאפיין האורך של מערך ההיסטוריה הוא גלוי וזמין לכל אפליקציה ועל ידי ניצול תכונה זו , ניתן להפר את מנגנון SOP ולפגוע בפרטיות של המשתמשים . אנו חושדים כי התוקפים ידעו וניצלו את הפרצה הזאת ולכן רצינו לחשוף אותה לקהילת המפתחים ומומחי אבטחת מידע.

נמחיש את השימוש באובייקט היסטוריה על ידי דוגמא פשוטה זאת: נניח כי משתמש פותח דפדפן וגולש לחמישה דפים שונים. כעת המשתמש ניגש לדף שישי אשר שנפתח מהאתר הפורץ. הפורץ קודם כל יכול לדגום את אורך אובייקט היסטוריה ולגלות שערכו 6, משמע המשתמש גלש לחמישה דפים/אתרים לפני שהגיע לאתר הפורץ. כפי שנראה מייד , פגיעה בפרטיות המשתמש עלולה להיות הרבה יותר גדולה במקרים מסוימים ואפילו לגרום לדליפת מידע מאוד רגיש . להתקפה שמתבססת על ניצול הגישה לאורך ההיסטוריה קראנו Cross-Site History Manipulation או XSHM. כל אפליקציה שמכילה תבנית הבאה של הקוד, חשופה במידה כזאת או אחרת להתקפה הזאת:

Page A: If (CONDITION)

Redirect(Page B)

הווקטור של ההתקפה הוא די פשוט:

1. Create IFRAME with src=Page B
2. Remember the current value of history.length
3. Change src of IFRAME to Page A
4. If the value of history.length is the same– then the CONDITION is TRUE

ננסה להבין כיצד ההתקפה עובדת: כצעד ראשון, הפורץ אמור לאתר אפליקציה שמכילה תבנית מותנית של redirect. נניח שהוא מצא כזאת אפליקציה שמעבירה מדף א' לדף ב' במידה ותנאי מסוים מתקיים. כעת, הפורץ יכול לפתח דף משלו שיכיל בתוך IFRAME דף ב', מהאפליקציה החשופה אותה הוא מעוניין לתקוף. נשים לב שבשלב הזה האיבר האחרון במערך רכיב ההיסטוריה הוא דף ב'. הפורץ מודד את אורך רכיב ההיסטוריה אחרי שדף ב' נפתח בתוך IFRAME ומייד אחר כך פותח דף א' מתוך אותו IFRAME. במידה ודף א' יעביר לדף ב' ודף ב' כבר למעלה במערך רכיב ההיסטוריה, אז לא יתווסף להיסטוריה עוד איבר משום שההיסטוריה מנוהלת כך שאין שני איברים זהים ברצף. אם כך, אורך ההיסטוריה יישאר כמו שהוא היה לפני פתיחת דף א', מה שמעיד על כך שהתנאי של redirect מתקיים.

במידה ודף א' לא יעביר לדף ב', אז אורך רכיב ההיסטוריה יגדל באחד- וזה מה יעיד על כך שהתנאי לא מתקיים. אם כן, התוקף יכול, מתוך הדף שלו, לקבל אינדיקציה על ערך של התנאי באפליקציה שרצה

ממקום אחר- זליגה של מידע בין דומיינים שונים. כאן נציג שתי דוגמאות כיצד זליגה כזאת יכולה להביא לפגיעה בפרטיות וסודיות של משתמשים.

אחד הדברים שחשובים לפורץ הוא לדעת האם המשתמשים אותם הוא רוצה לתקוף מחוברים ברגע נתון לאפליקציה מסוימת ועברו כבר את התהליך של זיהוי משתמש. למשל, אם משתמש הזדהה לאפליקציה בנקאית, וידוע שהאפליקציה הזו חשופה ל-CSRF, אז התוקף יכול לשלוח payload של CSRF ולהיות בטוח שההתקפה שלו תצליח. השיטות הקיימות לגילוי סטאטוס של זיהוי המשתמש בתוך האפליקציה מתבססות על פתיחה של תמונה מ-"האזור המוגן של האפליקציה" ודגימה של אירוע onerror. שיטה זו לא תמיד עובדת כי לא תמיד קיימות תמונות באזור המוגן שנפתחות אך ורק כאשר המשתמש מזוהה.

אנו מציעים שיטה אחרת של גילוי סטאטוס של זיהוי משתמש בעזרת XSHM. האפליקציה תהיה חשופה להתקפה כזאת במידה והיא מממשת תבנית הבאה בקוד:

```
If (!isAuthenticated())
      Response.Redirect("Login.aspx")
```

או בקונפיגורציה:

```
      <authentication mode="Forms">
            <forms loginUrl="Login.aspx”/>
</authentication>http://he.wikipedia.org/wiki/Advanced_Encryption_Stand
ard
```

כעת התוקף יכול לבצע את הצעדים הבאים:

1. Create IFRAME with src='Login.aspx'
2. Remember the current value of history.length
3. Change src of IFRAME to 'Protected.aspx'
4. If the value of history.length remains the same– then a user is not authenticated

אם כן, בהנחה כי האפליקציה מגדירה אזור מוגן וגישות לאזור הזה ללא זיהוי משתמש מעבירות לדף של זיהוי משתמש, התוקף יכול בקלות לדעת מהו סטאטוס הזהוי של המשתמש בתוך האפליקציה. משמעות הדבר שמשתמש יכול לגלוש לאתר כלשהו שמציג לדוגמא מכונית יפה, אך מאחורי הקלעים האתר יכול לקבל אינדיקציה אמינה מה הן האפליקציות שאתם כרגע מזוהים בהן. דרך אגב, אפליקציות רבות וביניהן Twitter ו-Facebook חשופות לסוג התקפה זאת. אתם יכולים לראות הדגמה עבור Facebook בלינק הזה.

דבר נוסף שאפשרי לעשות בעזרת מתקפת ה-XSHM, הוא לקבל מידע רגיש מתוך אפליקציה חשופה. תתארו מערכת לניהול משאבי אנוש שמאפשרת לחפש עובדים לפי פרמטרים שונים כמו שם העובד, גיל ומשכורות. כאשר מבצעים חיפוש, נשלח הלינק הבא:

http://Intranet/SearchEmployee.aspx?name=Jon&SalaryFrom=3000&SalaryTo=3500

במידה ואין אף עובד שעונה לקריטריונים של החיפוש, אז האפליקציה מעבירה לדף של Not found (או לחיפוש מתקדם). בנסיבות אלה תוקף יכול לבצע את הצעדים הבאים:

1. Create IFRAME with src='NotFound.aspx'
2. Remember the current value of history.length
3. Change src of IFRAME to 'SearchEmployee.aspx?name=Jon&SalaryFrom=3000&SalaryTo=3500'
4. If the value of history.length remains the same– then your search has no results

כך שלמעשה, התוקף יכול לדעת כמה מרוויח כל עובד כאשר אין לתוקף גישה ישירה לאפליקציה והוא נעזר אך ורק בטכניקה של XSHM. ישנם עוד תסריטים רבים של XSHM שמאפשרים לדלות מידע רגיש מאפליקציה חשופה לתוך אפליקציה של תוקף, על כך אפשר לקרוא במאמר הזה.

יצרני הדפדפנים, כעקרון, יכולים למנוע את ההתקפה של XSHM על ידי הכנסת מגבלות גישה לאובייקט ההיסטוריה. למשל, אם משתמש גלש לדפים הבאים:

- http://Site1.com
- http://Site2.com
- http://Site3.com/Home.aspx
- http://Site3.com/Report.aspx

וכעת הדף Report.aspx מתוך Site3 ניגש לאובייקט ההיסטוריה, אז אורך רכיש ההיסטוריה המוחזר אמור להיות 2 עבור Site3 ולא 5, כך אפשר לעשות הפרדה בין הדומיינים השונים גם בתוך האובייקט של ההיסטוריה ולא להפר מנגנון SOP. עם זאת, יצרני דפדפנים אינם ממהרים לבצע את השינוי הזה כי הוא יפגע בפונקציונאליות של אפליקציות קיימות שמשתמשות באובייקט של היסטוריה. לכן, הדרך למנוע את ההתקפה בשלב זה היא דרך אפליקטיבית – קודם כל המפתחים חייבים להיות מודעים להתקפה.

מאוד חשוב לדעת לזהות את תבניות הקוד שמובילות להתקפה הזאת. הכלי האוטומטי היחיד לניתוח קוד שקיים בשוק ומסוגל לזהות תבניות שחשופות ל- XSHM הוא הכלי של חברה ישראלית בשם "צ'קמרקס" וניתן להוריד גרסת דמו שסורקת קוד ומתריעה על XSHM.

על מנת להתגונן כנגד התקפת XSHM יש לשלב פרמטר עם ערך שרירותי בתוך URL. למשל, על מנת למנוע גילוי סטאטוס של זיהוי משתמש, אפשר להיעזר בקוד הזה:

```
If ( !isAuthenticated)
      Redirect('Login.aspx?r=' + Random())
```

חשוב להבין שלא כמו עבור Anti-CSRF, אין לבדוק את הפרמטר השרירותי באפליקציה – המטרה של הפרמטר היא להוסיף אנטרופיה ל-URL ובכך למנוע מהפורצים לבצע את XSHM. החידוש של התקפת XSHM מול הפרצות האחרות הידועות במנגנון ה-SOP ובפרט, מול פרצת CSS history hacking הוא בכך שהתקפת XSHM משתמשת רכיב ההיסטוריה הזמנית ולא ברכיש ההיסטוריה הקבועה. ההתקפה של CSS לא מסוגלת לגלות סטאטוס של זיהוי משתמש ולא לדלות מידע על משכורות העובדים משתי סיבות עיקריות:

1. היסטוריה הקבועה נשארת בין ה- sessions השונים, ולכן הלינק שמתוך האזור המוגן יישאר visited להרבה זמן גם אם המשתמש מזמן כבר לא בתוך האפליקציה, אבל ב-XSHM התוקף כן מקבל אינדיקציה לגבי ה- session הנוכחי וזה מאוד חשוב להתקפות על רכיב הלקוח כמו CSRF ו-Clickjacking.

2. אם פותחים קישור מתוך IFRAME, אז הצבע שלו לא ישתנה ל- visited, לכן אי אפשר לדלות מידע על משכורות עובדים בעזרת CSS, אלא רק בעזרת XSHM.

שורה תחתונה: התקפה של XSHM היא עוד התקפה על מנגנון SOP ויכולה להביא לזליגה של מידע מאתר חשוף לאתר של פורץ. חשוב להכיר את ההתקפה הזאת וגם לדעת להתגונן כנגדה.

על המחבר:

אלכס רויכמן הוא ה ארכיטקט הראשי ומנהל מעבדת המחקר של צ'קמרקס (www.Checkmarx.com). החברה מתעסקת בפיתוח כלים אוטומטיים לניתוח סטאטי של קוד המאתרים בעיות באבטחת המידע. החברה נוסדה ב- 2006 ואלכס היה בין הראשוניים שעבד על מחקר ופיתוח של אלגוריתמים שמזהים בעיות אבטחה בקוד. לאלכס מעל 10 שנות נסיון בבנית מערכות מתוחכמות ומורכבות. הוא גילה פרצות אבטחה רבות ובינהן, למשל, פרצת האבטחה בשם ReDoS שתפסה תשומת לב רבה בשנה אחרונה. אלכס פרסם מאמרים במגזינים מובילים כמו ACM ו-Springlink. אלכס רויכמן סיים בהצטיינות את לימודי התואר השני במדעי המחשב עם התמחות באבטחת אפליקציות ובסיס נתונים, נושא התזה שלו הוא "מניעה וגילוי חדירות לבסיסי נתונים ווביים". אלכס נגיש לשאלותיכם ב: Alexr@Checkmarx.com