

## דוח סיכום פרויקט: הרעלת מטמון (Web Cache Poisoning)

מגישים: יובל בץ ואביב עמידור

### תיאור החולשה והרקע התאורטי:

Web cache poisoning היא טכניקת תקיפה מתקדמת שבה התוקף מתמרן את השרת כדי לגרום לו לייצר תגובה זדונית (לרוב דף HTML המכיל קוד JavaScript פוגעני), ולשמור את התגובה הזו בתוך Cache של המערכת. התוצאה היא שמנגנון ה-Cache, שנועד לשפר ביצועים, הופך למפיץ של המתקפה: כל משתמש לגיטימי שיבקש את אותו הדף, יקבל מה-Cache את התגובה "המורעלת" מבלי שהתוקף יצטרך לבוא איתו באינטראקציה ישירה. במימוש זה התמקדנו בשיטה המורכבת של **Base URL Override**.

**כיצד היא נוצרת:** החולשה נוצרת כתוצאה מפער טכני בין האופן שבו ה-Cache מזהה בקשות לבין האופן שבו השרת מייצר את התגובות. מנגנוני Cache מסתמכים על "מפתח מטמון" (Cache Key) - לרוב צירוף של נתיב ה-URL ומתודת הבקשה (למשל GET /). במקביל, בקשות HTTP מכילות "קלטים שאינם חלק מהמפתח" (Unkeyed Inputs), כגון כותרות (Headers) כמו X-Forwarded-Host. החולשה מתרחשת כאשר מתקיימים שני שלבים: הראשון שלב ההרעלה poisoning גרימת השרת לייצר תגובה המכילה Payload במקרה זה, שינוי כתובת הבסיס של האתר. השני וידוא שהתגובה נשמרה ב-Cache ומוגשת למשתמשים אחרים המבקשים את אותו ה-URL. כך, השרת מחזיר דף המכיל את התוכן של התוקף, וה-Cache מאחסן אותו עבור כלל המשתמשים.

**מה תוקף יכול להשיג באמצעות:** באמצעות Base URL Override תוקף יכול לבצע "חטיפת משאבים" ע"י שינוי הכתובת ממנה נטענים קבצי ה-JavaScript של האתר. התוקף יכול לגרום לדפדפן של כל קורבן להוריד קוד זדוני משרת חיצוני, מה שמאפשר גניבת פרטי אשראי, Session Tokens והשתלטות מלאה על חשבונות המשתמשים בחנות.

ברגע שהדף הראשי של האתר מורעל, התוקף יכול:

- לגנוב עוגיות (Cookies) ו-Session Tokens של כל משתמש שמבקר באתר.
- לבצע פעולות בשם המשתמשים מבלי ידיעתם (Session Hijacking).
- לבצע השחתה (Defacement) של דף הבית.
- **טכנולוגיות:**
  - בחרנו במסלול B הוספת אתגר לפרויקט קוד פתוח קיים שממנו גם למדנו בסמסטר juice-shop.
  - שפות תכנות:** TypeScript עבור הלוגיקה של השרת ורישום האתגר. **סביבת צד-שרת (Backend)** Node.js המריץ שרת מסוג Express.js.
  - יצרנו Middleware ייעודי שמדמה מנגנון Caching פגיע בזיכרון השרת.
  - סביבת צד-לקוח (Frontend):** Angular (שפת הfrontend שבה כתבו את Juice Shop, שלתוכה הוזרק הסקריפט). בנוסף הם השתמשו גם במסד הנתונים של sqlite.

כלים המשמשים לביצוע הדגמה :

- **Node Package Manager (npm)** לקימפול של קבצי ה Typescript ל javascript והרצת השרת המקומי ע"י הפקודות `npm run build:server` ו- `npm start`.
- **cURL (Command Line Tool)** : כלי שורת פקודה ששימש לשליחת בקשות HTTP מותאמות אישית. באמצעות כלי זה התאפשרה שליחת הבקשה עם ה-Header הזדוני (X-Forwarded-Host: attacker.com) כדי לעקוף את הבדיקות הרגילות של הדפדפן ולהרעיל את השרת.
- **דפדפן אינטרנט (Web Browser)**: לאימות סופי של הצלחת המתקפה וקבלת האישור החזותי בלוח הניקוד (Score Board) של המערכת.

### תיאור ה-POC

- ה-POC ממחיש כיצד תוקף יכול לשבש את תשתית האתר (Infrastructure) של אפליקציית OWASP Juice Shop על ידי שינוי ה-Base URL שלה. במקום הזרקת סקריפט בודד, התוקף משתמש ב-Web Cache Poisoning כדי "להשתלט" על המקור ממנו הדפדפן טוען את כל משאבי המערכת (JS, CSS ותמונות). שלבי הניצול:
- שלב א': יצירת התגובה המורעלת (Poisoning)
  - זיהוי החולשה: התוקף מזהה שהשרת (בקובץ `routes/angular.ts`) משתמש בכותרת `X-Forwarded-Host` כדי להזריק באופן דינמי תגית `<base>` לתוך ה-HTML.
  - הזרקת ה-Payload: התוקף שולח בקשת HTTP הכוללת את הכותרת הזדונית `X-Forwarded-Host: attacker.com`.
  - עיבוד בשרת: השרת מקבל את הבקשה, ומייצר דף HTML שבו מופיעה השורה: `base > </href="//attacker.com`.
  - שלב ב': אחסון ב-Cache והגשה לקורבן (Serving)
  - כניסה למטמון: ה-Middleware של ה-Cache (בקובץ `lib/cacheMiddleware.ts`) שומר את התגובה המורעלת בזיכרון תחת מפתח המטמון `"/` (דף הבית).
  - בקשת הקורבן: קורבן תמים ניגש לאתר (`localhost:3000`) בבקשה רגילה.
  - מסירת הרעל: שרת ה-Cache מזהה שיש לו תגובה מוכנה בזיכרון ומגיש לקורבן את ה-HTML המורעל.
  - ביצוע המתקפה בדפדפן: הדפדפן של הקורבן קורא את תגית ה-`base` המורעלת ומנסה לטעון את כל קבצי ה-JavaScript (כמו `main.js`) מהדומיין `attacker.com` במקום מהשרת המקורי.
  - פקודת התוקף להרעלת המטמון:

```
curl.exe -v -H "X-Forwarded-Host: attacker.com" http://localhost:3000
```

- . בקשת הקורבן (בדפדפן או ב-cURL ללא Headers):

```
/ curl.exe http://localhost:3000
```

תוצאה בפועל של הניצול

כתוצאה מהמתקפה, השרת מחזיר מה-Cache קוד HTML הוכחת הצלחה:

חזותית: האתר מוצג כדף אפור/ריק מאחר והמשאבים הקריטיים (JS) לא נטענו מהמקור הנכון. טכנית (Console): בדפדפן מופיעות שגיאות מסוג "Access to script at 'http://attacker.com/...' has been blocked by CORS policy". זה מוכיח שהדפדפן אכן ניסה לטעון את הקוד משרת התוקף. מערכתית: בלוח הניקוד (Score Board) של Juice Shop, האתגר "Cache Poisoning" מסומן ב-V-ירוק (Solved).

## כיצד ניתן למנוע את החולשה:

הדרך היעילה ביותר למנוע את החולשה היא לוודא שכל קלט שמשפיע על תוכן התגובה יהיה חלק בלתי נפרד ממפתח המטמון כך שהמערכת לא תתבלבל בין גרסאות שונות של אותו הדף. ניתן להשתמש בכותרת Vary המורה למטמון לשמור עותקים נפרדים עבור כל ערך שונה של כותרת הבקשה ובמקביל מומלץ להימנע לחלוטין משימוש בכותרות חיצוניות לצורך בניית כתובות URL בתוך הקוד ולהסתמך רק על הגדרות שרת קבועות ומאובטחות. כהגנה נוספת כדאי להגדיר מדיניות אבטחה מסוג CSP שתמנע מהדפדפן לטעון משאבים ממקורות לא מורשים גם במקרה שבו המטמון עבר הרעלה מוצלחת.

## קבצים חדשים שיצרנו

lib/cacheMiddleware.ts

מימוש שכבת ה Cache הפגיעה

קובץ זה מכיל את הלוגיקה ששומרת דפי HTML בזיכרון השרת לפי נתיב ה URL בלבד תוך התעלמות מכותרות Headers. הקובץ בודק אם התגובה המוגשת מה Cache מכילה כתובת בסיס מורעלת ואם כן קורא לפונקציית הפתרון של האתגר.

## קבצים קיימים ששינינו

data/static/challenges.yml

הגדרת האתגר בלוח המשימות Score Board הוספת בלוק נתונים הכולל את שם האתגר Cache Poisoning הקטגוריה Vulnerable Infrastructure רמת הקושי 4 והרמזים למשתמש.

models/challenge.ts

רישום מפתח האתגר במודל הנתונים הוספת המפתח cachePoisoningChallenge לרשימת האתגרים המוכרים על ידי השרת לצורך מעקב אחר פתרונות.

routes/angular.ts

שינוי פונקציית serveAngularClient כך שתזהה את הכותרת X Forwarded Host ותזריק באופן אקטיבי תגית base מורעלת לתוך ה HTML של דף הבית.

server.ts

הגדרת מחזור החיים של בקשות בשרת Middleware Pipeline רישום ה cacheMiddleware ומיקומו בתחילת השרשרת לפני הגשת קבצים סטטיים כדי להבטיח שהרעלת המטמון תתבצע ותוגש כראוי.

## מקורות:

1. [PortSwigger Web Security Academy](#): [Web cache poisoning vulnerability tutorial](#)

2. [OWASP](#): [Cache Poisoning](#)

3. [MDN Web Docs \(Express.js\)](#): שמכאן בעצם נעזרנו בכתיבת הקוד תיעוד רשמי הנוגע לכתיבת Middlewares [Express Middleware](#).

