

Http Request Smuggling

שם: עמרו תרתיר

בית הספר: בית הספר להנדסת תוכנה _ מכללת עזריאלי

תאריך: 9/2/2026

1. תיאור החולשה והרקע התאורטי

מהי החולשה

HTTP Request Smuggling היא חולשת אבטחה קריטית במערכות ווב, המתרחשת כאשר רכיב Front-End כגון (Proxy / Load Balancer) ורכיב Back-End (שרת אפליקציה) מפרשים באופן שונה את גבולות בקשת ה-HTTP. כתוצאה מכך, תוקף יכול "להבריח" בקשה נוספת בתוך חיבור TCP קיים מבלי שה Front-End יזהה אותה כבקשה נפרדת.

כיצד היא נוצרת

החולשה נוצרת עקב חוסר סנכרון בפרשנות של בקשות HTTP בין רכיבי המערכת השונים. במקרים רבים, רכיב אחד כגון (Reverse Proxy) מסתמך על הכותרת Content-Length לצורך קביעת אורך גוף הבקשה, בעוד שרכיב אחר (כגון שרת האפליקציה) מסתמך על הכותרת Transfer-Encoding: chunked. חוסר התאמה זה גורם לכך שבקשה אחת, כפי שנשלחה מהלקוח, מתפרשת בפועל כשתי בקשות נפרדות בצד השרת.

באיזה הקשר היא מופיעה במערכות ווב

החולשה מופיעה בעיקר בארכיטקטורות ווב מודרניות המבוססות על מספר שכבות, כגון Reverse Proxy או Load Balancer המתווכים בין הלקוח לשרת האפליקציה. שימוש בשרתי אפליקציה נפוצים (כגון Node.js, Gunicorn, או Tomcat) יחד עם חיבורים מתמשכים בפרוטוקול HTTP/1.1 (Keep-Alive) מגביר את הסיכון להיווצרות חוסר סנכרון בין רכיבי המערכת.

מה תוקף יכול להשיג באמצעותה

באמצעות ניצול חולשת HTTP Request Smuggling, תוקף עשוי לעקוף מגננוני אימות והרשאה, לבצע גניבת Sessions של משתמשים אחרים, להחדיר בקשות זדוניות למטמון (Cache Poisoning) לשלוח בקשות בשם משתמשים לגיטימיים, לקבל גישה לנתיבים פנימיים שאינם חשופים ישירות, ואף לגרום לפגיעה בזמינות השירות באמצעות DoS לוגי.

2. סביבת העבודה

טכנולוגיות

- Programming language: (Back-End) python
- Framework: Flask
- Application server: Gunicorn

- **Reverse Proxy:** Nginx 1.23
- **Containerization:** Docker + Docker Compose
- **Protocol:** HTTP/1.1 + Keep-Alive

כלים המשמשים לביצוע ההדגמה

- TCP (ncat from Nmap) לשליחת בקשות
- curl.exe לבדיקות HTTP בסיסיות
- Docker CLI
- PowerShell (Windows)

3. מימוש והדגמת המתקפה (POC)

תיאור ה-POC

במעבדה הודגמה מתקפת HTTP Request Smuggling מסוג **TE.CL** שבה:

- ה- Frontend (Nginx) מפרש את הבקשה לפי Transfer-Encoding: chunked
 - ה- Backend (Gunicorn + Flask) מפרש את הבקשה לפי Content-Length
- התוקף שולח בקשת Login תקינה, שבתוכה "מוחבאת" בקשה נוספת לנתיב /profile.

שלבי הניצול

1. פתיחת חיבור TCP ישיר ל-Nginx
2. שליחת בקשת POST ל- /login עם Transfer-Encoding: chunked
3. סיום גוף הבקשה (0\r\n\r\n)
4. הזרקת בקשת HTTP נוספת (GET /profile) באותו חיבור
5. ה- Backend מפרש את הבקשה השנייה כבאה ממשתמש אחר

בקשה לדוגמה (Payload)

```
POST /login HTTP/1.1
Host: localhost
Transfer-Encoding: chunked
Content-Type: application/x-www-form-urlencoded

e
username=alice
0

GET /profile HTTP/1.1
Host: localhost
```

תוצאה בפועל של הניצול

- בקשת login/מוחזרת עם 200 OK
- השרת יוצר Session למשתמש alice
- הבקשה השנייה (/profile) מגיעה ללא אימות תקף
- מתקבלת תגובת 401 Unauthorized, המוכיחה שהבקשה נותחה כבקשה עצמאית

דיאגרמה – שלבי המתקפה (תיאור מילולי)

1. Client → Nginx: בקשה אחת
2. Nginx → Backend: מפצל לבקשה אחת
3. Backend: מזהה שתי בקשות נפרדות
4. בקשה שנייה מבוצעת מחוץ להקשר האימות

4. מנגנוני הגנה ומניעה

מדוע ההגנות הקיימות נכשלו

- Nginx אפשר העברת כותרות סותרות
- שימוש ב-HTTP/1.1 Keep-Alive
- חוסר אחידות בין פרשנות ה-Frontend וה-Backend
- כיבוי proxy_request_buffering

כיצד ניתן למנוע את החולשה

- חסימת בקשות המכילות גם Content-Length וגם Transfer-Encoding
- נרמול בקשות בפרונט-אנד
- ביטול Keep-Alive כאשר אין צורך
- שימוש ב-HTTP/2
- עדכון שרתים ו-Reverse Proxies

5. מקורות

מאמרים ותיעוד

- [PortSwigger – HTTP Request Smuggling](#)
- [OWASP Web Security Testing Guide](#)