

# Http Request Smuggling

שם: עמרו תרטיר

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 (Reverse Proxy / Load Balancer) מפרשים באופן שונה בקשות HTTP. כתוצאה לכך, תוקף יכול "להבריח" בקשה נוספת בתוך חיבור TCP קיים מבלי שהיא Front-End יזהה אותה כבקשה נפרדת.

### כיצד היא נוצרת

החולשה נוצרת עקב חוסר סyncronization בפרשנות של בקשות HTTP בין רכיב המערכת השונים. במקרים רבים, רכיב אחד כגון (Reverse Proxy) מסתמן על הכותרת Content-Length לצורך קביעת אורך גוף הבקשה, בעוד שרכיב אחר (כגון שרת האפליקציה) מסתמן על הכותרת Transfer-Encoding: chunked. חוסר התאמה זה גורם לכך שבבקשה אחת, כפי שנשלחה מהלך, מתפרקת בפועל לשתי 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 הלוקו' לשרת האפליקציה. שימוש בשרת אפליקציה נפוץ (כגון Tomcat, Node.js או Gunicorn), יחד עם חיבורים משתמשים ב프וטוקול HTTP/1.1 (Keep-Alive). מגביר את הסיכון להיווצרות חוסר סינכרון בין רכיבי המערכת.

### מה תוקף יכול להשיג באמצעותה

באמצעות ניצול חולשת HTTP Request Smuggling, תוקף עשוי לעקוף מנגןוני אימות והרשאה, לבצע גיבוב Sessions של משתמשים אחרים, להחדיר בקשות זדוניות למטען (Cache Poisoning) לשלו' בקשות בשם משתמשים לגיטימיים, לקבל גישה לנティיבים פנימיים שאינם חשופים לשירות, ואף לגרום לפגיעה בזיכרון השירות באמצעות SDo log.

## 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 הදגמה

- ncat from Nmap (ncat from Nmap) לשילוח בקשות TCP •
- curl.exe לבדיקה HTTP בסיסיות •
- Docker CLI •
- PowerShell (Windows) •

### 3. מימוש והדגמת המתקפה(POC)

#### תיאור ה-POC

במקרה הוגדרה מתקפה 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 שלוח בקשה Login תקינה, שבתוכה "מוחבאת" בקשה נוספת לנטייב /profile.

#### שלבי הניצול

- 1.פתיחה חיבור TCP ישיר ל-Nginx
- 2.שליחת בקשה POST ל-/login עם Transfer-Encoding: chunked
- 3.סיום גוף הבקשה(0x\0\0\0)
- 4.זרקת בקשה HTTP נוספת (GET /profile) באותו חיבור
- 5.ה- Backend מפרש את הבקשה השנייה כבאה משתמש 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 הnicol

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

## דיאגרמה – שלבי המתקפה (תיאור מילולי)

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

---

## 4. מנגנוני הגנה ומונעה

### מדוע ההגנות הקיימות נכשלן

- NGXINX מאפשר העברת כותרות סותרות
- שימוש ב-`Keep-Alive` HTTP/1.1
- 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`
- גרמול בקשות בפרונט-אנד
- ביטול `Keep-Alive` כאשר אין צורך
- שימוש ב-`HTTP/2`
- עדכון שרתים ו-`Reverse Proxies`

---

## 5. מקורות

### מאמרם ותיעוד

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