digital whisper Technologic Papers

Path Traversal

דוד סטרינסקי

הקדמה

דמיינו לכם את הסיטואציה הבאה: בנינו אתר מרשים ויפה לחברה שלנו או ללקוח גדול, האתר מכיל הרבה אפשרויות וחלקן כוללות גם ניהול של קבצים, חשבוניות מסמכים אישיים תמונות ועוד סוגי קבצים למיניהם.
תוקף מצליח "לשכנע" את השרת שלנו לשלוח לו קבצים שאנחנו לא רוצים שיגיעו לידיים זרות:
קבצי מערכת, סיסמאות, מידע על משתמשים אחרים... איך הוא עושה את זה?

הכירו את Path/Directory Traversal, במאמר הבא ארחיב על איך הפריצה עובדת, איפה זה פוגש אותנו בפועל ,נציג דוגמאות מתוך מעבדות שבוצעו בנושא וננתח ביחד חולשה שנמצאה בשירות מוכר.

אז מה זה בעצם Path/Directory Traversal?

החולשה מתארת מצב בו תוקף יכול לגשת לקבצי השרת, לעבור בין הנתיבים בהן מאוחסנים הקבצים ובכך לשים ידיו על קבצים אשר לא אמורה להיות לו גישה אליהם ולא אמורים להיות זמינים לו. החולשה מנוצלת כאשר יישום אינטרנט (למשל אתר, ממשק API) מקבל בקשה מהמשתמש שם של קובץ או נתיב אליו, אך לא מבצע את הבדיקות המתאימות לוודא כי אכן מדובר במשתמש עם ההרשאות המתאימות או שהקובץ מוגדר כקובץ "בטוח" להחזירו למשתמש.

באמצעות הוספת רצפים מהסוג /.. כמו במערכות הפעלה לינוקס, מתאפשר לתקוף "לטייל" בעץ התיקיות במערכת הקבצים בשרת ולהגיע לקבצים הרגישים למשל: קבצי קונפיגורציה, קבצי סיסמאות או מידע רגיש אחר שאותו אנחנו לא רוצים לחשוף.

כמה באמת החולשה נפוצה?

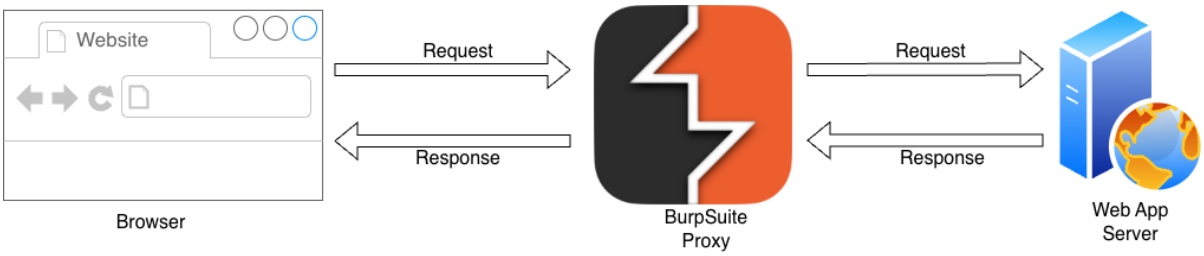
בשנת 2024 פרצות מסוג Path Traversal ממשיכות להוות איום ממשי באבטחת מידע, גם בעולם המודרני של אפליקציות ווב ו־SPA. מחקר של חברת Aikido זיהה ש־2.75% מסך כלל הפגיעויות בקוד פתוח נובעות מ־Path Traversal, כאשר בפרויקטים סגורים (קוד קנייני) השיעור הגיע ל־3.5% - עלייה של כ־85% ביחס ל־שנת 2023. בנוסף, דו"ח Edgescan מצא כי **10.5%** מהפרצות הקריטיות בשנת 2024 היו מסוג זה ו-2.67%

מהפרצות החמורות. סוכנות CISA אף ציינה 55 מקרים פעילים של Directory Traversal כפרצות בסיכון גבוה במערכות תוכנה שונות במהלך אותה שנה.

נכון ל-2024, הנתונים ממחקרים מובילים מצביעים על כך שהבעיה ממשיכה למרות שהטכנולוגיות משתנות - מה שמראה שזו פגיעות בסיסית, אך עדיין לא מטופלת במקרים רבים.

הסבר קצר על הכלי העיקרי בו השתמשתי במהלך העבודה

במהלך המעבדות השתמשתי בעיקר בכלי Burp Suite, זהו כלי הנמצא בשימוש נפוץ בתעשייה המשמש לבדיקות אבטחה על אפליקציות ושירותי ווב. הכלי "יושב" בין הדפדפן לבין המטרה הנתקפת שלנו ונותן לנו את האפשרות לתפוס את הבקשות הנשלחות לפני שהן עוברות לעיבוד ובכך ניתן לבצע פעולות נוספות שלא קורות באופן טבעי כמו שמי שפיתח תכנן שיקרו ממשתמש רגיל.



למשל ניתן לתפוס את הבקשה ובאמצעות הפיצ'ר Repeater המובנה בכלי לערוך את הבקשה, לשנות חלק מהמידע או להוסיף כותרות לבקשה שנשלחה ולראות את התשובה המתקבלת לאחר השליחה.

פיצ'ר העיקרי בו ניתן להשתמש לבדיקות של Path Traversal נקרא Intruder, הפיצ'ר מאפשר לנו ליצור אוטומציה של שליחת בקשות http מרובות כאשר אנחנו יכולים לבחור חלק מסוים מהמידע ואפילו מספר חלקים ולהחליפם בכל בקשה במידע אחר מתוך רשימת מילים שמגדירים מראש. כך למשל ניתן לנצל בקשה תקינה אחת שנשלחת, ליצור סדרה של בקשות שונות ונוכל לבחון את התשובות שקיבלנו.

הסבר קצר על איך להגדיר את Burp Suite, אקדים ואומר כי בתוך הכלי קיים דפדפן Chromium מובנה שניתן להשתמש בו כמו כל דפדפן אחר, אך אם בחרתם להשתמש בדפדפן שלכם הנה מה שצריך לעשות:

1. ראשית נגדיר מאזין לפרוקסי בתוך Burp Suite, ניגש להגדרות באופן הבא:

Tools -> Proxy -> Options

שם נבדוק אם קיים כבר מאזין על הכתובת 127.00.0.1:8080, במידה וקיים אז הכל תקין, אם לא נלחץ על כפתור Restore to default שמופיע לאחר לחיצה על גלגל השיניים בחלון. אם המאזין עדיין לא רץ ניתן לערוך את המאזין הקיים על ידי סימונו ולחיצה על כפתור Edit, מה שמאפשר לנו לערוך אותו ולשנות את הפורט אם 8080 כבר בשימוש באפליקציה אחרת.

2. כעת נעבור להגדרות הדפדפן,למשתמשי Chrome נפתח את חלון ההגדרות בדפדפן וזין בחיפוש את המילה Proxy. לאחר מכן נלחץ על Open your computer proxy settings מה שיעביר אותנו להגדרות הפרוקסי של המחשב, נוודא כי אנחנו מגדירים את הכתובת 127.0.0.1 ופורט 8080 (אפשר לבחור פורט אחר כפי שהוגדר במאזין מסעיף קודם במידה ולא פנוי) גם עבור HTTP וגם עבור HTTPS, חשוב בשלב זה לוודא כי שאר השדות לא מכילים מידע נוסף כמו דומיינים למעקף של הפרוקסי.

למשתמשי Firefox, ניגש להגדרות הדפדפן, שם נחפש את המילה proxy, לאחר מכן תחת החלק של Proxy settings נלחץ על לחצן ה-Settings וניגש להגדרות הפרוקסי. נבחר את האופציה של Manual proxy configuration ונזין את הכתובת 127.0.01 ופורט 8080, נוודא שמסומנת הבחירה של Use this proxy for all protocols, וגם שאין מידע נוסף בשדות אחרים כמו דומיינים שיכולים לעקוף את הפרוקסי שהגדרנו.

משתמשי Safari, נכנסים ל-Preferences ושם בוחרים Advanced. לאחר מכן בחלק של Proxies לוחצים על Change settings. נזין את הכתובת 127.0.01 ופורט 8080 כפי שהוגדר במאזין, נוודא ששאר השדות ריקים והסימון על HTTP ו-HTTPS מסומן ונלחץ OK.

3. כעת נבדוק שהמיירט של הפרוקסי פועל, בתוך BurpSuite ניגש ללשונית Proxy, כעת כדי להדליק את המיירט נזיז את הבורר Intercept is off ונראה שהוא על מצב On. נעבור אל הדפדפן שלנו ונגלוש לאתר כלשהו, הדפדפן אמור "להתקע" במצב ממתין. כעת נחזור ל-BurpSuite ובלשונית Proxy תחת Intercept נוכל לראות את הבקשה שלנו כדי להעביר אותה הלאה נלחץ על Forward.

4. כעת ייתכן והדפדפן יראה שגיאה של Certificates, לכן נצטרך להתקין Certificates שנקבל מ-BurpSuite עצמם, את ה-Certificates ניתן להוריד מהכתובת http://burpsuite. מידע נוסף ומפורט כיצד להתקינם בכל דפדפן ומערכת הפעלה שונות ניתן לראות בביליוגרפיה תחת:
Configuring Burp to work with an external browser

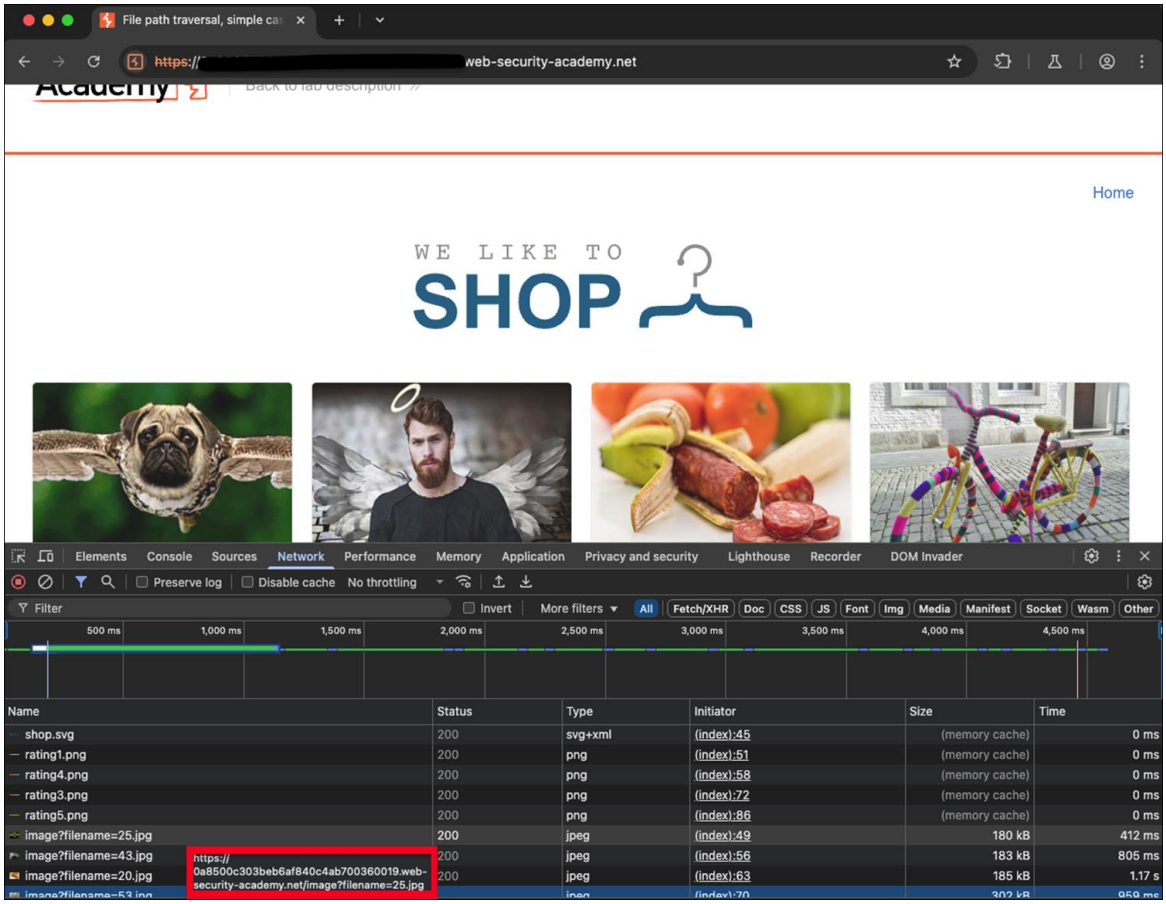
אז איך החולשה מנוצלת בפועל? נצלול אל הדברים

כחלק מהקורסים המוצעים באתר Port Swigger ניתן למצוא קורס מרחיב בנושא ואיך הפרצות מנוצלות בפועל עם מעבדות בהן ניתן להתנסות על אתר בו מיושמות הגנות שונות, את חלקן נראה כאן ונראה כיצד הצלחנו לעקוף אותן ובכל זאת לחלץ את המידע "המסווג".

המעבדות מיושמות בעזרת חנות אינטרנטית עם מוצרים, המטרה: להשיג את המידע בקובץ הנמצא בנתיב:

```
/etc/passwd
```

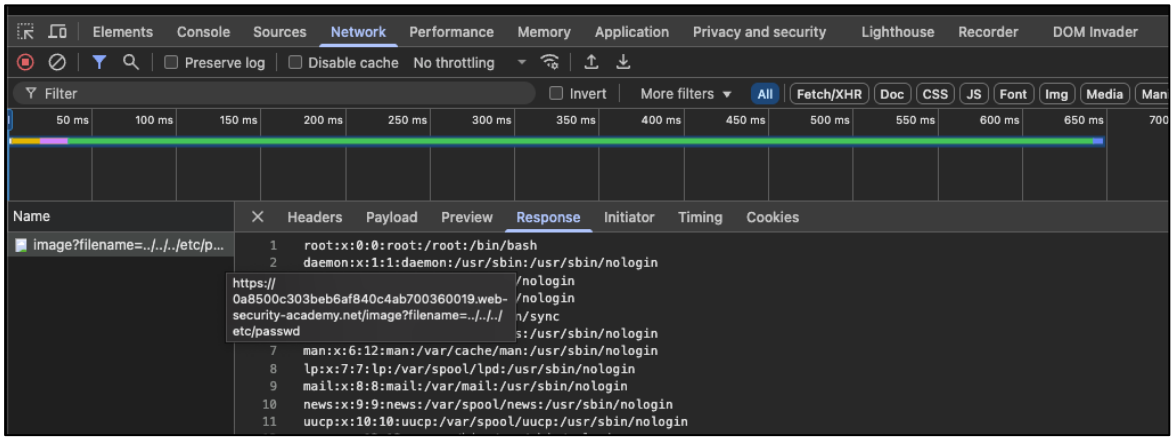
נתחיל מהמקרה הפשוט בו לא מיושמת כל הגנה מפני החולשה, ראשית נבחן כיצד האתר מציג את התמונות באמצעות כלי הפיתוח בדפדפן:



אפשר לראות בלשונית network שהאתר פונה לנתיב:

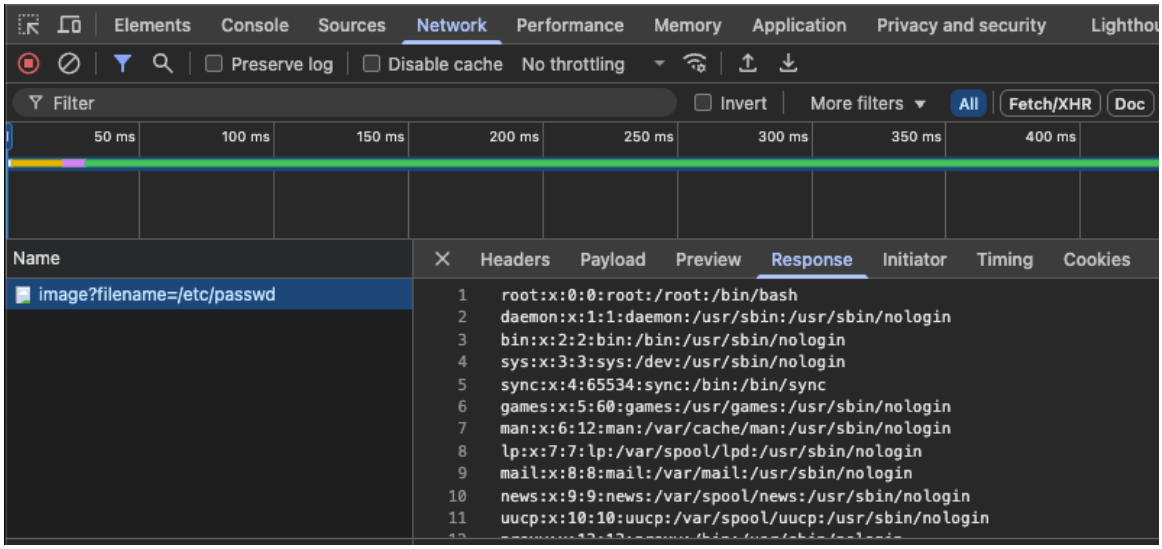
image?filename=25.jpg

על מנת להביא את התמונה מהשרת באמצעות GET request, ננצל את המידע הזה וננסה למצוא את הנתיב לקובץ הנדרש:

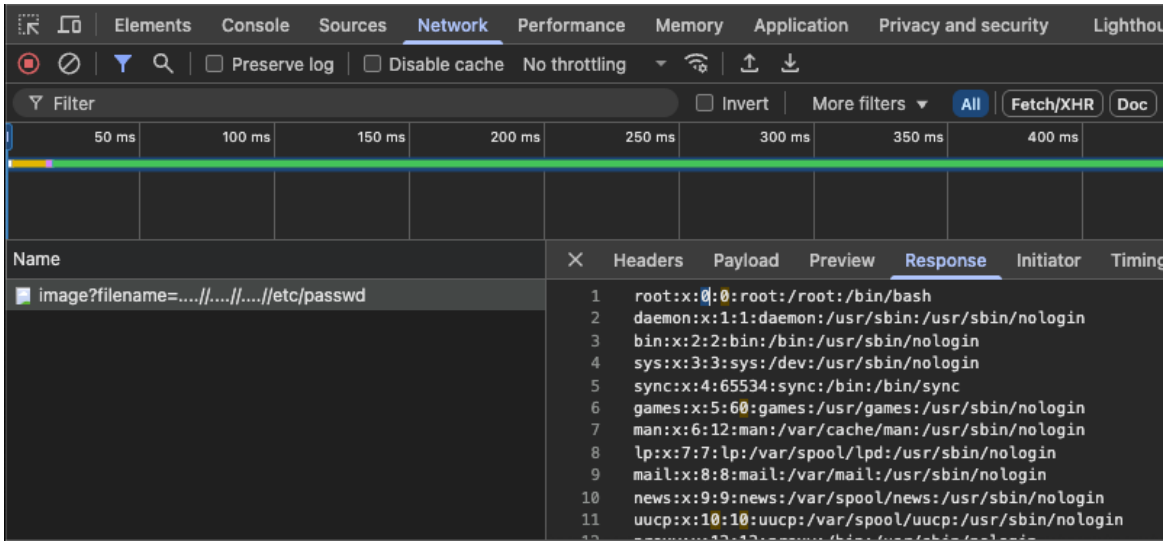


מניפולציה קלה על הנתיב על ידי הוספת "../../.." וקיבלנו את תוכן הקובץ הנדרש.

נעבור למקרה נוסף,במקרה הזה התחיליות "../../.." מוסרות מהפרמטר הנשלח בבקשה, נעקוף את זה באמצעות הנתיב המוחלט של הקובץ:



מקרה נוסף שנבחן הוא המקרה שבו התחיליות "../.." מוסרות מהבקשה באופן לא רקורסיבי, את הווליד‌ציה הזאת נעקוף באמצעות הכפלה של התווים בתחילית, במקום "../.." נשתמש ב-"..//.."



וגם כעת הגענו לתוכן הקובץ.

נעבור למקרה בו השרת מקבל את הפרמטר מהבקשה, מוריד את הקידוד, מסיר את כל הרצפים מהסוג "../.", מסיר שוב את הקידוד ומשתמש בתוצאה כנתיב.

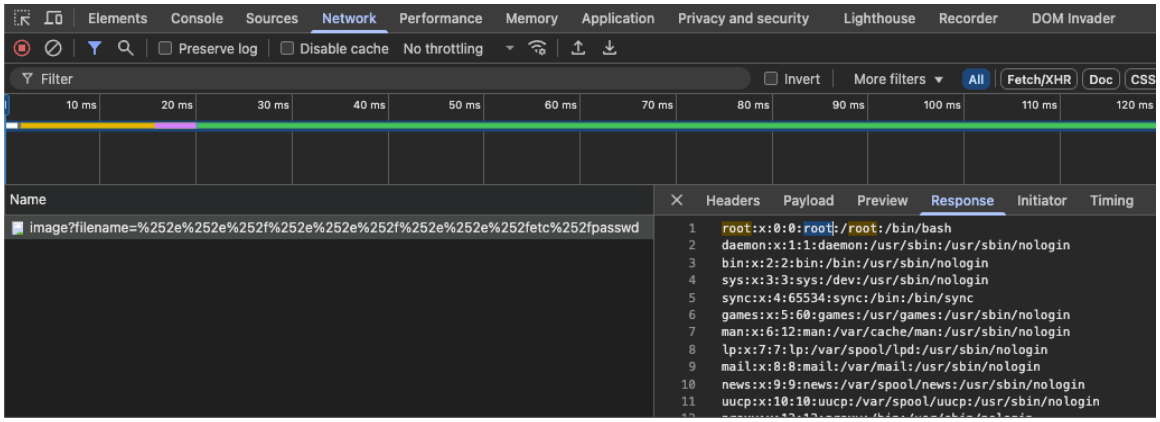
במצב הזה אם נשלח:

../../../etc/passwd או שנשלח %2 e%2e%2f

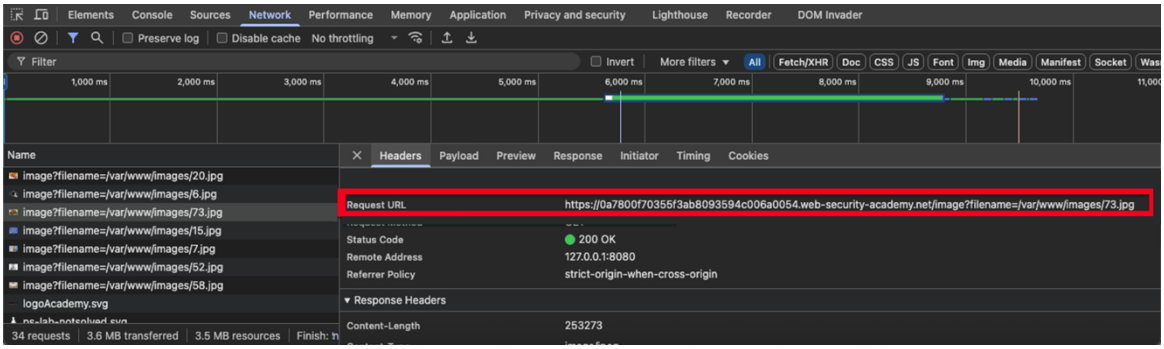
%2e%2e%2f מקודד להיות "../"

הרי שזה ייחסם, לכן נשתמש בקידוד כפול כדי לרמות את השיטה של השרת להסיר את המעבר בנתיבים, לכן נשלח:

%252e%252e%252f%252e%252e%252f%252e%252e%252fetc%252fpasswd

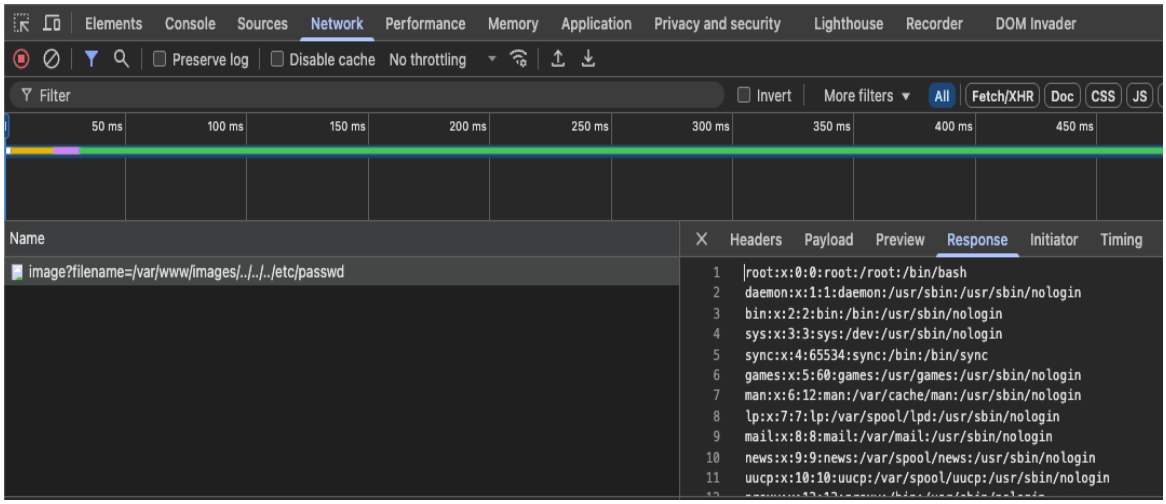


בצילום ניתן לראות שההמרה שרצינו התבצעה כמו שתוכנן. יכול להיות מקרה שבו השרת מוודא את תחילת הנתיב שנשלח כך שיהיה נתיב המיועד לקבצים שמותר לגשת אליהם, גם כאן נבצע מניפולציה, ראשית נבדוק את התחילית המתאימה:



לאחר שהאתר עלה, נשלחו בקשות דומות למה שראינו קודם, אך כעת כדי להביא את התמונה מהשרת נוסף החלק: var/www/images לבקשה, ננצל את זה, במקום לבקש את התמונה נוסיף את החלק:

../../../etc/passwd



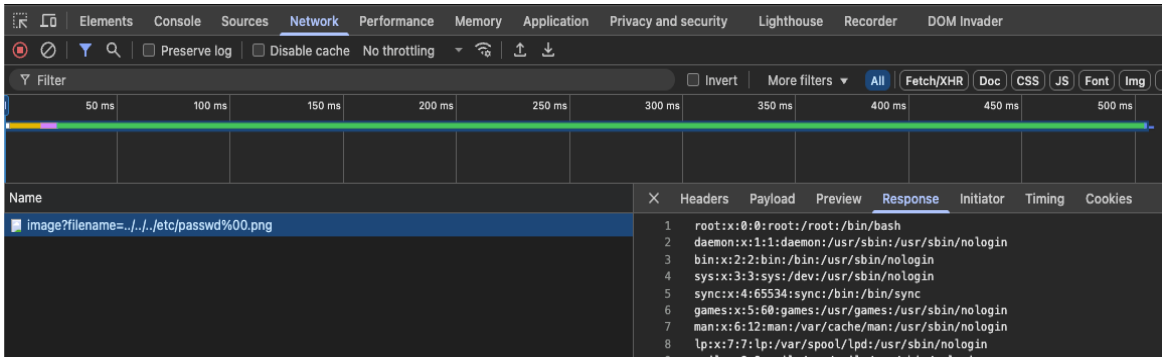
מה קורה כאשר האתר מוודא את סיומת הקובץ כדי להגביל את סוג הקבצים אליהם יש גישה? נבצע מניפולציה באמצעות בייט ריק מקודד (%00), נשלח בבקשה שלנו לשרת את הנתיב הבא:

../../../etc/passwd%00.png

במקרה הזה האתר בודק אם סיומת הקובץ תקינה, והיא אכן תקינה (במעבדה זו צריך קובץ מסוג png), אך בפועל בשרת זה מתקבל כ:

../../../etc/passwd

ואם אין ולידציה נוספת בשרת, נקבל את התוכן המבוקש ללא שום הפרעה:



הגיע הזמן, הדגמה מוחשית לפיתוח שגוי המוביל לחולשה.

Arbitrary file write by path traversal

הסבר קטן על מה שנראה, החולשה אומרת שאני יכול להעלות קבצים לשרת ולברוח מהתיקייה המיועדת להם. לצורך ההדגמה הקמתי שתי אפליקציות קטנות עם פייתון, אחת עם החולשה כשבעצם מה שקורה זה שבעת העלאת הקובץ הנתיב של התיקייה משורשר ישירות לקובץ שעולה ככה שהנתיב לקובץ הופך להיות: נתיב הבסיס+הקובץ. לכן כשנוסיף את התווים "../" ניתן לחמוק מתיקיית הבסיס, ולהלן הקוד:

```
from flask import Flask, request, jsonify
import os

app = Flask(__name__)

BASE_DIR = os.path.join(os.getcwd(), "uploads")
os.makedirs(BASE_DIR, exist_ok=True)

@app.route("/write", methods=["POST"])
def write_file():
    filename = request.form.get("filename", "")
    content = request.form.get("content", "")

    # here is the vulnerable part
    target_path = os.path.join(BASE_DIR, filename)

    with open(target_path, "w", encoding="utf-8") as f:
        f.write(content)

    return jsonify({"status": "ok", "path": target_path})
```

נריץ בדיקות קטנות להמחיש את החולשהת הקובץ הראשון note.txt עולה לתיקייה הנדרשת והקובץ השני עולה עם שם קובץ שגורם להתחמקות מהתיקייה המיועדת:

```
curl -X POST -F "filename=note.txt" -F "content=hello" [redacted]:5000/write
curl -X POST -F "filename=../escape.txt" -F "content=escape" [redacted]:5000/write
```

לפי הלוגים שהגדרנו בשרת קיבלנו סטאטוס 200 על שתי הבקשותת כלומר שהקבצים עברו:

```
"POST /write HTTP/1.1" 200
"POST /write HTTP/1.1" 200
```

וכעת ניתן לראות כי הקובץ note.txt אכן עלה לתיקייה המתאימה:

```
[redacted]:~/path_traversal_lab/uploads$ ls
note.txt
```

והקובץ escape.txt עלה לתיקיית האב בעקבות השימוש בתווים הנוספים בשם הקובץ:

```
[redacted]:~/path_traversal_lab$ ls
escape.txt [redacted] uploads [redacted]
```

נעבור לאפליקציה המאובטחת, הוספנו רשימת שמות מותרים לקבצים והוספנו פונקציה שתבדוק אם שם הקובץ מכיל תווים אסורים, ויש בדיקות נוספות שניתן להוסיף במידת הצורך אך למען ההדגמה נשארנו רק עם אלו כרגע, זה הקוד שבו השתמשתי:

```python
# whitelist of allowed basenames
ALLOWED_BASENAMES = {"note.txt", "report.txt", "user_upload.txt"}

def is_path_safe(base_dir, user_path):
    base = os.path.realpath(base_dir)
    target = os.path.realpath(os.path.join(base_dir, user_path))
    return target == base or target.startswith(base + os.sep)
@app.route("/write", methods=["POST"])
def write_file():
    filename = request.form.get("filename", "")
    content = request.form.get("content", "")

    # compute resolved path for internal checks
    resolved = os.path.realpath(os.path.join(BASE_DIR, filename))

    # check whitelist by basename
    if os.path.basename(filename) not in ALLOWED_BASENAMES:
        cid = log_rejection(request, filename, "basename-not-allowed", resolved_path=resolved)
        return jsonify({"status":"error","message":"not found","id": cid}), 404

    # ensure canonical path is inside base dir
    if not is_path_safe(BASE_DIR, filename):
        cid = log_rejection(request, filename, "path-outside-base", resolved_path=resolved)
        return jsonify({"status":"error","message":"not found","id": cid}), 404

    # passed checks -> perform write
    target_path = resolved
    try:
        with open(target_path, "w", encoding="utf-8") as f:
            f.write(content)
    except Exception as e:
        cid = str(uuid.uuid4())
        app.logger.error("cid=%s action=write-failed client=%s filename=%s err=%s", cid, request.remote_addr, filename, str(e))
        return jsonify({"status":"error","message":"not found","id": cid}), 404

    return jsonify({"status":"ok"}), 200
```

הרצתי שוב את אותן הבדיקות, התוצאה המבוקשת היא שהקובץ notes.txt עלה כנדרש והקובץ escape.txt נדחה ולא עלה בכלל.

מצרף את צילומי התשובות מהשרת עבור הקובץ notes.txt:

```
"POST /write HTTP/1.1" 200 -
```

עבור הקובץ escape.txt קיבלנו:

```
"POST /write HTTP/1.1" 404
```

לצורך הדוגמא עבור הקובץ שנדחה החזרתי סטטוס 404 כדי לא למסור מידע מיותר כמו למה נדחה או מידע נוסף שיכול לשרת את התוקף לשכלול הניסיונות שלו.

התקפה כזו יכולה להיות מסוכנת מאוד אם לא מתבצעות הבדיקות המתאימות, והתוקפים משכללים את עצמם מדי יום לכן חשוב לוודא כבר במהלך הפיתוח שלא משאירים מקום למזל או "לאנשים טובים" ומשתמשים בכלים ושיטות מתאימות לביצוע.

למעשה כמעט בכל תכנות קיימות פונקציות מובנות שניתן להשתמש בהן כדי להגן על השרת מהתקיפה, פונקציות אלו בדרך כלל ממשות עיקרון פשוט בו הן יוצרות נתיב מלא מהנתיב שהתקבל בבקשה ומשוות אותו לרשימת נתיבים מאושרים לגישה ובכך הן מונעות גישה לנתיבים שלא רוצים לחשוף בשרת.

מספר תבניות מוכרות של תוספות ל-URL שנמצאות בשימוש נרחב

הורדת קבצים או חשיפת נקודות קצה:

/download?file=report.csv → file=../../../etc/passwd

צפייה בלוגים או דיבאגרים:

/view-log?path=latest.log

תבניות שמשתמשות ב-preview להצגת קבצים:

/preview?template=invoice.html

נקודות קצה המשמשות להורדת גיבויים:

/api/fetch?name=backup.tar.gz

מספר קבצים ידועים עם פוטנציאל סיכון גבוה במערכות הפעלה השונות.

נתיבים לקבצים שונים במערכות לינוקס:

/etc/passwd - משמש בעיקר להוכחת קריאה של קבצים מהמערכת

/etc/shadow - מחזיק פרטי התחברות מוצפנים

/proc/self/environ - לרוב מחזיק משתני סביבה כמו מפתחות לחיבור שירותים חיוניים

.env, config.json, settings.py, config.yaml - קבצים אלו מחזיקים סודות כמו מפתחות חיבור לשירותים

/var/run/secrets/kubernetes.io/serviceaccount/token - מחזיק את טוקן החיבור לקוברנטיס

~/.bash_history, .git/config - מידע על הגדרות של גיט

במערכות Windows:

C:\Windows\win.ini - הוכחת יכולת הקלאסית לגישה לקבצים

C:\Users\Administrator\NTUser.dat - מידע על המשתמשים

C:\inetpub\wwwroot\web.config - פרטי התחברות לשירותי האינטרנט של מיקרוסופט

%APPDATA%\Microsoft\Credentials - פרטי התחברות למערכות מיקרוסופט שונות בשרת

מה עוד אפשר לעשות חוץ מלקרוא קבצים מהשרת?

הרבה פעמים נתקל במקרים בהם החולשה מנוצלת בשילוב עם מתקפה נוספת, מה שהופך את המתקפה לחמורה יותר. למשל תוקף שמנצל מערכות להעלאת קבצים כמו בדוגמת השרת ומעלה קובץ המכיל קוד זדוני או קוד המאפשר לו ליצור Reverse Shell.

תוקפים מנצלים את מנגנון הצגת הקבצים מהשרת, כלומר כשרוצים לקבל דף מסוים באתר, בשרת הוא מופיע כקובץ, כאשר הקובץ מסוים מוצג הוא גם מבוצע על כל הקוד שבתוכו, כך שאם תוקף הצליח להזריק קטע קוד לתוך הקובץ כאשר הקובץ יוצג קטע הקוד יבוצע. למשל אם תוקף הזריק קטע קוד שיוצר Reverse Shell, כאשר ניגש לקובץ הלוג באמצעות Path Traversal יבוצע הקוד והתוקף מקבל גישה ישירה לבצע פקודות בשרת.

ננתח ביחד חולשה אמיתית שנמצאה

בחודש אוקטובר שנת 2021, פרויקט Apache HTTP Server שחררו המלצות לתיקון הגדרות בגרסאות 2.4.49 ו-2.4.50 בעקבות חולשות שהתגלו (CVE-2021-41773 ו-CVE-2021-42013).

בגרסא ביצעו שינוי בקוד לצורך נרמול של נתיבים שמגיעים מכתובת ה-URL כך שיוסרו מהנתיב חלקים לא רצויים או מסוכנים, המטרה הייתה לבצע פענוח לתווים מקודדים שמגיעים כך שכל מחרוזת תווים תפוענח תו אחר תו. הלוגיקה בה השתמשו כשלה, ונוצר מצב שבו כאשר מנסים לבצע Path Traversal עם קידוד על שתי הנקודות, מתבצעת המרה רק על התו הראשון כך שהקידוד %2e על הנקודה השניה לא פוענח ועבר את הבדיקה בשרת. כלומר שכאשר נשלח נתיב עם הקידוד %2e/. (שאמור להיות מפוענח כ-../) עבר את הבדיקה (CVE-2021-41773).

נוסף על כך, כדי שהחולשה תוכל להיות מנוצלת, בקובץ ההגדרות בשרת צריכה להופיע ההגדרה הבאה:

```
<Directory />
   Require all granted
</Directory>
```

משמעות ההגדרה הזו היא שהגישה לכל התיקיות בשרת מותרת, או לחילופין לא תהיה בקובץ הגדרה לאילו תיקיות הגישה מורשת. שילוב של הכשל בלוגיקה ביחד עם הגדרה לא נכונה כפי שצוין מעלה, עלולים להוביל לגישה לקבצים רגישים במערכת הקבצים בשרת כמו קבצי סיסמאות.

אך לא די בכך, למרות שבהתחלה החולשה פורסמה כחולשת Path traversal ו-File Disclosure מחקר מעמיק נוסף גילה כי במידה והגדרות מסוימות מופעלות ניתן גם לבצע קוד מרחוק מתוך הבקשות הנשלחות לשרת.

החוקרים גילו כי אם המודול mod_cgi (מודול המאפשר הרצת סקריפטים מסוג ל-CGI) מאופשר בשרת שכן כברירת מחדל הוא אינו מאופשר, ניתן להריץ כל קובץ binary במערכת ולקבל את פלט ישירות לתוך התשובה שחוזרת לבקשה, ניתן לראות בדוגמא איך השתמשו בקידוד לעבור את הבדיקה והריצו פקודת echo על id ומה התקבל בתשובה:

Request:

```
POST /cgi-bin/.%2e/.%2e/.%2e/.%2e/bin/sh HTTP/1.1
Host: 127.0.0.1:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:92.0) Gecko/20100101 Firefox/92.0
Accept: */*
Content-Length: 7
Content-Type: application/x-www-form-urlencoded
Connection: close
echo;id
```

והתשובה:

Response:

```
HTTP/1.1 200 OK
Date: Mon, 18 Oct 2021 09:58:23 GMT
Server: Apache/2.4.49 (Unix)
Connection: close
Content-Length: 45
uid=1(daemon) gid=1(daemon) groups=1(daemon)
```

אז לאחר שהלוגיקה תוקנה יצאה גרסא חדשה 2.4.50, וכעת הקידוד של הנקודה השניה לא עובר את הבדיקה. אך החוקרים כמו כל חוקרים טובים לעולם לא מתייאשים מלשבור דברים גילו שכעת לאחר התיקון אם נבצע קידוד כפול, כלומר שאם נקודה מקודד להיות %2e נקודד את 2 להיות %32 ואת e להיות %65 את ה-% המקורי של הקידוד הראשון נשאר כמו שהוא) נקבל כעת קידוד שנראה כך:

. >> %2e >> %%32%65

ובכך ניתן לעבור שוב את הבדיקות בשרת ולהוציא לפועל את אותן התקיפות שצוינו בחולשה הקודמת של שליפת מידע והרצת קוד מרחוק (CVE-2021-42013).

ואם חשבתם שזה קורה רק באתרים או שירותי אינטרנט, תרשו לי לשתף אתכם על פרצה חדשה שהתגלתה לא מזמן בתוכנה שכולנו מכירים : WinRAR.

הסבר קצר למי שלא מכיר, WinRAR היא תוכנה המשמשת ליצירת ארכיון של קבצים דחוסים וחילוץ של קבצים כדי שהמשקל שלהם יהיה קל יותר.

בשנת 2025 דווח על ידי החוקרים של חברת ESET על חולשה שנמצאה בתוכנה (CVE-2025-8088). החולשה סווגה כ-Path traversal flaw וקיבלה דירוג גבוה של 8.4. החולשה מאפשרת לתוקפים ליצור קבצי ארכיון כך שמי שייחלץ את הקבצים הם יחולצו לנתיב שהתוקף בוחר מראש, ומכאן שניתן להשתמש בחולשה זו כדי לחלץ ישירות לנתיב בו נמצאים קבצים המופעלים בעת עליית מערכת ההפעלה ב-Windows.

לכן שמי שמשתמש בתוכנה הזאת מומלץ לעדכן לגרסאות חדשות בהן כבר נסגרה החולשה.

אז איך מתגוננים?

חולשות מסוג Path Traversal הן דוגמה טובה לכך שגם פרצות ישנות ממשיכות לאיים.

לא כי הן מתוחכמות, אלא כי לעיתים מפתחים לא שמים לב שהקלט מהמשתמש עובר ישירות למערכת הקבצים, או שפשוט עם מניפולציה קלה על הבקשה אפשר להגיע לדברים שממש לא רצינו שיגיעו אליהם, תוקף יכול לנצל את זה כדי לגשת לקבצים רגישים בשרת, גם אם הם לא אמורים להיות זמינים בכלל.

כדי להגן על האפליקציה שלנו, מומלץ לשלב כמה שכבות הגנה:

1. לא לאפשר למשתמש להגדיר שמות קבצים בעצמו. במקום שהמשתמש ישלח שם של קובץ, עדיף שהשרת יקבע בעצמו שמות לפי מזהים פנימיים או קבועים מראש.
2. לבדוק את הקלט שהמשתמש שולח. אם כן צריך לאפשר הזנת שם קובץ כדאי לבדוק מראש שמדובר בשם תקני בלבד (למשל: בלי תווים מיוחדים כמו /.., ~, או %), או להשוות את השם לרשימה מוכנה מראש של שמות מותרים.
3. לוודא שהקובץ נמצא במקום מותר. אחרי שחיברנו את שם הקובץ לתיקיית הבסיס, כדאי להשתמש בפונקציה שמחשבת את הנתיב האמיתי (למשל getCanonicalPath) ולבדוק שהוא עדיין בתיקייה שאנחנו מרשים לגשת אליה.
4. תמיד לבדוק בצד השרת. גם אם יש בדיקות בצד הלקוח (ב־JavaScript), חייבים לעשות את כל הבדיקות גם בצד השרת. תוקף, גם לא הכי מתוחכם, יכול בקלות לעקוף את מה שקורה בדפדפן.
5. לתעד ניסיונות חשודים. כדאי לתעד בעזרת לוגים או התראות כל ניסיון לגשת לנתיב חריג (למשל קבצים שמכילים /..) כדי לזהות ניסיונות תקיפה בזמן אמת וללמוד מהם בעתיד.
6. להגדיר הרשאות מתאימות, ולבדוק את ההרשאות בכל פעולה. השרת עצמו צריך להיות מוגבל בגישה - אם מישהו בכל זאת מצליח לעקוף את ההגנות, הרשאות מערכת צריכות למנוע ממנו לגשת לקבצים רגישים.
7. לא להחזיר מידע מיותר. במקום לחזיר הודעה שיכולה לתת מידע האם האובייקט קיים או לא, נשתמש בהודעות כלליות למשל: Access Denied.

אוסיף ואומר שהשקעת הזמן וחשיבה מראש על על נקודות כשל כמו אלו יכולה לחסוך הרבה זמן וכאבי ראש בהמשך וכמו כן אולי להיות נקודת המפנה שבין להיות בצד המותקף לבין להיות בצד ששומע על מתקפה שקרתה.

לכן שעוד בשלב התכנון כדאי להתייעץ עם מומחים שיכולים להיכנס לראש של תוקף ובהמשך גם להוסיף בדיקות נוספות מעבר לווידוא כי הרכיב שפותח עובד כנדרש.

לסיכום, החוויה האישית שלי

במהלך העבודה על המעבדות של PortSwigger, התנסיתי בעצמי והופתעתי לגלות כמה קל לעקוף הגנות שלא נבנו כמו שצריך. המסקנה העיקרית שלי היא: אבטחה טובה מתחילה עוד משלב התכנון.

לסיום, אני ממליץ לכל מפתח לא משנה אם אתם בתחילת הדרך או כבר עם ניסיון להתנסות בעצמכם במעבדות כאלה. אין תחליף ללמידה מעשית כשזה מגיע להבנה של חולשות אמיתיות.

תרגישו חופשי ליצור איתי קשר בלינקדין:
https://www.linkedin.com/in/david-starinsky

ביבליוגרפיה

Aikido - Path Traversal in 2024 - The year unpacked: https://www.aikido.dev/blog/path-traversal-in-2024-the-year-unpacked

Edgescan - 2024 Vulnerability Statistics Report: https://www.edgescan.com/wp-content/uploads/2025/04/2024-Vulnerability-Statistics-Report.pdf

Configuring Burp to work with an external browser: https://portswigger.net/burp/documentation/desktop/external-browser-config

CISA - Secure by Design Alert (May 2024): https://www.cisa.gov/sites/default/files/2024-05/Secure_by_Design_Alert_Eliminating_Directory_Traversal_Vulnerabilities_in_Software_508c%20(3).pdf

PortSwigger - Web Security Academy: https://portswigger.net/web-security

CVE-2025-8088 Detail: https://nvd.nist.gov/vuln/detail/CVE-2025-8088

Practical guide to path traversal: https://www.yeswehack.com/learn-bug-bounty/practical-guide-path-traversal-attacks

Apache HTTP Server Path Traversal & Remote Code Execution (CVE-2021-41773 & CVE-2021-42013): https://blog.qualys.com/vulnerabilities-threat-research/2021/10/27/apache-http-server-path-traversal-remote-code-execution-cve-2021-41773-cve-2021-42013