Internship: Cybersecurity

Intern: Umar Farooq

DHC-1327

Contents

Week 4: Advanced Threat Detection & Web Security Enhancements

Task 1: Intrusion Detection & Monitoring — Fail2Ban Setup

Objective

To implement real-time intrusion detection on the server by setting up **Fail2Ban**, which actively monitors authentication logs, detects multiple failed login attempts (brute-force attacks), and automatically bans offending IP addresses. This enhances the security of the server environment against unauthorized access.

Tools & Technologies

- **Fail2Ban** – Log monitoring and IP banning tool
- **Ubuntu 20.04+ / Debian-based Linux OS**
- **Systemd** – For service management
- **SSH Service** – Monitored for brute-force protection
- **Sendmail (optional)** – To send alert emails

Implementation Steps

1. Installation of Fail2Ban

```
sudo apt update
sudo apt install fail2ban -y
```

Fail2Ban is installed to monitor logs and automatically take action on suspicious activity.

2. Configuration of Fail2Ban

Created a custom jail configuration file at /etc/fail2ban/jail.local to define monitoring rules.

```
[DEFAULT]
bantime = 3600              # IP ban duration (1 hour)
findtime = 600              # Time frame to count failures (10 minutes)
maxretry = 3                # Failed login attempts before banning
action = %(action_mwl)s     # Email with log and whois info

[sshd]
enabled = true
port = ssh
logpath = %(sshd_log)s
backend = systemd
```

- The [sshd] section enables monitoring of SSH authentication attempts.
- %(action_mwl)s ensures the administrator is notified with detailed logs and attacker info.

3. Service Restart & Enable

```
sudo systemctl restart fail2ban
sudo systemctl enable fail2ban
```

Fail2Ban is restarted and enabled to run on boot.

4. Verifying the Setup

To check overall status:

```
sudo fail2ban-client status
```

To check specific jail (SSH):

```
sudo fail2ban-client status sshd
```

If intrusion attempts are detected, banned IPs will appear in the output.

5. Testing the Detection System

To verify the setup:

- Attempt 3 incorrect SSH logins from a remote machine.
- The IP address is automatically banned for 1 hour.
- View log: /var/log/fail2ban.log

```
┌──(mutex㉿kali)-[~/Desktop]
└─$ sudo fail2ban-client status sshd

Status for the jail: sshd
|- Filter
|  |- Currently failed: 0
|  |- Total failed:     0
|  `- Journal matches:  _SYSTEMD_UNIT=ssh.service + _COMM=sshd
`- Actions
   |- Currently banned: 0
   |- Total banned:     0
   `- Banned IP list:
```

```
C:\Users\umarfarooq>ssh wronuser@192.168.209.247
The authenticity of host '192.168.209.247 (192.168.209.247)' can't be established.
ED25519 key fingerprint is SHA256:M2TKzBbCg0ZDD9hC5Mai3V6hs5tXSg9GpC/THvpO3zw.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.209.247' (ED25519) to the list of known hosts.
wronuser@192.168.209.247's password:
Permission denied, please try again.
wronuser@192.168.209.247's password:
Permission denied, please try again.
```

```
┌──(mutex㉿kali)-[~/Desktop]
└─$ sudo fail2ban-client status sshd

Status for the jail: sshd
|- Filter
|  |- Currently failed: 0
|  |- Total failed:     3
|  `- Journal matches:  _SYSTEMD_UNIT=ssh.service + _COMM=sshd
`- Actions
   |- Currently banned: 1
   |- Total banned:     1
   `- Banned IP list:   192.168.209.132
```

```
┌──(mutex㉿kali)-[~/Desktop]
└─$ sudo systemctl status ssh

[sudo] password for mutex:
○ ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/usr/lib/systemd/system/ssh.service; disabled; preset: disabled)
     Active: inactive (dead)
       Docs: man:sshd(8)
             man:sshd_config(5)

┌──(mutex㉿kali)-[~/Desktop]
└─$ sudo systemctl start ssh


┌──(mutex㉿kali)-[~/Desktop]
└─$ sudo systemctl enable ssh

Synchronizing state of ssh.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable ssh
Created symlink '/etc/systemd/system/sshd.service' → '/usr/lib/systemd/system/ssh.service'.
Created symlink '/etc/systemd/system/multi-user.target.wants/ssh.service' → '/usr/lib/systemd/system/ssh.service'.

┌──(mutex㉿kali)-[~/Desktop]
└─$ sudo ss -tulpn | grep ssh

tcp   LISTEN 0      128          0.0.0.0:22        0.0.0.0:*    users:(("sshd",pid=5565,fd=3))
tcp   LISTEN 0      128             [::]:22           [::]:*    users:(("sshd",pid=5565,fd=4))
```

```
Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::d939:baa1:8b31:1621%4
   IPv4 Address. . . . . . . . . . . : 192.168.209.132
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.209.50
```

Log Sample

From **/var/log/fail2ban.log**:

```
2025-06-14 11:42:51,984 fail2ban.server          [5426]: INFO    Starting Fail2ban v1.1.0
2025-06-14 11:42:51,984 fail2ban.observer        [5426]: INFO    Observer start...
2025-06-14 11:42:51,991 fail2ban.database        [5426]: INFO    Connected to fail2ban persistent database '/var/lib/fail2ban/fail2ban.sqlite3'
2025-06-14 11:42:51,992 fail2ban.jail            [5426]: INFO    Creating new jail 'sshd'
2025-06-14 11:42:51,995 fail2ban.jail            [5426]: INFO    Jail 'sshd' uses systemd {}
2025-06-14 11:42:51,996 fail2ban.jail            [5426]: INFO    Initiated 'systemd' backend
2025-06-14 11:42:51,996 fail2ban.filter          [5426]: INFO      maxLines: 1
2025-06-14 11:42:52,007 fail2ban.filtersystemd   [5426]: INFO    [sshd] Added journal match for: '_SYSTEMD_UNIT=ssh.service + _COMM=sshd'
2025-06-14 11:42:52,007 fail2ban.filter          [5426]: INFO      maxRetry: 3
2025-06-14 11:42:52,007 fail2ban.filter          [5426]: INFO      findtime: 600
2025-06-14 11:42:52,007 fail2ban.actions         [5426]: INFO      banTime: 3600
2025-06-14 11:42:52,007 fail2ban.filter          [5426]: INFO      encoding: UTF-8
2025-06-14 11:42:52,009 fail2ban.filtersystemd   [5426]: INFO    [sshd] Jail is in operation now (process new journal entries)
2025-06-14 11:42:52,010 fail2ban.jail            [5426]: INFO    Jail 'sshd' started
2025-06-14 11:47:48,242 fail2ban.filter          [5426]: INFO    [sshd] Found 192.168.209.132 - 2025-06-14 11:47:48
2025-06-14 11:47:56,341 fail2ban.filter          [5426]: INFO    [sshd] Found 192.168.209.132 - 2025-06-14 11:47:56
2025-06-14 11:48:06,094 fail2ban.filter          [5426]: INFO    [sshd] Found 192.168.209.132 - 2025-06-14 11:48:05
2025-06-14 11:48:06,505 fail2ban.actions         [5426]: NOTICE  [sshd] Ban 192.168.209.132
```

Outcome

- Repeated SSH login failures are detected and blocked in real-time.
- Admins are alerted via email for further investigation.
- Significantly reduces risk of brute-force attacks.

Task 2: API Security Hardening

Objective

Secure backend API endpoints by:

- Preventing brute-force attacks via **rate limiting**
- Restricting unauthorized access with **CORS**
- Adding **API Key or OAuth** authentication for access control

Technologies Used

- **Node.js** with **Express.js**
- **`express-rate-limit`** for brute-force protection
- `cors` package for **CORS** configuration
- API Key middleware for simple authentication

Implementation

1. Rate Limiting using express-rate-limit

Install the package:

```
npm install express-rate-limit
```

Add to your Express app:

```
const rateLimit = require('express-rate-limit');

// Apply rate limiting to all API routes
const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100, // Max 100 requests per IP
  message: 'Too many requests from this IP, please try again after 15 minutes.'
});

app.use('/api/', limiter);
```

Purpose:

Prevents brute-force attacks and abuse of API endpoints.

2. CORS Configuration using cors package

Install CORS:

```
npm install cors
```

Setup:

```
const cors = require('cors');

const corsOptions = {
  origin: 'https://your-frontend-domain.com', // Replace with your frontend
  methods: 'GET,POST,PUT,DELETE',
  credentials: true
};

app.use(cors(corsOptions));
```

Purpose:

Restricts access to trusted origins only.

3. API Authentication using API Key (simple method)

Set API key in `.env`:

```
API_KEY=secure-key-12345
```

Create middleware:

```
require('dotenv').config();

function authenticateApiKey(req, res, next) {
  const key = req.headers['x-api-key'];
  if (!key || key !== process.env.API_KEY) {
    return res.status(401).json({ error: 'Unauthorized' });
  }
  next();
}
```

Apply to protected routes:

```
app.use('/api/secure', authenticateApiKey);
```

Purpose:

Ensures only authenticated requests can access secure routes.

```
C:\Users\umarfarooq>curl -H "x-api-key: mysecureapikey123" http://localhost:5000/profile
<!DOCTYPE html>
<html>
  <head>
    <title>Profile</title>
    <style>
      body {
        font-family: 'Segoe UI', sans-serif;
        background: #fff7e6;
        display: flex;
        justify-content: center;
        align-items: center;
        height: 100vh;
      }
      .card {
        background: white;
        padding: 2rem;
        border-radius: 10px;
        box-shadow: 0 0 15px rgba(0,0,0,0.1);
        text-align: center;
        width: 350px;
      }
      h2 {
        color: #ff8800;
      }
      p {
        color: #555;
      }
    </style>
  </head>
  <body>
    <div class="card">
      <h2>Welcome, User!</h2>
      <p>This is your profile page.</p>
```

Task 3: Security Headers & CSP Implementation

Objective:

To enhance web application security by:

- Preventing XSS and injection attacks using **Content Security Policy (CSP)**
- Enforcing HTTPS using **Strict-Transport-Security (HSTS)**
- Applying security headers using the helmet middleware

Technologies Used

- **Node.js + Express.js**
- **helmet** – for HTTP security headers

Implementation

1. Install helmet middleware

```
npm install helmet
```

2. Basic Helmet Setup

```
const express = require('express');
const helmet = require('helmet');

const app = express();

// Apply Helmet middleware
app.use(helmet());
```

This automatically adds several security headers like:

- X-Content-Type-Options: nosniff
- X-DNS-Prefetch-Control: off
- X-Frame-Options: DENY
- Strict-Transport-Security (if HTTPS is used)
- X-XSS-Protection: 0 (modern browsers handle this themselves)

3. Add a Custom Content Security Policy

```
app.use(
  helmet.contentSecurityPolicy({
    directives: {
      defaultSrc: ["'self'"], // allow from same origin
      scriptSrc: ["'self'", "https://trusted.cdn.com"],
      styleSrc: ["'self'", "'unsafe-inline'"],
      imgSrc: ["'self'", "data:"],
      objectSrc: ["'none'"],
    },
  })
);
```

This prevents attackers from injecting scripts from untrusted domains.

4. Add Strict-Transport-Security (HSTS)

Only effective **if your app uses HTTPS**.

Helmet includes HSTS by default. You can customize it:

```
app.use(
  helmet.hsts({
```

```
    maxAge: 31536000, // 1 year in seconds
    includeSubDomains: true,
    preload: true,
  })
);
```

Week 5: Ethical Hacking & Exploiting Vulnerabilities

Goal:

Learn ethical hacking techniques, identify vulnerabilities, and implement fixes in your web application.

Task 1: Ethical Hacking Basics

Objective:

Conduct reconnaissance on a test web app using tools like **Kali Linux**, **Nmap**, and **Burp Suite**.

Step-by-Step Reconnaissance

1. Nmap: Scan Target Web App

```
nmap -sV -p- 127.0.0.1
```

- `-sV` — Version detection
- `-p-` — Scan all 65535 ports
- Replace `127.0.0.1` with your target's IP

```
┌──(mutex㉿kali)-[~/Desktop]
└─$ nmap -sV -p- 127.0.0.1

Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-06-15 13:09 EDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000090s latency).
Not shown: 65534 closed tcp ports (conn-refused)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 9.7p1 Debian 7 (protocol 2.0)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 3.33 seconds
```

2. Dirb or Gobuster: Find Hidden Directories

```
gobuster dir -u http://localhost:3000 -w /usr/share/wordlists/dirb/common.txt
```

Finds hidden paths like /admin, /login, /api

```
┌──(mutex㉿kali)-[~/Desktop]
└─$ gobuster dir -u http://192.168.209.132:5000 -w /usr/share/wordlists/dirb/common.txt

===============================================================
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://192.168.209.132:5000
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /usr/share/wordlists/dirb/common.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.6
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/login               (Status: 200) [Size: 1536]
/Login               (Status: 200) [Size: 1536]
/profile             (Status: 403) [Size: 40]
/signup              (Status: 200) [Size: 1663]
Progress: 4614 / 4615 (99.98%)
===============================================================
Finished
===============================================================
```

Task 2: SQL Injection & Exploitation

Objective:

Identify and exploit SQL Injection (SQLi) vulnerabilities, then secure the app using **prepared statements**.

Test Vulnerable Input Field:

Suppose your app has a login form:

```
<input name="username">
<input name="password">
```

Try this as input:

```
Username: ' OR 1=1 --
Password: anything
```

If it logs in, you have SQLi!

Automate SQLi Detection with SQLMap

1. Intercept login request in Burp
2. Save request to a file, e.g., request.txt
3. Run SQLMap:

```
sqlmap -r request.txt --batch --dbs
```

Fix: Use Prepared Statements (Node.js + MySQL Example)

Vulnerable:

```
const query = `SELECT * FROM users WHERE username = '${username}' AND password =
'${password}'`;
```

Safe:

```
const query = 'SELECT * FROM users WHERE username = ? AND password = ?';
db.query(query, [username, password], (err, result) => {
  // handle result
});
```

Task 3: CSRF Protection

Objective:

Implement CSRF protection in the backend and test it with **Burp Suite**.

Implement with csurf in Express.js

Install:

```
npm install csurf cookie-parser
```

Setup Middleware:

```
const csrf = require('csurf');
const cookieParser = require('cookie-parser');

app.use(cookieParser());
app.use(csrf({ cookie: true }));

// CSRF token route
app.get('/form', (req, res) => {
  res.json({ csrfToken: req.csrfToken() });
});
```

The client must send the token in future requests.

Test CSRF in Burp Suite

- Capture a POST request with a valid token
- Try replaying without or with an old token → should fail with **403 Forbidden**

Week 6: Advanced Security Audits & Final Deployment Security Goal:

Conduct security audits, apply deployment best practices, and finalize a fully secured application ready for production.

Task 1: Security Audits & Compliance

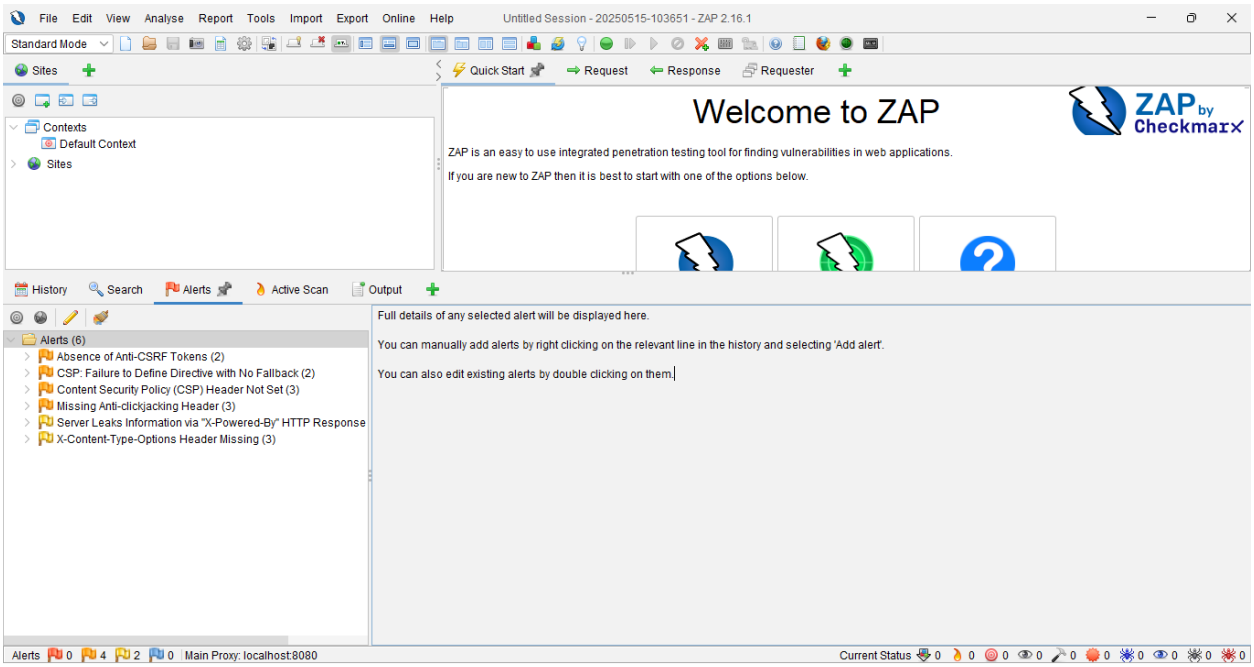
Objective:

Use auditing tools to identify vulnerabilities and ensure OWASP Top 10 compliance.

Tools:

- OWASP ZAP
- Nikto
- Lynis

1. Run OWASP ZAP Scan



2. Run Nikto (Web Server Scanner)

Detects:

- Outdated software
- Dangerous files and directories
- Default files (e.g., /admin, /test)

```
┌──(mutex㉿kali)-[~/Desktop]
└─$ nikto -h http://192.168.209.132:5000
- Nikto v2.5.0
---------------------------------------------------------------------------
+ Target IP:          192.168.209.132
+ Target Hostname:    192.168.209.132
+ Target Port:        5000
+ Start Time:         2025-06-15 14:40:30 (GMT-4)
---------------------------------------------------------------------------
+ Server: No banner retrieved
+ /: Retrieved x-powered-by header: Express.
+ /: Retrieved access-control-allow-origin header: http://localhost:5000.
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
+ /: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type. See: https
://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/
+ Root page / redirects to: /login
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ /login/: This might be interesting.
+ 8103 requests: 0 error(s) and 5 item(s) reported on remote host
+ End Time:           2025-06-15 14:40:59 (GMT-4) (29 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

3. Run Lynis (System Security Audit)

```
sudo apt install lynis
sudo lynis audit system
```

Checks:

- OS-level vulnerabilities
- SSH config
- File permissions
- Logging policies

OWASP Top 10 Checklist (Compliance Review)

- Injection (SQLi)
- Broken Auth
- Sensitive Data Exposure
- XML External Entities (XXE
- Broken Access Control
- Security Misconfig
- XSS

- Insecure Deserialization
- Using Components with Known Vulns

Task 2: Secure Deployment Practices

Enable Automatic Security Updates

Ubuntu/Debian:

```
sudo apt install unattended-upgrades
sudo dpkg-reconfigure --priority=low unattended-upgrades
```

Dependency Scanning in Node.js

Install audit tools:

```
npm audit fix
```

To view known issues:

```
npm audit
```

Use `snyk` for deeper scans:

```
npm install -g snyk
snyk test
```

Task 3: Final Penetration Testing

Manual Test with Burp Suite:

Test:

- Input fields (SQLi, XSS)
- Cookies (HTTPOnly, Secure)
- Authentication flows (Session hijacking)
- CSRF defenses