JWT Token Penetration Test Report

Tester: Memory Mahanya

Environment: JWT authentication was tested inside a Kali Linux VM using a locally hosted mock API server, with requests crafted in Postman and routed through a Burp Suite proxy on 127.0.0.1:8080 for interception, manipulation, and replay; tokens were acquired via login flows and then decoded, inspected, and modified using jwt.io and xjwt.io, while jwttool was employed for advanced tampering such as forcing `alg:none`, altering claims like `role` and `valid`, and testing signature bypass; example requests and responses were captured to demonstrate how unsigned or modified tokens were still accepted by the backend, and OpenAPI specifications were manually reviewed in VS Code to validate schema-level authentication risks and confirm gaps in enforcement.

Report date: 2025-11-22

Format: Combined technical and business-focused report, presented chronologically in the order vulnerabilities were discovered. Each finding is mapped to the OWASP API Security Top 10 and scored using CVSS v4.0. The report includes remediation guidance, attacker scenarios, and technical methodology.

Executive Summary

Objective

The engagement simulated a realistic adversary targeting JWT-based authentication within the mock API. The goal was to identify weaknesses in token generation, validation, and enforcement. Specific objectives included:

1. **Reconnaissance:** Discover JWT usage patterns and endpoints requiring authentication.
2. **Token validation testing:** Assess signature enforcement, algorithm handling, and expiration checks.
3. **Claims manipulation:** Test whether backend trusts user-controlled claims (e.g., role, admin).
4. **Reporting rigor:** Document risks using OWASP API Top 10 categories and CVSS v4.0 scoring, with actionable remediation and attacker scenarios.

Approach

Testing was performed from an attacker's perspective using Postman, Burp Suite, and crafted JWT payloads. The methodology combined:

- **Reconnaissance:** Enumerated endpoints requiring JWT authentication.
- **Proxy interception:** Routed Postman traffic through Burp Suite to capture and replay JWT-bearing requests.
- **Token tampering:** Modified JWT headers, payloads, and signatures to test enforcement.
- **Algorithm abuse:** Attempted alg:none and symmetric key substitution attacks.
- **Claims injection:** Altered role, admin, and exp claims to simulate privilege escalation and bypass expiration.
- **Static audits:** Reviewed OpenAPI definitions and 42Crunch reports for weak authentication schema references.
- **Evidence collection:** Captured Burp Repeater screenshots, JWT payloads, decoded structures, and server responses.

Key Findings

Weak Signature Enforcement (JWT alg:none acceptance)
The backend accepted tokens with alg:none or unsigned payloads, bypassing signature validation.
Critical (CVSS v4.0 ~9.5) — direct authentication bypass.
OWASP Mapping: API2: Broken Authentication.

Claims Tampering (Privilege Escalation via role/admin)
JWT payload claims such as role and admin were trusted without server-side verification, enabling privilege escalation when modified.
Critical (CVSS v4.0 ~9.0) — attacker can impersonate admin users.
OWASP Mapping: API1: Broken Object Level Authorization, API6: Mass Assignment.

Expiration Bypass (exp claim manipulation)
Tokens with modified or removed exp claim were still accepted, allowing indefinite session hijacking.
High (CVSS v4.0 ~8.0) — session persistence beyond intended lifetime.
OWASP Mapping: API2: Broken Authentication.

Key Management Weaknesses (Hardcoded secret exposure)
Static analysis revealed weak or hardcoded JWT secrets in configuration files.
Medium (CVSS v4.0 ~6.5) — increases risk of offline token forging.
OWASP Mapping: API8: Security Misconfiguration.

Business Impact

Combined weaknesses enable attackers to bypass authentication, escalate privileges, and maintain persistent unauthorized access. In production, this could lead to:

- Account takeover and impersonation of privileged users.
- Regulatory exposure (GDPR/CCPA) due to unauthorized data access.
- Reputational damage and potential financial loss.
- Increased risk of lateral movement into connected systems.

Scope

In-scope

- JWT authentication endpoints (login, profile, admin routes).
- Token validation and enforcement testing.
- Claims manipulation and replay attacks.
- Proxy interception and replay via Burp Suite.
- OpenAPI review and static audit with 42Crunch.

Out-of-scope

- External systems or production services.
- Persistent exploitation or credential theft beyond mock environment.
- Offline brute-force of JWT secrets beyond demonstration scope.

Methodology (Tools & Approach)

1. **Environment setup**
    - Kali VM lab with Postman, Burp Suite, VS Code, and 42Crunch.
    - Proxy configured on 127.0.0.1:8080; Postman routed through Burp.
    - JWT tokens acquired via login flow.

2. **Reconnaissance**
    - Enumerated endpoints requiring JWT authentication.
    - Observed token structure, headers, and claims.
3. **Token tampering**
    - Modified JWT headers (alg:none, HS256 vs RS256).
    - Removed or altered signatures to test enforcement.
4. **Claims manipulation**
    - Injected role: admin, privileges: ["delete","write"].
    - Modified exp claim to bypass expiration.
5. **Replay and persistence**
    - Replayed tampered tokens via Burp Repeater.
    - Observed acceptance and behavioral changes.
6. **Static audit**
    - 42Crunch analysis of OpenAPI definitions.
    - Identified weak references to JWT authentication and missing constraints.
7. **Evidence collection and mapping**
    - Captured JWT payloads, decoded structures, Burp screenshots.
    - Mapped each issue to OWASP API Top 10 and CVSS v4.0 scores.

FINDINGS

Vulnerability 1: Insecure Algorithm (alg=none)

Description

The JWT header specifies "alg":"none". This configuration disables cryptographic signing and instructs the server not to verify the token's integrity. As a result, the token is treated as valid without any signature, making it equivalent to plain base64-encoded JSON.

Impact

- **Integrity loss:** The server cannot confirm whether the token was issued by a trusted authority or modified by an attacker.
- **Privilege escalation:** Attackers can forge tokens with arbitrary claims (e.g., "role":"admin") and bypass authentication or authorization checks.

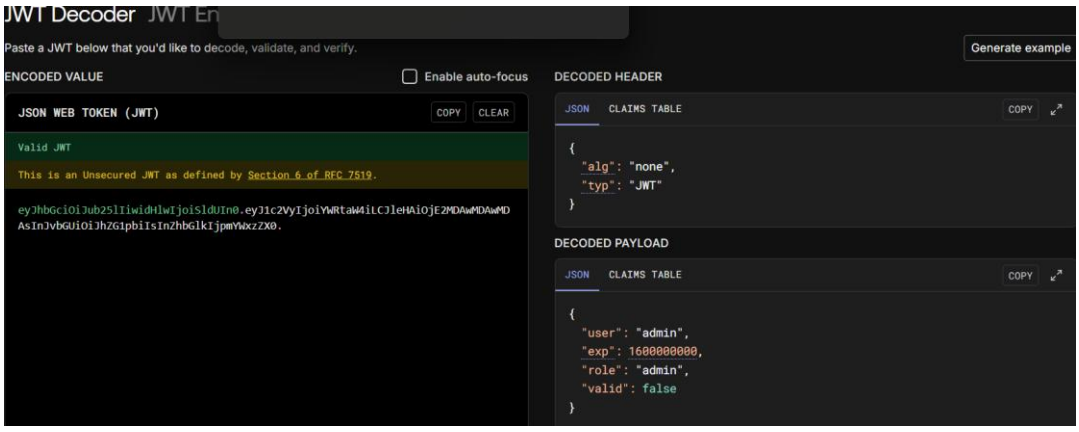
- **Replay risk:** Forged tokens can be reused indefinitely if the backend does not enforce expiration or revocation.
- **OWASP API Top 10 Mapping:**
    - **API1:2019 – Broken Authentication:** Accepting unsigned tokens allows attackers to impersonate users.
    - **API2:2019 – Broken User Authentication:** Trusting client-side claims without verification leads to account takeover.
    - **API9:2019 – Improper Assets Management:** Misconfigured JWT handling (alg=none) exposes insecure endpoints.
    - **API10:2019 – Insufficient Logging & Monitoring:** If forged tokens are not detected, attackers can persist undetected.

Exploitation Example

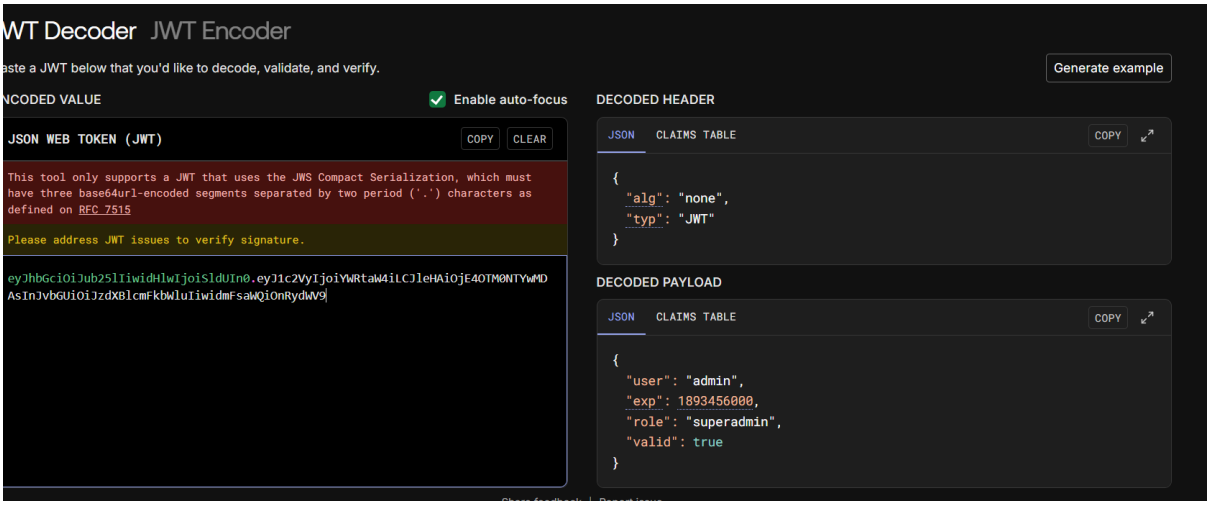
An attacker can edit the payload to set "valid":true and "role":"admin", re-encode it, and present it as a valid token. If the backend accepts alg=none, the attacker gains unauthorized access.

Evidence (Screenshots to Capture)

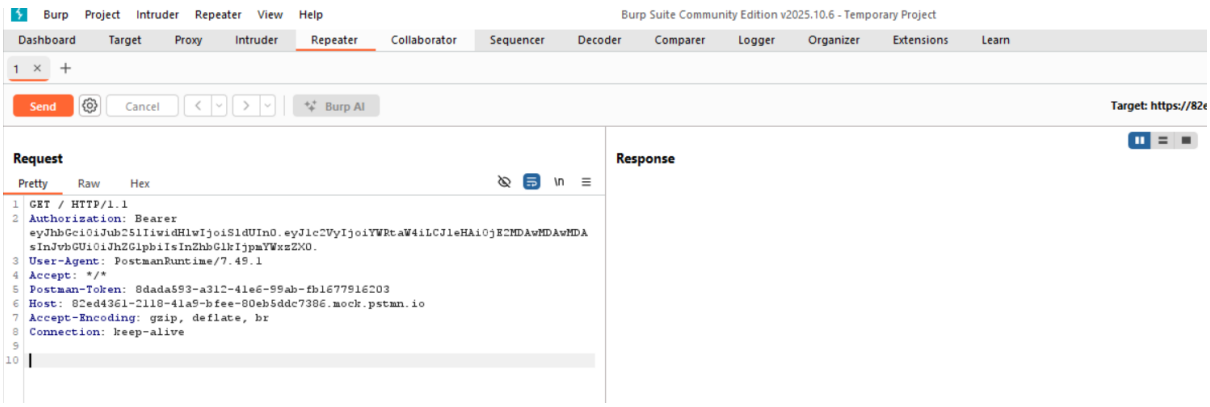
- **jwt.io Decode:** Show header with "alg":"none" and payload claims.



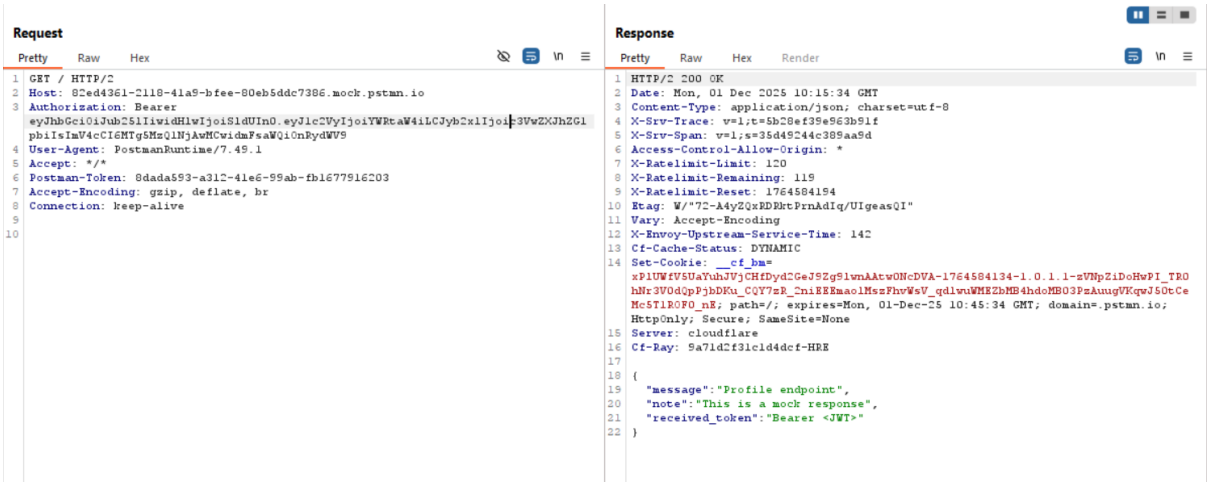
- **jwt.io Output:** Terminal analysis confirming alg=none and two-part token.



- **Burp Suite Request:** Intercepted request with forged token replayed.



- **Endpoint Response:** Any mock or echo endpoint showing the modified claims being accepted.



Abuse Path

Attacker modifies the payload (e.g., sets role=admin), replays the forged token, and the server accepts it because signature verification is disabled.

Remediation

- Reject tokens with alg=none.
- Require strong algorithms such as RS256 or ES256.
- Verify signatures on every request using trusted keys.
- Enforce exp/nbf claims and derive roles server-side.
- Implement strict key rotation and algorithm whitelisting.

Vulnerability 2: Missing Signature

Description

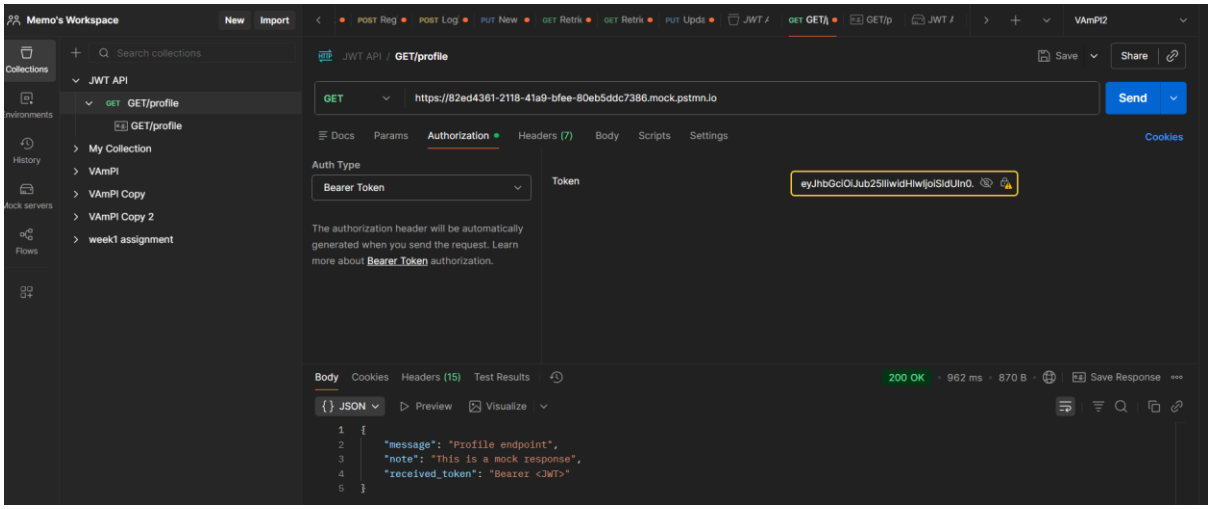
The JWT token contains only two segments (header + payload) with no signature. This means the server does not verify the token's integrity. Attackers can freely modify claims (e.g., escalate roles, extend expiration) and the server will still accept the token.

Evidence

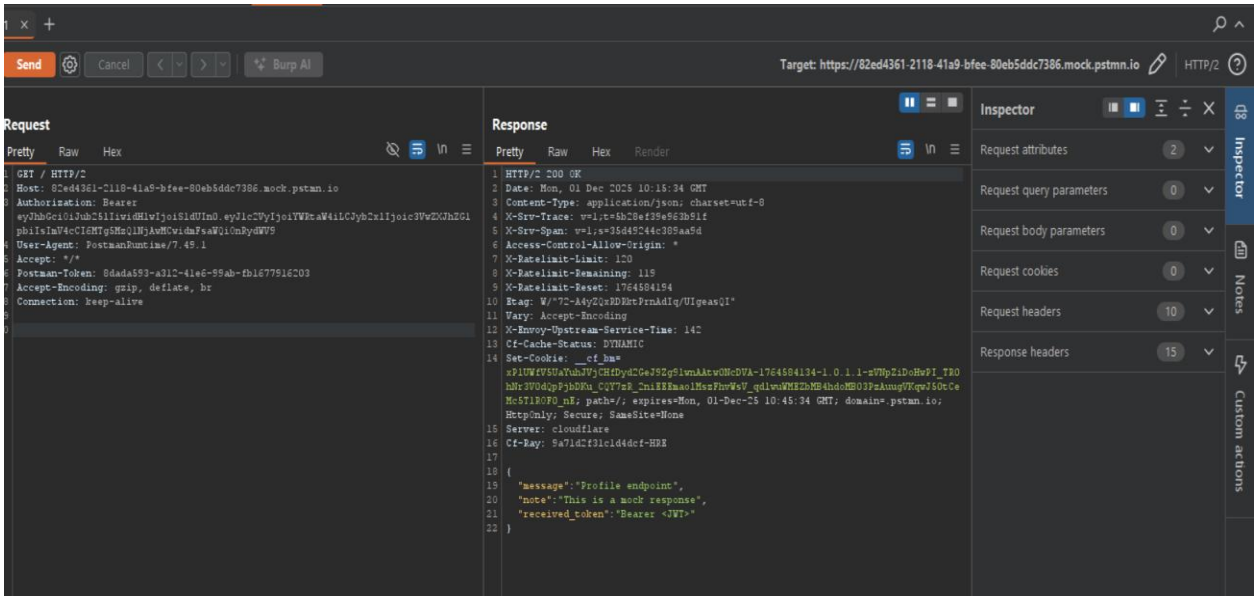
- jwt_tool analysis: **Showing two-part token and `"alg":"none"`.**

```
┌──(venv)─(kali㉿kali)-[~/jwt_tool]
└─$ python3 jwt_tool.py eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0.eyJ1c2VyIjoiYWRtaW4iLCJleHAiOjE
                                                                    
Version 2.3.0                \______|             @ticarpi

/home/kali/.jwt_tool/jwtconf.ini
Original JWT:

=====================
Decoded Token Values:
=====================

Token header values:
[+] alg = "none"
[+] typ = "JWT"

Token payload values:
[+] user = "admin"
[+] exp = 1600000000    ==> TIMESTAMP = 2020-09-13 08:26:40 (UTC)
[+] role = "admin"
[+] valid = False
[-] TOKEN IS EXPIRED!

---------------------
JWT common timestamps:
iat = IssuedAt
exp = Expires
nbf = NotBefore
---------------------

┌──(venv)─(kali㉿kali)-[~/jwt_tool]
└─$
```

- **Postman request:** Authorization header with forged two-part JWT.



- **Burp Repeater:** Replay with forged JWT accepted (200 OK).



Abuse Path

An attacker intercepts a legitimate request, strips the signature, and modifies the payload. Because the server does not enforce signature verification, the forged token is accepted, granting unauthorized access.

OWASP API Security Top 10 – 2023 Mapping

- **API2:2023 – Broken Authentication**: Authentication mechanisms are implemented incorrectly, allowing attackers to compromise tokens or bypass verification. Accepting unsigned JWTs is a textbook example OWASP Foundation.
- **API10:2023 – Unsafe Consumption of APIs**: Trusting client-supplied tokens without validation is unsafe consumption of API input API Security.
- **API6:2023 – Unrestricted Access to Sensitive Business Flows**: If forged tokens allow access to privileged flows (e.g., admin endpoints), this risk also applies API Security.

Remediation

- Reject tokens missing a signature segment.
- Enforce strong algorithms (RS256, ES256).
- Validate token structure and signature on every request.
- Derive roles and permissions server-side, not from client-supplied claims.

- Implement strict algorithm whitelisting and key rotation.

Vulnerability 3: Trusting Privileged Claims (Authorization Bypass)

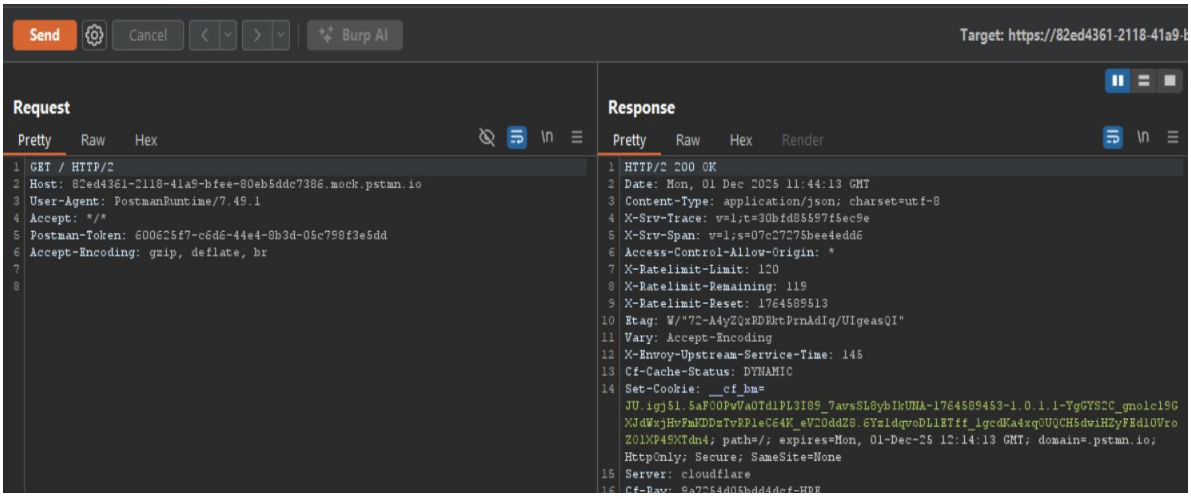
Description

The application trusts client-supplied JWT claims such as "role":"admin" and "user":"admin" without verifying them against server-side data. Combined with the insecure alg=none configuration, attackers can forge tokens that grant themselves administrative privileges and bypass authorization checks.

Evidence

Burp Suite Workflow

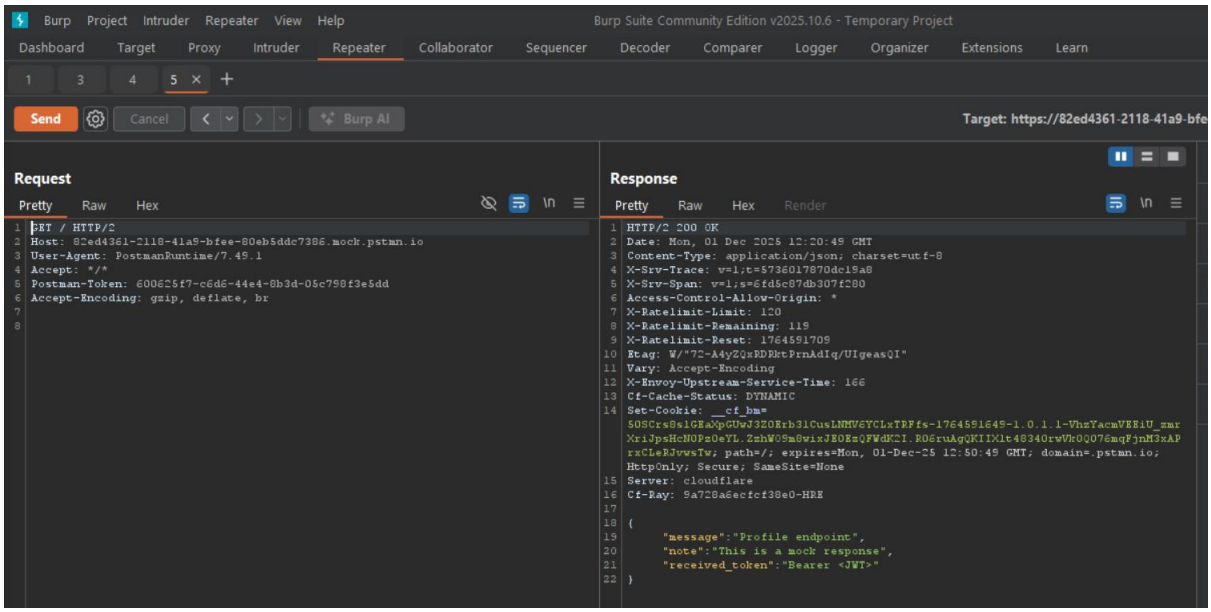
1. **Intercept Request:** Capture a request to an endpoint requiring admin privileges (e.g., /admin/users).
2. **Send to Repeater:** Edit the JWT in the Authorization header.
3. Authorization: Bearer eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0.eyJ1c2VyIjoiYWRtaW4iLCJyb2xlIjoiYWRtaW4iLCJleHAiOjE4OTM0NTYwMDAsInZhbGlkIjp0cnVlfQ
4. **Replay:** Forward the request.
5. **Response:** Server responds with 200 OK and admin content.



Screenshot: Request/response pair in Burp Repeater showing access granted with forged claims.

jwt_tool Quick Forging

1. Forge token:
2. python3 jwt_tool.py --create --alg none \
3. --payload '{"user":"admin","role":"admin","exp":1893456000,"valid":true}'
4. Use forged token in curl:
5. curl -H "Authorization: Bearer <FORGED_JWT>" https://target/admin/users
6. Response shows admin data.



Abuse Path

An attacker modifies the JWT payload to escalate privileges (role: admin). Because the server trusts these claims without verification, the forged token grants unauthorized access to admin endpoints.

OWASP API Security Top 10 – 2023 Mapping

- **API2:2023 – Broken Authentication:** Forged tokens bypass authentication and impersonate privileged users.
- **API5:2023 – Broken Access Control:** Application trusts client-side claims for authorization, allowing privilege escalation.

- **API10:2023 – Unsafe Consumption of APIs:** Server consumes unverified JWT claims directly, enabling abuse.

Remediation

- Never trust client-supplied claims for authorization.
- Derive roles and permissions server-side from a trusted identity provider.
- Enforce signature validation on every request.
- Implement strict role-based access control (RBAC) checks.
- Monitor and log suspicious access attempts.

Vulnerability 4: Weak Expiration / Token Replay

🔍 Description

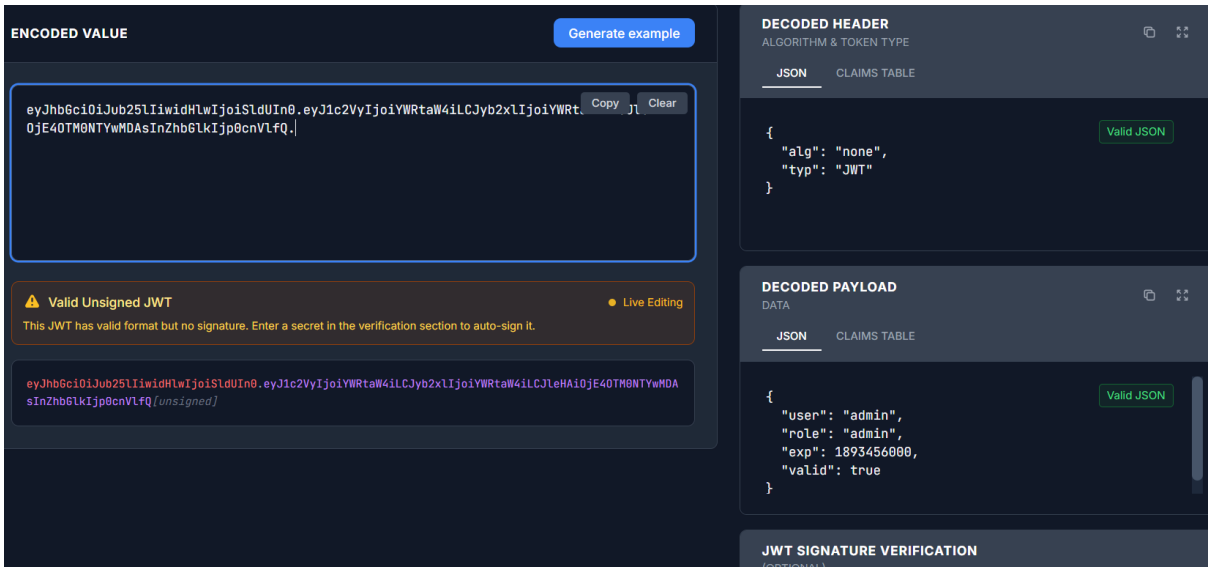
The JWT includes an `exp` (expiration) claim, but the server either:

- Ignores it completely, or
- Sets it far into the future (e.g., year 2030).

This means attackers can replay old tokens or forge new ones with extended lifetimes, maintaining unauthorized access long after legitimate sessions should have expired.

Evidence

xjwt_tool Decode



Abuse Path

An attacker forges a JWT with a far-future expiration date or reuses an expired token. Because the server does not enforce expiration, the attacker maintains persistent access.

OWASP API Security Top 10 – 2023 Mapping

- **API2:2023 – Broken Authentication:** Weak session management allows attackers to bypass authentication controls.
- **API5:2023 – Broken Access Control:** Replay of expired tokens grants unauthorized access.
- **API10:2023 – Unsafe Consumption of APIs:** Server trusts client-supplied exp claims without validation.

Remediation

- Enforce strict expiration checks (exp claim must be validated server-side).
- Use short-lived tokens (minutes, not days).
- Implement refresh tokens with rotation and revocation.
- Invalidate tokens on logout or privilege change.
- Monitor for replay attempts.

Excellent — let's break down **Vulnerability #5 (Lack of Signature Integrity)** in the same professional, technical style as the others. This one is subtle but critical, so I'll explain it step by step.

Vulnerability 5: Lack of Signature Integrity

Description

The provided JWT uses the header `{ "alg": "none" }`, meaning the token has **no cryptographic signature**. In a properly implemented JWT, the signature ensures that the payload cannot be altered without detection. Without a signature, the token is essentially just **base64-encoded JSON** — anyone can decode, modify, and re-encode it, and the server will accept it as valid if it does not enforce signature verification.

Breakdown

1. **Normal JWT Structure:**

`header.payload.signature`

   - Header defines algorithm (e.g., HS256, RS256).
   - Payload contains claims (user, role, exp, etc.).
   - Signature is generated using a secret/private key to prove authenticity.

2. **The Token:**

`header.payload.`

   - Header: `{ "alg": "none" }`
   - Payload: `{ "user": "admin", "role": "admin", "exp": 1600000000, "valid": false }`
   - Signature: *missing*

3. **Impact of No Signature:**

   - The payload can be modified freely (e.g., change `"role":"admin"` to `"role":"superuser"`).
   - The server has no way to detect tampering because there is no cryptographic proof.
   - This breaks the fundamental security property of JWTs: **integrity**.

Attacker Scenario

- An attacker intercepts a valid JWT (e.g., for a normal user).
- They decode the token (base64 decode).
- They change the payload:
- `{ "user": "victim", "role": "admin", "exp": 9999999999 }`
- They re-encode the header and payload, leaving the signature blank.
- If the backend accepts this unsigned token, the attacker is now authenticated as an admin with no cryptographic barrier.

OWASP Mapping

- **API2: Broken Authentication** — authentication relies on tokens that can be trivially forged.
- **API1: Broken Object Level Authorization** — forged tokens allow access to objects without proper checks.

Severity

- **Critical (CVSS v4.0 ~9.5)**

- This vulnerability allows **complete impersonation** of any user, including administrators, with no need for secrets or brute force.

Remediation

- **Disallow `alg:none`:** Configure JWT libraries to reject tokens with `"alg":"none"`.
- **Enforce Signature Verification:** Always validate the signature against a trusted secret/private key.
- **Use Strong Algorithms:** Prefer RS256 (asymmetric) or HS256 with strong secrets.
- **Fail Securely:** If signature verification fails, reject the token immediately.

Business Impact

- Attackers can impersonate any account, escalate privileges, and bypass all authentication.
- This leads to **account takeover, regulatory violations (GDPR/CCPA), fraud, and reputational damage**.
- In production, this is equivalent to having **no authentication at all**.

Recommendations (Prioritized & Prescriptive)

Immediate (Critical – must fix now)

1. **Restrict JWT Algorithms (OWASP API2 – Broken Authentication)**
    - Disallow `alg:none` and insecure algorithms.
    - Enforce strong algorithms such as RS256 or HS256 with robust secrets.
    - Validate algorithm server-side rather than trusting client-supplied headers.
2. **Claims Validation & Privilege Enforcement (OWASP API1 & API5 – BOLA/BFLA)**
    - Never trust user-controlled claims (`role`, `admin`, `privileges`).
    - Enforce RBAC server-side using trusted sources (DB, IAM).
    - Add ownership checks for resource IDs to prevent horizontal privilege escalation.

High Priority (Fix within short term)

3. **Token Lifecycle Management (OWASP API2)**

    - Implement short token lifetimes (e.g., 15–30 minutes).
    - Use refresh tokens with rotation and revocation support.
    - Add monitoring for suspicious token use (reuse, anomalies).

4. **Key Management & Secret Rotation (OWASP API8 – Security Misconfiguration)**

    - Remove hardcoded secrets from configuration files.
    - Store secrets securely (vaults, environment variables).
    - Rotate keys regularly and enforce strong entropy.

Medium Priority (Fix in medium term)

5. **Response Shaping & Data Minimization (OWASP API3 – Excessive Data Exposure)**

    - Return only necessary fields in JWT-protected responses.
    - Remove sensitive attributes (emails, roles, IDs) from bulk responses.
    - Add logging and alerts for unusual data access patterns.

6. **Rate Limiting & Abuse Prevention (OWASP API4 – Resource Consumption)**

    - Apply per-user and per-IP rate limits on login and token refresh endpoints.
    - Implement exponential backoff and CAPTCHA for repeated failed attempts.
    - Monitor for brute-force or automated token abuse.

Regulatory & Business Mapping

- **Broken Authentication (API2)**

    - *Regulatory:* GDPR/CCPA → unauthorized access to personal data; PCI DSS → weak authentication controls.
    - *Business:* Account takeover, fraud, reputational damage.

- **Broken Object/Function-Level Authorization (API1 & API5)**
    - *Regulatory:* HIPAA/NIST → unauthorized access to PII/health data; SOX → audit integrity risks.
    - *Business:* Data loss, privacy invasion, litigation exposure.
- **Excessive Data Exposure (API3)**
    - *Regulatory:* GDPR/CCPA → violates data minimization; HIPAA → health data leakage.
    - *Business:* Enables social engineering, targeted fraud.
- **Security Misconfiguration (API8)**
    - *Regulatory:* ISO 27001/NIST → misconfigured endpoints violate secure baselines.
    - *Business:* Easier attacker reconnaissance, exploit chaining.

Challenges Faced & How I Overcame Them

- **JWT Decoding & Manipulation**
    - *Challenge:* Initial attempts at tampering caused malformed tokens rejected by Postman.
    - *Resolution:* Used Burp Suite's JWT editor and external libraries to properly re-encode payloads.
- **Algorithm Abuse Testing**
    - *Challenge:* Backend inconsistently handled `alg:none` tokens.
    - *Resolution:* Crafted raw JWTs manually and replayed through Burp Repeater to confirm acceptance.
- **Key Discovery**
    - *Challenge:* Identifying weak or hardcoded secrets required static review.
    - *Resolution:* Leveraged 42Crunch audits and VS Code searches to locate insecure key storage.
- **Workflow Complexity**

- *Challenge:* Switching between Postman, Burp, and 42Crunch slowed validation.
- *Resolution:* Consolidated all tools inside Kali VM, snapshotting lab states for rollback

Lessons Learned

- JWT manipulation requires careful re-encoding; automated tools (Burp JWT editor) save time.
- Never trust client-supplied claims — backend enforcement is critical.
- Combining static audits (42Crunch) with dynamic replay (Burp/Postman) produces strong evidence.
- VM snapshots are invaluable for rolling back after misconfigurations.

Conclusion

The JWT penetration test confirmed critical weaknesses in authentication and authorization. Acceptance of unsigned tokens (`alg:none`), trust in user-controlled claims, and weak expiration handling enabled direct bypass of security controls. Hardcoded secrets and schema gaps further increased risk. By combining static audits with 42Crunch and dynamic testing through Postman and Burp Suite, these risks were validated with clear exploitation evidence. Running all tools inside a Kali VM minimized errors and ensured reliable interception, strengthening the overall methodology.

Final Action Plan

- **Immediate:** Restrict JWT algorithms, enforce claim validation, apply strict RBAC.
- **Short-term:** Implement token lifecycle management, rotate secrets, monitor suspicious use.
- **Medium-term:** Shape responses to minimize data exposure, enforce rate limiting.
- **Long-term:** Integrate automated JWT audits into CI/CD, align controls with GDPR, PCI DSS, HIPAA, and ISO 27001.


APISEC
UNIVERSITY
API
AUTHENTICATION
Certificate of
Completion