JWT authentication bypass via jku header injection

Executive Summary

This report describes the discovery of a critical security vulnerability within the target application's JWT-based authentication system. The application supports the `jku` (JSON Web Key Set URL) header parameter, but fails to validate the origin of the provided URL. This flaw allows attackers to forge valid JWTs and impersonate privileged users, leading to full administrative access. The vulnerability severity is **high** due to the potential for complete account compromise.

Introduction

This penetration test was conducted to evaluate the security posture of the application's authentication mechanisms, particularly its implementation of JSON Web Tokens (JWT). The focus was on assessing the validation process for public key references via the `jku` header.

Methodology

1. Used Burp Suite along with the JWT Editor extension to manipulate and inspect JWTs.

2. Logged in with valid user credentials to capture a legitimate session token.

3. Generated a custom RSA key pair and hosted the corresponding public key as a JWK on an attacker-controlled server.

4. Forged a JWT by injecting a `jku` header pointing to the hosted key and re-signing the token as the admin user.

5. Verified access to protected admin resources and executed privileged actions.

Vulnerability Findings

- **Type**: JWT Authentication Bypass via jku Header Injection

- **Location**: JWT header (`jku` parameter)

- **Severity**: High

Kama Ganah

Description

The application supported public key retrieval through the `jku` header in JWTs, but does not validate the origin of the URL. This allowed attackers to host their own JSON Web Key Set (JWKS) and craft a JWT header that references it. By re-signing a forged token using their own private key, attackers can gain unauthorized access as any user, including administrators.

Proof of Concept (PoC)

1. Logged in as user and captured a valid JWT from a `GET /my-account` request.

2. Generated a custom RSA key using Burp's JWT Editor.

3. Hosted the public key as a JWKS on the exploit server:

   json

   CopyEdit

   `{ "keys": [{ "kty": "RSA", "e": "AQAB", "kid": "custom-kid-id", "n": "custom-modulus-data" }] }`

4. Modified the JWT header to include:

   json

   CopyEdit

   `{ "alg": "RS256", "jku": "https://attacker.exploit-server.net/keys.json", "kid": "custom-kid-id" }`

5. Changed the payload to impersonate the administrator:

   json

   CopyEdit

   `{ "sub": "administrator" }`

6. Re-signed the JWT using the corresponding private key.

7. Sent a request to `GET /admin` with the forged token and accessed the admin panel.

8. Performed a `GET /admin/delete?username=user` to successfully delete the target user.

Kama Ganah

Impact Assessment

This vulnerability enabled unauthorized access to any account, including administrative roles. Exploitation could result in:

- Privilege escalation

- Sensitive data exposure

- Destructive actions such as account deletion

- Full application compromise

If deployed in a production environment, this issue could lead to serious reputational and financial damages.

Recommendations

1. Reject untrusted `jku` headers unless explicitly validated against a whitelist of known domains.

2. Use local key verification instead of fetching public keys from external URLs.

3. Ensure strict validation of all JWT header parameters and digital signatures.

4. Regularly audit JWT handling code and adopt secure cryptographic libraries.

5. Implement monitoring and alerting for abnormal JWT use patterns.

Conclusion

The discovery of an insecure `jku` header implementation highlighted a critical gap in the authentication architecture. By allowing the loading of arbitrary keys, the application exposed itself to severe risks of privilege escalation and full account compromise. Immediate remediation and enhanced JWT validation controls are strongly recommended.

Kama Ganah