JWT Misconfiguration & Abuse Detector in Web Application

Nivetheni B S
B.Sc Digital & Cyber Forensics Science
Rathinam College of Arts and Science, Coimbatore, India

Abstract

This project presents a Python-based command-line tool designed to evaluate the security of JSON Web Tokens (JWTs) signed using the HS256 algorithm. JWTs are widely used for authentication and authorization in modern

web applications, but improper key management or weak secrets can lead to critical security vulnerabilities.

The tool provides a structured framework for analyzing token components, detecting weak secrets, and demonstrating potential exploitation scenarios in a controlled environment.

Token Analysis:

The system begins by decoding and parsing the JWT into its header, payload, and signature components. Base64 URL decoding and JSON parsing are applied to extract claims and cryptographic metadata for inspection.

This allows security analysts to identify potentially sensitive information, token expiration details, and current authorization levels, providing a foundation for further testing and manipulation.

Brute Force Engine:

At its core, the tool implements a multithreaded dictionary-based brute force engine targeting the HMAC secret.

Each candidate secret from the user-supplied wordlist is used to recompute the signature and compared against the original token in a secure, constant-time manner. This approach highlights the risk posed by weak or predictable secrets, demonstrating how attackers could recover signing keys and compromise token integrity.

Token Forgery:

Upon successful discovery of the secret, the tool allows for controlled modification of payload claims to simulate privilege escalation or injection of custom attributes. The modified payload is re-encoded and signed with the discovered secret, generating a valid forged JWT. This demonstrates the practical impact of weak key management and the potential for unauthorized access when token validation is insufficient.

Performance and Usability:

The tool is designed with multithreading support to improve brute force efficiency, verbose logging for detailed inspection, and optional output file generation for saving forged tokens. Its modular structure separates decoding, brute forcing, and forging functions, enhancing maintainability and potential future extensions to cover additional JWT attack vectors.

Limitations and Security Implications:

While effective against weak HS256 secrets, the tool does not address advanced attack vectors such as algorithm confusion, asymmetric key misuse, or header-based vulnerabilities like kid manipulation. Nonetheless, it

provides a clear demonstration of the consequences of insecure token handling, emphasizing the importance of strong cryptographic keys, secure secret management, and comprehensive server-side.

1. INTRODUCTION

Modern web applications widely use JSON Web Tokens (JWT) for authentication and authorization because they enable stateless session management and improve scalability. JWTs store user information such as identity, roles, and permissions within the token, allowing servers to validate requests without maintaining session data.While this improves performance, improper implementation of JWT security can introduce critical vulnerabilities.

A common issue occurs when applications use the HS256 algorithm with weak or predictable secret keys.Attackers can perform brute-force or dictionary attacks to discover the signing secret. Once the secret is known, they can modify the token payload, change user privileges, and generate valid forged tokens. This breaks the authentication mechanism and can lead to authentication bypass, privilege escalation, and unauthorized access.

This project demonstrates these vulnerabilities by developing a JWT HS256 brute-force and token forging tool. The tool decodes tokens, recovers weak secrets, allows payload modification, and re-signs tokens to create valid forged JWTs. It simulates real-world attacker behavior and highlights the risks of insecure JWT implementations.

The project also emphasizes secure practices such as using strong secret keys and proper validation. It aligns with OWASP Top 10 risks, particularly Identification and Authentication Failures and Cryptographic Failures, showing the importance of secure design in modern applications.

2. LIMITATIONS OF EXISTING SYSTEM

Algorithm Limitation

- The tool only supports HS256 and cannot handle asymmetric algorithms such as RS256 or ES256, which are widely used in production environments.
- Dependence on Weak Secrets
- The effectiveness of the brute-force attack is entirely dependent on the strength of the JWT secret and the quality of the wordlist. Strong or randomly generated secrets cannot be cracked using this approach.

Performance Constraints

The tool relies on CPU-based multithreading and lacks GPU acceleration, making it significantly slower and less efficient compared to industry-standard tools like hashcat for large-scale attacks.

Limited Exploitation Capability

The payload manipulation is generic and does not consider application-specific authorization logic or critical JWT claims (e.g., exp, aud, iss), which may result in forged tokens being rejected.

3. PROPOSED SYSTEM

The proposed system is a command-line security testing tool aimed at evaluating vulnerabilities in JWT (JSON Web Token) implementations utilizing the HS256 signing algorithm. It identifies weak secret keys and allows controlled exploitation to test authentication and authorization weaknesses in web applications. The tool accepts a JWT token and a wordlist as input, decodes the token, and conducts a multithreaded brute-force attack to recover the signing secret. Once identified, it enables modification of payload claims and generates a new signed token. Additionally, it supports custom claim injection

to simulate privilege escalation scenarios, assessing the effectiveness of access control mechanisms. Lightweight and easy to integrate into penetration testing workflows, it can be used with interception tools to validate forged tokens. The system is intended for controlled and ethical testing environments to help identify insecure JWT implementations and weak secret management practices.

Targeted JWT Security Assessment

The tool is specifically designed to assess JWT implementations that use the HS256 algorithm. This focused approach allows penetration testers to quickly identify weak or misconfigured secret keys without dealing with unnecessary features, making the testing process more efficient in scenarios where symmetric signing is used.

Automated Brute-Force Capability

The tool automates the process of testing large numbers of potential secrets using a multithreaded approach. This significantly reduces the time and manual effort required compared to testing secrets individually, enabling faster identification of weak keys during security assessments.

Integrated Token Forging

Once the correct secret key is identified, the tool allows immediate modification of the JWT payload and re-signing of the token. This enables testers to validate real-world attack scenarios such as privilege escalation and unauthorized access without relying on external tools.

Custom Claim Injection Support

The tool provides flexibility to modify or inject custom claims into the JWT payload. This allows testers to simulate different user roles, permissions, or attributes, helping in identifying weaknesses in authorization logic and access control mechanisms.

Lightweight and Easy Integration

The command-line interface ensures that the tool is lightweight and does not require complex setup or dependencies. It can be easily integrated into existing penetration testing workflows and used alongside tools like Burp Suite for manual or semi-automated testing.

Useful for Learning and Controlled Testing

The tool is suitable for educational purposes and lab environments, allowing beginners to understand how JWT-based attacks such as brute-force and token forgery work. It provides a hands-on approach to learning common vulnerabilities in authentication mechanisms.

4. METHODOLOGY

The tool follows a structured approach to identify and exploit weaknesses in JWT implementations using the HS256 algorithm:

Token Acquisition

The JWT is obtained from the target application through interception mechanisms such as HTTP requests, cookies, or API responses during normal application interaction.

Token Decoding and Analysis

The token is decoded to extract the header and payload. The algorithm (alg) field is verified to confirm that HS256 is being used, and the payload is analyzed to understand user roles, permissions, and other relevant claims.

Wordlist Preparation

A suitable wordlist containing potential secret keys is provided as input. The effectiveness of the attack depends on the quality and relevance of this wordlist.

Brute-Force Attack Execution

The tool performs a multithreaded brute-force attack by generating HMAC-SHA256 signatures using each candidate secret and comparing them with the original token signature to identify a match.

Secret Key Identification

When a matching signature is found, the corresponding secret key is identified as the valid signing key used by the application.

Payload Manipulation

The original payload is modified by injecting or altering claims (e.g., role escalation or privilege flags) to simulate unauthorized access scenarios.

Token Re-signing and Generation

A new JWT is generated by signing the modified payload with the discovered secret key, producing a valid forged token.

Validation and Exploitation

The forged token is used in the target application to verify whether authorization controls can be bypassed and to assess the impact of the vulnerability.

5. RESULTS AND ANALYSIS

1. The tool was tested in a controlled environment using JWT tokens signed with the HS256 algorithm. The results demonstrate that the tool is effective in identifying weak secret keys when predictable or commonly used values are present in the provided wordlist.

2. During testing, the brute-force module successfully recovered the signing secret when weak secrets (e.g., short strings or commonly used passwords) were used. The time required to identify the secret varied depending on the size and quality of the wordlist, as well as the system's processing capability.

3. Once the secret key was identified, the tool successfully generated forged tokens by modifying payload claims such as user roles and privilege indicators. These forged tokens were accepted by the target application in cases where proper validation and authorization checks were not enforced, demonstrating the risk of insecure JWT implementations.

4. However, the analysis also showed that the tool is ineffective against tokens signed with strong, randomly generated secrets. In such cases, the brute-force attack did not succeed within a reasonable time frame. Additionally, tokens with strict validation of claims such as expiration time (exp) and issuer (iss) limited the success of forged tokens.

5. Overall, the results highlight that the tool is useful for identifying weak secret management practices and basic authorization flaws, but its effectiveness is limited to specific misconfigurations and does not extend to secure, properly implemented JWT systems.

6. CONCLUSION

The developed tool demonstrates how weak secret management in JWT implementations using the HS256 algorithm can lead to serious security vulnerabilities. By successfully performing brute-force attacks and generating forged tokens, the system highlights the risk of using predictable or poorly protected secret keys in authentication mechanisms. The results confirm that applications relying on weak secrets are vulnerable to token forgery, which can lead to unauthorized access and privilege escalation. This emphasizes the importance of implementing strong, randomly generated secrets and enforcing proper validation of JWT claims.

However, the tool also reveals its own limitations, as it is ineffective against securely implemented systems that use strong cryptographic practices or asymmetric signing algorithms. This reinforces the need for developers to adopt secure configurations rather than relying solely on the secrecy of keys.In conclusion, the tool serves as a practical demonstration of common JWT-related vulnerabilities and provides value in controlled testing environments. It also underlines the importance of following secure development practices to prevent exploitation of authentication and authorization mechanisms.

7. FUTURE WORK

Future improvements may include:

- Integration with multiple systems
- Advanced anomaly detection
- User-friendly interface
- Real-time monitoring enhancements

REFERENCES

[1] NIST Cybersecurity Framework

[2] CIS Critical Security Controls

[3] OWASP Top 10

[4] Stallings, W., Network Security Essentials

[5] Bishop, M., Computer Security Principles