**JSON Web Tokens Attack**

**Khái niệm:**

JSON web tokens là 1 định dạng được chuẩn hóa để gửi dữ liệu JSON giữa các hệ thống. Chúng có thể chứa bất cứ loại dữ liệu nào nhưng được dùng phổ biến nhất trong xác thực, xử lí phiên và kiểm soát truy cập.

Không như mã session cũ, tất cả dữ liệu mà máy chủ cần lưu trữ ở phía máy khách trong chính JWT. Nó làm cho JWT là 1 lựa chọn phổ biến cho những trang web có tính phân tán cao khi người dùng cần tương tác liền mạch với nhiều máy chủ.

**Định dạng JWT :**

1 JWT chứa 3 phần: header, payload và 1 signature. Mỗi phần được tách nhau bởi dấu chấm.

eyJraWQiOiI5MTM2ZGRiMy1jYjBhLTRhMTktYTA3ZS1lYWRmNWE0NGM4YjUiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJwb3J0c3dpZ2dlciIsImV4cCI6MTY0ODAzNzE2NCwibmFtZSI6IkNhcmxvcyBNb250b3lhIiwic3ViIjoiY2FybG9zIiwicm9sZSI6ImJsb2dfYXV0aG9yIiwiZW1haWwiOiJjYXJsb3NAY2FybG9zLW1vbnRveWEubmV0IiwiaWF0IjoxNTE2MjM5MDIyfQ.SYZBPIBg2CRjXAJ8vCER0LA\_ENjII1JakvNQoP-Hw6GG1zfl4JyngsZReIfqRvIAEi5L4HV0q7\_9qGhQZvy9ZdxEJbwTxRs\_6Lb-fZTDpW6lKYNdMyjw45\_alSCZ1fypsMWz\_2mTpQzil0lOtps5Ei\_z7mM7M8gCwe\_AGpI53JxduQOaB5HkT5gVrv9cKu9CsW5MS6ZbqYXpGyOG5ehoxqm8DL5tFYaW3lB50ELxi0KsuTKEbD0t5BCl0aCR2MBJWAbN-xeLwEenaqBiwPVvKixYleeDQiBEIylFdNNIMviKRgXiYuAvMziVPbwSgkZVHeEdF5MQP1Oe2Spac-6IfA

Header và payload của JWT là những đối tượng JSON được mã hóa base64. Header chứa dữ liệu về chính token, trong khi payload chứa dữ liệu về người dùng. Ví dụ, bạn có thể giải mã payload từ token trên để nhận được thông tin sau:

{ "iss": "portswigger", "exp": 1648037164, "name": "Carlos Montoya", "sub": "carlos", "role": "blog\_author", "email": "carlos@carlos-montoya.net", "iat": 1516239022 }

Dữ liệu này có thể dễ dàng đọc hay điều chỉnh bởi bất cứ ai truy cập token. Do đó, bảo mật của cơ chế JWT sẽ phụ thuộc rất nhiều vào signature.

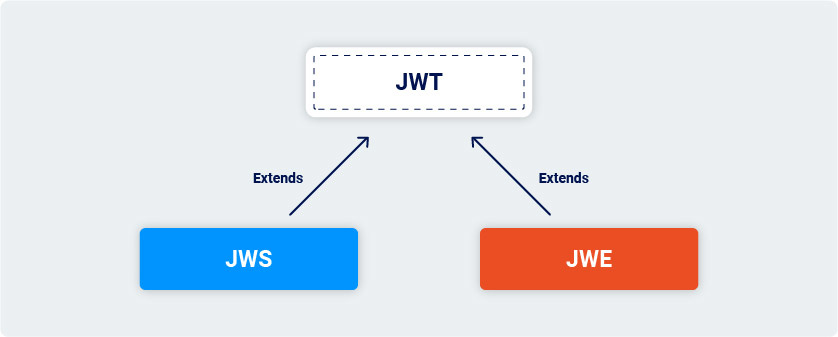
**JWT signature:**

Máy chủ mà phát hành token thường sẽ tạo ra signature bằng việc hash header và payload. Trong một vài trường hợp, kết quả đã hash còn được mã hóa thêm. Quá trình này có liên quan đến 1 chìa khóa bí mật. Cơ chế này giúp máy chủ xác thực không có dữ liệu nào trong token bị giả mạo kể từ khi nó được phát hành:

* Vì signature có nguồn gốc từ phần còn lại của token, thay đổi 1 byte của header hay payload sẽ dẫn đến signature không phù hợp.
* Không biết chìa khóa bí mật của máy chủ, không thể tạo ra 1 signature đúng cho 1 header và payload.

**JWT vs JWS vs JWE:**

Khả năng của JWT vẫn bị hạn chế. Nó chỉ là 1 định dạng để đại diện cho thông tin giống như JSON object có thể được truyền đi giữa các hệ thống. Trong thực tế, JWTs không được dùng như 1 thực thể độc lập. JWT được mở rộng bởi JSON Web Signature (JWS) và JSON Web Encrypt (JWE).



**Tấn công JWT**

Tấn công JWT là 1 ngưởi dùng gửi JWTs đã chỉnh sửa đến máy chủ để đạt được mục tiêu xấu như bỏ qua xác thực và kiểm soát truy cập bằng cách giả mạo người khác đã được xác thực.

Ảnh hưởng của tấn công JWT là rất nghiêm trọng. Nếu kẻ tấn công có thể tạo 1 token hợp lệ với giá trị bất kì, chúng có thể leo thang đặc quyền hay giả mạo ngưởi khác.

Lỗ hổng JWT sinh ra vì lỗi xử lí JWT trong ứng dụng. Các thông số kỹ thuật khác nhau liên quan đến JWT tương đối linh hoạt theo thiết kế, cho phép nhà phát triển web quyết định nhiều thực thi cho chính họ. Nó có thể dẫn đến lỗ hổng JWT mặc dù những thư viện rất cứng rắn.

Lỗi thực thi đồng nghĩa với việc signature của JWT không được xác thực phù hợp. Nó cho phép kẻ tấn công gửi các giá trị giả mạo đến ứng dụng qua payload của token. Thậm chí nếu signature được xác thực nghiêm ngặt thì nó có thực sự đáng tin cậy hay không phù thuộc vào chìa khóa bí mật của máy chủ có còn bí mật không. Nếu key bị rò rỉ hay có thể đoán và tấn công dồn dập, kẻ tấn công có thể tạo 1 signature hợp lệ cho bất cứ token nào hắn muốn.

**Khai thác lỗi xác thực của JWT signature**

Theo thiết kế, các máy chủ thường không lưu bất cứ thông tin gì về JWTs mà nó phát hành. Thay vào đó, mỗi token là 1 thực thể độc lập. Máy chủ không biết bất cứ gì về nội dung gốc của mã hay signature gốc là gì. Do đó, nếu máy chủ không xác thực token phù hợp, không gì có thể ngăn kẻ tấn công thay đổi phần còn lại của token.

Ví dụ, xem xét 1 JWT sau:

{

"username": "carlos",

"isAdmin": false

}

Nếu máy chủ xác thực phiên dựa trên username này, việc điều chỉnh giá trị của nó cho phép kẻ tấn công giả mạo những người dùng khác. Tương tự nếu giá trị của isAdmin được dùng để kiểm soát truy cập, kẻ tấn công có thể leo thang đặc quyền.

**Chấp nhận signature bất kỳ**

Các thư viện JWTs đem lại 1 phương thức để xác thực tokens và 1 cái khác chỉ để giải mã chúng. Ví dụ, thư viện Node.js jsonwebtoken có verify() và decode().

Hiếm khi, nhà phát triển nhầm lẫn hai phương thức này và chỉ gửi token đến phương thức decode(). Điều này cũng có nghĩa là ứng dụng không xác thực token.

**Chấp nhận token mà không có signature**

Trong số những thứ khác, header của JWT chứa tham số alg. Điều này cho máy chủ biết thuật toán nào đã được sử dụng để sign token và do đó thuật toán nào cần dùng khi xác thực signature.

{

"alg": "HS256",

"typ": "JWT"

}

Điều này vốn dĩ là thiết sót vì máy chủ không có lựa chọn nào ngoài việc hoàn toàn tin tưởng vào đầu vào do người dùng kiểm soát từ token mà tại thời điểm đó nó chưa được xác minh. Nói cách khác, kẻ tấn công có thể quyết định cách máy chủ kiểm tra token.

JWTs có thể được sign bằng nhiều thuật toán khác nhau nhưng cũng có thể unsign. Trong trường hợp đó, tham số alg đặt là none hay unsecured JWT. Do sự nguy hiểm rõ ràng của việc này, các máy chủ thường từ chối những token không có signature. Tuy nhiên vì kiểu lọc này dựa vào phân tích chuỗi, bạn thường có thể bỏ qua chúng bằng kĩ thuật obfuscation như viết hoa hỗn hợp và mã hóa không mong muốn.

Lưu ý: Thậm chí nếu token unsigned, payload vẫn phải được kết thúc bằng dấu chấm.

**Tấn công dồn dập secret keys**

Một vài thuật toán sign như HS256 (HMAC + SHA-256) dùng 1 chuỗi bất kỳ như secret key. Giống mật khẩu, nó không dễ đoán hay tấn công dồn dập bởi kẻ tấn công. Ngược lại, họ có thể tạo ra JWTs với bất cứ header và payload nào họ muốn, sau đó dùng key để re-sign token với 1 signature hợp lệ.

Khi ứng dụng JWT, nhà phát triển thường mắc phải sai lầm như quên thay đổi secret mặc định. Họ còn sao chép đoạn mã trên mạng mà quên thay đổi secret. Trong trường hợp đó, kẻ tấn công có thể tấn công dồn dập secret của máy chủ bằng wordlist những secret đã có.

**Tấn công dồn dập secret keys bằng hashcat**

Bạn chỉ cần 1 JWT đã sign, hợp lệ từ máy chủ và 1 danh sách những secret quan thuộc. Sau đó bạn có thể chạy câu lệnh sau, với JWT và wordlist là đối số:

hashcat -a 0 -m 16500 <jwt> <wordlist>

Hashcat sign header và payload từ JWT bằng mỗi secret trong wordlist sau đó so sánh với signature hợp lệ của máy chủ. Nếu giống nhau, hashcat in ra secret đó theo định dạng sau:

<jwt>:<identified-secret>

Vì hashcat chạy cục bộ trong máy của bạn và không gửi yêu cầu đến máy chủ, quá trình này rất nhanh khi dùng 1 wordlist lớn.

Khi bạn đã xác định được secret key, bạn có thể dùng nó để tạo 1 signature hợp lệ với bất cứ JWT header và payload nào bạn thích.

Nếu máy chủ dùng 1 secret yếu, có thể tấn công dồn dập từng kí tự hơn là 1 wordlist.

**JWT header parameter injections**

Theo thông số kỹ thuật của JWS chỉ có tham số alg là bắt buộc. Trong thực tế, header của JWT thường chứa nhiều tham số khác. Những cái sau đây được kẻ tấn công đặc biệt quan tâm:

* jwk (JSON Web Key): cung cấp 1 đối tượng JSON đại diện cho key
* jku (JSON Web Key Set Url): cung cấp 1 Url mà các máy chủ có thể fetch 1 danh sách key
* kid (Key ID): cung cấp 1 ID mà các máy chủ có thể dùng để xác thực đúng key trong những trường hợp mà có nhiều keys để chọn. Dựa vào định dạng của key, nó có thể có tham số kid.

Như bạn có thể thấy, mỗi tham số do người dùng kiểm soát này cho máy chủ nhận biết key nào để xác thực signature. Trong mục này, bạn sẽ học cách khai thác chúng để tiêm JWT đã được chỉnh sửa với key bất kỳ hơn là secret của máy chủ.

**Tiêm JWT self-signed qua tham số jwk**

Tham số header jwk mà máy chủ dùng để nhúng public key trực tiếp trong token theo định dạng của JWK

JWK là 1 định dạng chuẩn đại diện cho key như 1 đối tượng JSON.

{

"kid": "ed2Nf8sb-sD6ng0-scs5390g-fFD8sfxG",

"typ": "JWT",

"alg": "RS256",

"jwk": {

"kty": "RSA",

"e": "AQAB",

"kid": "ed2Nf8sb-sD6ng0-scs5390g-fFD8sfxG",

"n": "yy1wpYmffgXBxhAUJzHHocCuJolwDqql75ZWuCQ\_cb33K2vh9m"

}

}

Lý tưởng nhất là các máy chủ chỉ nên sử dụng 1 danh sách public key giới hạn để xác thực JWT. Tuy nhiên, những máy chủ bị cấu hình sai thường dùng bất cứ key nào được nhúng trong tham số jwk.

Bạn có thể khai thác hành vi này bằng cách sign 1 JWT đã chỉnh sửa bằng private key RSA, sau đó nhúng public key phù hợp trong jwk header.

Mặc dù bạn có thể thêm hay chỉnh sửa tham số jwk trong Burp, JWT Editor cung cấp 1 tính năng hữu ích để giúp bạn kiểm tra lỗ hổng này:

1. Vào tab JWT Editor Keys

2. Tạo 1 key RSA (nó sẽ tạo cho bạn 1 public key và 1 private key)

3. Gửi 1 yêu cầu chứa JWT đến Repeater

4. Chuyển sang JSON Web Token tab và điều chỉnh payload của token

5. Chọn Attack sau đó chọn Embedded JWK. Trong dialog, chọn RSA key mới tạo

(Nó sẽ làm hai bước: sign JWT bằng private key; thêm tham số jwk là public key)

6. Gửi yêu cầu để kiểm tra phản hồi

Bạn cũng có thể làm thủ công bằng việc thêm header jwk. Tuy nhiên, bạn cũng cần chỉnh sửa tham số kid của JWT để phù hợp với kid của key đã nhúng. Tấn công bằng extension sẽ giúp bạn làm điều này.

**Tiêm JWT self-signed qua tham số jku**

Thay vì nhúng public key trực tiếp trong tham số jwk, một vài máy chủ cho phép bạn dùng tham số jku để tham khảo danh sách các key. Khi xác thực signature, máy chủ fetch key tương ứng từ url đó.

JWK set

Là 1 đối tượng JSON chứa 1 mảng JWKs đại diện cho nhiều key khác nhau. Bạn có thể xem ví dụ sau:

{

"keys": [

{

"kty": "RSA",

"e": "AQAB",

"kid": "75d0ef47-af89-47a9-9061-7c02a610d5ab",

"n": "o-yy1wpYmffgXBxhAUJzHHocCuJolwDqql75ZWuCQ\_cb33K2vh9mk6GPM9gNN4Y\_qTVX67WhsN3JvaFYw-fhvsWQ"

},

{

"kty": "RSA",

"e": "AQAB",

"kid": "d8fDFo-fS9-faS14a9-ASf99sa-7c1Ad5abA",

"n": "fc3f-yy1wpYmffgXBxhAUJzHql79gNNQ\_cb33HocCuJolwDqmk6GPM4Y\_qTVX67WhsN3JvaFYw-dfg6DH-asAScw"

}

]

}

JWK sets thường được public qua 1 endpoint như /.well-known/jwks.json

Những trang web bảo mật hơn sẽ chỉ fetch keys từ những tên miền đáng tin cậy, nhưng bạn có thể tận dụng sự khác biệt trong Url parsing để bỏ qua bộ lọc này (SSRF) .

**Tiêm JWT self-signed qua tham số kid**

Các máy chủ có thể dùng một số key cryptographic để sign các loại dữ liệu khác nhau không chỉ JWT. Vì lí do đó, header của 1 JWT có thể chứa tham số kid giúp máy chủ xác định key để xác thực signature.

Những keys xác thực thường được lưu trữ như JWK set. Trong trường hợp đó, máy chủ có thể tìm kiếm JWK giống với kid của token. Tuy nhiên, thông số kĩ thuật của JWS không xác định cấu trúc cụ thể cho ID này - nó chỉ là 1 chuỗi tùy ý do nhà phát triển lựa chọn. Ví dụ, họ có thể dùng tham số kid để trỏ đến cơ sở dữ liệu hay tên tập tin.

Nếu tham số này bị dính lỗ hổng Path Traversal, kẻ tấn công có thể buộc máy chủ dùng 1 tập tin bất kỳ từ hệ thống của nó để xác thực keys.

{

"kid": "../../path/to/file",

"typ": "JWT",

"alg": "HS256",

"k": "asGsADas3421-dfh9DGN-AFDFDbasfd8-anfjkvc"

}

Điều này đặc biệt nguy hiểm nếu máy chủ hỗ trợ JWTs được sign bằng thuật toán đối xứng. Trong trường hợp đó, kẻ tấn công có thể trỏ tham số kid đến 1 tập tin tĩnh, có thể dự đoán được sau đó sign JWT bằng 1 secret mà phù hợp với nội dung của file.

Về lý thuyết bạn có thể làm điều này với bất kỳ tệp nào nhưng phương thức đơn giản nhất là dùng /dev/null có trên đa số hệ thống Linux. Vì nó là 1 tệp rỗng, đọc nó sẽ trả về giá trị rỗng. Do đó, sign 1 token với chuỗi rỗng sẽ dẫn đến 1 signature hợp lệ.

Nếu bạn dùng JWT Editor extension, nó không cho bạn sign 1 token bằng chuỗi rỗng. Vì 1 bug trong extension, bạn có thể giải quyết vấn đề này bằng cách sử dụng byte rỗng được mã hóa Base64.

Nếu máy chủ lưu trữ các key xác thực trong cơ sở dữ liệu, tham số kid có thể bị tấn công SQL Injection.

**Một số tham số JWT header khác**

Những tham số header sau có thể là mục tiêu của kẻ tấn công:

cty (Content Type) : thường dùng để xác định loại truyền thông cho nội dung trong JWT payload. Nó thường bị bỏ qua trong header, nhưng một số thư viện có thể vẫn hỗ trợ. Nếu bạn đã tìm được cách để bỏ qua xác thực signature, bạn có thể thử tiêm cty để thay đổi content type sang text/xml hay application/x-java-serialized-object để thực hiện tấn công XXE hay deserialization.

x5c (X.509 Certificate Chain) : thường dùng để gửi public key X.509 hay 1 chuỗi key để sign JWT. Tham số này có thể tiêm self-signed như jwk. Do sự phức tạp của định dạng X.509 và extension của nó, việc phân tích cú pháp các chứng chỉ này có thể gây ra nhiều lỗ hổng. Chi tiết các cuộc tấn công này nằm ngoài phạm vi của bài học nhưng để biết thêm chi tiết hãy xem CVE-2017-2800 và CVE-2018-2633.

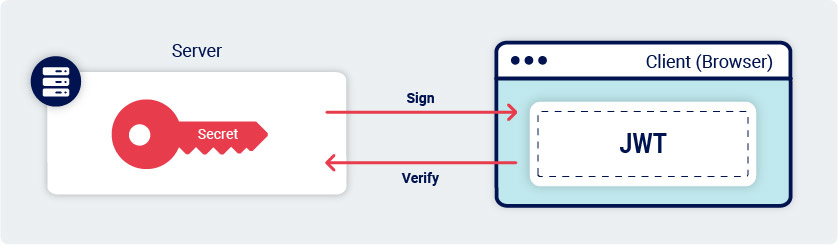
**JWT algorithm confusion**

Thậm chí nếu máy chủ dùng secret key mạnh mẽ mà bạn không thể tấn công dồn dập, bạn vẫn có thể làm giả JWTs hợp lệ bằng cách sign 1 token dùng 1 thuật toán mà nhà phát triển không lường trước được. Nó được biết đến như algorithm confusion attack.

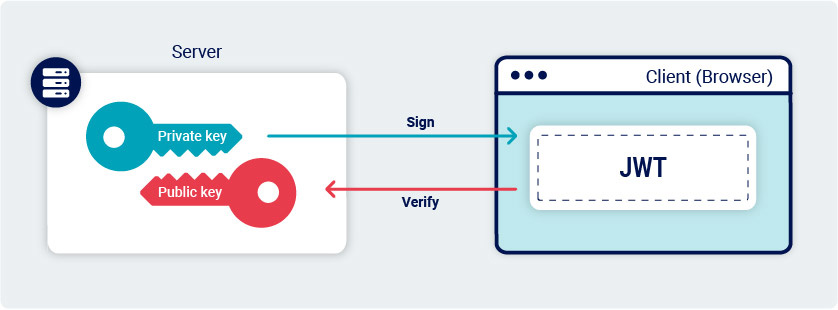
Các cuộc tấn công nhầm lẫn thuật toán hay key confusion attack xảy ra khi 1 kẻ tấn công có thể lừa máy chủ xác thực signature của JWT bằng 1 thuật toán khác với mong muốn của nhà phát triển. Nếu trường hợp này không được xử lí đúng cách, nó cho phép kẻ tấn công làm giả JWT hợp lệ chứa giá trị bất kỳ mà không cần biết secret key của máy chủ.

**Thuật toán đối xứng và bất đối xứng**

JWTs có thể được sign bằng nhiều thuật toán khác nhau. Một vài trong số đó như HS256 (HMAC + SHA256) dùng 1 key đối xứng. Điều đó có nghĩa là máy chủ dùng 1 key duy nhất để vừa sign và xác thực token. Rõ ràng, nó cần được giữ bí mật như mật khẩu.



Thuật toán khác như RS256 (RSA + SHA256) dùng 1 cặp key bất đối xứng. Nó chứa 1 private key mà máy chủ dùng để sign token và 1 public key có thể dùng để xác thực signature.



Private key bắt buộc giữ bí mật nhưng public key thường chia sẻ để mà bất cứ ai cũng có thể xác thực signature của token.

**Lỗ hổng rối loạn thuật toán sinh ra như thế nào?**

Lỗ hổng rối loạn thuật toán sinh ra do lỗi thực thi của thư viện JWT. Quá trình xác thực khác nhau phụ thuộc vào thuật toán mình dùng, nhưng rất nhiều thư viện cung cấp 1 phương pháp duy nhất, không dựa vào thuật toán để xác thực signature. Những phương pháp này dựa trên tham số alg trong header để xác định loại xác thực chúng thực hiện.

Mã giả sau đây cho thấy 1 ví dụ đơn giản về cách khai báo cho phương thức verify() có thể trông như thế nào trong thư viện JWT:

function verify(token, secretOrPublicKey){

algorithm = token.getAlgHeader();

if(algorithm == "RS256"){

// Use the provided key as an RSA public key

} else if (algorithm == "HS256"){

// Use the provided key as an HMAC secret key

}

}

Vấn đề sinh ra khi nhà phát triển web giả sử phương thức verify() xử lí JWT đã sign bằng 1 thuật toán bất đối xứng là RS256. Do giả định sai, họ có thể gửi 1 public key đã chỉnh sửa đến 1 phương thức như sau:

publicKey = <public-key-of-server>;

token = request.getCookie("session");

verify(token, publicKey);

Trong trường hợp đó, nếu máy chủ nhận 1 token được sign bằng 1 thuật toán đối xứng như HS256, phương thức verify sẽ xử lí public key như 1 HMAC secret. Điều này có nghĩa là kẻ tấn công có thể sign token bằng HS256 và public key; máy chủ sẽ dùng chính public key ấy để xác thực signature.

Lưu ý: public key mà bạn dùng để sign token phải hoàn toàn giống với public key được lưu trữ trên máy chủ. Nó phải chung định dạng (như X.509 PEM) và giữ nguyên bất cứ kí tự nào không in được như \n. Trong thực tế, bạn cần thử nhiều định dạng khác nhau để cuộc tấn công này hoạt động.

**Thực hiện 1 cuộc tấn công rối loạn thuật toán**

1 cuộc tấn công rối loạn thuật toán bao gồm những bước sau:

1. **Lấy được public key của máy chủ**

Các máy chủ đôi khi hiển thị public key của chúng dưới dạng các đối tượng JSON Web Key được ánh xạ qua /jwks.json hay /.well-known/jwks.json. Chúng có thể được lưu trữ trong mảng jwks gọi là keys.

{

"keys": [

{

"kty": "RSA",

"e": "AQAB",

"kid": "75d0ef47-af89-47a9-9061-7c02a610d5ab",

"n": "o-yy1wpYmffgXBxhAUJzHHocCuJolwDqql75ZWuCQ\_cb33K2vh9mk6GPM9gNN4Y\_qTVX67WhsN3JvaFYw-fhvsWQ"

},

{

"kty": "RSA",

"e": "AQAB",

"kid": "d8fDFo-fS9-faS14a9-ASf99sa-7c1Ad5abA",

"n": "fc3f-yy1wpYmffgXBxhAUJzHql79gNNQ\_cb33HocCuJolwDqmk6GPM4Y\_qTVX67WhsN3JvaFYw-dfg6DH-asAScw"

}

]

}

Thậm chí nếu key không được hiển thị public, bạn có thể trích xuất nó từ 1 cặp JWT hiện có.

1. **Chuyển đổi public key sang định dạng phù hợp**

Mặc dù máy chủ có thể hiển thị public key ở định dạng JWK, khi xác thực signature của 1 token, nó sẽ dùng bản sao của key từ hệ thống tệp hay cơ sở dữ liệu. Nó có thể được lưu trữ ở 1 định dạng khác.

Để tấn công được hoạt động, key bạn dùng để sign token phải hoàn toàn giống với cái được lưu trên máy chủ. Ngoài việc ở cùng định dạng, mỗi byte đơn lẻ phải khớp nhau bao gồm mọi ký tự không in được.

Giả sử chúng ta cần key ở định dạng X.509 PEM. Bạn có thể chuyển đổi JWK sang PEM bằng JWT Editor như sau:

* Vào tab JWT Editor Keys, chọn New RSA key.
* Trong hộp thoại, dán JWK mà bạn đã nhận được
* Chọn PEM và sao chép key PEM nhận được
* Đến Decoder và mã hóa Base64 PEM
* Quay lại JWT Editor Keys và chọn New Symmetric Key
* Trong hộp thoại, chọn Generate để tạo 1 key mới ở định dạng JWK
* Thay thế giá trị của tham số k với PEM đã được mã hóa
* Lưu key.

1. **Chỉnh sửa payload và tham số alg là HS256**

Khi bạn có public key ở định dạng phù hợp, bạn có thể điều chỉnh JWT theo cách bạn muốn. Chỉ cần đảm bảo rằng tham số alg được đặt là HS256.

1. **Sign token với HS256 bằng public key**

Sign token bằng thuật toán HS256 với RSA public key là secret.

**Lấy public key từ token hiện có**

Trong trường hợp public key không được hiển thị, bạn vẫn có thể tấn công rối loạn thuật toán bằng cách lấy key từ hai token hiện có. Quá trình này sử dụng công cụ như jwt\_forgery.py. Bạn có thể tìm thấy nó cùng với một số tập lệnh hữu ích khác trên rsa\_sign2n Github repository.

Chúng tôi cũng đã tạo ra 1 phiên bản đơn giản của nó trên docker.

docker run --rm -it portswigger/sig2n <token1> <token2>

Bạn cần docker cli để chạy nó. Đầu tiên bạn chạy câu lệnh này, nó sẽ tự động lấy image từ Docker Hub, quá trình này sẽ mất 1 vài phút.

Nó sử dụng JWTs mà bạn cung cấp để tính ra các giá trị tiềm năng của n. Và chỉ 1 trong số chúng hợp với giá trị n được dùng bởi key của máy chủ. Với mỗi giá trị n, tập lệnh in ra:

* PEM key được mã hóa Base64 ở cả hai định dạng X.509 và PKCS1.
* 1 JWT giả được sign bằng từng key này.

Để xác định key đúng, dùng Burp Repeater để gửi 1 yêu cầu có chứa JWT giả đó. Chỉ một cái được chấp nhận bới máy chủ. Sau đó bạn có thể dùng key thích hợp để tấn công rối loạn thuật toán.

Để biết thêm thông tin về quá trinh này hoạt động như thế nào hay cách sử dụng công cụ jwt\_forgery.py, hãy tham khảo tài liệu trong repository.

**Ngăn chặn cuộc tấn công JWT**

Bạn có thể ngăn chặn tấn công JWT bằng những phương pháp sau:

* Dùng 1 thư viện được cập nhật liên tục để xử lí JWTs và đảm bảo các nhà phát triển hiểu rõ về cách nó hoạt động, cùng với mọi tác động bảo mật.
* Đảm bảo rằng bạn xác thực signature nghiêm ngặt trên mọi JWTs mà bạn nhận được và tính đến các trường hợp mà JWTs dùng những thuật toán không mong muốn.
* Thực thi 1 whitelist nghiêm ngặt những máy chủ được phép cho tham số jku.
* Đảm bảo bạn không bị dính lỗ hổng path traversal hay sql injection qua tham số kid.

**Cách xử lí JWT tốt nhất**

* Thường xuyên đặt thời gian hết hạn cho bất cứ token nào bạn phát hành
* Tránh gửi token trong tham số của url khi có thể
* Bao gồm aud (audience) claim (hay tương tự) để chỉ định người nhận mã mong muốn. Điều này ngăn chặn sử dụng token trên các trang web khác.
* Cho phép máy chủ phát hành thu hồi token (ví dụ, khi đăng xuất)