THAT JWT TALK

JSON WEB TOKENS CONSIDERED HARMFUL

silent signal

András Veres-Szentkirályi Camp++ 0x7e6

$ whoami

András Veres-Szentkirályi

- CISSP, OSCP, GWAPT, SISE
- Silent Signal co-founder
- pentester, toolmaker

Fahrplan

1. High level
   - About JWTs
   - Stateless approach
2. Design issues
3. Cryptography
   - HMAC
   - RSA
   - ECDSA
4. Misc

Why now?

- JWTs are literally everywhere by now
  - PSD2 APIs
  - long-term tokens for mobile apps
- our RSA public key recovery tool from February 2022
- CVE-2022-21449: Psychic Signatures in Java from April 2022
- and we still encounter low-entropy HMAC secrets in 2022

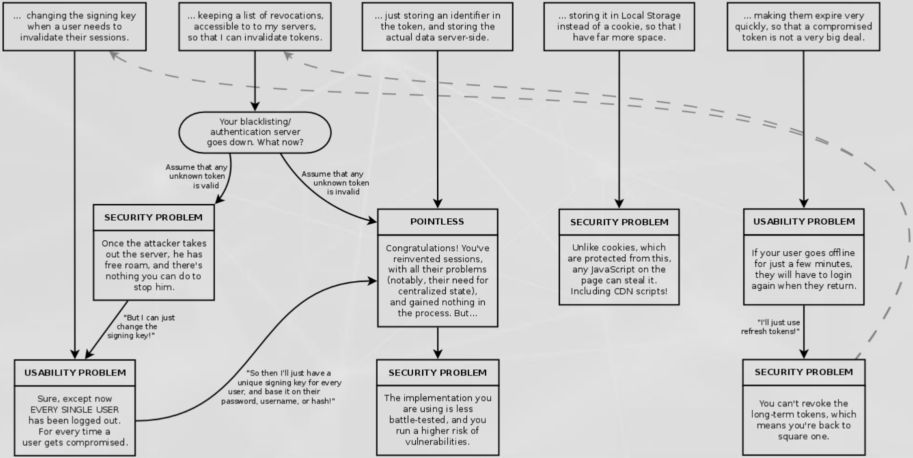
JWS

- JSON Web Signature, RFC 7515
- `BASE64URL(UTF8(JWS Protected Header)) || ’.’ || BASE64URL(JWS Payload) || ’.’ || BASE64URL(JWS Signature)`
- signature is calculated on `ASCII(BASE64URL(UTF8(JWS Protected Header)) || ’.’ || BASE64URL(JWS Payload))`
- payload might be detached, see Appendix F
  - header and signature goes into metadata such as HTTP header
  - payload is replaced with empty string
  - similar to XML signatures and WS-Security in the SOAP world

JWT

- JSON Web Token, RFC 7519
  - pronounced like the word “jot”
- builds on JWS
- payload contains set of claims
  - username
  - Unix timestamps for issuance and/or expiration
- people love using them for stateless session management
  - `http://cryto.net/~joepie91/blog/2016/06/19/stop-using-jwt-for-sessions-part-2-why-your-solution-doesnt-work/`

I can make JWT sessions work by...



... changing the signing key when a user needs to invalidate their sessions.

... keeping a list of revocations, accessible to to my servers, so that I can invalidate tokens.

... just storing an identifier in the token, and storing the actual data server-side.

... storing it in Local Storage instead of a cookie, so that I have far more space.

... making them expire very quickly, so that a compromised token is not a very big deal.

Your blacklisting/ authentication server goes down. What now?

Assume that any unknown token is valid

Assume that any unknown token is invalid

SECURITY PROBLEM

Once the attacker takes out the server, he has free roam, and there's nothing you can do to stop him.

"But I can just change the signing key!"

USABILITY PROBLEM

Sure, except now EVERY SINGLE USER has been logged out. For every time a user gets compromised.

"So then I'll just have a unique signing key for every user, and base it on their password, username, or hash!"

POINTLESS

Congratulations! You've reinvented sessions, with all their problems (notably, their need for centralized state), and gained nothing in the process. But...

SECURITY PROBLEM

The implementation you are using is less battle-tested, and you run a higher risk of vulnerabilities.

SECURITY PROBLEM

Unlike cookies, which are protected from this, any JavaScript on the page can steal it. Including CDN scripts!

USABILITY PROBLEM

If your user goes offline for just a few minutes, they will have to login again when they return.

"I'll just use refresh tokens!"

SECURITY PROBLEM

You can't revoke the long-term tokens, which means you're back to square one.

Fahrplan

1. High level
   - About JWTs
   - Stateless approach
2. Design issues
3. Cryptography
   - HMAC
   - RSA
   - ECDSA
4. Misc

Cryptographic agility

- the `alg` header offers too much flexibility
- that parameter comes from an untrusted source
- easiest and thus earliest vulnerability: set it to `none`
- parser differentials
  - WAF catches `none` (case sensitive)
  - parser accepts `nOnE` (case insensitive)
- all that assuming that the server even checks it: fail-open
  - `verify()` vs. `decode()`
  - assuming another node checked it vs. zero-trust

JWS replay attacks

- just resending a valid message can cause problem for non-idempotent things
- WS-Security used Timestamp and Nonce
- JWS/JWT has `jti` (JWT ID)
- order does matter
  - the verifier must maintain a list of “used” `jti` values until expiration
  - parsing and storing `jti` *before* verifying the signature → storage DoS

Key management

- “signing . . . is not a tooling problem, but a trust and key distribution problem” (Filippo Valsorda)
  - `https://docs.google.com/document/d/11yHom20CrsuX8KQJXBBw04s80Unjv8zCg_A7sPAX_9Y/preview`
- trusting `kid` too much can be a problem
- self-signed tokens can be created using the `jwk` and `jku` parameters

Confidentiality vs. integrity

- Base64 layer adds a false sense of confidentiality for some
- cf. HTTP Basic authentication
- JWE (JSON Web Encryption) can help with this
  - now you have $n + 1$ problems
  - invalid curve attack (2017)
  - Bleichenbacher’s attack (pre-finalized versions only)

Fahrplan

1. High level
   - About JWTs
   - Stateless approach
2. Design issues
3. Cryptography
   - HMAC
   - RSA
   - ECDSA
4. Misc

HMAC intro

- symmetric MAC
- easy to understand
- HS256 required: HMAC + SHA-256
- HS384 and HS512 optional

HMAC problems

- HMAC and the underlying SHA-2 is designed to be fast
- secret can have low entropy
- John the Ripper supports it out of the box

RSA intro

- asymmetric signatures
- can be verified with the public key
- multiple keys → `kid`
- RS256 recommended: RSASSA-PKCS-v1_5 + SHA-256
- RS384 and RS512 optional
- PSnnn variants are RSASSA-PSS using SHA-256 and MGF1

RSA problems

- verifier trusts the header regarding algorithm
- what if we replace RSA with HMAC?
  - key confusion attacks, such as CVE-2017-11424
  - will the verifier treat the RSA public key as a HMAC key?
- do we know the public key at all?
  - use-case might or might now involve publishing the public key
  - public keys being kept in secret are not a common threat model
  - `https://blog.silentsignal.eu/2021/02/08/abusing-jwt-public-keys-without-the-public-key/`

RSA public key recovery

- Although public key cryptosystems guarantee that the *private key* can't be derived from the public key, signatures, ciphertexts, etc., there are usually no such guarantees for the *public key*!
- Although RSA involves large numbers, really efficient algorithms exist to find the GCD of numbers since the ancient times (we don't have to do brute-force factoring).
- Although the presented method is probabilistic, in practice we can usually just try all possible answers. Additionally, our chances grow with the number of known message-signature pairs.
- The main lesson is: one should not rely on the secrecy of public keys, as these parameters are not protected by mathematical trapdoors.
- `https://github.com/silentsignal/rsa_sign2n`

ECDSA intro

- asymmetric signatures
- can be verified with the public key
- multiple keys → `kid`
- ES256 recommended “plus”: P-256 + SHA-256
  - compatible with iOS Secure Enclave
- ES384 (P-384) and ES512 (P-521) optional

ECDSA app-level problems

- G – elliptic curve base point, $n \times G = O$ where O is the identity element
- d_A – private key
- $Q_A = d_A \times G$ – public key
- z – leftmost bits of the hash of the message
- k – cryptographically secure random integer
- $(x_1, y_1) = k \times G$
- signature consists of $r = x_1 \bmod n$ and $s = k^{-1}(z + rd_A) \bmod n$
- if k is ever reused, private key d_A can be calculated
  - see PlayStation 3 signing key

CVE-2022-21449

- Psychic Signatures in Java
  - `https://neilmadden.blog/2022/04/19/psychic-signatures-in-java/`
- affects not only JWT but also SAML assertions, OIDC id tokens
- Java 15-18 since the C++ $\rightarrow$ Java port in 15 introduced the bug
- verification steps:
  - $u_1 = zs^{-1}$ and $u_2 = rs^{-1}$
  - $(x_1, y_1) = u_1 \times G + u_2 \times Q_A$
  - signature is valid if $r \equiv x_1 \pmod n$
- what if we allow r and s to be 0?

Fahrplan

1. High level
   - About JWTs
   - Stateless approach
2. Design issues
3. Cryptography
   - HMAC
   - RSA
   - ECDSA
4. Misc

JWT injection

- JWT might include attributes from an untrusted source
- artisanal JSON serialization: `'{"name": "' + untrusted + '", ...}'`
- some JSON parsers even accept colliding keys

Leaks

- if something is URL-safe, people will put it into the URL
- HTTP Referrer headers
- logs: HTTPd, reverse proxy, application server, forward proxy
- caches
- browser history

Handy Burp tool: JWT Editor

- `https://portswigger.net/bappstore/26aaa5ded2f74beea19e2ed8345a93dd`
- `https://github.com/PortSwigger/jwt-editor`
- detection
- verification
- editing
- signing
- encryption (JWE)
- basic attacks

PortSwigger JWT labs

- `https://portswigger.net/web-security/jwt`
- detailed explanations
- 8 live labs hosted by PortSwigger
- they link to our `rsa_sig2n` repository ;)
  - they even offer a dockerized version of it
- all the labs are free

Wrapping up

- JWS can be used securely for some purposes
- JWT should only be used with caution
- you shouldn’t pick technologies based on hype
- especially if your security depends on it
- if something has lots of knobs on it, eventually someone will use it wrong

THANKS!

ANDRÁS VERES-SZENTKIRÁLYI

vsza@silentsignal.hu

facebook.com/silentsignal.hu

@SilentSignalHU

@dn3t

silent signal