Using and abusing JWTs

Dev vs CISO

WhatTheStack 2024 Workshop

Wekoslav Stefanovski
Bozidar Spirovski

What the f* is Dev vs CISO?

Dev focus is on functionality

“dialing Accessibility to zero means you have no product at all, whereas dialing Security to zero can still get you a reasonably successful product” (Steve Yegge)

CISO focus is on security (act surprised)

Security is implicitly expected
It’s not what you pay for
It’s rarely in the happy path of testing
It’s very easy to be optimistic
It’s not important, until it’s very important (usually after your system gets hacked)

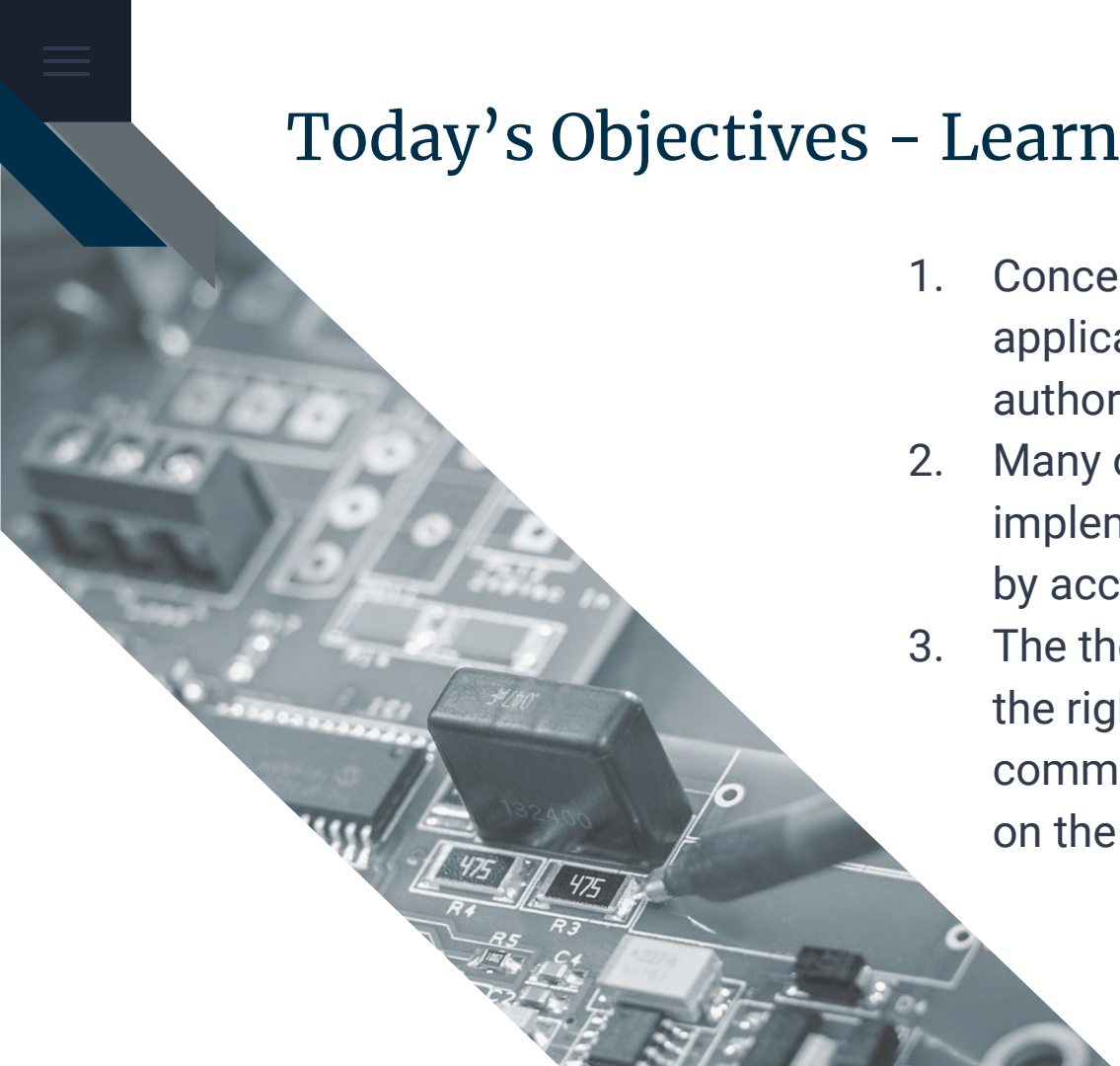
happy devs = happy users && happy management && happy regulators

What our lawyers make us say*

- All examples, concepts, and code in this workshop are for educational purposes only
- Use this knowledge to improve security on platforms you work with.
- Do not use any information or techniques from this training for illegal or harmful purposes.
- We are not responsible for any misuse of the content provided.

* if we had lawyers

Today's Objectives - Learn through doing



1. Concept of JWTs, and their application in authentication / authorization systems.
2. Many of the ways a JWT implementation can be broken (mostly by accident)
3. The thought process and tools to ask the right questions, check the common issues (before they hit you on the head)

If you must remember only ~~one~~ five things?

JWTs are not magic - You have to do the work to use them properly.

JWTs are not perfect - They have inherent deficiencies, so consider whether you need them

Don’t reinvent the wheel - when possible, build on tested and mature frameworks.

JWT bugs are outside the happy path - Most flaws remain undiscovered until you actively try to hack the system.

Never underestimate the power of stupid deadlines in large numbers

/usr/bin/whoami

- CISO of Blue dot and Sourcico
- Trying to get people to do the right thing for better part of two decades
- Big enterprise and software product companies
- Terrified of people chasing deadlines and being pressured into MVPs
- Many mistakes, incidents and a lot of stress

Email: b.spirovski@beyondmachines.net
Youtube: https://www.youtube.com/@spirovskib
LinkedIn: https://www.linkedin.com/in/spirovskibozidar/

- Head of development at Sourcico
- Coding professionally since last century
- C# / .net
- JavaScript / TypeScript
- I love programming, I love programmers

Email: swekster@gmail.com
Twitter: @swekster
Youtube: https://www.youtube.com/@swekster
Linked In: https://www.linkedin.com/in/swekster/

Agenda

Build it

- Overview of JWTs
- Common use cases for JWTs in authentication and authorization.
- Build a JWT (including bad JWTs)
- Integrate JWT in web app

Break it

- Mandatory security rant
- Decode vs Verify
- Bad Signing Algorithms
- Time is relative - reminder - get a token after deployment, save it and share it with the group.
- Denial of service
- Storing in wrong places
- Leaking secrets

Fix it

- Step by step addressing the flaws we just exploited

Workshop checklist

- Account on GitHub and logged on
- Logged on to Dev vs Ciso Discord
- Browser(s) that’s cleared of history, cookies, local storage.
- Your favorite editor running (VScode, Pycharm, vi)
- Installed development environment (see Discord #whatthestack2024)

- Don’t Panic! - every exercise has a solution in the GitHub repos.

How does this work?

And check for well protected

https://wts2024-well-protected.onrender.com

Pay attention to the cookie

- We are going to walk you through a problem
- We will share with you the repository and/or URL of a running website
- We will show or hint at a solution or direction of progress
- We will ask you to try to solve the problem yourself
- You can ask questions, but better to try first.
- We will walk through a solution
- We will have a short discussion

Build it

What are JWTs

JWT’s are NOT magic

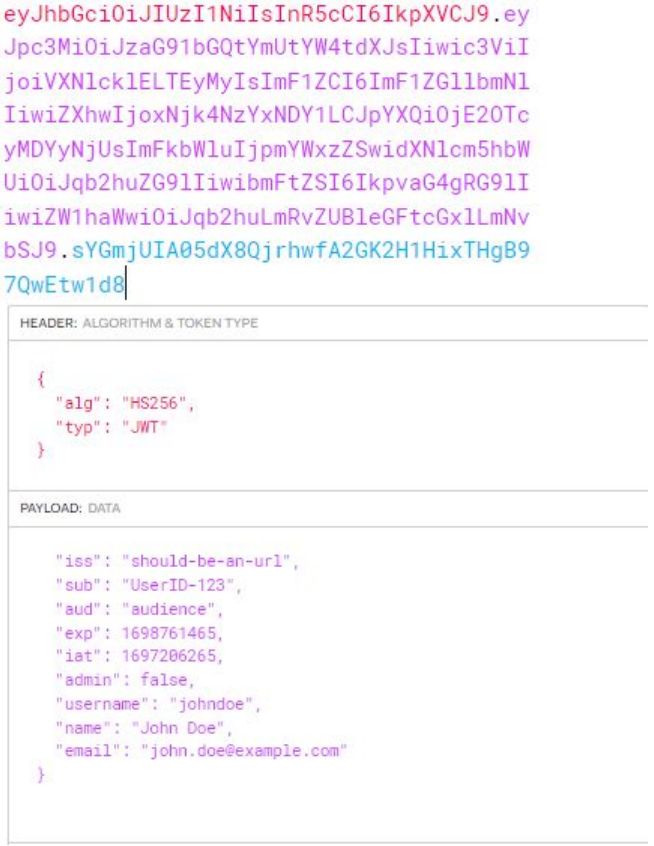
Encoded text (usually not encrypted)

The text is a piece of JSON data

A Header

A Payload

A Signature (or is there?)



How are JWTs usually used?

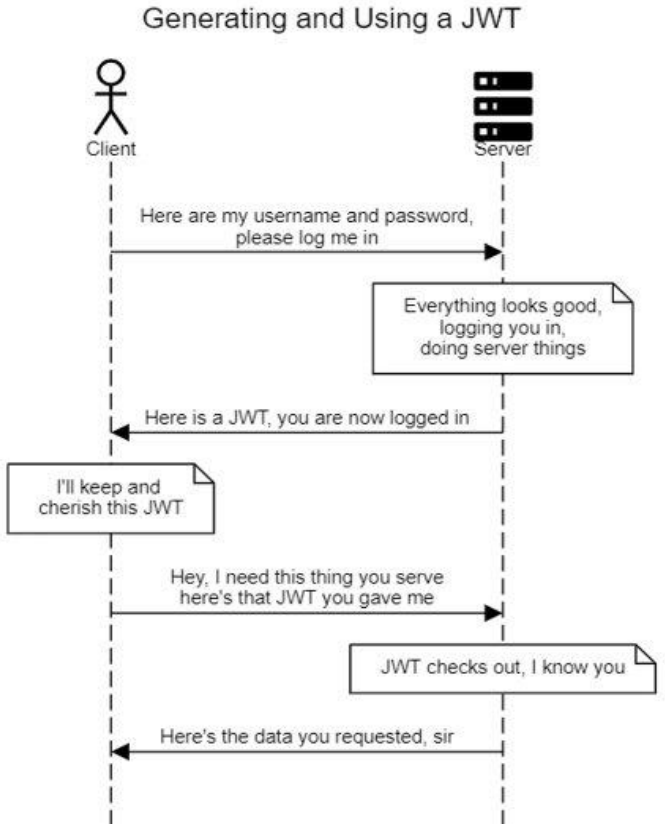
Generated on some kind of server

Sent to the client

Communicated from the client

Verified on the server

No magic required



Why are JWTs usually used?

Yes

- Near infinite scalability
- Stateless - no need to store data
- Self-contained - “Omnia mea mecum porto”

But

- The (missing) magic of revocation
- Stateful is still needed
- It’s considered “dark art” - (bus factor 1)

Exercise I - Making tokens from scratch

Do as I say

Do as I do

Follow the process to generate JWT from scratch

HS vs RS keys

- RSA Signature with SHA-256
- HMAC with SHA-256

Repo:

https://github.com/dev-vs-ciso/generating-jwt

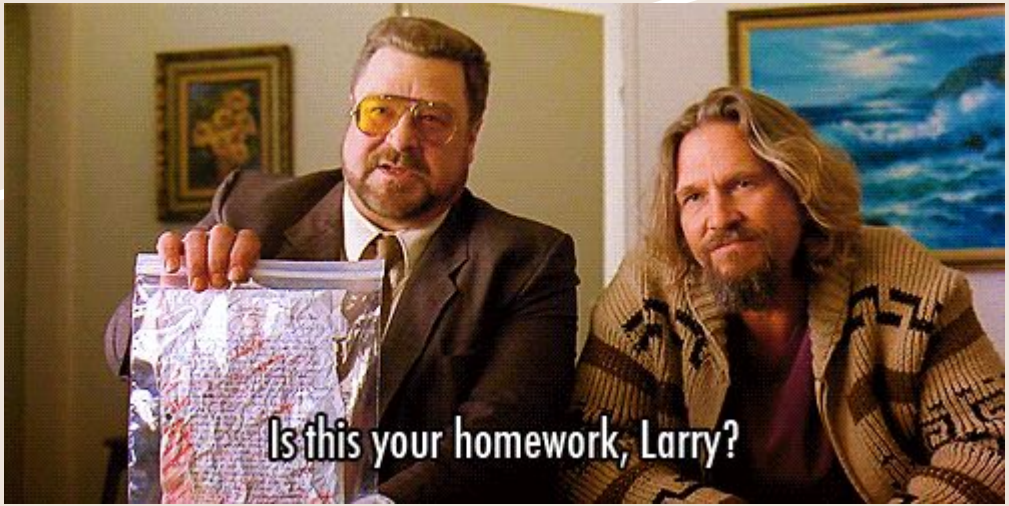
Let’s bring it into a web application

Exercise II - Integrate in a Web application

- Repeat the process to integrate JWT authentication in the application.
- Run it locally to confirm this is ok.
- Repository to work with: https://github.com/dev-vs-ciso/integrating-jwt
- URL https://integrating-jwt.onrender.com/


Have a break,
have a KitKat.®

Break it



Mandatory security rant

Security by obscurity in JWTs doesn’t work

- Code leaks
- Hacked Employee
- Hacked Customer endpoint
- Customer pentester finds the flaw
- Your pentester find the flaw (and customer reads report)
- Malicious employee
- Logs with JWTs

Swiss cheese failure model is a real real thing

- Sensitive data WILL end up in JWTs
- Someone else will code a flaw assuming your code protects them
- Product MVP are hardcoded for demo to be properly done later
- Management decides to release MVP to production
- Nobody gives you the time to do it properly later

Exercise III - Decode vs Verify

What are we really checking here?

- Consider that methods “decode” and “verify” both return True or 1 for success.
- Visit https://wts2024-decode-only.onrender.com
- Log in as user1
- Investigate
- Try to become an admin and access the secure site

Exercise IV – Bad Signing

Whose signature is this?

- Consider that there are multiple signing methods (HS, RS, none)
- Visit https://wts2024-algo-check.onrender.com
- Log in as user
- Investigate
- Try to become an admin and access the secure site
- Helpful repo: https://github.com/dev-vs-ciso/whathestack2024-vuln-python
    - branch jwt_encoder_none

Exercise V - Time is relative

How old is this token?

- JWT tokens need to expire, quickly. Consider what happens after expiry?
- Visit https://wts2024-expiry-management.onrender.com
- Try to log in
- Get the cookie, and experiment with the expiry


Have a break,
have a KitKat.®

Exercise VI - Application DoS

We don’t have to flood with traffic to break the app

- Consider that JWTs need to be decoded. It creates an opportunity for app level DoS
- Use https://github.com/dev-vs-ciso/whathestack2024-vuln-python
  - Branch overloader
- Install and try to overload our https://wts2024-well-protected.onrender.com
- Hint - more is not always better.

Exercise VII – Stuff in wrong places

Does it matter where we store the JWTs?

- Visit https://wts2024-session-storage.onrender.com/ and log in.
- Check where the JWT is set.
- Use https://github.com/dev-vs-ciso/whathestack2024-vuln-python
  - Branch rogue_data_stealer
- Try out the XSS
- Consider - what about logging stuff?

Exercise VII
-one more thing

Does it matter where we store the JWTs?

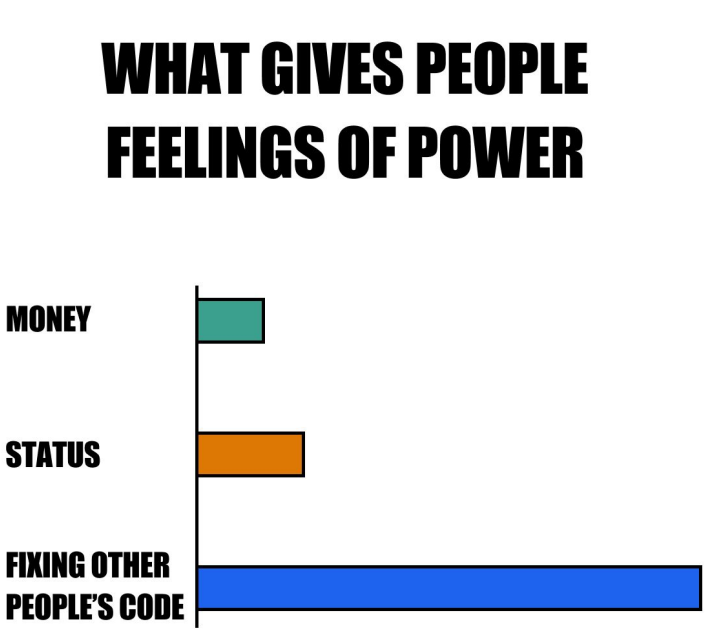
- What about logging?
  - March 1
  - 4:55PM - Received an email raising concerns about the security of our endpoints/potential leaking of our token.
  - 6:41PM - Discovered logs of an unrecognized device accessing API endpoints.
  - 6:51PM - Revoked all existing GitHub user access tokens.
  - 6:51PM-11PM - Rotated our internal access tokens.

Exercise VIII -Leaking secrets

Let me sign that for you

- Consider that even in a well protected code there can be terrible mistakes.
- Visit https://wts2024-leaking-secret.onrender.com
- Review the code https://github.com/dev-vs-ciso/whathestack2024-vuln-python
  - Branch leaking-secret
- Log in as user1
- Become admin

Fix it


WHAT GIVES PEOPLE
FEELINGS OF POWER
MONEY
STATUS
FIXING OTHER
PEOPLE'S CODE

Decode vs Verify

- This can happen, but the test is fairly easy.
- Always check for random JWT
- Add it to a test criteria for QA
- Use well known trusted frameworks

Bad Signing Algorithms

- Always check for None
- Remember that RS can be used with the wrong part of the key
- You can add a test criteria for QA
- Use well known trusted frameworks

Handle token Expiry

- Make sure the expiry process doesn't allow very old tokens
- Be careful of re-issuing only after checking expiry, but not checking validity of the token.
- If you are re-issuing tokens, check whether you have already re-issued (this is a stateless state)
- Use well known trusted frameworks

Denial of service

- Check for size of token before anything else - much faster than the decoding math.
- Standard throttling
- Make sure you do similar checks in your other business logic.

Storing in wrong places

- Use cookies, HttpOnly, Secure.
- Never store authentication credentials in client accessible JavaScript
- Assume that someone else will mess up somewhere else (like XSS)
- Use well known trusted frameworks

Leaking secrets

- Awareness
- Pre-commit hooks
- Automatic scanning of all repositories

If you must remember only ~~one~~ five things?

JWTs are not magic - You have to do the work to use them properly.

JWTs are not perfect - They have inherent deficiencies, so consider whether you need them

Don’t reinvent the wheel - when possible, build on tested and mature frameworks.

JWT bugs are outside the happy path - Most flaws remain undiscovered until you actively try to hack the system.

Never underestimate the power of stupid deadlines in large numbers

Ask Us Anything

Contact us

- Presentation: https://github.com/dev-vs-ciso/whatthestack2024

- Email: swekster@gmail.com
- Twitter: @swekster
- Youtube: https://www.youtube.com/@swekster
- Linked In: https://www.linkedin.com/in/swekster/

- Email: b.spirovski@beyondmachines.net
- Youtube: https://www.youtube.com/@spirovskib
- Linked In: https://www.linkedin.com/in/spirovskibozidar/