



Don't Use JWTs for Authorization

Sohan Maheshwar
Developer Relations @ AuthZed



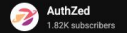
Don't use JWTs for Authorization!



0:00 / 5:34

⏪ 🔊 0:00 / 5:34 ⏩ 🎛️ 📺 🗑️ 📌 🗑️

Don't use JWTs for Authorization!



Subscribe

👍 891 🗨️ 📄 Share 📌 Save ⋮

38K views 11 months ago
Update:
Lots of discussions in the comments with different approaches. Check out this blog which refutes most of these <http://crypto.net/%7Ejoepie91/blog/201...>
...more



Authorization!

0:00 / 5:34

Don't use JWTs for Authorization!



AuthZed

1.82K subscribers

Subscribe

38K views 11 months ago

Update:

Lots of discussions in the comments with different approaches. Check out this blog which refutes most of these <http://crypto.net/%7Ejoepie91/blog/201...>

...more



jwt is one the most used things out there to authorization and authentication and one guy says no, please make your own implementation and make it opensource, geniuss 😏

Reply

0 replies



1



Probably a jwt marketplace somewhere in the shade

Reply

0 replies



What are you talking about? Expires
But you don't offer any solution to



Has this guy even read the RFC for JWT?

Reply

0 replies



Bullshit 😏

Reply

0 replies

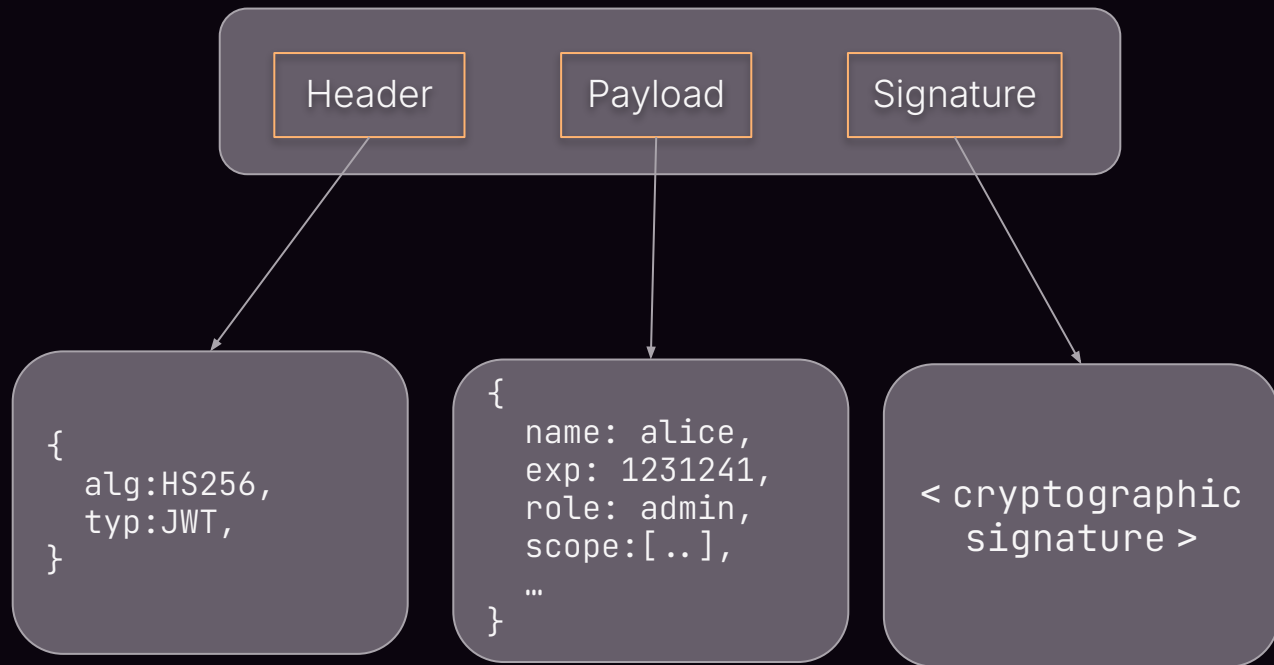


JSON Web Token aka **jot**

- Header
- Payload (claims)
- Signature

Common claims:

- **exp**
- **iss**
- **aud**
- **scope**



JSON Web Token aka **jot**

- **scope** - not even from JWT spec!
- OAuth2 Token Exchange spec

4.2. "scope" (Scopes) Claim

The value of the scope claim is a JSON string containing a space-separated list of scopes associated with the token, in the format described in [Section 3.3](#) of [\[RFC6749\]](#).

[Figure 7](#) illustrates the scope claim within a JWT Claims Set.

```
{
  "aud": "https://consumer.example.com",
  "iss": "https://issuer.example.com",
  "exp": 1443904177,
  "nbf": 1443904077,
  "sub": "dgaf4mvfs75Fci_FL3heQA",
  "scope": "email profile phone address"
}
```

Figure 7: Scopes Claim

[OAuth 2.0 Token Introspection \[RFC7662\]](#) already defines the scope parameter to convey the scopes associated with the token.

<https://www.rfc-editor.org/rfc/rfc8693#name-scope-scopes-claim>



Assumption



"Permissions don't change during token lifetime"

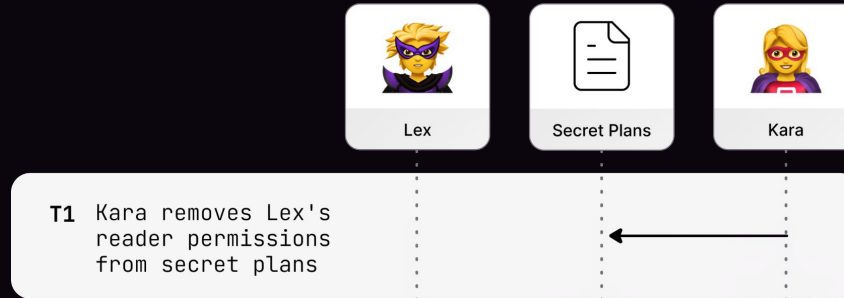
Problem #1 - No Meaningful Revocation

- You can't invalidate a JWT early
 - User removed from org?
 - Role downgraded?
 - Access revoked?

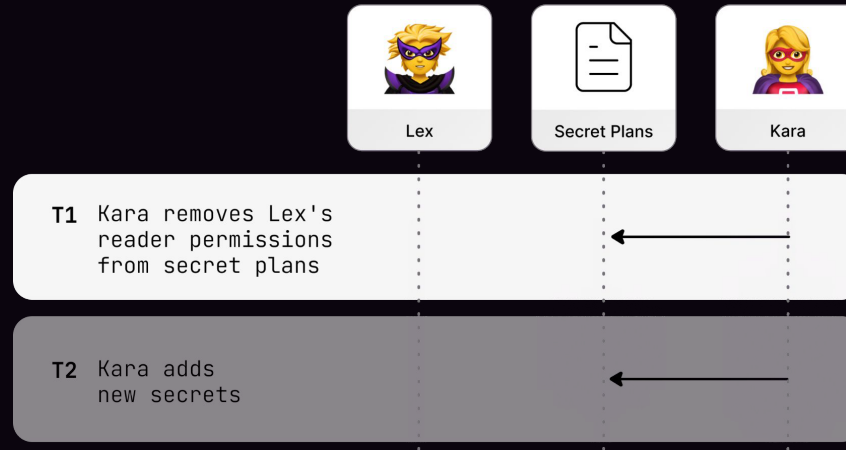
- JWTs are
 - Self-contained
 - Stateless
- No central authority to check reality
- Token still works until **exp**



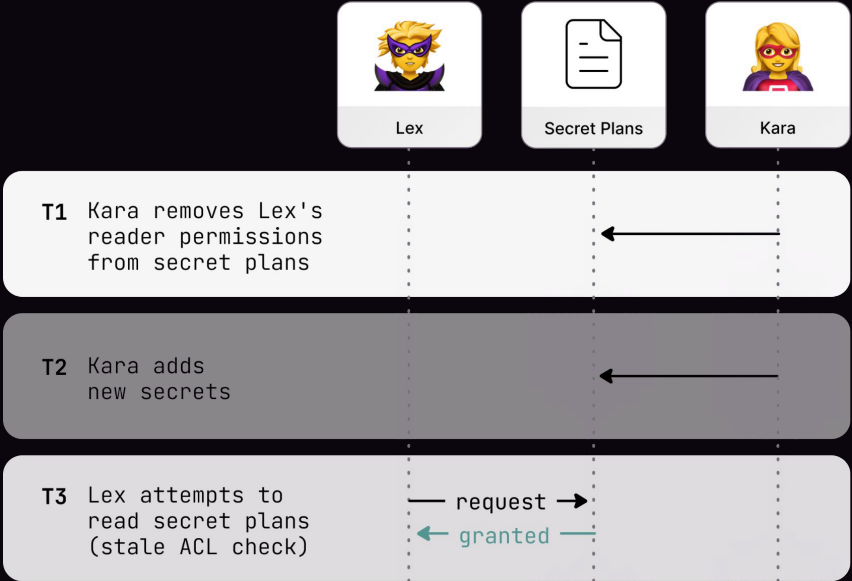
New Enemy Problem



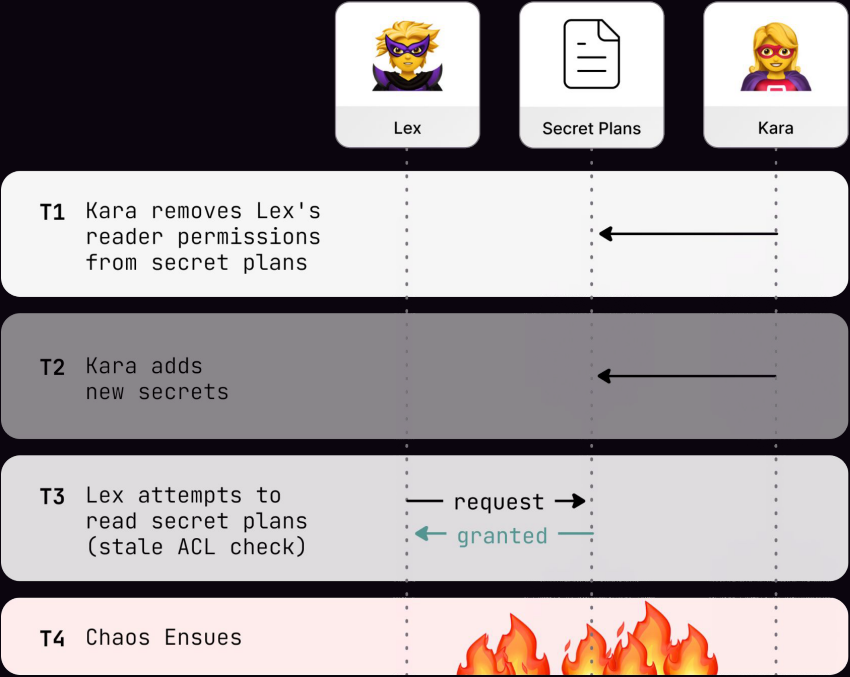
New Enemy Problem



New Enemy Problem



New Enemy Problem



Google Zanzibar

- Globally distributed authorization system
- Used in Docs, Sheets, Maps, YouTube, Gmail & more

Example Tuple in Text Notation	Semantics
<code>doc:readme#owner@10</code>	User 10 is an owner of <code>doc:readme</code>
<code>group:eng#member@11</code>	User 11 is a member of <code>group:eng</code>
<code>doc:readme#viewer@group:eng#member</code>	Members of <code>group:eng</code> are viewers of <code>doc:readme</code>
<code>doc:readme#parent@folder:A#...</code>	<code>doc:readme</code> is in <code>folder:A</code>

Table 1: Example relation tuples. “#...” represents a relation that does not affect the semantics of the tuple.

“new enemy” problem, which can arise when we fail to respect the ordering between ACL updates or when we apply old ACLs to new content. Consider these two examples:

Example A: Neglecting ACL update order

1. Alice removes Bob from the ACL of a folder;
2. Alice then asks Charlie to move new documents to the folder, where document ACLs inherit from folder ACLs;
3. Bob should not be able to see the new documents, but may do so if the ACL check neglects the ordering between the two ACL changes.

Example B: Misapplying old ACL to new content

1. Alice removes Bob from the ACL of a document;
2. Alice then asks Charlie to add new contents to the document;
3. Bob should not be able to see the new contents, but may do so if the ACL check is evaluated with a stale ACL from before Bob’s removal.

$\geq T_c$ ensures that all ACL updates that happen causally before the content update will be observed by the ACL check.

To provide external consistency and snapshot reads with bounded staleness, we store ACLs in the Spanner global database system [15]. Spanner’s TrueTime mechanism assigns each ACL write a microsecond-resolution timestamp, such that the timestamps of writes reflect the causal ordering between writes, and thereby provide external consistency. We evaluate each ACL check at a single snapshot timestamp across multiple database reads, so that all writes with timestamps up to the check snapshot, and only those writes, are visible to the ACL check.

To avoid evaluating checks for new contents using stale ACLs, one could try to always evaluate at the latest snapshot such that the check result reflects all ACL writes up to the check call. However, such evaluation would require global data synchronization with high-latency round trips and limited availability. Instead, we design the following protocol to allow most checks to be evaluated on already replicated data with cooperation from Zanzibar clients:

1. A Zanzibar client requests an opaque consistency token called a *zookie* for each content version, via a *content-*

www.zanzibar.tech

Problem #2 - Scopes don't scale

- `scope: "profile:admin"`
- Questions:
 - What data?
 - Whole website?
 - Current user's profile?





Fine-grained authorization

- Example:
 - `issue/authzed/spicedb/52:author` instead of `issue:author`
- Users can have access to millions of software objects
- JWTs can't carry this at scale


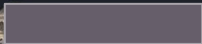



Moar Scopes!

- Tokens get huge
- Network overhead explodes
- Tokens can still be stale and/or ambiguous

  **Ways to support large number of scopes in OAuth?**




Get Help [auth0](#) [jwt](#)


  1  Oct 2022




Hi team,

As the product grows so does the number of entities, and permissions pool. The scopes are encoded in JWT and the web servers have a limit of 8kb (going with least as of Node < 14 and default Tomcat limit). What are some of the ways to bypass this. Are there any established ways to mapping or compressing scopes to avoid hitting the limit?

Thanks,

 Solved by [ty.frith](#) in post 2  

Hi there 

What are some of the ways to bypass this. Are there any established ways to mapping or compressing scopes to avoid hitting the limit?

Have you had a chance to look into using [RBAC](#)? This might help cut down on scopes/permissions per token by having a collection of permissions under each role.




Problem #3 - Predicting Permissions

- JWT is created at request entry point
- It must know every:
 - Downstream service
 - Permission needed
- Impossible in microservices

- **Over-scoped tokens**
 - Extra permissions leak downstream
 - Attack surface increases
 - Privilege escalation risk



Modern Systems

- Modern systems are:
 - Distributed
 - Dynamic
 - Composable
- JWT assumes:
 - Static permission model
-  Mismatch

Fixes

- Macaroons
 - Attenuation (reduce scope)
 - More flexible
- Reality:
 - Complex
 - Hard to adopt
 - Rarely used correctly

[Home](#) > [Publications](#) >

Macaroons: Cookies with Contextual Caveats for Decentralized Authorization in the Cloud

[Arnar Birgisson](#) · [Joe Gibbs Politz](#) · [Úlfar Erlingsson](#) · [Ankur Taly](#) · [Michael Vrable](#) · [Mark Lentczner](#) · Network and Distributed System Security Symposium, Internet Society (2014)

[Google Scholar](#) [Copy Bibtext](#)

Abstract

Controlled sharing is fundamental to distributed systems; yet, on the Web, and in the Cloud, sharing is still based on rudimentary mechanisms. More flexible, decentralized cryptographic authorization credentials have not been adopted, largely because their mechanisms have not been incrementally deployable, simple enough, or efficient enough to implement across the relevant systems and devices.

This paper introduces macaroons: flexible authorization credentials for Cloud services that support decentralized delegation between principals. Macaroons are based on a construction that uses nested, chained MACs (e.g., HMACs) in a manner that is highly efficient, easy to deploy, and widely applicable.

Although macaroons are bearer credentials, like Web cookies, macaroons embed caveats that attenuate and contextually confine when, where, by who, and for what purpose a target service should authorize requests. This paper describes macaroons and motivates their design, compares them to other credential systems, such as cookies and SPKI/SDSI, evaluates and measures a prototype implementation, and discusses practical security and application considerations. In particular, it is considered how macaroons can enable more fine-grained authorization in the Cloud, e.g., by strengthening mechanisms like OAuth2, and a formalization of macaroons is given in authorization logic.

<https://research.google/pubs/macaroons-cookies-with-contextual-caveats-for-decentralized-authorization-in-the-cloud/>

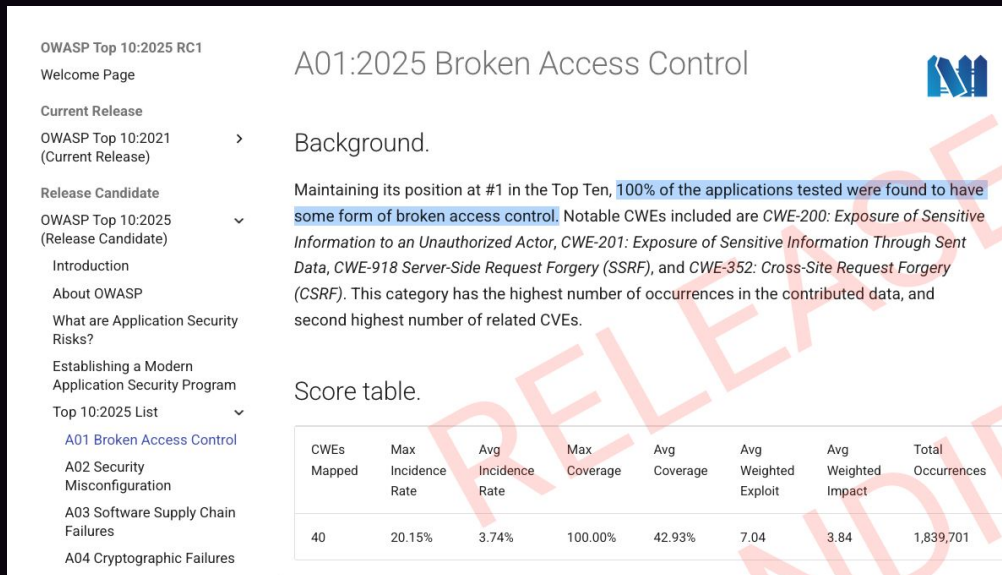




Access Control Is Complex!

OWASP Top 10: 2025

- Broken Access Control at #1
- Was #1 in 2021
- 100% of apps tested had broken access control



The screenshot shows the OWASP Top 10: 2025 RC1 website. The left sidebar contains a navigation menu with items like 'Welcome Page', 'Current Release', 'OWASP Top 10:2021 (Current Release)', 'Release Candidate', 'OWASP Top 10:2025 (Release Candidate)', 'Introduction', 'About OWASP', 'What are Application Security Risks?', 'Establishing a Modern Application Security Program', and 'Top 10:2025 List'. Under 'Top 10:2025 List', 'A01 Broken Access Control' is highlighted.

A01:2025 Broken Access Control

Background.

Maintaining its position at #1 in the Top Ten, 100% of the applications tested were found to have some form of broken access control. Notable CWEs included are *CWE-200: Exposure of Sensitive Information to an Unauthorized Actor*, *CWE-201: Exposure of Sensitive Information Through Sent Data*, *CWE-918 Server-Side Request Forgery (SSRF)*, and *CWE-352: Cross-Site Request Forgery (CSRF)*. This category has the highest number of occurrences in the contributed data, and second highest number of related CVEs.

Score table.

CWEs Mapped	Max Incidence Rate	Avg Incidence Rate	Max Coverage	Avg Coverage	Avg Weighted Exploit	Avg Weighted Impact	Total Occurrences
40	20.15%	3.74%	100.00%	42.93%	7.04	3.84	1,839,701

owasp.org/Top10/2025/0x00_2025-Introduction/

Recommendation: Use Centralized AuthZ

Mental model:

"Token tells you access" → "Ask a system if access is allowed"

Query on demand: `can(user, action, resource) → true/false`

- Real-time decisions
- Revocation works instantly
- Fine-grained access
- No over-scoped tokens



Modern Authorization Methods

- Google Zanzibar Systems
 - AuthZ as a graph of relationships (ReBAC)
 - Eg: SpiceDB, OpenFGA, Ory Keto
- Policy Decision Points (PDPs)
 - AuthZ as a logic problem solved by evaluating **policies**.
 - Eg: OPA, Cedar, Cerbos



Tradeoffs

You trade:

- Stateless simplicity

For:

- Correctness
- Security
- Flexibility



When to use JWTs

Valid use case:

- One-time, non-revocable grants
- Examples:
 - Email verification links
 - Short-lived download tokens

AVOID for:

- Long-lived sessions
- App Authorization decisions
- Fine-grained permissions



Key Takeaways

- JWTs are not for app authorization
- Revocation matters (a lot)
- Scopes don't scale
- Prediction is impossible
- Centralized authZ fixes these



Next Steps

- Stop using JWT for sessions:

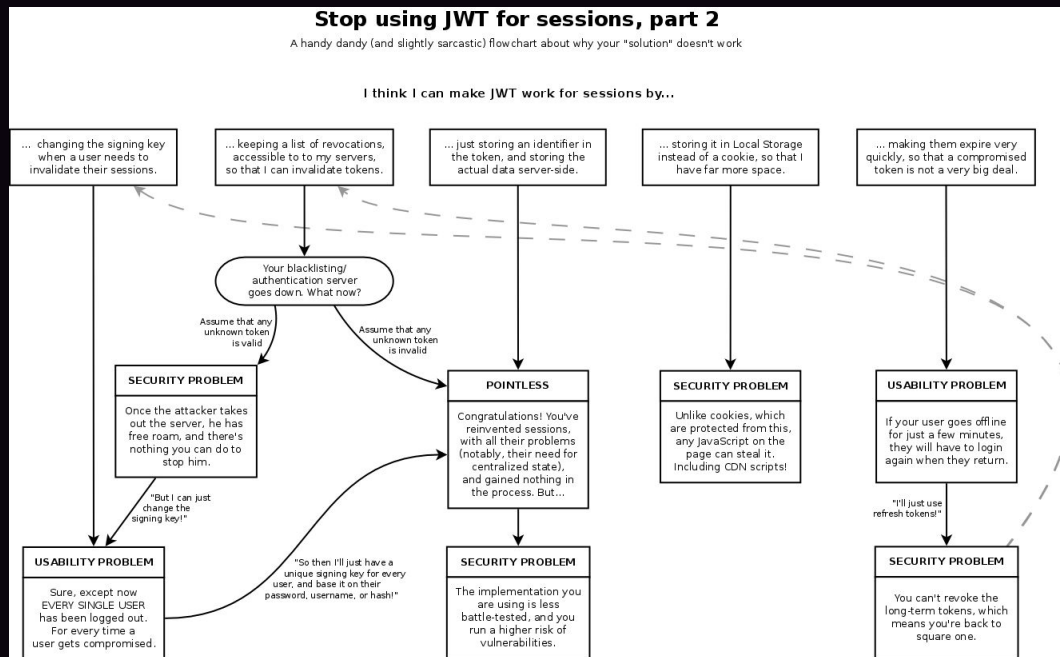
- crypto.net/~joepie91/blog/2016/06/19/stop-using-jwt-for-sessions-part-2-why-your-solution-doesnt-work

- API Tokens: A Tedious Survey

- fly.io/blog/api-tokens-a-tedious-survey/#jwt

- Read the Zanzibar paper

- zanzibar.tech



Thank You!

Sohan Maheshwar

[linkedin.com/in/sohanmaheshwar/](https://www.linkedin.com/in/sohanmaheshwar/)



Sohan Maheshwar

Lead Developer Advocate @ AuthZed |
Developer Relations

