



KCMapper

Auditing Keycloak Configurations with Neo4j

03/07/2025





Kévin Schouteeten

Pentester

@Scouty__



- French offensive security company
- 172 security experts
- 5 departments :
 - Pentest/ Redteam
 - RE / VR
 - Development
 - IR
 - Revelio




- Keycloak presentation and definition
- Why it's challenging to audit Keycloak configuration
- How KCMapper works
- Demo
- Conclusion



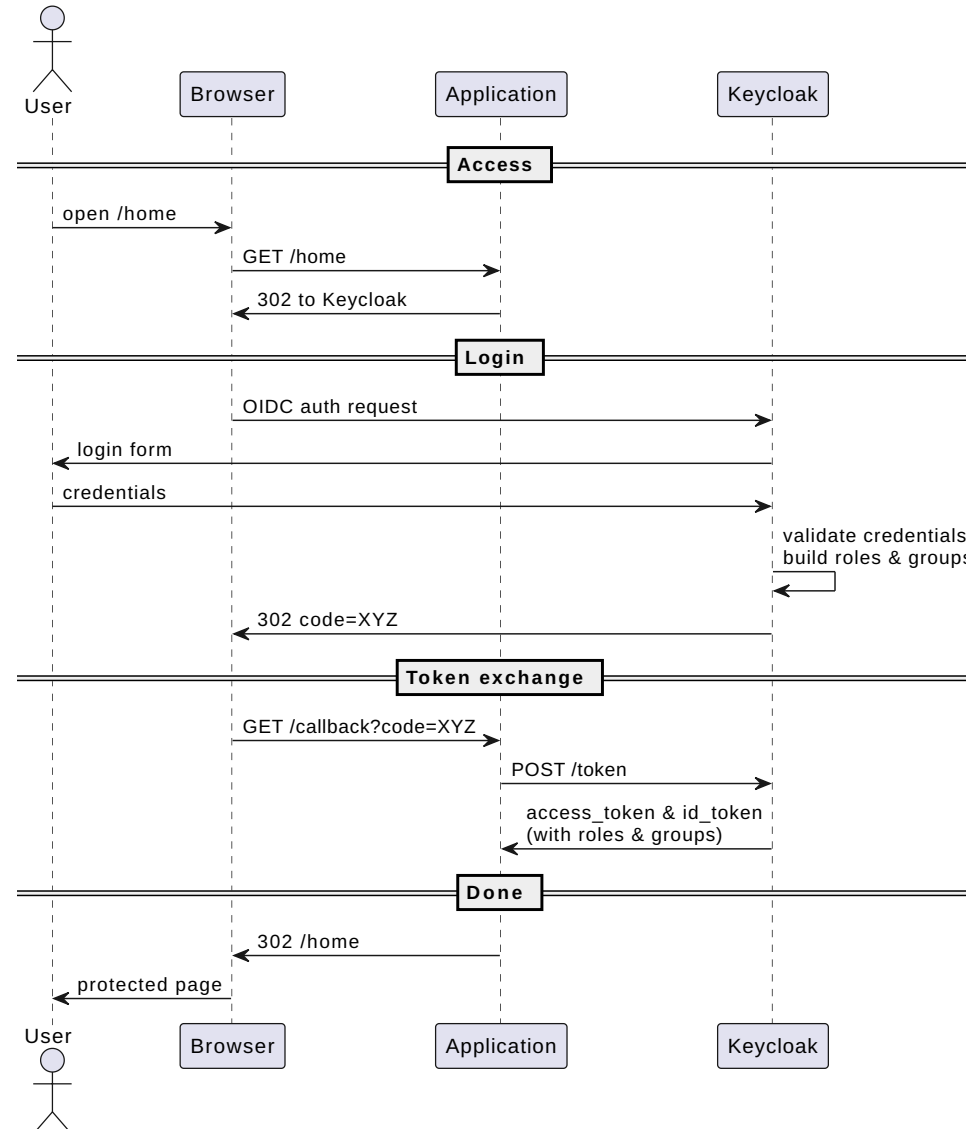
Keycloak



- Open-source Identity & Access Management (IAM) platform maintained by Red Hat
- Provides Single Sign-On, identity brokering & social login
- Supports OIDC, OAuth 2.1, SAML 2.0 protocols
- Community momentum: 50 000+ GitHub , 400+ contributors



Keycloak login workflow



Keycloak

Core entities snapshot

Entity	Purpose	Supports nesting?
Realm	Top-level security boundary; isolates users, clients, roles	—
Client	Application or API being protected; holds credentials & protocol settings	—
Role	Coarse-grained permission; can be realm-level or client-level	Composite roles (role-of-roles)
Group	Logical collection of users; inherits roles	Sub-groups
User	Human or service account; inherits groups & roles	—
Mapper	Adds/transforms claims in tokens, federation, or user storage	—



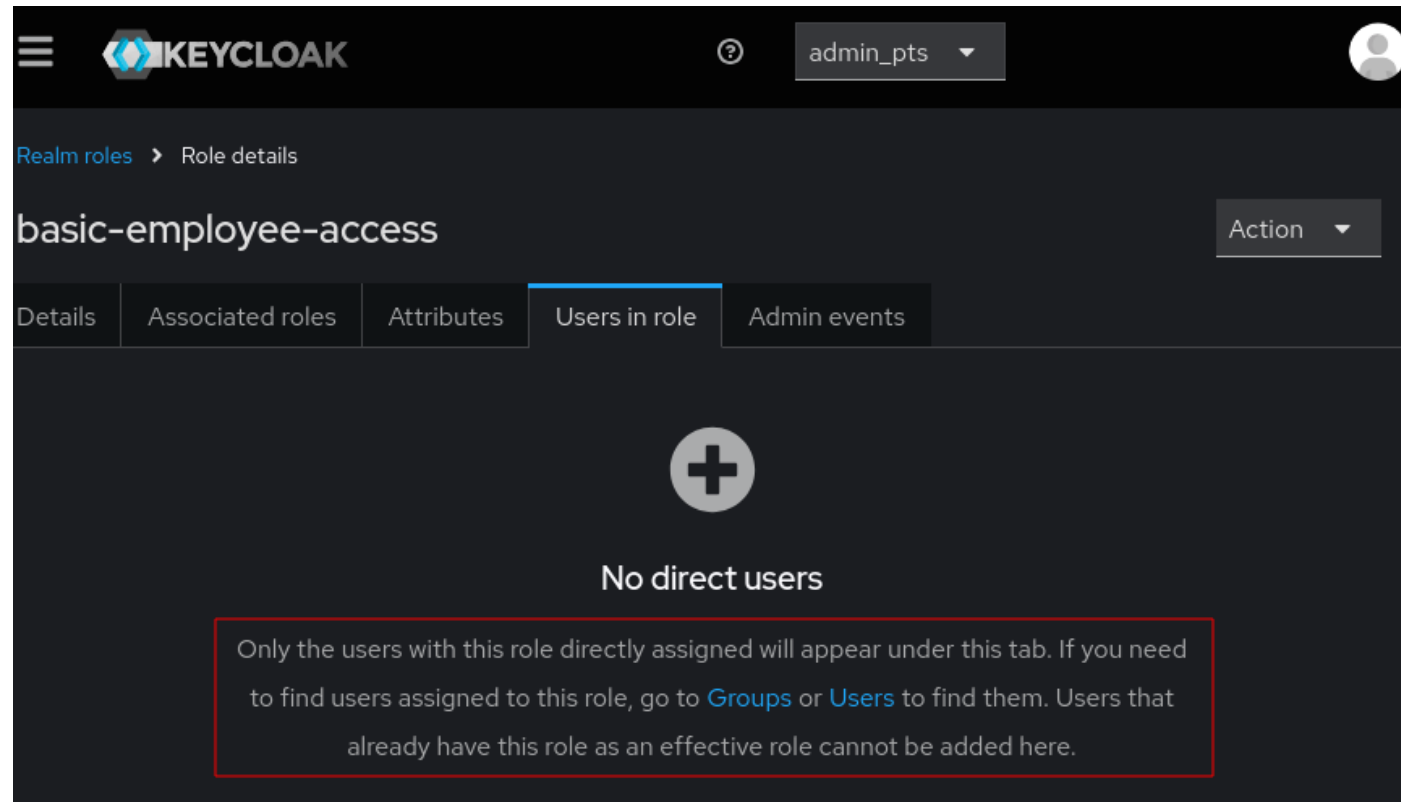
Audit challenges



Audit challenges

Why it's challenging to audit Keycloak configuration

- Deep nesting of **roles & groups** obscures effective permissions
- Keycloak UI can be misleading – a role tab may show **“No direct users”** while many users inherit the role via groups or composite roles



Audit challenges

Why it's challenging to audit Keycloak configuration

- Need to check many screens to collect permissions
- No single report showing all effective rights
- Easy to miss inherited roles through composites or groups
- Manual review is time-consuming and error-prone



KCMapper



- **Export** – uses the *python-keycloak* library by **Marcos Pereira** (<https://github.com/marcospereirampj/python-keycloak>) to pull realms via the Admin REST API
- **Map** – normalise JSON into a **Neo4j** graph
- **Analyze** – dynamic web UI: pre-written queries



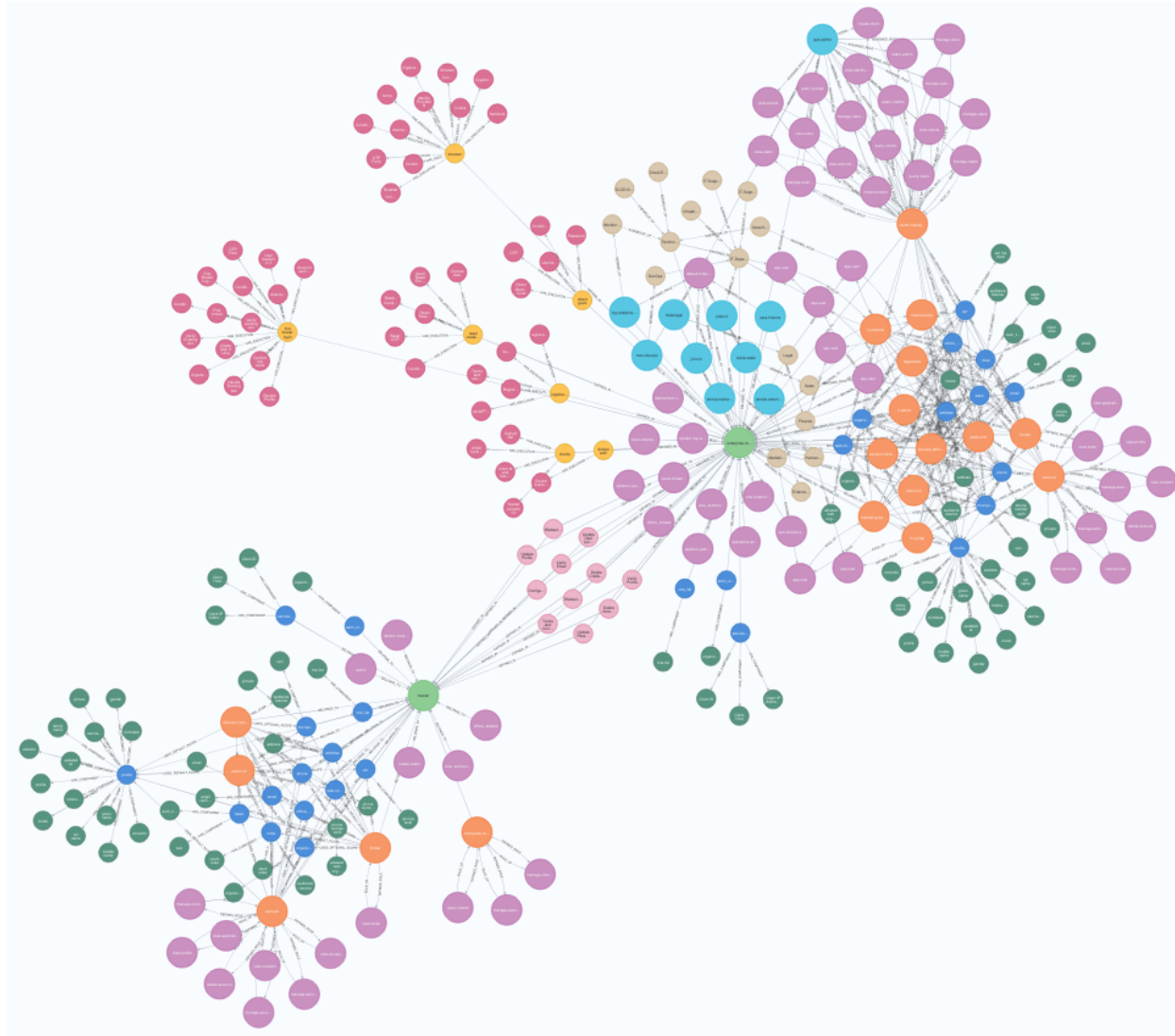
What is Neo4j?

- Native **graph database** storing nodes & relationships
- Query language: **Cypher** (SQL-like for graphs)
- Supports fast traversals on large graphs
- Used by security tools such as **BloodHound**, which maps privilege-escalation paths in Active Directory
- KCMapper loads each realm snapshot into Neo4j for interactive analysis



KCMapper

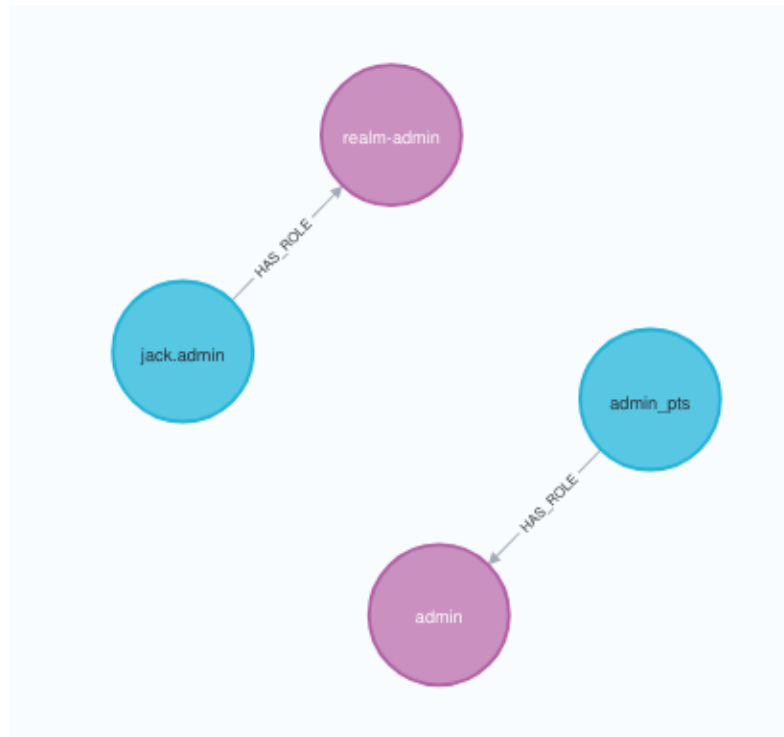
Noe4j graph after export



Cypher query samples

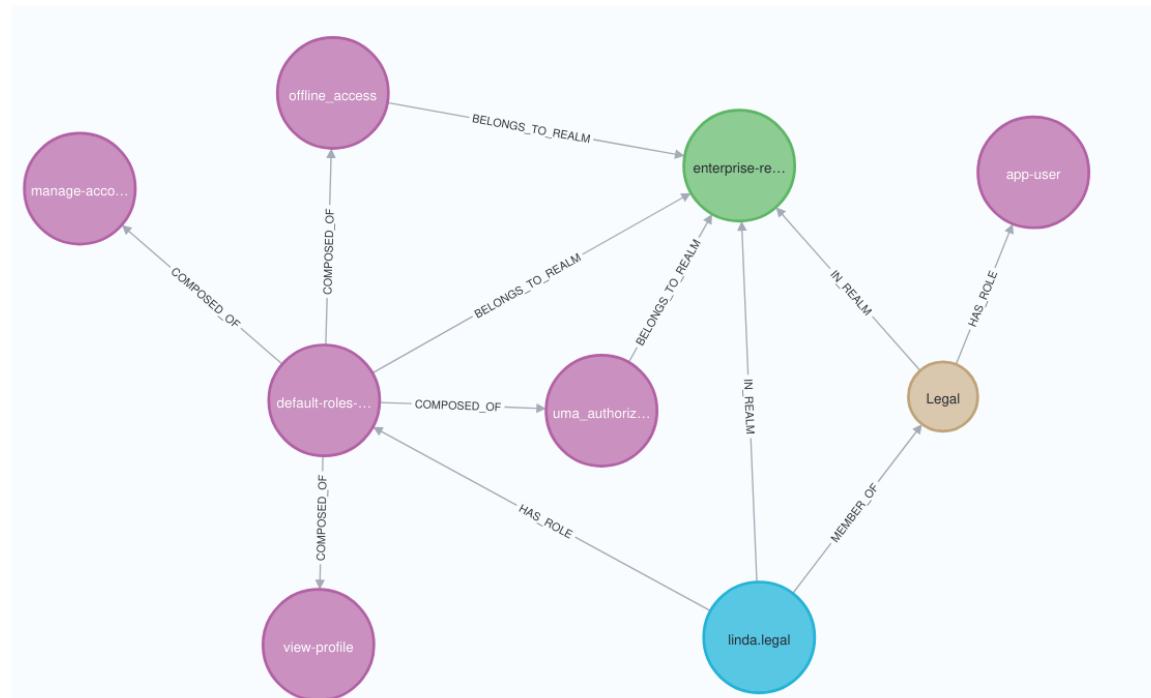
1. List all path between roles with the string **admin** and user

```
MATCH path = (r:Role)-[]-(u:User)
WHERE toLower(r.name) CONTAINS 'admin'
RETURN path;
```



2. List objects link to the user **linda.legal**

```
MATCH path = (u:User {username: "linda.legal"})-[*1..2]->()  
RETURN path  
LIMIT 25;
```



Demo



Conclusion



Conclusion

- Manual Keycloak audits are brittle and slow
- **KCMapper** automates mapping & analysis with a graph-based approach
- Dynamic UI + Cypher queries expose hidden inheritance issues
- Open-source: try it, give feedback, share rule ideas!

-> <https://github.com/synacktiv/kcmapper>



Thanks!





<https://www.linkedin.com/company/synacktiv>



<https://x.com/synacktiv>



<https://bsky.app/profile/synacktiv.com>



<https://synacktiv.com>