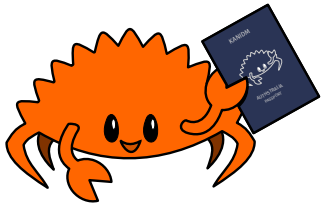
Hacker-ID Workshop



Contents

Contents (ii)

1 Intro

What's Hacker-ID?

> Hacker-ID is an *Identity Provider* (IdP) that can be used for *Identity Management* (IdM) at Hackeriet.

Scope

- Anything used by Hackeriet members which implements authentication

Why IdM?

Starting out at Hackeriet, I found:

- Managing SSH keys was a pain
- Shared credentials
- Some projects require someone™ to onboard you manually (account + keys)
- One (1) service supported Github as SSO

- Password vault requires using GPG
  - gpg.fail
  - Repeat winner of “Worlds Worst UX” award
  - Breaks regularly due to expired keys
  - `pass` is not good at the scale Hackeriet uses it for

The consequence:

Getting started/joining a project is difficult → ~~Gjørokrati~~

Why Kanidm?

1. It's opinionated in favor of:
   - Security defaults (e.g. OAuth2, modern authentication schemes)
   - User data practices
2. Supports the bare minimum we might need:
   - OAuth2 / OpenID Connect
   - RADIUS (Network gear)
   - SSH (We're a Linux shop, folks!)
   - LDAP (Nearly anything that doesn't support the above)
3. Featureful API
4. I've used it before
5. It's *very easy* to go from 0 to production

2 Kanidm basics

Persons

User accounts (i.e. humans) are defined using the “Person” object

Metadata

- Account name (`d404d`)
- Display name (“404′d”)
- Legal name (*Optional*)
- Primay email
- Secondary emails

Credentials

- Passkeys (CTAP2 key, Browser, Pw.Mgr.)
- Password w/OTP
- SSH keys

Special:

- Unix password
- RADIUS password
- Client certificates

Persons (ii)

Users identify themselves using their *account name*:

- On the web interface
- In the CLI
- Over RADIUS
- Over SSH
- For Unix auth

Users can change *all* metadata on themselves.

This includes:

- The email address(es)
- The *account name*

```
% kanidm self whoami | grep -i '^name:'
name: d404d
% kanidm person update --newname marty d404d
% kanidm self whoami | grep -i '^name:'
name: marty
```

↑ This is a *feature*

Persons (iii)

> Users are identified by their *UUID*

```
% kanidm person get d404d | grep -i uuid
uuid: a3cc01b9-1bf1-4166-857b-af15302e45db

% kanidm person get a3cc01b9-1bf1-4166-857b-af15302e45db | grep -i '^name:'
name: d404d
```

All applications **must** use *UUID* for identifying users

Groups

Accounts (persons + service accounts) are given access through groups

Special groups

- idm_all_accounts
- idm_all_persons

Role groups

- domain_admins
- idm_admins
- idm_service_desk
- system_admins

General functions

- idm_account_mail_read
- idm_mail_servers
- idm_message_senders
- idm_people_self_mail_write
- idm_people_self_name_write
- idm_radius_servers
- idm_ui_enable_experimental_features
- idm_unix_authentication_read

Groups (ii)

Administrative functions

- idm_access_control_admins
- idm_account_policy_admins
- idm_application_admins
- idm_client_certificate_admins
- idm_group_admins
- idm_mail_service_admins
- idm_message_admins
- idm_oauth2_account_admins
- idm_oauth2_admins
- idm_oauth2_client_admins
- idm_people_admins
- idm_people_on_boarding
- idm_people_pii_read
- idm_radius_service_admins
- idm_recycle_bin_admins
- idm_schema_admins
- idm_service_account_admins
- idm_unix_admins

Groups (iii)

TL;DR:

- `idm_admins` get access to anything account management
- `system_admins` get access to operational admin features

Fun fact

`idm_high_privilege` serves as a taint role, enforcing stricter access control and object security for affected users

Groups (iv)

Hackeriet's groups are roughly structured as follows:

- Role groups
- Service role groups
- Project groups

Groups (v)

> **Roles** follows the pattern `<scope>-<role>`

These groups assigns broad rights across sets of users.

Examples

Hackeriet

- `hackeriet-members`: Active (paying) members
- `hackeriet-styret`: The board

Nettlauget

- `nettlaug-tenants`: People making use of the server room etc.
- `nettlaug-operators`: Network operations

Groups (vi)

> **Service roles** follows the pattern `service-<service>-<role>`

These groups grants context-specific rights for groups (preferably) or users.

Common roles:

- `users`: General login rights. You'd usually bind OAuth2 clients to these roles.
- `admins`: Adminitrative rights.

Examples

- `service-mobilizon-users`: Grants access to Mobilizon.
- `service-netbox-superusers`: Grants the "superuser" Django flag in Netbox to this user.

Groups (vii)

> **Project groups** follows the pattern `project-<name>`

Groups for members who want to utilize Hacker-ID for access control in their own projects.

Examples

- `project-avinfra`: LEDFX, and some testing for NDI video streaming in the space.
- `project-vendingmachine`: SSH to the RPi within *you know what*.
- `project-hackradio`: RPi for testing UX improvements on music playback.

The CLI

Read and modify account data

```
% kanidm person get d404d
directmemberof: idm_all_persons@idp.hackeriet.no
directmemberof: idm_all_accounts@idp.hackeriet.no
directmemberof: hackeriet-members@idp.hackeriet.no
gidnumber: 1882080731
name: d404d
spn: d404d@idp.hackeriet.no
uuid: a3cc01b9-1bf1-4166-857b-af15302e45db
```

The CLI (ii)

Read group information

```
% kanidm group get hackeriet-members
description: Actively subscribed members
directmemberof: service-mobilizon-users@idp.hackeriet.no
directmemberof: service-dokuwiki-users@idp.hackeriet.no
directmemberof: service-librenms-users@idp.hackeriet.no
directmemberof: service-hedgedoc-users@idp.hackeriet.no
directmemberof: service-netbox-users@idp.hackeriet.no
entry_managed_by: svc-hackerhula@idp.hackeriet.no
gidnumber: 1906302590
member: hackeriet-styret@idp.hackeriet.no
member: d404d@idp.hackeriet.no
name: hackeriet-members
spn: hackeriet-members@idp.hackeriet.no
uuid: 27be9add-f1f4-4e8f-a9ee-32e9a19fde7e
```

The CLI (iii)

Add group members

```
% kanidm group add-members service-idp-sysops atluxity apt der_metzger aleksil
```

Remove group members

```
% kanidm group remove-members service-idp-sysops someone_idk
```

Create new groups

```
% kanidm group create project-test d404d
```

Navigate the command tree

```
% kanidm -h
```

Documentation

https://kanidm.github.io/kanidm/stable/

3 `idp1.hackeriet.no`

Getting familiar with the host setup

Currently running on a single VM in Klynge1, dualhomed to DMZ and NETT-INFRA-HOSTS.

`idp.hackeriet.no`

- HTTPS
- OAuth2
- API

`int-idp.hackeriet.no`

- SSH
- RADIUS
- LDAP

1. SSH to `int-idp.hackeriet.no`
2. Check Docker stack with `docker ps`
3. Check folder contents at `/srv/kanidm`

Full details: https://wiki.hackeriet.no/infra/services/hacker-id#system_setup

Break-the-glass accounts

Two break-the-glass accounts exists by default inside Kanidm itself:

- `admin`, for operational actions
- `idm_admin`, for account management actions

This only grants access to Kanidm

Passwords are generated on-demand:

```
docker compose exec kanidm kanidmd recover-account admin
docker compose exec kanidm kanidmd recover-account idm_admin
```

Accounts must not be used for day-to-day work.

More info: https://kanidm.github.io/kanidm/stable/server_configuration.html#default-admin-accounts

Break-the-glass accounts (ii)

For emergency access to `idp1.hackeriet.no`, the `btg` local account can be used.

Password for proxmox console, SSH key for shell

To-do:

- Cache-prime keys from users in `service-idp-sysops` onto the `btg` account key list
- Password stored at TBD

Backups

Kanidm automatically makes backups every night:

```
root@idp1:/srv/kanidm/data/kanidm/backups# ls
backup-2026-04-04T22:00:00.225524009Z.json.gz
[...]
backup-2026-04-10T22:00:00.496787259Z.json.gz
```

Restoring:

```
cd /srv/kanidm
docker compose stop kanidm
docker run --rm -i -t -v /srv/kanidm/data:/data kanidm/server:latest \
  /sbin/kanidmd database restore -c /data/server.toml \
  /data/kanidm/backups/backup-[...].json.gz
docker compose start kanidm
```

https://kanidm.github.io/kanidm/stable/backup_and_restore.html

Unix integration

Kanidm provides a dedicated service `kanidm-unixd` for integrating with Linux and other platforms.

- Manages lookups of users, groups, SSH keys, and more
- Falls back to local cache if Kanidm server is down
- Local accounts are prioritized over Kanidm-managed ones

```
btg@idp1:~$ kanidm-unix status
system: online
Kanidm: online
```

Unix integration (ii)

Built for resiliency; if you're in the cache already then you should be good.

```
btg@idp1:~$ kanidm-unix cache-invalidate -h
```

> Invalidate, but don't erase the content of the unixd resolver cache. This will force the unixd daemon to refresh all user and group content immediately. If the connection is offline, entries will still be available and will be refreshed as soon as the daemon is online again

Unix integration (iii)

Integration is managed through three config files:

- `/etc/kanidm/config`: Points to the Kanidm instance
- `/etc/kanidm/unixd`: Describes IDP-group-to-local-group mappings (e.g. sudo, docker rights)
- `/etc/ssh/sshd_config.d/kanidm.conf`: Hooks OpenSSH up to Kanidm's ssh key lookup command

First time install is a bit involved

See https://kanidm.github.io/kanidm/stable/integrations/pam_and_nsswitch.html for details

Also see https://wiki.hackeriet.no/infra/services/hacker-id#using_hacker-id_for_sshlinux_login

4 Hooking up OAuth2 apps

Creating an app

Apps have:

- An object name (e.g. `hackeriet-mobilizon`)
- A display name
- An origin (the URL we should send the user *back* to after auth.)
- A landing (a URL we can use on `idp.hackeriet.no` which preferably will automatically trigger SSO sign-in)

The origin, as well as other needed metadata, must be gandered from the documentation of whatever service you're integrating.

```
% kanidm system oauth2 create -h
Create a new oauth2 confidential client that is protected by basic auth

% kanidm system oauth2 create-public -h
Create a new OAuth2 public client that requires PKCE. You should prefer using
confidential client types if possible over public ones
```

Granting rights

People are granted access to OAuth apps by being assigned scopes on said app. This doubles as the information the app is allowed to request from the user.

This is where the “general login rights” service role comes in:

```
kanidm system oauth2 create-scope-map hackeriet-grafana service-grafana-users email
groups
```

We can then assign `service-grafana-admins` as members of `service-grafana-users`, avoiding the need to manage separate scope maps for all role groups.

Granting custom claims

Some applications makes use of custom claims to control access, limits etc.

Grafana for example lets us assign rules through custom claims:

```
kanidm system oauth2 create-claim-map hackeriet-grafana grafana_role service-grafana-
admins GrafanaAdmin
```

OAuth2 pitfalls

The untold truths in SSO-land:

- Services will violate spec:
  - `sub`: Subject - *Identifier for the End-User at the Issuer.*
  - `preferred_username`: *Shorthand name by which the End-User wishes to be referred to at the RP, such as* `janedoe` *or* `j.doe`*. [...] The RP MUST NOT rely upon this value being unique*
  - `email`: *End-User's preferred e-mail address. [...] The RP MUST NOT rely upon this value being unique*
- Services will collect more than they need
- Services will skip on role/group mapping capabilities

https://openid.net/specs/openid-connect-core-1_0.html#StandardClaims

5 Further details

API

https://idp.hackeriet.no/docs/

Radius sidecar

Radius is implemented using a dedicated container: `kanidm/radius`

Radius does not give us strong access controls; need to run one container per separate RADIUS “context”.

Currently only used by `nettlaug-operators` towards switches.

```
# if the user is in one of these Kanidm groups,
# then they're allowed to authenticate
radius_required_groups = ["nettlaug-operators@idp.hackeriet.no"]
```

Hula

Hula is very loosely integrated with Hacker-ID:

- Members can create a Hacker-ID account through Hula
- Members can recover credentials through Hula if none is configured
- Subsequent management is done through Hacker-ID

Details to be determined in https://github.com/hackeriet/hackerhula/issues/61

Hula has one service account: `svc-hackerhula`

- Member of `idm_people_on_boarding`
- Entry manager of
  - `hackeriet-alumni`
  - `hackeriet-members`
  - `hackeriet-styret`
  - `nettlaug-tenants`
  - `nettlaug-operators`