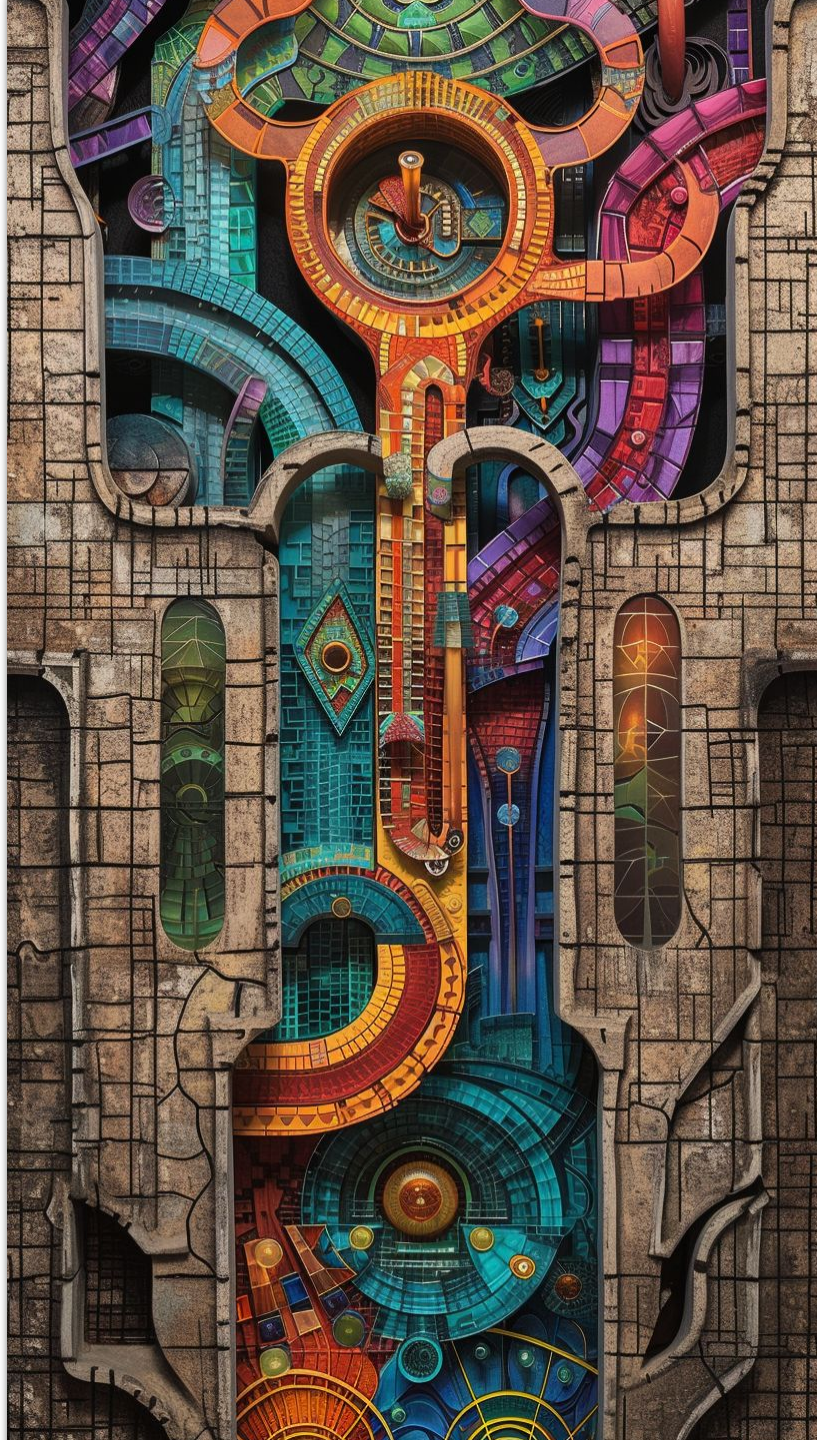
sambaXP'2024

POSIX identities out of OAuth2 identity providers

How to redesign SSSD and Samba

Alexander Bokovoy || Andreas Schneider || Sumit Bose



Who are we?

Alexander Bokovoy

- Software engineer at Red Hat
- Focus on identity management and authentication in Red Hat Enterprise Linux and Fedora Project
  - FreeIPA, SSSD, Samba, MIT Kerberos
- Samba Team member, FreeIPA core developer

Andreas Schneider

- Software engineer at Red Hat
- Samba maintainer for Red Hat Enterprise Linux and Fedora Project
  - Samba, libssh, cmocka, ...
- Samba Team member

Sumit Bose

- Software engineer at Red Hat
- SSSD core developer

POSIX identities

- POSIX identities
  - Stable user/group information (UID and GID values) are used to run processes in environments, compatible with POSIX standards
  - File system access is arbitrated with IDs, not user/group names. Names are resolved to IDs by the operating system components
  - POSIX identity metadata: what shell to run at login, where to find default home directory

- Focus: traditional workstations and servers in enterprise environments
  - Users have the same UID/GID values on all machines they can login to
  - Data stored locally under different user/group IDs belong to different users

POSIX ID needs and their coverage by OAuth2 IdPs

- POSIX users
  - User name
  - UID and (primary) GID numbers (32-bit)
  - [may be] Description (`gecos`)
  - home directory
  - Shell

- POSIX groups
  - Group name
  - GID number (32-bit)
  - [may be] Description
  - List of group members

OIDC Connect default claims (excerpt from OIDC Connect specification)

Member	Type	Description
sub	string	Subject - Identifier for the End-User at the Issuer.
name	string	End-User's full name in displayable form including all name parts, possibly including titles and suffixes, ordered according to the End-User's locale and preferences.
given_name	string	Given name(s) or first name(s) of the End-User. Note that in some cultures, people can have multiple given names; all can be present, with the names being separated by space characters.
family_name	string	Surname(s) or last name(s) of the End-User. Note that in some cultures, people can have multiple family names or no family name; all can be present, with the names being separated by space characters.
middle_name	string	Middle name(s) of the End-User. Note that in some cultures, people can have multiple middle names; all can be present, with the names being separated by space characters. Also note that in some cultures, middle names are not used.
nickname	string	Casual name of the End-User that may or may not be the same as the `given_name`. For instance, a `nickname` value of `Mike` might be returned alongside a `given_name` value of `Michael`.
preferred_username	string	Shorthand name by which the End-User wishes to be referred to at the RP, such as `janedoe` or `j.doe`. This value MAY be any valid JSON string including special characters such as `@`, `/`, or whitespace. The RP MUST NOT rely upon this value being unique, as discussed in **Section 5.7**.

Authenticated access

- User information is needed before user session is established
  - SSH server or console login process needs to know POSIX identity and user metadata before login

- OAuth2 Identity Provider (IdP) requires client identification and user consent to get access to user information
  - OAuth2 client identification ~ host enrollment into enterprise domain
  - OAuth2 client credentials need to be guarded on the host side if anything non-trivial is exposed through their permissions
    - Trusted Platform Module (TPM, a chip which improves the security of your system) integration is needed

OAuth2: an open standard for access delegation, commonly used as a way for internet users to grant websites or applications access to their information

OpenID Connect (OIDC): an auth protocol that verifies user IDs when they sign in. It is an extension of OAuth2.

Authenticated access (2)

- Host enrollment
  - Simplest case = create OIDC client creds for this host
    - Users can do so with public IdPs, an enrollment tool can handle the details on behalf of a user
    - Protect OIDC client creds locally with systemd-creds or similar interface (binding to TPM)
  - Advanced case: Entra ID and Intune service allow a host enrollment with a special endpoint
    - Authenticate against a Broker application endpoint on user's behalf
      - Windows does it with the pre-authorized (private) Windows OIDC client creds
      - These credentials owned by Microsoft and trusted by all tenants
      - Everyone else will need to register own OIDC client (for each machine)
    - Client then registers by exchanging cryptographically signed data with a Device Registration Service (DRS)
    - Expects integration with TPM and derivation of tokens based on the primary resource token's possession

Azure AD integration from David Mulder (Samba Team, SUSE): https://github.com/himmelblau-idm/himmelblau/

SambaXP talk on April 18th

Bridging Worlds: Linux and Azure AD

Enrolled and (dangerous)

- Enrolled host is really an OIDC client
  - Define IdP claims to POSIX ID metadata mapping
  - Process data to retrieve or generate POSIX information
  - Perform authorization against IdP to delegate authentication on login
- Online only
  - IdP is not available offline
  - offline login as a PAM stack option with pam_sss (SSSD PAM module) with passwords or pam_sss_gss with Kerberos tickets

Generate POSIX information

- IdPs do have POSIX information
  - No IdP provided one so far, green field
- Solution: use IdP-integrated OAuth2 application
  - Provide ID ranges
  - Provide user POSIX ID metadata
  - Enforce data consistency
  - Provide access control extensions
- Local system configuration
  - Store mapping locally, allow admins to adjust
  - Pull system configuration from an OAuth2 application
    - Self-provisioning in large environments

POSIX OAuth2 application

- Host enrollment mechanism
  - Same enrollment process for all IdPs
  - Same integration mechanism for different enterprise domain systems
- POSIX ID self-management for users (read/write for specific data)
  - Customizable by admin and users
- May implement algorithmic mapping

Can we trust federation?

- Federation is common
  - User authentication is delegated to other OAuth2 IdP (Google, Azure, Github, Gitlab, etc.)
  - Some claims from the federated IdP response used to fill in user claims in our IdP
  - There is no way to know origin of the claims in a response

- What should we trust for POSIX needs?
  - Multiple IdPs run POSIX OAuth2 app
    - Use Identity chaining to communicate between them and coordinate POSIX ID mapping in trusted environments

draft-ietf-oauth-identity-chaining is promising

- Requires explicit cross-domain trust agreement, unrealistic for public IdPs
- Similar to S4U extensions and constrained delegation in Kerberos

Generate POSIX information

- IdPs have no POSIX information
  - Algorithmic mapping
    - Have N non-overlapping ID ranges defined by (startID, sizeID) for each range
    - num = **hash**(unique_identifier_of_IdP_server) % N
    - offset = **f**(unique_attribute_value) % sizeID[num]
    - POSIX-ID = startID[num] + offset
  - Hash is a configurable message digest function with configurable seed
  - f is configurable function depending on unique attribute/claims from the object properties
  - SSSD:
    - hash = murmurhash3 with a common seed
      - 0xdeadbeef is used by SSSD
    - startID from [200000 ... 2000000000]
    - sizeID is 200000
    - f()
      - for AD domains it is identity function of the object's RID
      - for OAuth2 object is murmurhash3 of a chosen unique attribute
    - Uses automatic private groups by default

Fully qualified names

- username@idp.suffix
  - Generate ID range off the idp.suffix
  - Generate ID offset in the range by username value
- Works for multiple IdPs
- Stable ID mapping on multiple workstations without additional requirements from IdP
- No support for username aliases (ID collisions)

Generate POSIX information

```
ID ranges: [(200000, 200000), (400000, 200000), (600000, 200000), (800000, 200000), (
1000000, 200000), (1200000, 200000), (1400000, 200000), (1600000, 200000), (1800000,
200000), (2000000, 200000)]
Domain sambaxp.org
        ID(a@sambaxp.org)       [1428744, 6, 28744]
        ID(admin@sambaxp.org)   [1485907, 6, 85907]
        ID(testuser@sambaxp.org)        [1469896, 6, 69896]
        ID(testuser1@sambaxp.org)       [1451526, 6, 51526]
        ID(very long name with spaces@sambaxp.org)      [1445576, 6, 45576]
        ID(some.account@sambaxp.org)    [1452162, 6, 52162]
Domain example.com
        ID(a@example.com)       [428744, 1, 28744]
        ID(admin@example.com)   [485907, 1, 85907]
        ID(testuser@example.com)        [469896, 1, 69896]
        ID(testuser1@example.com)       [451526, 1, 51526]
        ID(very long name with spaces@example.com)      [445576, 1, 45576]
        ID(some.account@example.com)    [452162, 1, 52162]
Domain
        ID(a@)  [2028744, 9, 28744]
        ID(admin@)      [2085907, 9, 85907]
        ID(testuser@)   [2069896, 9, 69896]
        ID(testuser1@)  [2051526, 9, 51526]
        ID(very long name with spaces@) [2045576, 9, 45576]
        ID(some.account@)       [2052162, 9, 52162]
```

Fully qualified names

- username@idp.suffix
  - Generate ID range off the idp.suffix
  - Generate ID offset in the range by username value
- Works for multiple IdPs
- Stable ID mapping on multiple workstations without additional requirements from IdP
- No support for username aliases (ID collisions)

Samba integration

- ID mapping
  - Dynamically loaded modules (idmap) in winbind to represent a ‘domain’ of users and groups
    - idmap_passdb, idmap_nss, and idmap_sss
    - Needs running winbind
- Easiest way, with no changes:
  - idmap_passdb
    - Lookup in a passdb backend, by default SIDs in tdbsam and POSIX IDs in NSS
      - Makes use of nss_himmelblau and nss_sss possible in standalone mode
  - idmap_nss
    - Lookup POSIX IDs through NSS, then lookup SID through a domain controller with the help of winbindd
      - Will not work in standalone mode
  - idmap_sss
    - Lookup through SSSD
      - Makes use of any SSSD ID provider with SIDs possible
      - Will not work in standalone mode unless winbind is running

Samba modes

- **Standalone**: no domain joined, winbind is optional
- **Domain member**, winbind is required
- **Domain controller**, winbind is required

Samba integration

- Domain SID generation options
  - Domain member/controller
    - Map to existing domain user, reuse domain SID
      or
    - Define SID based on the properties of the IdP suffix
  - Standalone
    - Map to local machine SID
      or
    - Define SID based on the properties of the IdP suffix

Access control

- Who can login to that account on that machine
- Who is authorized to use these PAM services
- Who can raise privileges to run SUDO

Authentication at login time

- IdP authentication
  - Typically browser based
  - Needs a browser before login

- Login fact should be reusable in the session
    - SUDO reauthentication should not be constant
    - Local SSH access should be seamless
    - Browsers should be able to sign-on seamlessly to services from the same (organizational) domain

Browser is a new mainframe

- 2016: captive portals
  - Login over network needs … network access
  - Network access needs captive portal handling
    - Before login to the desktop/laptop

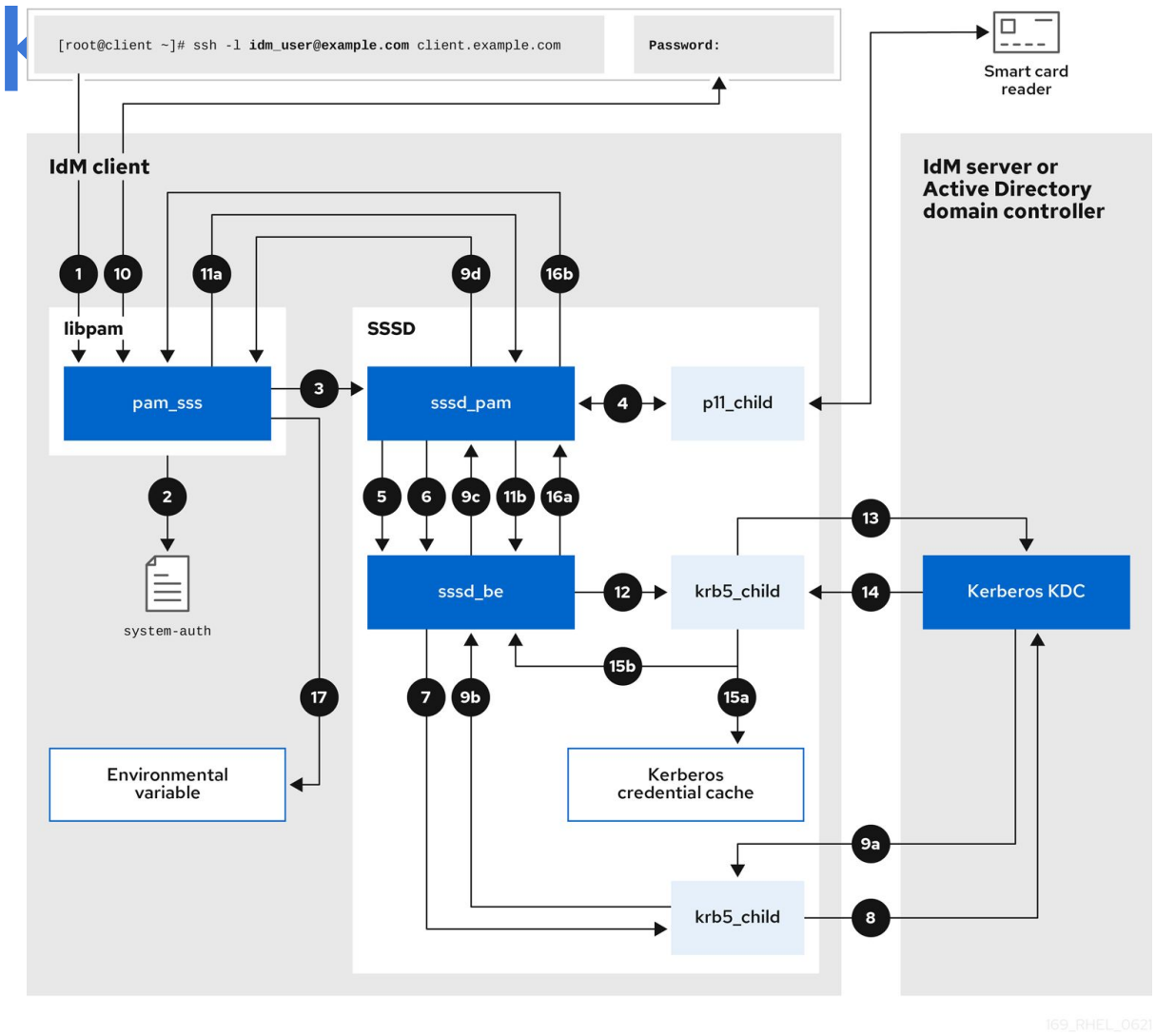
- 2024: OAuth 2.0 identity provider before login
  - Login with OAuth 2.0 implies user browser interaction
  - Still no browser view access prior to GDM login
    - Security issues with untrusted content

Somewhere else browser

- Remote access
  - We already have **other** system to run browser
  - Instruct user to visit OAuth 2.0 IdP end-point
    - Device authorization grant flow

- FreeIPA 4.9.10 or later
  - SSSD extends MIT Kerberos pre-authentication mechanism
  - Works with almost all public OAuth 2.0 IdPs
    - Requires Device authorization grant flow (RFC 8628)
- Demo at SambaXP'23, slides

Authentication with



Detailed description is in RHEL IdM guide 'Configuring and managing Identity Management': 8.3. Data flow when authenticating as a user with SSSD in IdM

Use OAuth2 behind Kerberos authentication

- Done already with FreeIPA
  - KDC authenticates user through OAuth2 device authorization grant flow against IdP
  - Issues Kerberos ticket with ‘idp’ authentication indicator
  - PAM module pam_sss_gss can check authentication indicator to limit Kerberos ticket use for PAM authentication and authorization
    - Gives selective SUDO authentication
  - Web browsers can already use Kerberos tickets for single sign-on
  - Use of Kerberos for VPN, SSH, network file systems’ access

Downsides:

Requires FreeIPA deployment

Flood of changes?

- Common
  - A library to handle algorithmic POSIX ID mapping for OAuth2-provided data
    - Handle SIDs and POSIX ID ranges together
- SSSD
  - Identity: Identity provider to talk to OAuth2 IdP
  - Authentication: no change if local KDC adopted
  - Access control: access provider to talk to OAuth2 IdP
- Samba
  - Make MIT Kerberos KDC fully supported
  - Make idmap modules handle multiple ID ranges and manage them automatically
  - Make Samba to support being enrolled to multiple “domains” properly
  - Add OAuth2 idmap/passdb support or rely on external projects

Local KDC adopted?

SambaXP talk on April 18th

Get rid of NTLM or become passwordless: choose both?

Thank you!


sssd
SAMBA
TEAM
FreeIPA
Open Source Identity Management Solution