Identity Synchronization & Authentication in eIRC

This whitepaper outlines the identity synchronization features introduced in eIRC, and describes the two pluggable authentication adapters—**SASL** and **NickServ Identify**—that can be used to strengthen the security of user identities. It covers architectural components, protocol flows, security considerations, and best-practice recommendations for deploying and operating eIRC in production.

1. Introduction

1.1. Background on eIRC

eIRC is a self-hosted web and terminal-based IRC application built to integrate modern web frameworks with traditional IRC networks. It leverages Laravel for user management and configuration, Lua/OpenResty for HTTP APIs, and a C++ client to bridge between web sockets and IRC servers.

1.2. The Need for Secure Identity Synchronization

Because eIRC's web accounts and IRC identities remain separate, two primary security gaps arise:

- Separate credentials and lack of synchronization: Rather than requiring users to manage a distinct NickServ password, eIRC generates a cryptographically secure IRC secret on their behalf and automatically propagates any secret changes from the web application to the IRC service.

- Variable authentication protocols: IRC networks differ in their supported methods (SASL vs. NickServ Identify), adding configuration complexity and potential for misconfiguration.

Our enhancements address these gaps by ensuring that:

- New users' IRC registrations are triggered automatically upon account creation.
- eIRC never propagates user passwords to NickServ; instead, it generates a cryptographically secure IRC secret on behalf of each user and rotates it automatically or on-demand.
- Administrators choose the appropriate authentication method—SASL or NickServ Identify—according to their network's supported protocols and security needs.

2. Identity Synchronization Features

2.1. Automatic Registration on Account Creation

This workflow ensures that new users are immediately registered with NickServ without manual intervention:

- **Laravel Model Hook**: The User model's *created* event dispatches a **RegisterNickServ** job.
- Queueing: Jobs run in the **irc_operations** queue, ensuring rate-limited, background processing.
- **NickServ Register HTTP endpoint:** `POST /irc/nickserv/register` launches a headless IRC client, performs the initial MOTD handshake, and sends input: `PRIVMSG NickServ :REGISTER <password> <email>`.

2.2. SASL Secret Generation & Rotation

- **Secret Creation:** `User::generateSaslSecret()` creates a 32-character secure secret.
- **Reset Command:** `artisan eirc:reset-sasl-secret <realname>` dispatches a **ResetSaslSecret** job.

- **Password-Change API:** `POST /irc/nickserv/password` (nickserv_password.lua) spawns a headless client, sends `PRIVMSG NickServ :SET PASSWORD <new>`, and returns notices.

2.3. Rate Limiting & Security Controls:

- **Nginx Rate-Limiting:** Limits `/irc/nickserv/*` endpoints to mitigate brute-force attacks.
- **Token Binding:** Laravel issues a one-time chat token via Passport, validated in the websocket server by making an API call to laravel.
- **Process Isolation:** Each request spawns a detached, ephemeral IRC client instance.

3. Pluggable Authentication Adapters

eIRC's C++ client supports two strategies via the **AuthStrategy** interface.

3.1. SASL Adapter Flow

- Request a list of the IRC server's capabilities with `CAP LS 302`
- On LS reply containing "**sasl**", send `CAP REQ :sasl`
- On `ACK :sasl`, send `AUTHENTICATE PLAIN <(Base64 of nick\secret)>`
- On a numeric **903** event, send `CAP END`
- **Pros:** Standardized, avoids plaintext, works over TLS.
- **Cons:** Requires network SASL support.

3.2. NickServ Identify Adapter Flow

- Wait for the **MOTD end** (376/422) event
- Send `PRIVMSG NickServ :IDENTIFY <secret>`
- Wait for numeric **900** event or successful **NOTICE**
- **Pros:** Universally supported.
- **Cons:** Sends secret in plaintext **PRIVMSG** (using TLS will ensure it's encrypted).

Note: Both adapters can operate over an encrypted TLS socket connection

4. Security Analysis

4.1. Threat Model

- **Network Observers:** Use TLS on both web and IRC.
- **Endpoint Attackers:** Single-use chat tokens are bound to the Nginx request ID.
- **Brute-Force:** Nginx rate-limits and Passport token lifetimes throttle abuse.

4.2. Recommendations

- **Immediate Actions**
    - Enable TLS on Nginx (HTTPS) and IRC (port 6697).
    - Prefer SASL where available to avoid plaintext exchanges.
- **Ongoing Practices**
    - Rotate secrets regularly.
    - IRC secret management is invisible to users.
    - Users only manage their Laravel issued account in the web interface.
    - Authentication synchronization of the IRC infra is automated.

5. Implementation Overview:

- **Configuration:** `bin/configure` sets up environment variables, supervisord, and the **irc_operations** queue.
- **Web API:** Lua scripts in **routes/api/** provide **/register** and **/password** endpoints.
- **Queue Jobs:** Laravel jobs (**RegisterNickServ**, **ResetSaslSecret**) run background tasks.
- **Client Integration:** C++ client picks **SaslAdapter** or **NickServAdapter** via the **--sasl** flag.

6. Best Practices & Deployment

- Choose **SASL** if supported, falling back to **NickServ Identify**.
- Run workers with least privilege under supervisord.
- Secure API with HTTPS and firewall rules.
- Educate site administrators on secret rotation and security rationale.

7. Conclusion

The identity synchronization features in eIRC bridge Laravel’s user management with IRC’s NickServ service, ensuring that registrations and secret rotations stay in sync. By offering both **SASL** and **NickServ Identify** adapters, eIRC provides flexible and secure authentication strategies suitable for most IRC networks.

Whitepaper prepared by the eIRC development team, May 2025.