Kitty Chat C++

Application de Messagerie Matrix

Master Cybersécurité - Janvier 2026

🔒 Zero-Trust

⚡ Client Natif C++

🌐 Protocole Matrix

Sommaire

Contexte et Objectifs

Objectifs Pédagogiques

- Comprendre un protocole de messagerie moderne (Matrix)
- Maîtriser le déploiement d'infrastructure Linux
- Développer une application native C++ performante
- Implémenter des communications HTTPS sécurisées
- Appliquer les principes Zero-Trust

Contexte du Projet

Kitty Chat est une **application de messagerie instantanée complète**, depuis l'infrastructure serveur jusqu'au client natif. Le projet couvre l'ensemble de la stack technique : backend Linux, protocole de communication, et interface utilisateur Windows.

Pourquoi Matrix ?

Décentralisé

Protocole open-source avec fédération entre serveurs

Sécurisé

Support E2EE avec Olm/Megolm (optionnel)

Standardisé

API REST complète et documentée (spec.matrix.org)

Interopérable

Compatible Element, FluffyChat, et autres clients

02 • STACK TECHNIQUE

Choix Techniques

Synapse (Python)
Serveur Matrix

Implémentation de référence, stable et bien documentée. Supporte la fédération et offre une API complète.

Nginx
Reverse Proxy

Performant, léger, support WebSocket natif. Gère les headers X-Forwarded et les timeouts longs pour le long polling.

Cloudflare Tunnel
Tunnel HTTPS

Aucun port ouvert, protection DDoS intégrée, certificats SSL automatiques. Connexion sortante sécurisée.

C++17 / Win32
Client Natif

Performance native, contrôle total sur le système, faible empreinte mémoire. Pas de dépendances lourdes.

Dear ImGui + DX11
Interface Graphique

Rendu GPU, personnalisation complète de l'UI, multiplateforme. Interface "immédiate" légère et rapide.

WinHTTP + JSON
Communication

API Windows native pour HTTPS, support SSL/TLS intégré. Parsing JSON moderne avec nlohmann/json.

03 • ARCHITECTURE

Architecture Globale

Flux de Données Complet

1. **Kitty Chat Client**
   C++ / ImGui / WinHTTP →
2. **Cloudflare Edge**
   HTTPS / DDoS Protection / SSL →
3. **Cloudflare Tunnel**
   Connexion sortante chiffrée →
4. **cloudflared**
   Agent tunnel local →
5. **Nginx**
   Reverse Proxy :80 →
6. **Synapse**
   Matrix Server :8008 →
7. **SQLite**
   Base de données

Avantages Sécuritaires

- ✓ Aucun port entrant ouvert
- ✓ Protection DDoS Cloudflare
- ✓ SSL/TLS automatique
- ✓ Synapse en localhost uniquement

Performances

- ⚡ Long-polling 30s timeout
- ⚡ CDN global Cloudflare
- ⚡ Compression HTTP activée
- ⚡ Caching intelligent

vault.buffertavern.com

Domaine de production

Matrix Synapse

Installation

```
sudo apt install -y matrix-synapse-py3
```

Installation via dépôt officiel Matrix avec clé GPG pour authentification des paquets.

Création Admin

```
register_new_matrix_user -c homeserver.yaml \ http://localhost:8008 -u admin -p password123 -a
```

Option -a pour les privilèges administrateur complets.

Service Systemd

```
sudo systemctl enable matrix-synapse
sudo systemctl start matrix-synapse
```

Activation au démarrage et gestion automatique du service.

Configuration homeserver.yaml

```
# === IDENTITÉ DU SERVEUR ===
server_name: "vault.buffertavern.com"
# === ÉCOUTE (localhost uniquement) ===
listeners:
- port: 8008
type: http
bind_addresses: ['127.0.0.1']
x_forwarded: true
# === BASE DE DONNÉES ===
database:
name: sqlite3
args:
database: /var/lib/matrix-synapse/homeserver.db
# === INSCRIPTION ===
enable_registration: true
enable_registration_without_verification: true
# === RATE LIMITING ===
rc_message:
per_second: 0.2
burst_count: 10
```

Nginx & Cloudflare Tunnel

Configuration Nginx

```
upstream synapse {
server 127.0.0.1:8008;
keepalive 32;
}
server {
listen 80;
server_name vault.buffertavern.com;
client_max_body_size 50M;
location /_matrix {
proxy_pass http://synapse;
proxy_set_header Host $host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header X-Forwarded-Proto $scheme;
proxy_http_version 1.1;
proxy_set_header Upgrade $http_upgrade;
proxy_set_header Connection "upgrade";
proxy_buffering off;
proxy_read_timeout 600s;
}
}
```

Headers X-Forwarded
IP réelle du client

Timeout 600s
Long polling /sync

Cloudflare Tunnel

Principe : Connexion sortante depuis le serveur vers Cloudflare. Aucun port entrant n'est ouvert.

```
tunnel: a1b2c3d4-e5f6-7890-abcd-ef1234567890
credentials-file: ~/.cloudflared/...json
ingress:
- hostname: vault.buffertavern.com
service: http://localhost:80
```

Avantages Sécuritaires

- **Pas de ports ouverts** - Surface d'attaque minimale
- **Protection DDoS** - Filtrage automatique Cloudflare
- **Certificats SSL auto** - Gérés par Cloudflare
- **Pas de NAT** - Fonctionne derrière tout firewall

Erreur 1033 = Tunnel inactif

Structure et Rendu

Structure du Projet

```
kitty-chat-cpp/
├── CMakeLists.txt
├── src/
│   ├── main.cpp # Point d'entrée
│   ├── matrix_client.cpp # Client Matrix
│   ├── chat_window.cpp # Interface UI
│   └── texture_manager.cpp
├── include/
│   ├── matrix_client.h
│   └── chat_window.h
└── assets/ # Ressources
```

Dépendances CMake

```
FetchContent_Declare(
imgui
GIT_REPOSITORY https://github.com/ocornut/imgui.git
GIT_TAG v1.90.1
)
FetchContent_Declare(
json
GIT_REPOSITORY https://github.com/nlohmann/json.git
GIT_TAG v3.11.3
)
```

Téléchargement automatique, versions garanties, compilation auto.

Chaîne de Rendu

1. **WinMain**
   Création fenêtre Win32
2. **DirectX 11**
   Device + Swap Chain
3. **Dear ImGui**
   Init backends Win32 + DX11
4. **Boucle Principale**
   NewFrame → Render → Present

MatrixClient

Classe MatrixClient

```
class MatrixClient {
private:
std::string m_baseUrl;
std::string m_accessToken;
std::string m_userId;
std::string m_deviceId;
std::string m_syncToken;
std::thread m_syncThread;
std::atomic<bool> m_stopSync;
public:
bool Login(const std::string& user,
const std::string& pwd);
bool Register(const std::string& user,
const std::string& pwd);
bool SendMessage(const std::string& roomId,
const std::string& msg);
void SyncLoop();
bool HttpRequest(const std::string& method,
const std::string& endpoint,
const std::string& body,
std::string& response);
};
```

Authentification

```
json loginRequest = {
{"type", "m.login.password"},
{"identifier", {
{"type", "m.id.user"},
{"user", user}
}},
{"password", password}
};
```

Token d'accès stocké **uniquement en mémoire**.

Synchronisation

```
while (!m_stopSync) {
std::string endpoint =
"/_matrix/client/v3/sync?timeout=30000";
if (!m_syncToken.empty())
endpoint += "&since=" + m_syncToken;
HttpRequest("GET", endpoint, ...);
}
```

Thread dédié + mutex pour les mises à jour UI.

Endpoints Utilisés

POST /login

POST /register

GET /sync

PUT /rooms/{id}/send

POST /createRoom

POST /join/{id}

05 • FRONTEND

Interface et UX

Écran de Login

```
/\_____/\
/ o o \
( == ^ == )
[Utilisateur ]
[Mot de passe ]
[Se connecter]
```

Chat ASCII **animé** qui change selon le focus :

- Yeux ouverts = champ actif
- Yeux fermés = saisie mot de passe
- Œil qui cligne = affichage mot de passe

Thème Visuel

Fond #1a1625	**Primaire** #9d4edd
Secondaire #c77dff	**Accent** #ff6b9d

Fond avec **particules animées** pour un effet dynamique.

Interface de Chat

Sidebar

Liste des salons avec indicateurs de messages non lus

Zone Messages

Bulles colorées (violet pour soi, rose pour autres)

Saisie

Zone de texte avec bouton "Miaou!" pour envoyer

Gestion des Salons

+ **Créer un salon** - Modale avec nom et visibilité
+ **Rejoindre un salon** - Saisie de l'ID ou alias
+ **Historique** - Chargement automatique des messages

60 FPS stable avec VSync

Protocole Matrix

API Client-Server v3

Matrix est un **protocole ouvert et décentralisé** avec une API REST complète en JSON.

POST /login	Authentification
POST /register	Création compte
GET /sync	Synchronisation
PUT /rooms/{id}/send	Envoi message
POST /createRoom	Création salon
POST /join/{id}	Rejoindre salon

Formats d'Identifiants

User ID : @alice:vault.buffertavern.com

Room ID : !room123:vault.buffertavern.com

Event ID : $event456:vault.buffertavern.com

Room Alias : #general:vault.buffertavern.com

Long Polling /sync

1 Client envoie GET

```
GET /_matrix/client/v3/sync?timeout=30000&since=S
```

↓

2 Synapse attend

Jusqu'à 30 secondes pour un nouvel événement

↓

3 Réponse JSON

```
{ "next_batch": "S"", "rooms": {...} }
```

↓

4 Client relance immédiatement

Nouvelle requête avec le token since mis à jour

07 • SÉCURITÉ

Architecture Zero-Trust

Côté Serveur

Aucun port entrant

Cloudflare Tunnel = connexion **sortante**. Pas de NAT, pas de règles firewall.

Firewall UFW

```
ufw default deny incoming
ufw default allow outgoing
```

Synapse localhost only

```
bind_addresses: ['127.0.0.1']
```

Inaccessible depuis l'extérieur, même sans Nginx.

Rate Limiting

0.2 msg/s (burst 10), 0.17 registration/s (burst 3)

Côté Client

HTTPS/TLS obligatoire

Toutes les requêtes passent par WinHTTP avec WINHTTP_FLAG_SECURE.

Tokens en mémoire uniquement

m_accessToken jamais écrit sur disque. Effacé à la déconnexion.

Validation des entrées

Échappement XSS, vérification des réponses HTTP avant parsing JSON.

Principe Zero-Trust

Aucun composant n'est considéré comme fiable par défaut.

Chaque requête est authentifiée et chiffrée.

Tests et Validation

Tableau des Tests

Test	Résultat	Détails
Compilation Windows 10/11	PASS	MSVC 2019/2022, CMake 3.20+
Connexion au serveur Matrix	PASS	Login/Logout réussi
Création de compte	PASS	Validation email/username
Envoi de messages	PASS	Formatage correct
Réception de messages	PASS	Synchronisation temps réel
Création de salon	PASS	Paramètres par défaut OK
Rejoindre un salon	PASS	ID de salon valide
Animations d'interface	PASS	60 FPS stable, VSync OK
Gestion des erreurs réseau	PASS	Reconnexion automatique

Outils de Test

- ✓ Postman (tests API)
- ✓ Wireshark (analyse réseau)
- ✓ Visual Studio Debugger
- ✓ Chrome DevTools

Métriques

100%
Taux de succès

<200ms
Temps de réponse

<5%
CPU

~50MB
Mémoire

Difficultés et Solutions

Tunnel Cloudflare Inactif

Problème : Erreur 1033 côté client, tunnel cloudflared arrêté.

```
cloudflared tunnel run matrix
sudo systemctl restart cloudflared
sudo systemctl status cloudflared
```

Solution : Redémarrage manuel ou supervision systemd.

GIFs Non Chargés

Problème : URLs Tenor invalides, dépendance réseau.

Solution : Retour à l'ASCII art fiable qui fonctionne sans dépendance réseau.

Le chat ASCII est devenu une signature de l'app !

Conflit Macro SendMessage

Problème : Windows définit SendMessage → SendMessageA/W

```
#ifdef SendMessage
#undef SendMessage
#endif
bool SendMessage(const std::string& roomId,
const std::string& msg);
```

Solution : #undef avant la définition de notre méthode.

Parsing JSON

Problème : Crash sur réponses non-JSON (erreur 1033, pages HTML).

```
if (httpCode == 200) {
json response = json::parse(body);
} else {
ShowError("Erreur serveur: " + std::to_string(httpCode));
}
```

Solution : Vérification code HTTP avant parsing.

Enseignements : Ces difficultés ont permis de renforcer la robustesse de l'application et d'implémenter une gestion d'erreurs complète.

Conclusion

Réalisations

Infrastructure complète de messagerie du **backend au client natif**, avec un accent particulier sur la **sécurité Zero-Trust** et les **performances natives**.

🎓 Compétences Acquises

Réseau
HTTP/HTTPS, REST API

Sécurité
TLS/SSL, Zero-Trust

Linux
Admin, Nginx, Systemd

Windows
Win32, WinHTTP, DX11

C++
C++17, Multithreading, ImGui

🚀 Améliorations Futures

- 🚀 **E2EE** - Chiffrement bout-en-bout
- 🚀 **Notifications** - Push natif
- 🚀 **Fichiers** - Envoi de pièces jointes
- 🚀 **Multiplateforme** - Linux/macOS

Merci !

Des questions ?