

Velvet Astro - Unified Cyber Defense Platform

Comprehensive Documentation

Version: 1.0.0

Date: December 18, 2025

Generated By: Antigravity

Table of Contents

- [Executive Summary](#)
- [System Architecture](#)
- [Features & Modules](#)
- [API Reference](#)
- [Frontend Structure](#)
- [Installation & Setup](#)
- [Deployment](#)
- [Troubleshooting](#)

1. Executive Summary

Velvet Astro is a cutting-edge Cyber Defense SaaS platform designed to empower organizations with advanced security capabilities. It bridges the gap between automated threat detection and human operator readiness by combining real-time monitoring, machine learning-driven anomaly detection, and interactive security training.

Key Value Proposition

- Real-time Visibility:** Monitor threats, active attacks, and system health in real-time.
- Proactive Defense:** Utilize deception technology ("honey tokens") and predictive AI to stop attacks before they breach critical systems.
- Human-Centric Security:** Integrated training modules to upskill employees and reduce phishing risks.
- Zero-Trust Architecture:** Rigorous identity verification and role-based access control (RBAC).

2. System Architecture

The platform follows a modern, decoupled **microservices-ready architecture**:

Backend

- Framework:** FastAPI (Python 3.10+) - High performance, async support.
- Database:** SQLAlchemy ORM with SQLite (Local) / PostgreSQL (Production).
- Migrations:** Alembic for database schema management.
- Authentication:** OAuth2 with JWT tokens and Argon2 password hashing.
- Data Science:** Scikit-Learn for anomaly detection/ML models.

Frontend

- Framework:** Next.js 15 (React) - Server-side rendering and static generation.
- Styling:** Tailwind CSS v4 - Utility-first styling for rapid UI development.
- Language:** TypeScript - For type-safe code and better maintainability.
- Visualization:** Recharts for data visualization (charts, graphs).

- **Icons:** Lucide React.

Infrastructure

- **Containerization:** Docker & Docker Compose for consistent environments.
- **Orchestration:** Capable of running on Kubernetes or simpler container services.

Project Structure

```

velvet-astro/
├── backend/                      # Python FastAPI Backend
│   ├── app/                         # API Routes (v1)
│   ├── core/                        # Config, Security, Events
│   ├── db/                           # Database Models & Session
│   ├── schemas/                     # Pydantic Schemas
│   └── services/                   # Business Logic
│
├── tests/                          # Pytest Tests
└── main.py                         # Entry Point
├── frontend/                      # Next.js Frontend
│   ├── app/                         # App Router Pages
│   ├── components/                 # Reusable UI Components
│   └── lib/                          # Utilities & API Clients
└── docker-compose.yml               # Container Orchestration

```

3. Features & Modules

The platform is divided into specialized modules handling different aspects of cyber defense:

1. Identity & Access Management (IAM)

- **Auth Module:** Login, Registration, MFA support.
- **Orgs Module:** Multi-tenancy support for different organizations.
- **RBAC:** Granular permissions for Admins, Analysts, and Users.

2. Detection & Response

- **Anomaly Detection:** ML-based outlier detection in user behavior.
- **Threat Intel:** Integration with external feeds to identify known malicious IPs.
- **Deception:** Deployment of fake assets to trap intruders.
- **Incidents:** Centralized management of security alerts.

3. Analysis & Forensics

- **Forensics:** Immutable logging of security events.
- **Profiling:** Attacker profiling and persona classification.
- **Predictions:** AI-driven forecasting of potential future attack vectors.

4. Training & Awareness

- **Phishing Simulations:** Campaigns to test user vigilance.
- **Training Modules:** Interactive courses on security best practices.

4. API Reference

All API endpoints are prefixed with `/api/v1`. Interactive documentation is available at `/docs` when running the backend.

Key Endpoints

Category	Endpoint Prefix	Description
Auth	<code>/auth</code>	Login, Refresh Token, User Profile
Organizations	<code>/orgs</code>	Manage Org details and members
Monitoring	<code>/monitor</code>	Web traffic and system health monitoring
Training	<code>/training</code>	User training progress and modules
Incidents	<code>/incidents</code>	List and manage security incidents
Defense	<code>/defense</code>	Active defense rules (IP blocking, firewall)
Attacks	<code>/threat</code>	Real-time threat data feed
Anomaly	<code>/anomaly</code>	ML-detected anomalies

5. Frontend Structure

The user interface is built with Next.js App Router:

- `/dashboard` : Main command center showing high-level metrics.
- `/map` : Visual "Threat Map" showing geographical attack origins.
- `/attacks` : Live feed of incoming attacks and their details.
- `/defense` : Configuration of active defense mechanisms.
- `/training` : Employee training dashboard and quiz interface.
- `/reports` : Generation of PDF/CSV security reports (planned).
- `/login` : Secure authentication entry point.

6. Installation & Setup

Prerequisites

- **Python:** 3.10 or higher
- **Node.js:** 18 or higher
- **Git:** Version control
- **Docker:** (Optional) For containerized deployment

A. Local Development

1. Backend Setup

```
cd backend
python -m venv venv           # Create virtual environment
source venv/bin/activate       # Activate (Windows: venv\Scripts\activate)
```

```
pip install -r requirements.txt  
alembic upgrade head          # Run DB migrations  
python -m uvicorn app.main:app --reload
```

Backend is now running at http://localhost:8000

2. Frontend Setup

```
cd frontend  
npm install           # Install dependencies  
npm run dev          # Start Dev Server
```

Frontend is now running at http://localhost:3000

B. Docker Deployment

To spin up the entire stack including database and redis:

```
docker-compose up --build
```

7. Deployment

For production deployment:

1. **Environment Variables:** Ensure all secrets in `.env` are set (DB_URL, SECRET_KEY, etc.).
 2. **HTTPS:** Use a reverse proxy like Nginx or Traefik with SSL termination.
 3. **Database:** Use a managed PostgreSQL instance instead of local SQLite/Postgres container.
 4. **Frontend Build:** Run `npm run build` and `npm start` for optimized performance.
-

8. Troubleshooting

Common Issues:

- **"Failed to fetch" on Frontend:**
 - Ensure Backend is running on port 8000.
 - Check CORS settings in `backend/app/main.py`.
 - Verify `NEXT_PUBLIC_API_URL` in frontend `.env.local` matches backend URL.
 - **Database Errors:**
 - Run `alembic upgrade head` to ensure schema is up to date.
 - Delete `velvet.db` if using SQLite and schema is hopelessly out of sync, then re-run migrations.
-