Cyber Deception Agent

Developer Onboarding Guide

Welcome to the **Cyber Deception Agent** project. This document will help you understand, set up, and start working with the codebase quickly.

---

1. Project Overview

The Cyber Deception Agent is an automated threat-response system that:

- Ingests security alerts mapped to **MITRE ATT&CK** techniques
- Uses **MITRE Engage** to select deception actions
- Applies **LLM-based reasoning** (Claude) or rule-based logic
- Maintains historical memory of alerts, deployments, and attackers
- Provides both CLI and interactive interfaces

High-Level Flow

```
Alert → Agent → Decision Engine → Action Plan
              ↓
           Memory
```

---

2. Repository Structure

```
project_root/
│
├── agent.py           # Main orchestrator
├── main.py            # CLI entry point
├── memory.py          # In-memory state and profiling
├── decision_engine.py # LLM + rule-based logic
├── engage_loader.py   # Engage + ATT&CK data loader
├── schemas.py         # Data models
├── config.py          # Configuration
├── data/
│   └── engage_data.json
└── requirements.txt   # Dependencies (if present)
```

---

3. Core Components

3.1 CyberDeceptionAgent (`agent.py`)

The main system controller.

Responsibilities:

- Initializes subsystems
- Routes alerts to the decision engine
- Stores results in memory
- Exposes status and maintenance APIs

Key methods:

- `initialize()`
- `process_alert()`
- `get_status()`
- `cleanup_memory()`

3.2 Decision Engine (`decision_engine.py`)

Handles threat analysis and action selection.

Two modes:

Mode	Class	Description
LLM	`DecisionEngine`	Uses Claude for reasoning
Rule	`RuleBasedDecisionEngine`	Deterministic fallback

Main workflow:

1. Classify threat
2. Load Engage activities
3. Build memory context
4. Query LLM (if enabled)
5. Parse structured output

3.3 Memory System (`memory.py`)

Maintains system state and intelligence.

Stores:

- Alert history
- Deception deployments
- Attacker profiles

Features:

- Correlation by IP, tactic, technique
- Kill-chain escalation detection
- Sophistication estimation
- Retention cleanup

3.4 Engage Loader (`engage_loader.py`)

Loads and manages Engage framework data.

Provides:

- ATT&CK → Engage mappings
- Activity metadata
- Threat-level constraints

Data source:

```
data/engage_data.json
```

3.5 Schemas (`schemas.py`)

Defines all major data models:

- `AlertInput`
- `ActionPlan`
- `EngageAction`
- `Deployment`
- `AttackerProfile`

All system components communicate using these schemas.

3.6 Configuration (`config.py`)

Centralized configuration.

Includes:

- LLM settings
- Threat thresholds
- Memory retention
- Kill-chain mappings
- Response strategies

Main class:

```
AgentConfig
```

---

4. Installation & Setup

4.1 Requirements

- Python 3.9+
- Anthropic SDK
- Internet access (for LLM mode)

Install dependencies:

```
pip install -r requirements.txt
```

4.2 API Key Configuration (LLM Mode)

The LLM engine uses Anthropic API.

Set your API key:

Linux / macOS

```
export ANTHROPIC_API_KEY="your-key"
```

Windows (PowerShell)

```
setx ANTHROPIC_API_KEY "your-key"
```

Verify:

```
echo $ANTHROPIC_API_KEY
```

---

5. Running the System

5.1 Standard Mode (LLM Enabled)

```
python main.py
```

5.2 Rule-Based Mode

```
python main.py --no-llm
```

5.3 Process a Single Alert

```
python main.py --alert '{
  "attack_id": "T1003",
  "probability": 0.8
}'
```

5.4 Interactive Mode

```
python main.py
```

Commands:

- `alert`
- `status`
- `memory`
- `techniques`
- `trigger`
- `quit`

---

6. Development Workflow

6.1 Adding New Engage Activities

1. Edit `data/engage_data.json`
2. Add activity metadata
3. Map to ATT&CK techniques
4. Restart agent

6.2 Extending Decision Logic

To customize LLM behavior:

- Edit `_build_prompt()`
- Modify `_parse_llm_response()`
- Adjust `THREAT_LEVEL_STRATEGIES`

For rule logic:

- Modify `RuleBasedDecisionEngine.decide()`

6.3 Adding New Correlation Rules

Edit `memory.py`:

- `get_related_alerts()`
- `detect_attack_escalation()`
- `_estimate_sophistication()`

---

7. Testing

7.1 Quick Test

```
python main.py --alert '{
  "attack_id": "T1566",
  "probability": 0.6,
  "affected_assets": ["pc-01"],
  "observed_indicators": {"source_ip": "1.2.3.4"}
}'
```

7.2 Escalation Test

Run multiple alerts in sequence:

1. T1566 (Phishing)
2. T1003 (Credential Dump)
3. T1021 (Lateral Movem**8. Architecture Principles**
4. Modular components
5. Schema-driven communication
6. Pluggable decision engines
7. Centralized configuration
8. Context-aware reasoning