# Advanced Network Threat Detection and Response: A Comprehensive Architecture for the "Sentinel" Project Template

## Executive Summary

The modern cybersecurity landscape is characterized by an asymmetric struggle between attackers, who leverage automation and rapidly evolving exploits, and defenders, who are often constrained by static, signature-based detection methodologies. Traditional Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) struggle to identify zero-day threats, polymorphic malware, and sophisticated "living off the land" techniques that blend with legitimate traffic. To bridge this gap, there is a critical operational need for flexible, intelligent, and autonomous network monitoring solutions that fuse the capabilities of offensive tooling with the analytical power of artificial intelligence.

This report presents the architectural design, technical specification, and implementation guide for "Sentinel," a GitHub project template that integrates Bettercap—a premier network attack and monitoring framework—with a Python-based Artificial Intelligence (AI) analysis engine. The proposed framework functions as a closed-loop Active Defense system. It utilizes Bettercap’s high-performance, event-driven architecture as a sensor to capture and parse network traffic in real-time. This data is streamed via WebSockets to a dedicated analysis subsystem where unsupervised machine learning algorithms, specifically Isolation Forests and Autoencoders, identify anomalous behavior patterns indicative of malicious activity.

Beyond mere detection, the Sentinel framework is engineered for automated response. Leveraging Bettercap’s REST API and host-level firewall controls, the system can autonomously execute mitigation strategies—ranging from isolating compromised endpoints via Wi-Fi deauthentication to blocking malicious IP addresses at the kernel level. This document serves as a definitive guide for security researchers and developers, detailing the necessary directory structures, code logic, feature engineering techniques, and operational considerations required to build and deploy this next-generation network security tool. The analysis draws upon extensive technical documentation, academic research in anomaly detection, and best practices in open-source software development to ensure the resulting template is robust, scalable, and operationally secure.

## 1. Introduction: The Convergence of Offensive and Defensive Engineering

### 1.1 The Evolution of Network Reconnaissance

The paradigm of network reconnaissance has shifted dramatically over the past decade. Early tools like tcpdump and Wireshark provided visibility but lacked the interactivity required for dynamic testing. The subsequent generation of tools, such as Ettercap, introduced active manipulation capabilities like ARP spoofing, yet they were often plagued by performance bottlenecks and instability when handling high-throughput modern networks. Bettercap emerged as a response to these limitations, rewritten entirely in the Go programming language to leverage its superior concurrency model and memory safety.1

Bettercap is frequently categorized solely as an offensive tool used by Red Teams for Man-in-the-Middle (MITM) attacks, credential harvesting, and Wi-Fi coercion. However, this classification overlooks its potential as a highly sophisticated network sensor. Its modular architecture allows for the granular inspection of layers 2 through 7 of the OSI model, making it an ideal foundation for a custom Network Security Monitoring (NSM) solution. By repurposing its offensive capabilities—such as the ability to parse complex protocols or interact with wireless drivers—defenders can gain a level of visibility and control that standard passive sensors cannot match.2

### 1.2 The Necessity of AI in Traffic Analysis

Conventional traffic analysis relies heavily on signature matching (e.g., Snort or Suricata rules). While effective against known threats, this approach is brittle. It fails against encrypted malicious traffic, novel data exfiltration techniques, or "low and slow" beaconing that does not trigger volumetric thresholds. Artificial Intelligence, and specifically unsupervised machine learning, offers a complementary approach. Instead of looking for known "bad" patterns, unsupervised models learn the "normal" baseline of network behavior and flag statistical deviations.4

Integrating AI with Bettercap creates a symbiotic relationship. Bettercap handles the heavy lifting of packet capture, protocol dissection, and event normalization—tasks that are computationally expensive and complex to implement from scratch. The AI engine, relieved of these low-level duties, can focus entirely on high-value tasks: feature extraction, inference, and decision-making. This separation of concerns allows for the creation of a "smart" sensor that is both performant and intelligent.6

### 1.3 Project Goals and Scope

The objective of this report is to define a standardized GitHub project template, herein referred to as "Sentinel," that developers can fork to jumpstart their own AI-driven network security tools. The template aims to satisfy the following requirements:

1. **Robust Monitoring:** Continuous, real-time capture of network events using Bettercap.
2. **AI Integration:** A modular Python pipeline for feature engineering and anomaly detection.
3. **Custom Output:** Structured logging and reporting capabilities beyond standard console output.
4. **Automated Actions:** A logic engine capable of triggering active responses to detected threats.
5. **User Interface:** A modern, web-based dashboard for visualization and control.
6. **Documentation:** Comprehensive guides to ensure usability and maintainability.

The following sections will deconstruct each of these components, providing the theoretical justification and practical implementation details necessary to construct the Sentinel framework.

## 2. Architectural Framework and Design Patterns

### 2.1 The Sidecar Design Pattern

To ensure system stability and modularity, the Sentinel framework adopts the "Sidecar" design pattern, a concept borrowed from cloud-native microservices architecture. In this model, the primary application (the Sensor) and the supporting service (the Analysis Engine) run as separate processes or containers but share the same host context and lifecycle.8

Component A: The Sensor (Bettercap)

Bettercap runs as the primary interface to the hardware. It requires elevated privileges (root or CAP\_NET\_ADMIN) to manipulate network interfaces, set promiscuous mode, and inject packets. Its role is strictly defined as the producer of data and the executor of low-level network commands. It is optimized for I/O operations and concurrency, leveraging Go's goroutines to handle thousands of packets per second without blocking.9

Component B: The Brain (Python AI Engine)

The analysis logic resides in a separate Python process. This decoupling is critical for several reasons:

* **Fault Isolation:** Data science pipelines involving large libraries like Scikit-learn or TensorFlow can be memory-intensive and prone to unhandled exceptions (e.g., during complex matrix operations). If the Python process crashes, it must not bring down the network sensor. The Sidecar pattern ensures that Bettercap continues to capture and buffer events even if the analysis engine is temporarily unavailable.
* **Security Segmentation:** The AI engine typically does not require root privileges. By separating it from the sensor, we adhere to the Principle of Least Privilege. The Python script interacts with the privileged sensor only through a defined, authenticated API, reducing the attack surface.8
* **Ecosystem Utilization:** While Go is excellent for networking, Python is the undisputed lingua franca of data science. This architecture allows developers to utilize the rich ecosystem of Python libraries (Pandas, NumPy, Scikit-learn) without trying to force-fit ML logic into Go.11

### 2.2 Data Flow Pipeline

The flow of information through the Sentinel framework follows a unidirectional pipeline with a feedback control loop:

1. **Ingestion:** Bettercap's net.sniff module captures raw packets from the network interface (e.g., wlan0, eth0).
2. **Normalization:** The net.sniff module dissects the packets (IP, TCP, UDP, HTTP, DNS) and converts them into structured internal event objects.
3. **Transmission:** The events.stream module pushes these events to the api.rest module, which broadcasts them over a WebSocket connection to the Python client.13
4. **Feature Extraction:** The Python client parses the JSON event stream, extracting relevant fields (e.g., packet size, flags, entropy) and transforming them into numerical feature vectors.
5. **Inference:** The feature vectors are passed to the Machine Learning model (e.g., Isolation Forest), which assigns an anomaly score to each event or flow.
6. **Decision:** The Action Engine evaluates the anomaly score against a configurable threshold.
7. **Response (Feedback):** If the score exceeds the threshold, the Action Engine triggers a response—either logging the alert, updating the UI, or sending a command back to Bettercap via the REST API to block the threat.15

### 2.3 Technology Stack Selection

The selection of technologies for the Sentinel template is driven by the need for performance, stability, and ease of development.

* **Core Framework:** **Bettercap** (Go). Chosen for its maturity, stability, and extensive feature set compared to writing a raw sniffer in Python using Scapy, which can be slower in high-throughput environments.1
* **Analysis Language:** **Python 3.10+**. Selected for its robust support for asynchronous programming (asyncio) and machine learning libraries.11
* **Communication:** **WebSockets**. Chosen over HTTP polling or file tailing to ensure millisecond-level latency for real-time threat detection.14
* **Machine Learning:** **Scikit-learn**. Offers efficient implementations of unsupervised algorithms like Isolation Forest and One-Class SVM that are suitable for tabular network data.7
* **Dashboard:** **Streamlit**. Selected for its ability to rapidly build interactive data dashboards using pure Python, lowering the barrier to entry for security researchers who may not be frontend developers.19
* **Database:** **SQLite** (for lightweight deployments) and **InfluxDB** (for high-volume time-series data). This dual support allows the template to scale from a Raspberry Pi to a dedicated monitoring server.21

## 3. The Bettercap Sensor: Configuration and Modules

### 3.1 The net.sniff Module: The Eyes of the Sentinel

The net.sniff module is the primary data source for the framework. It operates by listening to the network interface in promiscuous mode, capturing all traffic visible to the hardware. Unlike standard packet capture tools, Bettercap's sniffer is fully integrated with its event bus, allowing captured packets to be manipulated and broadcasted internally.10

To serve as an effective input for AI analysis, the sniffer must be configured with precision. The default settings are often optimized for human readability rather than machine consumption. The following parameters are essential for the Sentinel template:

* **net.sniff.verbose true**: By default, Bettercap operates in a "quiet" mode to avoid flooding the console. Setting verbose to true is mandatory for our use case. It instructs the module to parse every captured packet and emit a detailed net.sniff.\* event containing the full layer hierarchy (Ethernet, IP, Transport, Application). Without this, the AI engine receives only metadata, missing the payload data required for deep inspection.10
* **net.sniff.local true**: This parameter ensures that traffic originating from or destined for the sensor itself is captured. This is crucial for detecting attacks targeting the monitoring node or for analyzing traffic when the node is acting as a gateway (MITM).10
* **net.sniff.filter**: Bettercap supports Berkeley Packet Filter (BPF) syntax. The template must include a pre-configured filter to exclude the sensor's own management traffic (e.g., SSH, API WebSocket) to effectively prevent "feedback loops" where the sensor captures its own log stream, leading to a denial of service.10

### 3.2 The events.stream and api.rest Modules

The events.stream module aggregates logs, discovery events, and sniffer output into a unified timeline. While it can write to a file (events.stream.output), disk I/O is often a bottleneck. The architecture relies on the api.rest module to expose this stream over the network.13

The REST API serves two distinct functions in the Sentinel framework:

1. **Data Egress (Streaming):** By setting api.rest.websocket true, the /api/events endpoint upgrades standard HTTP requests to a persistent WebSocket connection. This allows Bettercap to push events to the Python engine as they happen, ensuring near real-time processing capabilities.13
2. **Command Ingress (Control):** The API exposes a /api/session/cmd endpoint that accepts Bettercap commands via POST requests. This is the mechanism by which the Python AI engine executes active response actions, such as wifi.deauth or net.probe on.13

### 3.3 Automation with Caplets

To ensure reproducibility and ease of deployment, the Sentinel template utilizes "Caplets"—Bettercap's scripting language. Caplets (.cap files) allow researchers to define the exact state of the sensor upon initialization, replacing manual configuration steps.15

The project template includes a modular Caplet system:

* **sentinel.cap**: The master entry point. It sets environment variables and includes sub-caplets.
* **config/api.cap**: Configures the API server, generating random credentials or reading them from secure environment variables, and enables HTTPS to secure the communication channel.13
* **config/sniff.cap**: Sets up the net.sniff module with the appropriate BPF filters and verbosity settings tailored for AI ingestion.10

Example api.cap configuration structure:

Bash

# Secure API Configuration  
set api.rest.username ${env.SENTINEL\_USER}  
set api.rest.password ${env.SENTINEL\_PASS}  
set api.rest.port 8081  
set api.rest.websocket true  
api.rest on

This approach treats "Infrastructure as Code," allowing the sensor configuration to be version-controlled alongside the software.24

## 4. Data Ingestion and Engineering Strategy

### 4.1 Asynchronous WebSocket Client

The link between Bettercap and the analysis engine is a high-velocity data pipe. To handle this, the Python component employs an asynchronous WebSocket client using the asyncio and websockets libraries. Traditional synchronous (blocking) networking is insufficient here; if the machine learning inference takes even a fraction of a second, a blocking client would stop reading from the socket, causing the TCP receive buffer to fill and eventually drop packets, leading to data loss.14

The implementation details of the Connector class (src/core/connector.py) are critical:

* **Authentication:** The client must construct the HTTP Basic Auth header using the credentials defined in the Caplet.
* **Reconnection Logic:** Network sensors operate for extended periods. The client must implement robust error handling to detect connection drops (e.g., if Bettercap restarts) and automatically reconnect using an exponential backoff strategy to avoid overwhelming the server.26
* **Message Dispatch:** The client acts as a dispatcher, routing incoming JSON messages to appropriate queues based on their tags (e.g., net.sniff to the ML queue, sys.log to the logging queue).23

### 4.2 Handling Complex Event Structures

Bettercap events are structured as nested JSON objects. The schema varies depending on the event type. For net.sniff events, the payload contains the parsed packet layers. However, the structure is dynamic; a TCP packet will have a tcp object, while a UDP packet has udp.

The DataProcessor module acts as a normalization layer. It flattens these nested structures into a standardized dictionary format expected by the feature extractor.

* **Protocol Normalization:** It maps protocol-specific fields (like tcp.dst\_port and udp.dst\_port) to generic feature fields (destination\_port).
* **Binary Decoding:** Bettercap encodes binary payloads (like packet content) as base64 or hexadecimal strings to ensure safe transport over JSON. The ingestion layer must decode these fields back into raw bytes to perform content analysis, such as calculating entropy.23

### 4.3 Advanced Feature Engineering with Scapy

While Bettercap's JSON output provides parsed metadata, advanced anomaly detection often requires "derived" features that are not explicitly present in the packet headers. To achieve this, the Sentinel template integrates the **Scapy** library.28

The integration strategy involves "Packet Reconstruction." The Python engine takes the raw hex payload provided by Bettercap and reconstructs a Scapy packet object:

Python

from scapy.all import IP  
packet = IP(bytes.fromhex(bettercap\_event['data']['packet']['payload']))

Once the packet is in the Scapy object model, we can extract sophisticated features:

* **Inter-Arrival Time (IAT):** Calculating the time delta between the current packet and the previous packet in the same flow. Consistent, low-variance IAT is a strong indicator of automated beacons or C2 channels.30
* **Payload Entropy:** Calculating the Shannon entropy of the payload. High entropy often indicates encryption or compressed data, which can identify data exfiltration in protocols that should be cleartext (like DNS or HTTP).30
* **TCP Flag Ratios:** Analyzing the ratio of SYN to ACK packets within a time window to detect scanning activity (SYN floods).31

This hybrid approach leverages Bettercap for high-speed capture and Scapy for deep analytical precision.

## 5. Artificial Intelligence and Anomaly Detection

### 5.1 The Case for Unsupervised Learning in Security

The application of AI in network security is often bifurcated into supervised and unsupervised learning. Supervised learning, which relies on labeled datasets of known attacks, faces significant challenges in operational environments. "Malicious" traffic is statistically rare compared to "benign" traffic, leading to severe class imbalance. Furthermore, supervised models are fundamentally incapable of detecting zero-day attacks—threats they have never seen before.4

The Sentinel framework, therefore, prioritizes **Unsupervised Anomaly Detection**. The premise is to model the "normal" behavior of the specific network environment where the sensor is deployed. Any traffic pattern that deviates significantly from this learned baseline is flagged as an anomaly. This approach allows the system to detect novel attacks, misconfigurations, and insider threats without requiring a database of attack signatures.7

### 5.2 Algorithm Selection: Isolation Forest

Among the various unsupervised algorithms, the **Isolation Forest** (iForest) is selected as the default model for the Sentinel template. Unlike density-based algorithms (like DBSCAN) which look for clusters of normal points, iForest explicitly isolates anomalies.

Operational Theory:

The algorithm constructs an ensemble of random decision trees. Because anomalies are "few and different," they are susceptible to isolation. In a random partitioning of the feature space, anomalous points are isolated closer to the root of the tree (requiring fewer splits) than normal points, which are clustered deep in the leaves. The average path length from the root to the point serves as the anomaly score.18

**Advantages for Network Security:**

* **Efficiency:** Isolation Forest has a linear time complexity ($O(n)$), making it highly scalable for high-velocity data streams.
* **High Dimensionality:** It performs robustly even when the feature vector contains many dimensions (ports, flags, sizes, timings), a common characteristic of network data.7
* **Parameter Tuning:** The contamination parameter allows operators to tune the sensitivity of the model based on the expected "noise" level of the network.32

### 5.3 Autoencoders for Deep Analysis

The modular design of the template (src/analysis/models.py) allows for the substitution of the machine learning backend. As an advanced alternative, the template supports **Autoencoders** using TensorFlow or PyTorch. An Autoencoder is a neural network trained to compress (encode) the input data into a lower-dimensional latent space and then reconstruct (decode) it back to the original input.

The model is trained exclusively on benign traffic. When it encounters a malicious packet, the compression-decompression process fails to accurately reconstruct the input because the pattern was not present in the training set. The **Reconstruction Error** (Mean Squared Error between input and output) serves as the anomaly score. This method is particularly effective for detecting subtle, non-linear anomalies in encrypted traffic patterns that linear models might miss.33

### 5.4 Feature Vector Construction and Normalization

The bridge between the raw packet and the ML model is the Feature Vector. The FeatureExtractor class converts the qualitative packet data into a quantitative NumPy array.

**Key Features:**

1. **Packet Size:** (Integer) Can indicate buffer overflow attempts or command injection.
2. **Source/Destination Port:** (Categorical/Integer) While ports are numbers, they represent categories. The template handles this by tracking "Service Ports" (0-1024) vs "Ephemeral Ports".
3. **Protocol ID:** (One-hot Encoded) TCP, UDP, ICMP.
4. **Flow Duration:** (Float) For aggregated flow records.

Normalization (Scaling):

Machine learning algorithms are sensitive to the scale of input data. A packet size of 1500 and a TCP flag value of 1 are numerically vastly different. The pipeline employs StandardScaler (Z-score normalization) or MinMaxScaler to scale all features to a common range (e.g., 0 to 1 or mean 0, variance 1). This ensures that features with larger raw values do not dominate the model's decision-making process.30

## 6. Automated Response and Active Defense

### 6.1 The Active Defense Feedback Loop

Detection without response is merely observation. To transform the system into a "Sentinel," it must possess the capability to react. The Sentinel framework implements a closed-loop feedback mechanism known as the OODA loop (Observe, Orient, Decide, Act) at machine speed. The ActionEngine module subscribes to the anomaly alerts generated by the AI core and executes countermeasures based on the confidence level and severity of the detection.

### 6.2 Layer 2 Mitigation: Leveraging Bettercap

Since Bettercap is already running as the privileged sensor, the most efficient response mechanism is to leverage its offensive capabilities for defensive purposes. The Python engine sends commands back to Bettercap via the REST API to manipulate Layer 2 (Data Link) traffic.

Scenario: Wireless Containment

If the AI detects a device exhibiting behavior consistent with a "Evil Twin" attack or a deauthentication flood:

1. **Detection:** The model flags a spike in wifi.client.deauth events associated with a specific MAC address.
2. **Action:** The Python script sends a POST request to /api/session/cmd with the payload wifi.deauth <Attacker\_MAC>.
3. **Effect:** The attacker is disconnected from the network, neutralizing the threat immediately. This "counter-deauth" is a standard capability of Bettercap that creates a defensive quarantine.35

Scenario: ARP Isolation

If an internal host is identified as compromised (e.g., performing ARP spoofing or scanning), the system can use the arp.ban command. This effectively removes the host from the network by instructing the gateway to ignore its traffic, isolating the infection without requiring switch-level access.2

### 6.3 Layer 3 Mitigation: Host Firewall Integration

For threats targeting the sensor itself or passing through it (if the sensor is acting as a gateway/router), the framework integrates with the Linux kernel's firewall via the python-iptables library.

Dynamic Blocking:

The Responder class manages a dynamic list of block rules. When a malicious IP is identified with high confidence:

Python

import iptc  
chain = iptc.Chain(iptc.Table(iptc.Table.FILTER), "INPUT")  
rule = iptc.Rule()  
rule.src = malicious\_ip  
rule.target = iptc.Target(rule, "DROP")  
chain.insert\_rule(rule)

This blocks the attacker at the kernel level, preventing any further interaction with the application layer.37

Temporal Rules:

To prevent permanent denial-of-service due to false positives, the template implements "Time-Based Blocking." Block rules are added with a timestamp, and a background thread monitors these rules, automatically removing them after a configurable cooldown period (e.g., 15 minutes). This ensures the system fails open rather than closed over long periods.39

## 7. Visualization: The Sentinel Dashboard

### 7.1 Streamlit Implementation

Visualizing network traffic and anomaly scores is essential for operator trust and investigation. The Sentinel template utilizes **Streamlit**, a Python framework that allows for the rapid creation of data-centric web applications. Streamlit is ideal for this use case as it integrates natively with Pandas and Scikit-learn, allowing the dashboard to render complex charts directly from the analysis dataframes without requiring separate frontend code (HTML/JavaScript).19

### 7.2 Dashboard Components

The dashboard (src/ui/app.py) is designed to provide a comprehensive operational picture:

A. Real-Time Telemetry HUD:

Top-level metrics display the current health of the network:

* **Packet Rate:** Packets per second (PPS).
* **Bandwidth:** Bits per second (BPS).
* **Threat Level:** A color-coded indicator (Green/Yellow/Red) derived from the aggregate anomaly score of the last minute.

B. The Anomaly Timeline:

A dynamic line chart plots the anomaly score over time. Spikes in this chart provide immediate visual correlation with network events. Operators can zoom into specific time windows to investigate what traffic caused a spike.41

C. Feature Importance View:

One of the challenges with AI is "explainability." Why was this packet flagged? The dashboard includes an "Explanation" panel. Using simple statistical heuristics (e.g., "Feature X deviated 3 standard deviations from the mean"), it translates the model's mathematical output into human-readable text:

* *Example:* "Anomaly Detected: Source IP 192.168.1.55. Reason: Packet Rate (5000/s) is 10x higher than the baseline."

D. Control Plane:

The dashboard provides manual overrides for the autonomous system:

* **Sensitivity Slider:** Adjusts the contamination parameter of the Isolation Forest in real-time.
* **Arm/Disarm Switch:** Toggles the Active Response module, allowing the system to run in "Passive Monitoring" mode during sensitive operations.

## 8. Data Persistence and Logging Strategy

### 8.1 Database Architecture

Network traffic generates massive amounts of data. The Sentinel template supports a dual-database strategy to balance performance and simplicity.

Primary Storage: SQLite

For metadata, configuration, and anomaly logs, the template uses SQLite. It is serverless, zero-configuration, and stores data in a single file (data/db/sentinel.db). This makes the system highly portable and suitable for embedded deployments like Raspberry Pis.21

Metric Storage: InfluxDB

For high-velocity time-series data (packet rates, byte counts, raw feature vectors), SQLite can become a bottleneck. The template includes an optional connector for InfluxDB, a database optimized for time-stamped data. This allows for long-term retention of traffic metrics without degrading performance, enabling historical trend analysis.21

### 8.2 Log Rotation and Management

To prevent disk exhaustion, especially when capturing PCAP files (net.sniff.output), the template includes a log rotation utility. This Python module runs as a background service, monitoring the data/pcaps/ directory. It compresses old capture files (.pcap.gz) and deletes files older than a configured retention period (e.g., 7 days) or when disk usage exceeds a threshold.23

## 9. Deployment and Operational Security

### 9.1 Containerization with Docker

To ensure consistency across different environments (dev laptops, cloud servers, edge devices), the Sentinel template is fully containerized.

The Dockerfile Strategy:

The build process uses a multi-stage Dockerfile to minimize image size:

1. **Builder Stage:** Uses golang:alpine to compile the latest version of Bettercap from source.
2. **Runtime Stage:** Uses python:3.10-slim. It installs the necessary system libraries (libpcap-dev, iptables) and Python dependencies (scikit-learn, websockets, streamlit).
3. **Integration:** The compiled Bettercap binary is copied into the Python runtime image.

Privileged Execution:

Because Bettercap interacts with network hardware, the Docker container must be run with specific privileges:

Bash

docker run -it --net=host --cap-add=NET\_ADMIN -v $(pwd)/data:/app/data sentinel:latest

* --net=host: Allows the container to see the host's network interfaces.
* --cap-add=NET\_ADMIN: Grants permission to modify network settings (promiscuous mode, iptables).9

### 9.2 Operational Security (OpSec)

Running a network sensor implies risks. If the sensor is compromised, it becomes a powerful pivot point for an attacker. The template enforces strict OpSec measures:

* **Interface Binding:** The API and Dashboard are bound to 127.0.0.1 (localhost) by default. Remote access should be achieved via SSH Tunnels, not by exposing these ports to the network.9
* **Stealth Mode:** The default Caplet disables active probing (net.probe off) to minimize the sensor's electronic footprint. Passive sniffing is undetectable by standard scans.
* **Least Privilege:** While Bettercap runs as root (inside the container), the Python analysis script is designed to run as a standard user where possible, interacting with the system only through the defined API.

## 10. Project Template Structure and Documentation

### 10.1 Directory Hierarchy

The project structure follows industry standards for Python application development, ensuring maintainability and scalability.8

sentinel/

├── bin/ # Startup scripts and wrappers

│ └── start\_sentinel.sh

├── caplets/ # Bettercap configuration scripts

│ ├── listener.cap # Main sensor config

│ ├── api.cap # API security config

│ └── filters.cap # BPF filter definitions

├── config/ # YAML configuration files

│ ├── settings.yaml # AI thresholds, DB paths

│ └── logging.conf

├── data/ # Persistent storage (Gitignored)

│ ├── db/ # SQLite/InfluxDB data

│ ├── models/ # Serialized ML models (.joblib)

│ └── pcaps/ # Raw packet captures

├── docs/ # Project documentation

│ ├── ARCHITECTURE.md

│ └── DEPLOYMENT.md

├── src/ # Source code

│ ├── init.py

│ ├── core/ # Core logic (Connection, Event Loop)

│ │ ├── connector.py

│ │ └── responder.py

│ ├── analysis/ # Data Science logic

│ │ ├── features.py # Feature extraction

│ │ └── models.py # ML wrappers

│ └── ui/ # Dashboard code

│ └── app.py

├── tests/ # Unit and Integration tests

│ ├── test\_features.py

│ └── test\_api.py

├── Dockerfile # Container definition

├── Makefile # Build automation

├── requirements.txt # Python dependencies

└── README.md # Overview and Quickstart

### 10.2 Documentation Strategy

The docs/ folder treats documentation as a first-class citizen.

* **Architecture Guide:** Explains the Sidecar pattern and data flow for developers extending the tool.
* **Deployment Guide:** Step-by-step instructions for installing on Kali Linux, Ubuntu, and Raspberry Pi.
* **Model Card:** Documents the AI model used (Isolation Forest), the features it uses, and its limitations (e.g., susceptibility to poisoning attacks).

## 11. Implementation Guide: Building the Core

### 11.1 Defining the Caplet (listener.cap)

This script orchestrates the Bettercap session. It initializes the API, sets up the sniffer, and applies necessary filters.

Bash

# listener.cap  
# Initialize API with secure credentials from ENV  
set api.rest.username ${env.SENTINEL\_USER}  
set api.rest.password ${env.SENTINEL\_PASS}  
set api.rest.port 8081  
set api.rest.websocket true  
api.rest on  
  
# Configure Sniffer for AI Ingestion  
set net.sniff.verbose true  
set net.sniff.local true  
# Vital: Filter out API traffic to prevent feedback loops  
set net.sniff.filter "not port 8081"  
net.sniff on  
  
# Optimization: Ignore noisy events we don't need  
events.ignore sys.log  
events.ignore endpoint.lost

### 11.2 The Python Connector (src/core/connector.py)

This module handles the persistent WebSocket connection.

Python

import asyncio  
import websockets  
import json  
import logging  
  
class BettercapConnector:  
 def \_\_init\_\_(self, uri, username, password):  
 self.uri = uri  
 # Construct Basic Auth Header  
 token = f"{username}:{password}"  
 self.auth\_header = {  
 "Authorization": f"Basic {base64.b64encode(token.encode()).decode()}"  
 }  
 self.queue = asyncio.Queue()  
  
 async def connect(self):  
 while True:  
 try:  
 async with websockets.connect(self.uri, extra\_headers=self.auth\_header) as ws:  
 logging.info("Connected to Sentinel Sensor")  
 while True:  
 msg = await ws.recv()  
 data = json.loads(msg)  
 # Push to async queue for processing  
 await self.queue.put(data)  
 except Exception as e:  
 logging.error(f"Connection lost: {e}. Retrying...")  
 await asyncio.sleep(5) # Exponential backoff logic here

### 11.3 Feature Extraction (src/analysis/features.py)

This module converts raw event JSON into Scikit-learn compatible vectors.

Python

import numpy as np  
from scapy.all import IP, TCP  
  
def extract\_features(event):  
 """  
 Transforms a Bettercap net.sniff event into a feature vector.  
 Vector: [packet\_size, is\_tcp, is\_udp, dst\_port\_norm, entropy]  
 """  
 try:  
 # Extract metadata directly from JSON for speed  
 pkt\_data = event['data']['packet']  
 size = int(pkt\_data.get('length', 0))  
   
 # Determine protocol (One-Hot Encoding concept)  
 is\_tcp = 1 if 'tcp' in pkt\_data else 0  
 is\_udp = 1 if 'udp' in pkt\_data else 0  
   
 # Port normalization (simplified)  
 dst\_port = 0  
 if is\_tcp:  
 dst\_port = int(pkt\_data['tcp']['dst\_port'])  
 elif is\_udp:  
 dst\_port = int(pkt\_data['udp']['dst\_port'])  
   
 # Example: Simple vector construction  
 return np.array([size, is\_tcp, is\_udp, dst\_port])  
   
 except KeyError:  
 return None

## Conclusion

The "Sentinel" project template represents a paradigm shift in how we approach network tool development. By moving beyond the limitations of monolithic, signature-based architectures and embracing a modular, AI-driven design, we empower security researchers with a tool that is proactive rather than reactive. The integration of Bettercap's robust sensor capabilities with Python's advanced analytical ecosystem creates a framework that is flexible enough to adapt to modern threats and accessible enough to be deployed by a wide range of professionals. This report provides the blueprint; the code serves as the foundation for the next generation of autonomous network defense.

## References

1

#### Bibliografia

1. bettercap | Kali Linux Tools, accesso eseguito il giorno gennaio 4, 2026, <https://www.kali.org/tools/bettercap/>
2. Introduction - Bettercap, accesso eseguito il giorno gennaio 4, 2026, <https://www.bettercap.org/project/introduction/>
3. bettercap | bettercap, accesso eseguito il giorno gennaio 4, 2026, <https://www.bettercap.org/>
4. Behavioral Machine Learning for Network Anomaly Detection - Fidelis Security, accesso eseguito il giorno gennaio 4, 2026, <https://fidelissecurity.com/threatgeek/network-security/network-behavior-anomaly-detection-at-scale/>
5. Network Anomaly Detection: A Comprehensive Guide - Kentik, accesso eseguito il giorno gennaio 4, 2026, <https://www.kentik.com/kentipedia/network-anomaly-detection/>
6. Anomaly Detection in Machine Learning Using Python | The PyCharm Blog, accesso eseguito il giorno gennaio 4, 2026, <https://blog.jetbrains.com/pycharm/2025/01/anomaly-detection-in-machine-learning/>
7. Practical Anomaly Detection using Python and scikit-learn | My Battles With Technology, accesso eseguito il giorno gennaio 4, 2026, <https://blog.shellnetsecurity.com/posts/2025/anomaly-detection-python-scikit-learn/>
8. Professional Python Project Architecture: A Comprehensive Guide for Cybersecurity Engineers | by Mohamed Gebril | Medium, accesso eseguito il giorno gennaio 4, 2026, <https://medium.com/@moh.ahmed.gebril/professional-python-project-architecture-a-comprehensive-guide-for-cybersecurity-engineers-16f55b066dd2>
9. How to Use Bettercap for Network Penetration Testing ? Beginner's Guide with Commands and Use Cases - Web Asha Technologies, accesso eseguito il giorno gennaio 4, 2026, <https://www.webasha.com/blog/how-to-use-bettercap-for-network-penetration-testing-beginners-guide-with-commands-and-use-cases>
10. net.sniff - net.fuzz - Bettercap, accesso eseguito il giorno gennaio 4, 2026, <https://www.bettercap.org/modules/ethernet/netsniff/>
11. Python for Cybersecurity: Key Use Cases and Tools - Panther | The Security Monitoring Platform for the Cloud, accesso eseguito il giorno gennaio 4, 2026, <https://panther.com/blog/python-for-cybersecurity-key-use-cases-and-tools>
12. Python for cybersecurity: use cases, tools and best practices - SoftTeco, accesso eseguito il giorno gennaio 4, 2026, <https://softteco.com/blog/python-for-cybersecurity>
13. api.rest - bettercap, accesso eseguito il giorno gennaio 4, 2026, <https://www.bettercap.org/modules/core/apirest/>
14. Consume a real-time Media Stream using WebSockets, Python, and Flask | Twilio, accesso eseguito il giorno gennaio 4, 2026, <https://www.twilio.com/docs/voice/tutorials/consume-real-time-media-stream-using-websockets-python-and-flask>
15. Interactive Session - Bettercap, accesso eseguito il giorno gennaio 4, 2026, <https://www.bettercap.org/usage/interactive_session/>
16. Custom active response scripts - Wazuh documentation, accesso eseguito il giorno gennaio 4, 2026, <https://documentation.wazuh.com/current/user-manual/capabilities/active-response/custom-active-response-scripts.html>
17. Websockets: Streaming input and output using websockets | AutoGen 0.2, accesso eseguito il giorno gennaio 4, 2026, <https://microsoft.github.io/autogen/0.2/docs/notebooks/agentchat_websockets/>
18. Anomaly Detection in Python Using Isolation Forest | CodeSignal Learn, accesso eseguito il giorno gennaio 4, 2026, <https://codesignal.com/learn/courses/data-cleaning-and-validation-for-machine-learning/lessons/anomaly-detection-in-python-using-isolation-forest>
19. Streamlit • A faster way to build and share data apps, accesso eseguito il giorno gennaio 4, 2026, <https://streamlit.io/>
20. Real-Time Network Traffic Analysis Dashboard with Streamlit | by Simanga Mchunu, accesso eseguito il giorno gennaio 4, 2026, <https://medium.com/@simanga.mchunu/real-time-network-traffic-analysis-dashboard-with-streamlit-0d1e0e4c1c6c>
21. InfluxDB vs SQLite comparison - PeerSpot, accesso eseguito il giorno gennaio 4, 2026, <https://www.peerspot.com/products/comparisons/influxdb_vs_sqlite>
22. Database - BAC0 documentation, accesso eseguito il giorno gennaio 4, 2026, <https://bac0.readthedocs.io/en/latest/database.html>
23. events.stream - Bettercap, accesso eseguito il giorno gennaio 4, 2026, <https://www.bettercap.org/modules/core/eventsstream/>
24. caplets | bettercap, accesso eseguito il giorno gennaio 4, 2026, <https://www.bettercap.org/modules/core/caplets/>
25. Python WebSocket connection script - Finxact.io, accesso eseguito il giorno gennaio 4, 2026, <https://console-sandbox-server.dev-10.finxact.io/docs/document-detail/document/python-websocket-connection-script/>
26. WebSocket Testing Essentials: Strategies and Code for Real-Time Apps - The Green Report, accesso eseguito il giorno gennaio 4, 2026, <https://www.thegreenreport.blog/articles/websocket-testing-essentials-strategies-and-code-for-real-time-apps/websocket-testing-essentials-strategies-and-code-for-real-time-apps.html>
27. 7.2. Feature extraction — scikit-learn 1.8.0 documentation, accesso eseguito il giorno gennaio 4, 2026, <https://scikit-learn.org/stable/modules/feature_extraction.html>
28. Scapy, accesso eseguito il giorno gennaio 4, 2026, <https://scapy.net/>
29. Python Network Programming: Forging and Sniffing Packets with Scapy - ExamCollection, accesso eseguito il giorno gennaio 4, 2026, <https://www.examcollection.com/blog/python-network-programming-forging-and-sniffing-packets-with-scapy/>
30. How to Build a Real-Time Intrusion Detection System with Python and Open-Source Libraries - freeCodeCamp, accesso eseguito il giorno gennaio 4, 2026, <https://www.freecodecamp.org/news/build-a-real-time-intrusion-detection-system-with-python/>
31. Exploring the World of Packets with Scapy: A Beginner's Hands-on Journey to Understanding - DEV Community, accesso eseguito il giorno gennaio 4, 2026, <https://dev.to/yo-shi/exploring-the-world-of-packets-with-scapy-a-beginners-hands-on-journey-to-understanding-1d0g>
32. Anomaly Detection in Python with Isolation Forest - DigitalOcean, accesso eseguito il giorno gennaio 4, 2026, <https://www.digitalocean.com/community/tutorials/anomaly-detection-isolation-forest>
33. Unsupervised Network Anomaly Detection with Autoencoders and Traffic Images The research presented in this paper was partially funded by the project “ISEEYOO - arXiv, accesso eseguito il giorno gennaio 4, 2026, <https://arxiv.org/html/2505.16650v1>
34. How I Built a Real-Time Anomaly Detection System for Enterprise Networks Using Python and Machine Learning | by Ndhilani Simbine | Medium, accesso eseguito il giorno gennaio 4, 2026, <https://medium.com/@ndhilani.simbine/how-i-built-a-real-time-anomaly-detection-system-for-enterprise-networks-using-python-and-machine-0078d8a26e84>
35. WiFi | bettercap, accesso eseguito il giorno gennaio 4, 2026, <https://www.bettercap.org/modules/wifi/>
36. Bettercap - Hackviser, accesso eseguito il giorno gennaio 4, 2026, <https://hackviser.com/tactics/tools/bettercap>
37. deepin-community/python-iptables - GitHub, accesso eseguito il giorno gennaio 4, 2026, <https://github.com/deepin-community/python-iptables>
38. Firewall Rules: iptables with Python - Tutorial | Krython, accesso eseguito il giorno gennaio 4, 2026, <https://krython.com/tutorial/python/firewall-rules-iptables-with-python/>
39. accesso eseguito il giorno gennaio 4, 2026, <https://github.com/SudoJvck/Block_IP#:~:text=Block_IP%20is%20a%20Python%20script,them%20from%20incoming%20network%20traffic.>
40. Streamlit Python: Tutorial - DataCamp, accesso eseguito il giorno gennaio 4, 2026, <https://www.datacamp.com/tutorial/streamlit>
41. How to Build a Real-time Network Traffic Dashboard with Python and Streamlit, accesso eseguito il giorno gennaio 4, 2026, <https://www.freecodecamp.org/news/build-a-real-time-network-traffic-dashboard-with-python-and-streamlit/>
42. Personal project: I'm logging sensor data to Sqlite on a Raspberry Pi. I want to make some pretty visuals with R using the data, but unsure about my options for making the data accessible to R. : r/dataengineering - Reddit, accesso eseguito il giorno gennaio 4, 2026, <https://www.reddit.com/r/dataengineering/comments/10oqjgf/personal_project_im_logging_sensor_data_to_sqlite/>
43. What is the best project structure for a Python application? [closed] - Stack Overflow, accesso eseguito il giorno gennaio 4, 2026, <https://stackoverflow.com/questions/193161/what-is-the-best-project-structure-for-a-python-application>
44. All Hail Bettercap 2.0, One Tool to Rule Them All. | evilsocket, accesso eseguito il giorno gennaio 4, 2026, <https://www.evilsocket.net/2018/02/27/All-hail-bettercap-2-0-one-tool-to-rule-them-all/>
45. Real-Time Anomaly Detection With Python - Towards Data Science, accesso eseguito il giorno gennaio 4, 2026, <https://towardsdatascience.com/real-time-anomaly-detection-with-python-36e3455e84e2/>
46. Building a Real-time Laptop Monitoring System with Flask and Redis - DEV Community, accesso eseguito il giorno gennaio 4, 2026, <https://dev.to/bansikah/building-a-real-time-laptop-monitoring-system-with-flask-and-redis-3mch>
47. How to Add Graphs to Flask apps - Python - GeeksforGeeks, accesso eseguito il giorno gennaio 4, 2026, <https://www.geeksforgeeks.org/python/how-to-add-graphs-to-flask-apps/>
48. Which DataBase for weather logging - Raspberry Pi Forums, accesso eseguito il giorno gennaio 4, 2026, <https://forums.raspberrypi.com/viewtopic.php?t=373651>
49. 8 Anomaly Detection Algorithms to Know | Built In, accesso eseguito il giorno gennaio 4, 2026, <https://builtin.com/machine-learning/anomaly-detection-algorithms>
50. Testing WebSockets - FastAPI, accesso eseguito il giorno gennaio 4, 2026, <https://fastapi.tiangolo.com/advanced/testing-websockets/>
51. unittest.mock — getting started — Python 3.14.2 documentation, accesso eseguito il giorno gennaio 4, 2026, <https://docs.python.org/3/library/unittest.mock-examples.html>