Undergrad Campus Recruitement Assignment

24-Hour Take-Home Challenge

---

0. Purpose & Ground Rules

You are being evaluated primarily on **how you think**, not on whether you “finish” everything.

We’re mostly interested in:

- How you break down an unfamiliar problem.
- How you prioritize limited time.
- How you communicate uncertainty and limitations.
- How you use analytics to support security decisions.

Integrity & Use of Tools

You **may not** use generative AI tools (ChatGPT, Copilot, Gemini, etc.) to write code, configuration, documentation, or prose for this assignment.

You **may** use:

- Library documentation (e.g., `pandas`, `scikit-learn`, `docker`, etc.)
- General reference material (e.g., “what is IsolationForest?”)
- StackOverflow–style answers for specific error messages, but **not** wholesale copy-pasting entire solutions.

You will be asked to **walk through and modify your own code live** in a follow-up interview.

You must include a short **INTEGRITY.md** file stating what tools you used and how you approached the assignment.

Scenario: BeaconHunter – Analytics for Covert C2 Detection

You are a junior security engineer on the Incident Response team of a mid-size company. Overnight, your EDR (endpoint detection & response) and network sensors flagged **suspicious beacon-like traffic** from several workstations to external IP addresses.

Your lead gives you a **synthetic dataset** of network/EDR events and asks you to:

1. Build an **analytics-driven detection pipeline** that can score events by likelihood of being **command-and-control (C2) beacons**.
2. Prioritize which hosts an on-call analyst should investigate first.
3. Package your logic into a **containerized Python service** that could eventually be deployed into the SOC environment.
4. Explain your reasoning and limitations clearly.

Provided Data (all synthetic)

Assume the following CSV files are provided in a `data/` folder:

1. `beacon_events_train.csv` – Labeled training data
2. `beacon_events_test_labeled.csv` – Labeled test data (for final evaluation)
3. `beacon_events_eval_unlabeled.csv` – Unlabeled "live" events from last night

Each row is one network/EDR event with the following columns:

- `event_id` – Unique ID for the event (string)
- `host_id` – Workstation identifier (string)
- `timestamp` – ISO timestamp (UTC)
- `src_ip` – Source IP (internal)
- `dst_ip` – Destination IP (external)
- `dst_port` – Destination port (int)
- `protocol` – e.g., `tcp`, `udp`, `https`, `dns`
- `bytes_out` – Bytes sent from host to destination (int)
- `bytes_in` – Bytes received (int)
- `inter_event_seconds` – Time since previous event for the same `(host_id, dst_ip, dst_port)` tuple (float; may be NaN for first event)
- `proc_name` – Process name that initiated the connection (e.g., `chrome.exe`, `powershell.exe`, `rundll32.exe`, `sliver-client.exe`, etc.)
- `user` – Logged-in user identifier
- `country_code` – GeoIP country of `dst_ip` (string; e.g., `US`, `RU`, `CN`, `DE`)
- `signed_binary` – 1 if executable is signed by a trusted publisher; 0 otherwise
- `label` – (only in train/test) 1 = confirmed malicious beacon; 0 = benign

Data is **noisy and imperfect** on purpose: there are missing values, weird process names, and some mislabeled rows.

Overview of Tasks & Weighting

1. **Part 1 – Analytics & Detection Pipeline (50 points)**
2. **Part 2 – Analyst-Facing Security Report (30 points)**
3. **Part 3 – Engineering Hygiene & GitHub Project (20 points)**

Part 1 – Analytics & Detection Pipeline (50%)

You will build an end-to-end Python pipeline to:

- Explore the data
- Engineer meaningful features
- Train and evaluate **at least one supervised** and **one unsupervised** ML approach
- Produce a scoring interface that assigns a **risk score** to new events

Your code should live under a top-level `src/` directory.

1.1. Exploration & Feature Engineering

Create a Jupyter notebook:

- `notebooks/01_exploration_and_features.ipynb`

In it, you should:

1. Load `beacon_events_train.csv`.
2. Perform **basic EDA**:
   - Distributions of key features (e.g., `inter_event_seconds`, `bytes_out`, `dst_port`, `proc_name`).
   - Relationships between features and `label` (e.g., boxplots, groupby aggregates).
3. Propose and implement **at least 4 derived features** motivated by security intuition, for example:
   - Beacon-ness measures (e.g., variance of `inter_event_seconds` per `(host_id, dst_ip)`).
   - "Weirdness" of ports (is `dst_port` in top common ports vs rare port).

- Process risk score (e.g., mapping certain processes to higher risk).
- GeoIP risk (e.g., country risk buckets).

4. Justify each derived feature in **markdown cells**:
   Why might this feature help distinguish benign browsing from C2 beacons?

You do **not** have to finish perfect EDA, but we want to see your **reasoning**.

1.2. Model Training Script

Create:

- `src/train_detector.py`

Requirements:

1. **Input & Config**
   - Read from `data/beacon_events_train.csv`.
   - You may hard-code file paths for this assignment.
2. **Preprocessing**
   - Handle missing values explicitly (e.g., imputation strategies).
   - Encode categorical variables (e.g. `protocol`, `proc_name`, `country_code`, `user`) via one-hot, target encoding, or another reasonable approach.
   - Use the derived features from your notebook (you may refactor into reusable functions in a `features.py` module if you want).
3. **Supervised Model (Classification)**
   - Train at least one supervised model (e.g., Logistic Regression, Random Forest, Gradient Boosting) using `label` as target.
   - Use **cross-validation** or a train/validation split.
   - Evaluate at least: **precision, recall, F1, ROC-AUC**, and preferably **PR-AUC**.
   - Make an explicit choice of **operating threshold** (e.g., risk score > 0.8 → malicious) and justify it.
4. **Unsupervised Model (Anomaly Detection)**
   - Train at least one unsupervised model (e.g., `IsolationForest`, `OneClassSVM`, `LocalOutlierFactor`) on **only benign-labeled events** (if you decide to trust that label), or on all data as "mostly benign."
   - Calibrate an anomaly score so that you can combine or compare it with supervised predictions.

5. **Fusion Strategy**
    - Implement a simple strategy to compute a final **risk score** for an event, combining supervised probability and unsupervised anomaly score.
        - e.g., weighted average, max, or a rule-based combiner.
    - Your final detector must output a **continuous risk score** between 0 and 1.
6. **Model Export**
    - Save the necessary trained artifacts (preprocessing steps + models) to `artifacts/` (e.g., via `joblib` or `pickle`).
    - Document what files get produced in a markdown cell in the notebook or in `README.md`.

You should be able to run:

```
python -m src.train_detector
```

and end up with trained models in `artifacts/`.

1.3. Evaluation on Labeled Test Data

Create:

- `src/evaluate_detector.py`

This script should:

1. Load your exported models.
2. Load `data/beacon_events_test_labeled.csv`.
3. Compute predictions and risk scores.
4. Print **summary metrics** (e.g., confusion matrix, ROC-AUC, precision/recall at your chosen threshold).
5. Print the **top 10 most misclassified events** (e.g., high risk but actually benign; low risk but actually malicious), including key feature values, so you can reason about errors.

We do **not** expect perfect performance – we want to see how you analyze and interpret results.

1.4. Scoring Interface (CLI)

Create:

- `src/score_events.py`

This script should:

- Accept a path to a CSV file of events (with no `label` column), e.g.:

```
python -m src.score_events --input data/beacon_events_eval_unlabeled.csv --output results/eval_scored.csv
```

- For each event, output at minimum:
  - `event_id`
  - `host_id`
  - `risk_score` (0–1)
  - `risk_label` (e.g., HIGH/MED/LOW, based on thresholds you define)
  - Optionally: top 2–3 feature values that contributed most to risk (your own heuristic with feature importance, SHAP, or simpler logic is fine; we care more about how you try than about perfection).

This CLI is what we will conceptually "deploy" to the SOC.

Part 2 – Analyst-Facing Security Report (30%)

Create a markdown or PDF report:

- `ANALYST_REPORT.md` **or** `ANALYST_REPORT.pdf`

Imagine you are writing for a **tier-2 SOC analyst** who will decide which hosts to investigate based on your work.

The report must be **at least ~1200 words** and include:

2.1. Executive Summary (Non-Technical)

- In 2–3 paragraphs, explain in plain language:
  - What you did.
  - What your detector can and cannot do.
  - Key results on the test data (successes and limitations).

2.2. Methodology & Analytics

Explain:

1. **Feature Engineering**
    - Which features you created and why (tie them to real-world C2 behavior).
    - Any interesting patterns you saw in the data (e.g., specific ports, countries, processes).
2. **Model Choices & Trade-offs**
    - Why you chose your supervised and unsupervised algorithms.
    - How you tuned/selected thresholds.
    - A brief comparison: what would have happened if you used only supervised or only unsupervised.
3. **Error Analysis**
    - Discuss patterns in false positives and false negatives from `evaluate_detector.py`.
    - Suggest at least **two concrete changes** (data collection, rules, model) that could reduce those errors in a real deployment.

2.3. Prioritization of Live Events

Using `beacon_events_eval_unlabeled.csv` and your scoring CLI:

1. Identify the **top 5 hosts** you would prioritize for investigation.
2. For each host, provide:
    - A short narrative ("mini incident note"):
        - Why is this host suspicious?
        - Which patterns triggered the detector? (e.g., regular 60-second beaconing to rare country, suspicious process, high bytes_out).
    - A **concrete recommendation** for the analyst:
        - e.g., "Isolate host from network," "Collect memory image," "Pull specific log types," "Block domain/IP."

You may include tables or charts if that helps clarity.

2.4. Limitations & Next Steps

In 2–3 paragraphs, discuss:

- Limitations of your current approach (data quality, coverage, model assumptions).
- Potential **adversarial behaviors** that could evade your detector.
- How you would evolve this system over the next 3 months if you stayed in the role.

Part 3 – Engineering Hygiene & GitHub Project (20%)

We also care about how you structure, package, and document your work.

3.1. Repository Structure

We expect a **public GitHub repository** (you'll share the link) with at least:

```
.
├── data/                     # (you may exclude large files if needed; note in README)
├── src/
|   ├── __init__.py
|   ├── train_detector.py
|   ├── evaluate_detector.py
|   ├── score_events.py
|   └── (optional helpers: features.py, models.py, utils.py, ...)
├── notebooks/
|   └── 01_exploration_and_features.ipynb
├── artifacts/                # Model artifacts created by training script
├── docker/
|   └── (optional extra docker files)
├── tests/
|   └── test_basic.py         # At least 3 simple tests
├── .github/
|   └── workflows/
|       └── ci.yml            # GitHub Actions pipeline
├── Dockerfile
├── requirements.txt
├── README.md
├── ANALYST_REPORT.md or ANALYST_REPORT.pdf
├── INTEGRITY.md
└── NOTES.md
```

3.2. Dockerization

Create a `Dockerfile` that:

- Uses an official Python base image.
- Installs dependencies from `requirements.txt`.
- Copies your project into the image.
- Sets an entrypoint so that running the container like:

  ```
  docker build -t beaconhunter .
  docker run --rm -v $(pwd)/data:/app/data beaconhunter \
      python -m src.score_events --input data/beacon_events_eval_unlabeled.csv --output /app/results/eval_scored.csv
  ```

  is conceptually possible.

We will mainly look at:

- Whether the Dockerfile builds (in principle).
- Whether your choices (layers, user, etc.) are reasonable.

3.3. Tests

Under `tests/`, provide **at least three** simple tests, e.g.:

- Sanity check on feature engineering functions.
- Sanity test that `train_detector.py` can run a tiny subset of data.
- Sanity test that `score_events.py` produces reasonable outputs for a hand-crafted mini-CSV.

You may use `pytest` or the built-in `unittest` framework.

3.4. GitHub Actions CI

Create:

- `.github/workflows/ci.yml`

The workflow should:

- Run on `push` and `pull_request` to `main` (or `master`).

- Set up Python.
- Install dependencies.
- Run:
  - Your tests (`pytest` or equivalent).
  - A linter (e.g., `flake8` , `ruff` , or `pylint`) on the `src/` directory.

It does **not** need to be perfect, but it should be functional in principle.

3.5. Documentation & Process

Required top-level docs:

1. **README.md**
   - One-paragraph description of the project.
   - How to set up the environment.
   - How to run:
     - `train_detector.py`
     - `evaluate_detector.py`
     - `score_events.py`
     - Basic Docker usage.
2. **INTEGRITY.md**
   - Brief description of how you approached the assignment.
   - A statement about whether you used any generative AI; if yes (not recommended), explain exactly how and where.
3. **NOTES.md**
   - A **chronological log** of your work:
     - Timestamps (approximate are fine).
     - What you tried.
     - Dead ends and bugs you hit.

- What you would do next if you had more time.

We care a lot about this file – it shows us your problem-solving process.

What to Submit

When you're done (or time runs out), submit:

1. **GitHub Repository Link** – public repo for this assignment.
2. (Optional but helpful) A short text note with:
   - Which parts you prioritized.
   - Anything you specifically want reviewers to look at.

Rubric (100 Points Total)

We will grade holistically. "Satisfactory" is roughly what we hope a strong undergrad can achieve with good time management; "Excellent" is intentionally aspirational.

Part 1 – Analytics & Detection Pipeline (50 pts)

Criteria	Needs Improvement	Satisfactory	Excellent
Feature Engineering & EDA (15)	Minimal or no EDA; features seem arbitrary or copy-paste from generic examples.	Reasonable EDA with some domain-driven features; basic justifications present.	Thoughtful, security-motivated features; clear discussion of patterns and why they matter for C2.
Supervised & Unsupervised Modeling (20)	Only one model type used; poor handling of labels; metrics missing or incorrect.	Both supervised and unsupervised attempted; basic metrics; thresholds chosen but not deeply justified.	Clear, well-implemented models; good use of metrics; thoughtful threshold selection and trade-off discussion.
Fusion & Scoring Interface (15)	No coherent risk score; scoring script missing or broken.	Working scoring script with simple fusion; reasonable outputs.	Robust scoring pipeline; well-designed risk labels; explanations for scores are meaningful and reproducible.

Part 2 – Analyst-Facing Security Report (30 pts)

Criteria	Needs Improvement	Satisfactory	Excellent
Clarity & Communication (10)	Hard to follow; very technical jargon or unclear structure.	Generally clear and structured; minor issues.	Highly readable; sharp executive summary; good separation of technical and non-technical content.
Technical Depth & Analytics (10)	Superficial descriptions; no real analysis of errors or patterns.	Solid explanation of methods; some discussion of limitations.	Deep, critical analysis; thoughtful discussion of errors, patterns, and realistic improvements.
Prioritization & Recommendations (10)	Hosts chosen without clear reasoning; recommendations vague.	Reasonable host prioritization with basic justification.	Convincing host prioritization; strong, actionable investigative recommendations grounded in your analytics.

Part 3 – Engineering Hygiene & GitHub (20 pts)

Criteria	Needs Improvement	Satisfactory	Excellent
Repo Structure & Code Quality (8)	Disorganized repo; hard to run; little to no structure.	Clear structure; scripts runnable with some effort.	Professional structure; clean, readable code; sensible modularization.
Docker & CI (6)	Dockerfile or CI missing/broken.	Docker and CI present and mostly working.	Well-structured Docker image; CI that runs tests and linting cleanly.
Documentation & Process (6)	Missing README, NOTES, or INTEGRITY; little insight into process.	Documentation present; basic process notes.	Excellent README; honest, detailed NOTES; clear integrity statement and reflection.

Final Notes

- You are **not** expected to complete everything in 24 hours. Focus on:
  - Clear reasoning.
  - Honest documentation of what you tried.
  - A coherent, end-to-end story, even if some components are rough.
- We will **not** penalize you for bugs you clearly identified and documented, especially if we can see your attempts and debugging steps.

Good luck, and have fun thinking like both a data scientist **and** a security engineer.