Configuration Manual

MSc Research Project
Programme Name MSc Cyber Security

Hassan Manzoor
Student ID: X23393475

School of Computing
National College of Ireland

Supervisor: Mark Monaghan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Hassan Manzoor

Student ID: X23393475

Programme: Cyber Security **Year:** ……2024,2025……..

Module: Research Practicum

Supervisor: ……

Submission Due Date: 11th Aug, 2025

Project Title: **Effectiveness of Zero Trust Security Against AI and Quantum Computing Threats: A Systematic Literature Review**

Word Count: …… **Page Count**……..

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Hassan Manzoor

Signature:

Date: ……11th Aug 2025……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

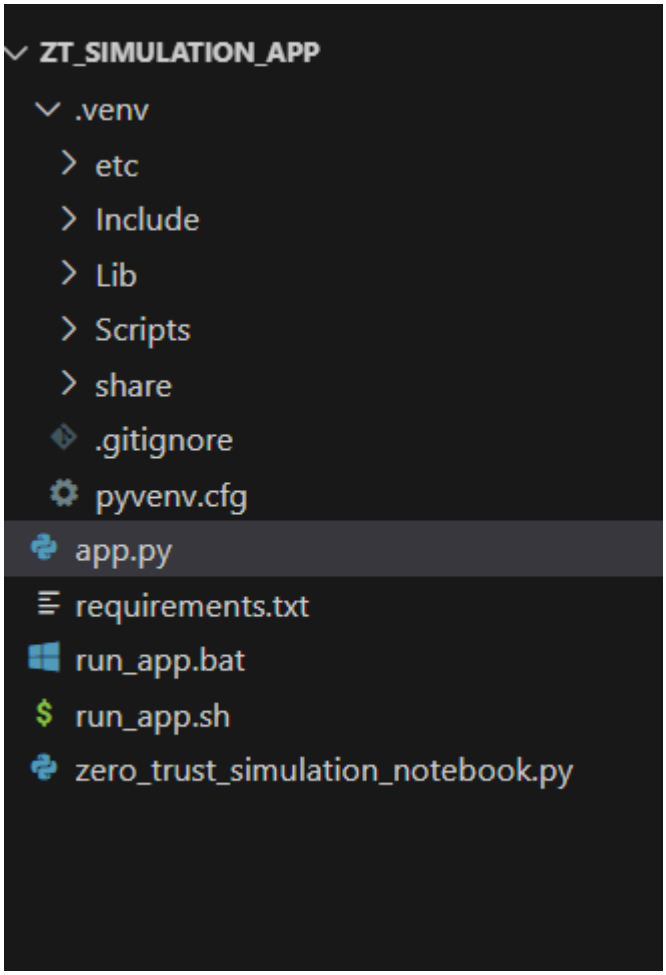
Configuration Manual for Simulation

Hassan Manzoor
X23393475

1 Introduction

A lightweight Streamlit app that simulates a Zero-Trust access gateway with tunable controls (threshold, cache/PDP, device posture, UEBA, JIT).
It generates thousands of synthetic requests, measures security (TPR/FPR/Precision/Recall) and performance (Mean/P50/P95/P99), and visualizes latency distributions.
Designed for classroom/demo use and policy tuning, it shows how configuration choices trade detection effectiveness against user experience.

2 Environment Setup



Each file/folder is

- **.venv/** – your project's isolated Python environment (activate this before running).

- **app.py** – the Streamlit UI for the Zero-Trust simulation (the thing you launch).
- **requirements.txt** – packages the app needs (Streamlit, NumPy, Pandas, Matplotlib, etc.).
- **run_app.bat** – one-click launcher for Windows (creates venv if needed, installs deps, runs app).
- **run_app.sh** – one-click launcher for macOS/Linux.
- **zero_trust_simulation_notebook.py** – a script version for batch runs/plots without the UI (good for generating CSVs/figures for your report).
- **.gitignore / pyvenv.cfg / etc, Include, Lib, Scripts** – venv internals; ignore in Git or when zipping.

Run it (Windows)

1. Open PowerShell in this folder.
2. Activate venv:

```
powershell
CopyEdit
.\.venv\Scripts\Activate.ps1
```

3. Install deps (only needed once or after changes):

```
powershell
CopyEdit
python -m pip install -U pip
pip install -r requirements.txt
```

4. Launch:

```
powershell
CopyEdit
streamlit run app.py
```

App opens at http://localhost:8501. Stop with `Ctrl+C`.

Run it (macOS/Linux)

```
bash
CopyEdit
cd /path/to/ZT_SIMULATION_APP
source .venv/bin/activate
python -m pip install -U pip
pip install -r requirements.txt
streamlit run app.py
```

3 Troubleshooting (fast)

- **'Activate.ps1 cannot be loaded' (Windows):**

```
powershell
CopyEdit
Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned
.\.venv\Scripts\Activate.ps1
```

- **'streamlit not found':** activate venv, then `pip install -r requirements.txt`.
- **Port busy:** `streamlit run app.py --server.port 8502`.
- **Black screen / caching issues:** hard refresh the browser (Ctrl+F5).

Generating artefacts for slides/report

- Use the **Download CSV** button in the app for per-request latency.
- For automated figures/tables (threshold sweeps, grid search), run:

```
powershell
CopyEdit
# after activating venv
python .\zero_trust_simulation_notebook.py
```

This script can save metrics CSVs and plots to a `results/` folder (add that folder if it doesn't exist).

Packaging for submission

- Zip **app.py**, **requirements.txt**, **run_app.bat/run_app.sh**, and any **results/** outputs.
- **Exclude** the **.venv/** folder to keep the zip small (it's recreated on the grader's machine).

Want a "results" subfolder prewired in the code (so figures/CSVs drop in one place)? Say the word and a tiny patch can be provided.

Code SS :

```
# Zero-Trust Simulation — Streamlit App
# ------------------------------------------------------------------------
-
# What this app does
#   This is a Monte-Carlo simulation of a Zero-Trust access gateway. For each
#   synthetic request, I (1) assign a risk score, (2) add realistic latency
#   components (PEP base cost, optional mTLS handshake, device-posture check,
#   PDP policy evaluation on cache miss, optional UEBA overhead, optional JIT
#   challenge), and (3) decide allow/deny using a policy threshold (and JIT if
#   enabled). We then compute security metrics (TPR/FPR/Precision/Recall) and
#   user-experience metrics (Mean/P50/P95/P99 latency). The UI lets you move
#   sliders to see the trade-offs live.
#
# How this maps to Zero-Trust terms:
#   - PEP (Policy Enforcement Point): the gateway here adding latency &
enforcing.
```

```
#   - PDP (Policy Decision Point): the policy engine (latency modeled as
lognormal).
#   - Device posture: a per-request check with time cost and small failure
prob.
#   - UEBA: raises risk a bit for flagged requests (behavior analytics).
#   - JIT (step-up auth): adds one-off friction for risky requests instead of
all users.
#
================================================================================
```

- Imports Streamlit/NumPy/Pandas/Matplotlib and sets the app layout to **wide** with `st.set_page_config(...)`.

- Initializes a reproducible random generator `rng = np.random.default_rng(42)` for consistent simulation runs.

- Defines `_draw_pdp_latencies(...)`: converts a target **arithmetic mean** and **sigma** into the lognormal parameter `mu`, then draws PDP latency samples with `rng.lognormal(...)`.

```python
import streamlit as st
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

st.set_page_config(page_title="Zero-Trust Simulation", layout="wide")

rng = np.random.default_rng(42)

def _draw_pdp_latencies(mean_ms: float, sigma: float, size: int) -> np.ndarray:
    mu = np.log(mean_ms) - (sigma ** 2) / 2.0
    return rng.lognormal(mean=mu, sigma=sigma, size=size)
```

```python
rng = np.random.default_rng(42)

def _draw_pdp_latencies(mean_ms: float, sigma: float, size: int) -> np.ndarray:
    mu = np.log(mean_ms) - (sigma ** 2) / 2.0
    return rng.lognormal(mean=mu, sigma=sigma, size=size)

def simulate(
    n_requests=20000,
    malicious_rate=0.006,
    benign_risk_mean=0.2, benign_risk_std=0.1,
    mal_risk_mean=0.7, mal_risk_std=0.15,
    policy_threshold=0.65,
    pep_base_ms=2.0,
    cache_hit_rate=0.80,
    pdp_lognorm_mean_ms=12.0,
    pdp_lognorm_sigma=0.35,
    mtls_handshake_ms=20.0,
    conn_reuse_rate=0.80,
    device_check_ms=6.0,
    device_fail_prob=0.002,
    ueba_enabled=False,
    ueba_extra_ms=0.0,
    ueba_risk_uplift_mean=0.0,
    ueba_flag_rate=0.0,
    jit_enabled=False,
    jit_trigger_rate=0.0,
    jit_ms=1200.0,
    jit_pass_prob_benign=0.95,
    jit_pass_prob_malicious=0.20,
):
    is_mal = rng.random(n_requests) < malicious_rate
    benign_risk = np.clip(rng.normal(benign_risk_mean, benign_risk_std, n_requests), 0, 1)
    mal_risk = np.clip(rng.normal(mal_risk_mean, mal_risk_std, n_requests), 0, 1)
    risk = np.where(is_mal, mal_risk, benign_risk)
```

```python
is_mal = rng.random(n_requests) < malicious_rate
benign_risk = np.clip(rng.normal(benign_risk_mean, benign_risk_std, n_requests), 0, 1)
mal_risk = np.clip(rng.normal(mal_risk_mean, mal_risk_std, n_requests), 0, 1)
risk = np.where(is_mal, mal_risk, benign_risk)

if ueba_enabled and ueba_flag_rate > 0:
    ueba_flag = rng.random(n_requests) < ueba_flag_rate
    uplift_sigma = ueba_risk_uplift_mean/3 if ueba_risk_uplift_mean>0 else 0.0
    uplift = np.maximum(0.0, rng.normal(ueba_risk_uplift_mean, uplift_sigma, n_requests))
    risk = np.clip(risk + uplift * ueba_flag, 0, 1)

jit_independent = (rng.random(n_requests) < jit_trigger_rate) if (jit_enabled and jit_trigger_rate>0) else np.zeros(n_requests,
cache_hit = rng.random(n_requests) < cache_hit_rate
pays_handshake = rng.random(n_requests) > conn_reuse_rate
device_fail = rng.random(n_requests) < device_fail_prob
pdp_lat = _draw_pdp_latencies(pdp_lognorm_mean_ms, pdp_lognorm_sigma, n_requests)

latency = np.zeros(n_requests, dtype=float)
latency += pep_base_ms
latency += np.where(pays_handshake, mtls_handshake_ms, 0.0)
latency += device_check_ms
latency += np.where(~cache_hit, pdp_lat, 0.0)
latency += np.where(ueba_enabled, ueba_extra_ms, 0.0)

at_risk = risk >= policy_threshold
```

```
at_risk = risk >= policy_threshold

if jit_enabled:
    jit_trigger = jit_independent | at_risk
    latency += np.where(jit_trigger, jit_ms, 0.0)
else:
    jit_trigger = np.zeros(n_requests, dtype=bool)

outcome_deny = device_fail.copy()
if jit_enabled:
    mask = jit_trigger & (~outcome_deny)
    passes = np.where(is_mal[mask],
                      rng.random(mask.sum()) < jit_pass_prob_malicious,
                      rng.random(mask.sum()) < jit_pass_prob_benign)
    jit_deny = ~passes
    temp = outcome_deny.copy()
    temp[mask] = jit_deny
    outcome_deny = temp
else:
    outcome_deny = outcome_deny | at_risk

outcome_allow = ~outcome_deny

tp = int((outcome_deny & is_mal).sum())
fn = int((outcome_allow & is_mal).sum())
fp = int((outcome_deny & ~is_mal).sum())
tn = int((outcome_allow & ~is_mal).sum())
```

```
tp = int((outcome_deny & is_mal).sum())
fn = int((outcome_allow & is_mal).sum())
fp = int((outcome_deny & ~is_mal).sum())
tn = int((outcome_allow & ~is_mal).sum())

allow_rate = outcome_allow.mean()
deny_rate = outcome_deny.mean()
tpr = tp / (tp + fn) if (tp + fn) > 0 else 0.0
fpr = fp / (fp + tn) if (fp + tn) > 0 else 0.0
precision = tp / (tp + fp) if (tp + fp) > 0 else 0.0
recall = tpr

mean_ms = float(latency.mean())
p50_ms = float(np.percentile(latency, 50))
p95_ms = float(np.percentile(latency, 95))
p99_ms = float(np.percentile(latency, 99))
jit_rate = float(jit_trigger.mean() if jit_enabled else 0.0)

return {
    "metrics": {
        "Allow %": round(100*allow_rate, 2),
        "Deny %": round(100*deny_rate, 2),
        "TPR": round(tpr, 3),
        "FPR": round(fpr, 3),
        "Precision": round(precision, 3),
        "Recall": round(recall, 3),
        "Mean (ms)": round(mean_ms, 1),
        "P50 (ms)": round(p50_ms, 1),
        "P95 (ms)": round(p95_ms, 1),
        "P99 (ms)": round(p99_ms, 1),
        "JIT Rate %": round(100*jit_rate, 2),
    },
    "latency": latency
}
```

```python
st.title("Zero-Trust Simulation — Click to Run")

st.sidebar.header("Controls")
n_requests = st.sidebar.number_input("Requests", min_value=1000, max_value=200000, value=20000, step=1000)
malicious_rate = st.sidebar.slider("Malicious Rate", 0.0, 0.05, 0.006, 0.001)
policy_threshold = st.sidebar.slider("Policy Threshold", 0.30, 0.95, 0.65, 0.01)
cache_hit_rate = st.sidebar.slider("Cache Hit Rate", 0.0, 1.0, 0.80, 0.01)
pdp_mean = st.sidebar.slider("PDP Mean (ms)", 5.0, 40.0, 12.0, 0.5)
pdp_sigma = st.sidebar.slider("PDP Sigma", 0.1, 0.8, 0.35, 0.01)
conn_reuse = st.sidebar.slider("Connection Reuse", 0.0, 1.0, 0.80, 0.01)
mtls_ms = st.sidebar.slider("mTLS Handshake (ms)", 0.0, 60.0, 20.0, 1.0)
device_ms = st.sidebar.slider("Device Check (ms)", 0.0, 30.0, 6.0, 0.5)
device_fail = st.sidebar.slider("Device Fail Prob", 0.0, 0.02, 0.002, 0.0005)

ueba_enabled = st.sidebar.checkbox("Enable UEBA", value=False)
ueba_extra = st.sidebar.slider("UEBA Extra (ms)", 0.0, 30.0, 8.0, 0.5) if ueba_enabled else 0.0
ueba_uplift = st.sidebar.slider("UEBA Risk Uplift Mean", 0.0, 0.4, 0.10, 0.01) if ueba_enabled else 0.0
ueba_flag = st.sidebar.slider("UEBA Flag Rate", 0.0, 0.2, 0.05, 0.01) if ueba_enabled else 0.0

jit_enabled = st.sidebar.checkbox("Enable JIT Re-Auth", value=False)
jit_trigger = st.sidebar.slider("Independent JIT Trigger Rate", 0.0, 0.2, 0.03, 0.01) if jit_enabled else 0.0
jit_ms = st.sidebar.slider("JIT Challenge Time (ms)", 100.0, 3000.0, 1200.0, 50.0) if jit_enabled else 1200.0
jit_pass_b = st.sidebar.slider("JIT Pass Prob (Benign)", 0.0, 1.0, 0.95, 0.01) if jit_enabled else 0.95
jit_pass_m = st.sidebar.slider("JIT Pass Prob (Malicious)", 0.0, 1.0, 0.20, 0.01) if jit_enabled else 0.20
```

```python
if st.button("Run Simulation"):
    res = simulate(
        n_requests=n_requests,
        malicious_rate=malicious_rate,
        policy_threshold=policy_threshold,
        cache_hit_rate=cache_hit_rate,
        pdp_lognorm_mean_ms=pdp_mean,
        pdp_lognorm_sigma=pdp_sigma,
        conn_reuse_rate=conn_reuse,
        mtls_handshake_ms=mtls_ms,
        device_check_ms=device_ms,
        device_fail_prob=device_fail,
        ueba_enabled=ueba_enabled,
        ueba_extra_ms=ueba_extra,
        ueba_risk_uplift_mean=ueba_uplift,
        ueba_flag_rate=ueba_flag,
        jit_enabled=jit_enabled,
        jit_trigger_rate=jit_trigger,
        jit_ms=jit_ms,
        jit_pass_prob_benign=jit_pass_b,
        jit_pass_prob_malicious=jit_pass_m,
    )
    st.subheader("Metrics")
    st.dataframe(pd.DataFrame([res["metrics"]]))

    st.subheader("Latency Histogram")
    fig = plt.figure()
    plt.hist(res["latency"], bins=60)
    plt.xlabel("Latency (ms)")
    plt.ylabel("Requests")
    st.pyplot(fig)
```

```
  st.subheader("Latency Histogram")
  fig = plt.figure()
  plt.hist(res["latency"], bins=60)
  plt.xlabel("Latency (ms)")
  plt.ylabel("Requests")
  st.pyplot(fig)

  st.subheader("Latency CDF")
  x = np.sort(res["latency"]); y = np.arange(1, len(x)+1)/len(x)
  fig2 = plt.figure()
  plt.plot(x, y)
  plt.xlabel("Latency (ms)")
  plt.ylabel("CDF")
  st.pyplot(fig2)

  csv = pd.DataFrame({"latency_ms": res["latency"]}).to_csv(index=False).encode("utf-8")
  st.download_button("Download per-request latency CSV", data=csv, file_name="latency.csv", mime="text/csv")

t.info("Tip: Enable UEBA and JIT to see security improve with targeted friction. Try lowering cache to see PDP latency impact.")
```

Notebook SS:

```
# For the bonus UEBA model
try:
    from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import roc_curve, precision_recall_curve, auc, confusion_matrix
    SKLEARN_AVAILABLE = True
except Exception:
    SKLEARN_AVAILABLE = False

plt.rcParams.update({
    "figure.figsize": (10, 6),
    "axes.grid": True,
})
rng = np.random.default_rng(42)
out_dir = Path(".")
out_dir.mkdir(exist_ok=True, parents=True)
```

```
"""## 2) Simulator

**Scenario knobs:**
- `policy_threshold` – risk threshold to deny (unless JIT passes)
- `cache_hit_rate` – % of requests skipping PDP latency
- `pdp_lognorm_*` – PDP latency distribution
- `conn_reuse_rate` and `mtls_handshake_ms` – mTLS cost and reuse
- `device_check_ms`, `device_fail_prob` – posture checks
- `ueba_*` – UEBA extra latency and risk uplift for flagged requests
- `jit_*` – just-in-time challenge probability, duration, pass rates

"""
```

```python
@dataclass
class Scenario:
    name: str
    n_requests: int = 30000
    malicious_rate: float = 0.008
    benign_risk_mean: float = 0.2
    benign_risk_std: float = 0.1
    mal_risk_mean: float = 0.7
    mal_risk_std: float = 0.15
    policy_threshold: float = 0.65
    pep_base_ms: float = 2.0
    cache_hit_rate: float = 0.80
    pdp_lognorm_mean_ms: float = 12.0
    pdp_lognorm_sigma: float = 0.35
    mtls_handshake_ms: float = 20.0
    conn_reuse_rate: float = 0.80
    device_check_ms: float = 6.0
    device_fail_prob: float = 0.002
    ueba_enabled: bool = False
    ueba_extra_ms: float = 0.0
    ueba_risk_uplift_mean: float = 0.0
    ueba_flag_rate: float = 0.00
    jit_enabled: bool = False
    jit_trigger_rate: float = 0.00
    jit_ms: float = 1500.0
    jit_pass_prob_benign: float = 0.95
    jit_pass_prob_malicious: float = 0.20

def _draw_pdp_latencies(mean_ms: float, sigma: float, size: int) -> np.ndarray:
    mu = np.log(mean_ms) - (sigma ** 2) / 2.0
    return rng.lognormal(mean=mu, sigma=sigma, size=size)
```

```python
def _draw_pdp_latencies(mean_ms: float, sigma: float, size: int) -> np.ndarray:
    mu = np.log(mean_ms) - (sigma ** 2) / 2.0
    return rng.lognormal(mean=mu, sigma=sigma, size=size)

def simulate(s: Scenario) -> Dict[str, object]:
    n = s.n_requests
    is_mal = rng.random(n) < s.malicious_rate

    benign_risk = np.clip(rng.normal(s.benign_risk_mean, s.benign_risk_std, n), 0, 1)
    mal_risk = np.clip(rng.normal(s.mal_risk_mean, s.mal_risk_std, n), 0, 1)
    risk = np.where(is_mal, mal_risk, benign_risk)

    # UEBA
    if s.ueba_enabled and s.ueba_flag_rate > 0:
        ueba_flag = rng.random(n) < s.ueba_flag_rate
        uplift_sigma = s.ueba_risk_uplift_mean/3 if s.ueba_risk_uplift_mean>0 else 0.0
        uplift = np.maximum(0.0, rng.normal(s.ueba_risk_uplift_mean, uplift_sigma, n))
        risk = np.clip(risk + uplift * ueba_flag, 0, 1)

    # Independent JIT triggers
    if s.jit_enabled and s.jit_trigger_rate > 0:
        jit_independent = rng.random(n) < s.jit_trigger_rate
    else:
        jit_independent = np.zeros(n, dtype=bool)

    cache_hit = rng.random(n) < s.cache_hit_rate
    pays_handshake = rng.random(n) > s.conn_reuse_rate
    device_fail = rng.random(n) < s.device_fail_prob
    pdp_lat = _draw_pdp_latencies(s.pdp_lognorm_mean_ms, s.pdp_lognorm_sigma, n)
```