REPORT: TRUST DEVELOPMENT ENGINE

Student: Leon Jacob — 13178938 — WSU INFO3016

What I Set Out to Do

My task was to **code and build a "trust development engine" that combines a rule-based system with a reinforcement learning (RL) component and** produces a **binary trust value** (`0` or `1`) for each simulated device.

The goal was to identify and flag suspicious device behaviour, (i added to the project by including stealthy threats like **reverse shells** and **zero-day-like patterns**, while maintaining a clear, explainable trust decision for each device)

How I Built It (My Architecture)

I built the trust engine using four core Python files:

File Name	What I Used It For
`main.py`	The main engine that runs everything and prints results
`simulator.py`	Generates random device behaviours
`rules.py`	My rule-based trust evaluation logic
`rl_agent.py`	A simulated reinforcement learning scoring system

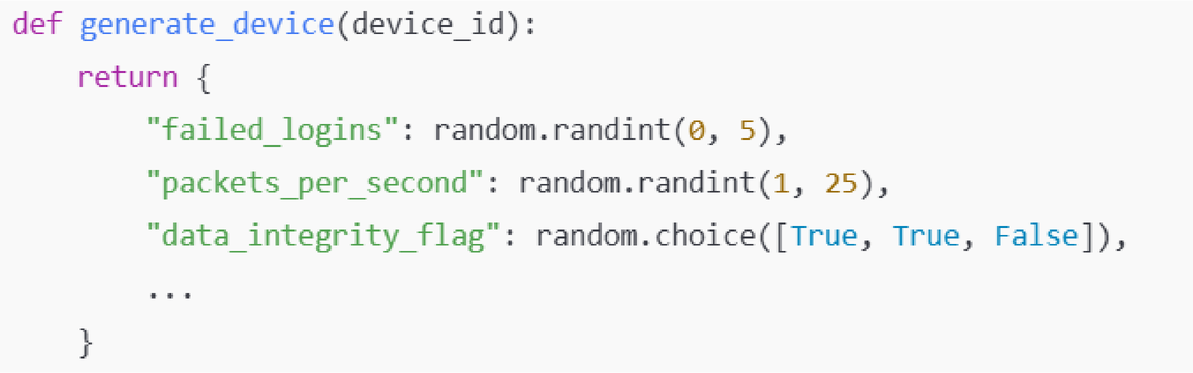
Step-by-Step: What I Did

1. Simulating Device Behaviour

`simulator.py`

I created a function that randomly generates 5 device entries every time the script runs. Each device includes fields like:

```
    *failed_logins
    *packets_per_second
    *data_integrity_flag
    *unknown_outbound_connection
    *suspicious_process_detected
```



```python
def generate_device(device_id):
    return {
        "failed_logins": random.randint(0, 5),
        "packets_per_second": random.randint(1, 25),
        "data_integrity_flag": random.choice([True, True, False]),
        ...
    }
```

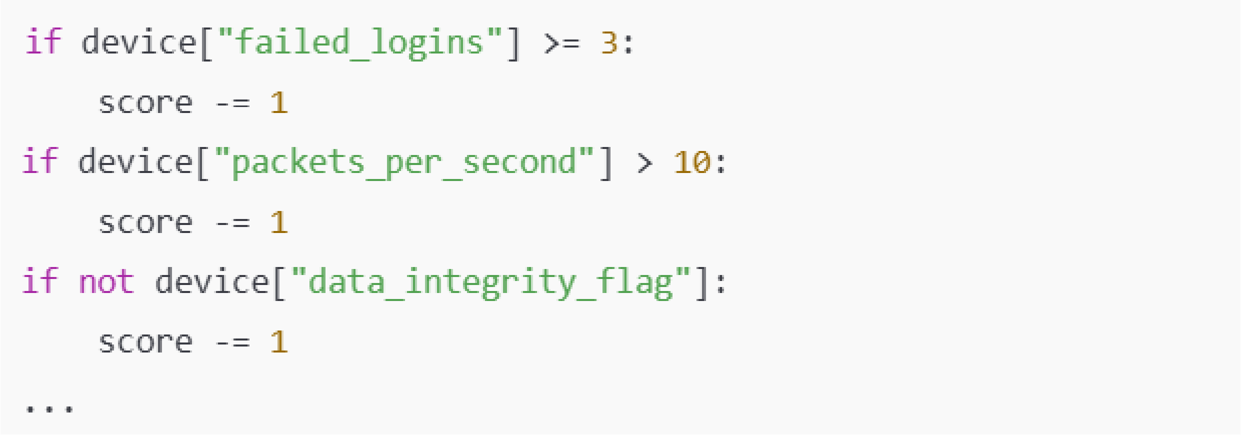
Why I did this:
This allowed me to simulate both **clean** and **suspicious** device behaviour, including

reverse shells (detected by outbound connections + suspicious processes) and stealthy activity that mimics zero-day exploits.

2. Applying Rule-Based Logic

`rules.py`

I wrote logic that penalizes devices for suspicious activity. Here's a snippet:



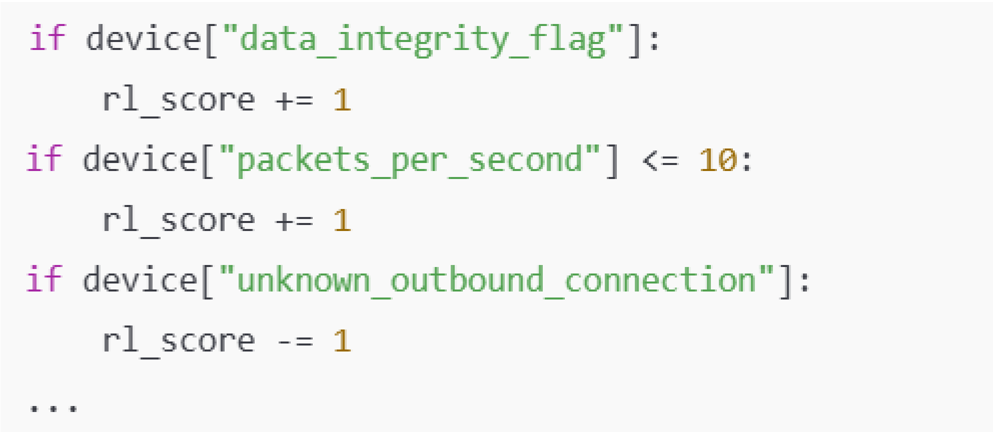
```
if device["failed_logins"] >= 3:
    score -= 1
if device["packets_per_second"] > 10:
    score -= 1
if not device["data_integrity_flag"]:
    score -= 1
...
```

Why I did this:
These rules mimic what a real system administrator or security policy might flag — such as brute force attempts, malformed data, or reverse shell behaviour.

3. Simulating an RL Model

`rl_agent.py`

I created a simulated reinforcement learning system. Instead of training over time, it mimics what an RL model would have learned after training.



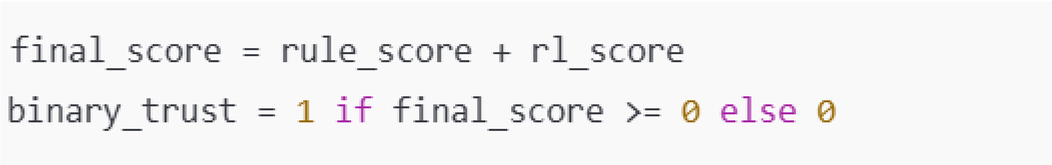
```
if device["data_integrity_flag"]:
    rl_score += 1
if device["packets_per_second"] <= 10:
    rl_score += 1
if device["unknown_outbound_connection"]:
    rl_score -= 1
...
```

Why I did this:
My version doesn't use actual Q-learning, but it reflects **adaptive logic** that rewards or punishes based on behaviour — just like a trained RL agent would.

4. Producing Binary Trust Decisions

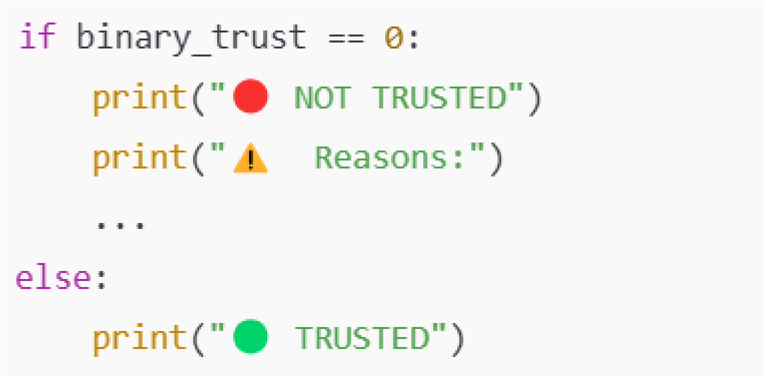
`main.py`

In the main script, I combined both scores and created a **binary trust value** (`0` or `1`):



```
final_score = rule_score + rl_score
binary_trust = 1 if final_score >= 0 else 0
```

Then, I printed full explanations for each decision:

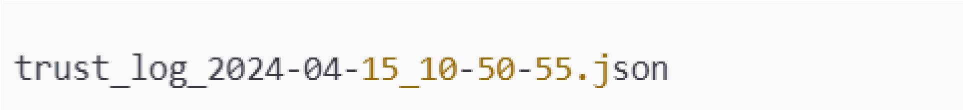


```python
if binary_trust == 0:
    print("🔴 NOT TRUSTED")
    print("⚠️  Reasons:")
    ...
else:
    print("🟢 TRUSTED")
```

Why I did this:
This gives full **human-readable reasoning** per device, perfect for demo, audit logs, or training.

5. Saving Trust Results to JSON Logs

Every time I run the engine, it creates a JSON file with a timestamp:



```
trust_log_2024-04-15_10-50-55.json
```

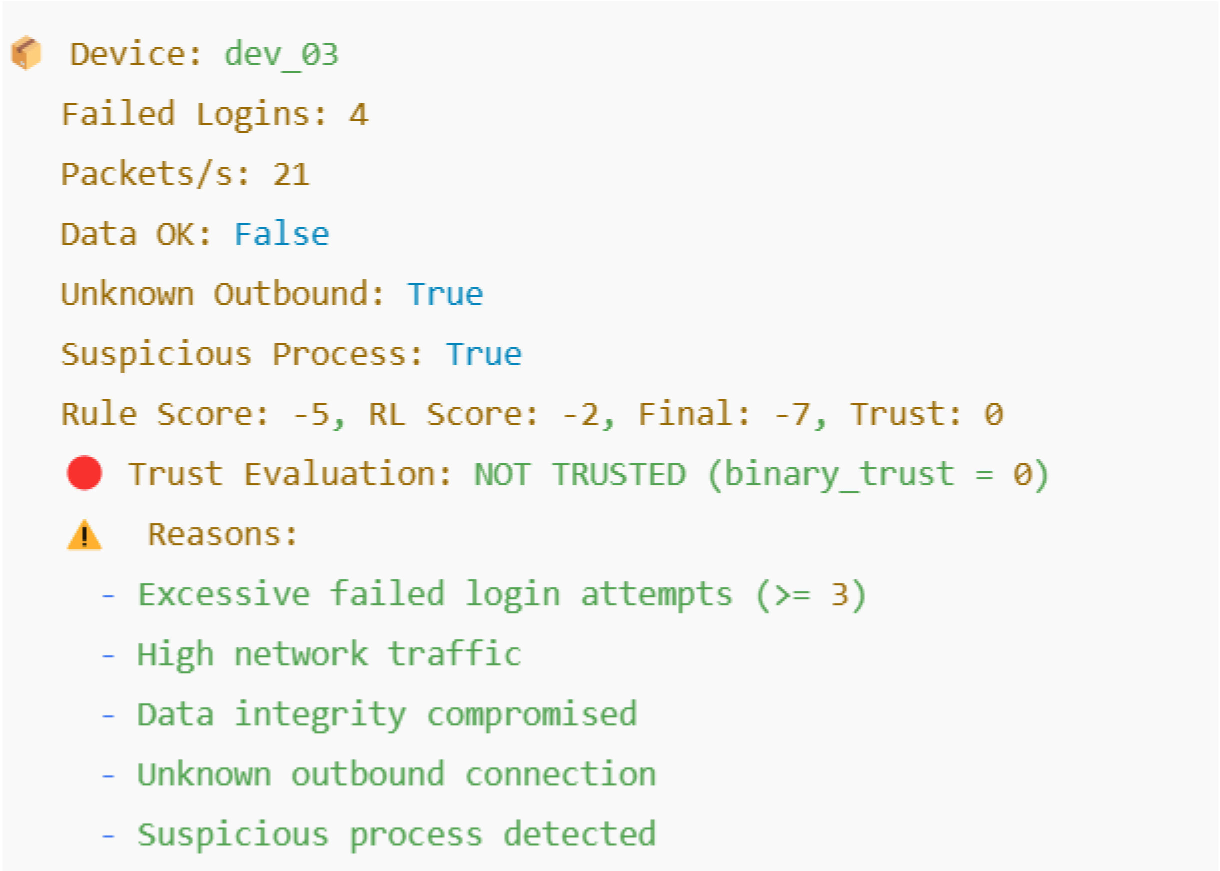
This helps track each run and makes it easy to review device decisions over time.

How I Met the Original Requirements

Requirement	How I Met It
✅ Use of RL	created a simulated RL scoring engine in `rl_agent.py`
✅ Use of rule-based logic	Done in `rules.py` using hardcoded security rules

✅ Hybrid system	`main.py` combines both score types into a final trust decision
✅ Random input data	`simulator.py` uses `random` module to create different behavior each time
✅ Binary output	`binary_trust = 0 or 1` clearly flags devices
✅ Explanations for decisions	printed reasons per device in the terminal
✅ Logging and reproducibility	Each run creates a timestamped JSON file

Example Terminal Output (From My Project)



```
📦 Device: dev_03
   Failed Logins: 4
   Packets/s: 21
   Data OK: False
   Unknown Outbound: True
   Suspicious Process: True
   Rule Score: -5, RL Score: -2, Final: -7, Trust: 0
   🔴 Trust Evaluation: NOT TRUSTED (binary_trust = 0)
   ⚠️ Reasons:
     - Excessive failed login attempts (>= 3)
     - High network traffic
     - Data integrity compromised
     - Unknown outbound connection
     - Suspicious process detected
```

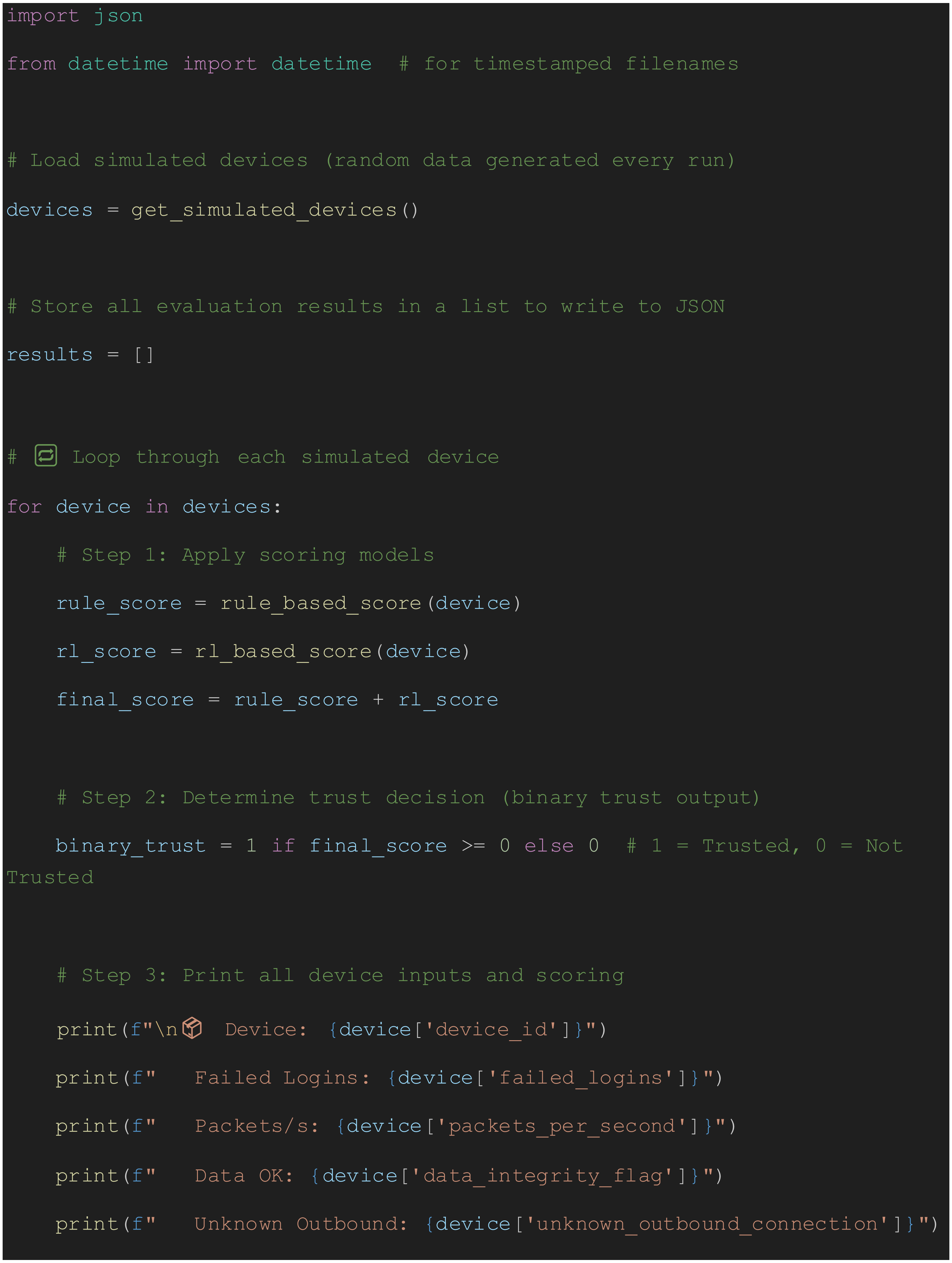
Leon Jacob 13178938

To run script in main.py

Powershell command in VSC ‘py main.py’

main.py file

```
#Leon Jacob 13178938 WSU info3016


# to begin we need to import other modules,
# run everything together and
# output JSON logs to a file



# Leon Jacob 13178938 WSU info3016
# main.py - Hybrid Trust Evaluation Engine

# This script:
# 1. Loads dynamically simulated device data from simulator.py
# 2. Applies both rule-based and simulated RL scoring logic
# 3. Combines scores to produce a final binary trust decision (0 = NOT TRUSTED, 1 = TRUSTED)
# 4. Explains the reasoning in the terminal
# 5. Saves output to a timestamped JSON file

from simulator import get_simulated_devices
from rules import rule_based_score
from rl_agent import rl_based_score
```



```python
import json
from datetime import datetime  # for timestamped filenames

# Load simulated devices (random data generated every run)
devices = get_simulated_devices()

# Store all evaluation results in a list to write to JSON
results = []

# 🔁 Loop through each simulated device
for device in devices:
    # Step 1: Apply scoring models
    rule_score = rule_based_score(device)
    rl_score = rl_based_score(device)
    final_score = rule_score + rl_score

    # Step 2: Determine trust decision (binary trust output)
    binary_trust = 1 if final_score >= 0 else 0  # 1 = Trusted, 0 = Not Trusted

    # Step 3: Print all device inputs and scoring
    print(f"\n📦 Device: {device['device_id']}")
    print(f"   Failed Logins: {device['failed_logins']}")
    print(f"   Packets/s: {device['packets_per_second']}")
    print(f"   Data OK: {device['data_integrity_flag']}")
    print(f"   Unknown Outbound: {device['unknown_outbound_connection']}")
    print(f"   Suspicious Process: {device['suspicious_process_detected']}")
    print(f"   Rule Score: {rule_score}, RL Score: {rl_score}, Final: {final_score}, Trust: {binary_trust}")


    # Step 4: Show human-friendly explanation of the result
    if binary_trust == 0:
        print("   ◍ Trust Evaluation: NOT TRUSTED (binary_trust = 0)")
        print("   ⚠  Reasons:")
        if device["failed_logins"] >= 3:
            print("     - Excessive failed login attempts (>= 3)")
        if device["packets_per_second"] > 10:
            print("     - High network traffic (packets/s > 10)")
        if not device["data_integrity_flag"]:
            print("     - Data integrity compromised (corrupted or malformed)")
        if device["unknown_outbound_connection"]:
            print("     - Unknown outbound connection (possible reverse shell)")
        if device["suspicious_process_detected"]:
            print("     - Suspicious process detected (e.g., bash, powershell)")
    else:
        print("   ◍ Trust Evaluation: TRUSTED (binary_trust = 1)")


    # Step 5: Store result for later output
    result = {
        "device_id": device["device_id"],
        "rule_score": rule_score,
        "rl_score": rl_score,
        "final_score": final_score,
        "binary_trust": binary_trust
    }

    results.append(result)

# Step 6: Create a timestamped filename for JSON log
timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
filename = f"trust_log_{timestamp}.json"

# Step 7: Save results to file
with open(filename, "w") as f:
    json.dump(results, f, indent=4)

# Final Confirmation
print(f"\n✅ Trust evaluation complete. Results saved to {filename}.")

# Step 8: Summary of suspicious devices
suspicious_devices = [res["device_id"] for res in results if res["binary_trust"] == 0]

if suspicious_devices:
    print(f"\n🚨 Suspicious Devices Detected: {', '.join(suspicious_devices)} (binary trust = 0)")
else:
    print("\n☑ All devices passed the trust evaluation.")
```

rules.py

```
#Leon Jacob 13178938 WSU info3016



# rules.py
# This module evaluates a device using rule-based logic.
# If rules are violated, penalties are applied.
# The total rule-based score is returned.

# def rule_based_score(device):
#     """
#     Calculates a trust score based on hard-coded rules.
#     Higher score = more trustworthy.
#     Negative scores indicate risky behavior.
#     """
#     score = 0  # start neutral

#     # Rule 1: Too many failed login attempts
#     if device["failed_logins"] >= 3:
#         score -= 1  # potential brute-force behavior

#     # Rule 2: High packet rate could mean DDoS or abuse
#     if device["packets_per_second"] > 10:
```

```
#         score -= 1


#     # Rule 3: Data was not clean or was malformed

#     if not device["data_integrity_flag"]:

#         score -= 1  # suggests tampering or corruption


#     # Rule 4: Connected to suspicious IP (reverse shell, C2 server,
etc.)

#     if device["unknown_outbound_connection"]:

#         score -= 2  # major red flag


#     # Rule 5: Suspicious process detected (e.g., bash, nc, powershell)

#     if device["suspicious_process_detected"]:

#         score -= 2  # major red flag


#     return score



# Explanation of Rule Logic:

# Rule #                      What It Checks                           Why It
Matters                                   Score Impact

# 1                           failed_logins >= 3
Possible brute-force or password-guessing       -1

# 2                           packets_per_second > 10
Potential DDoS or mass scanning                 -1

# 3                           data_integrity_flag == False             Data
might be corrupted, malicious, or fake      -1

# 4                           unknown_outbound_connection == True      Could
be a reverse shell calling home            -2
```

```
# 5                         suspicious_process_detected == True
Suggests malware/ransomware activity            -2


# What This Returns

# It gives you back an integer score, like:


# 0 = clean


# -1 = minor concern


# -3 = multiple red flags


# -4 or lower = likely hostile










# Leon Jacob 13178938 WSU info3016

# rules.py - Rule-Based Trust Scoring System


def rule_based_score(device):

    """

    Calculates a trust score using hard-coded security rules.

    More violations = lower trust score.

    """
```

```python
    score = 0

    if device["failed_logins"] >= 3:
        score -= 1  # Possible brute-force attempt

    if device["packets_per_second"] > 10:
        score -= 1  # Possible scanning or DDoS

    if not device["data_integrity_flag"]:
        score -= 1  # Corrupted or malformed data

    if device["unknown_outbound_connection"]:
        score -= 2  # Could be reverse shell or exfiltration

    if device["suspicious_process_detected"]:
        score -= 2  # Suspicious shell or malware-like behavior

    return score


# This score is then passed to main.py to be combined with the RL score.
```

rl_agent.py

```
#Leon Jacob 13178938 WSU info3016



# This is a simulated reinforcement learning (RL) trust evaluator,
# designed to evolve into a Q-learning agent later. For now, it's rule-informed
# and gives slightly different weightings from the rule-based system —mimicking a
# learning system that’s been trained on past behavior.



# rl_agent.py
# This module simulates a simple RL agent's trust evaluation.
# # Later, this could evolve into a proper Q-learning or SARSA agent.

# def rl_based_score(device):
#     """
#     Simulates a reinforcement learning trust score based on device behavior.
#     A real RL agent would update its Q-values from past actions and rewards.
#     Here, we just simulate intelligent weighting as a placeholder.
#     """
#     rl_score = 0

#     # Reward: clean data increases trust
#     if device["data_integrity_flag"]:
```

```
#         rl_score += 1

#     # Reward: low traffic is generally benign
#     if device["packets_per_second"] <= 10:
#         rl_score += 1

#     # Penalty: reverse shell-type behavior
#     if device["unknown_outbound_connection"]:
#         rl_score -= 1  # RL has "learned" this is risky

#     # Penalty: suspicious processes
#     if device["suspicious_process_detected"]:
#         rl_score -= 2  # this behavior should be heavily punished

#     return rl_score


# Leon Jacob 13178938 WSU info3016
# rl_agent.py - Simulated Reinforcement Learning Evaluator

def rl_based_score(device):
    """
    Assigns trust points based on simulated learning.
    Rewards expected behavior and penalizes risk.
    """
    rl_score = 0
    if device["data_integrity_flag"]:
        rl_score += 1  # Clean data rewarded

    if device["packets_per_second"] <= 10:
        rl_score += 1  # Low traffic rewarded

    if device["unknown_outbound_connection"]:
        rl_score -= 1  # Penalize suspicious outbound

    if device["suspicious_process_detected"]:
        rl_score -= 2  # Major penalty for suspicious shell behavior

    return rl_score
```

simulator.py

```
#Leon Jacob 13178938 WSU info3016



# need to create a simulator.py to simulate a few fake devices with different   behvaiours

# INCLUDING reverse shells AND stealthy zero day mimic



# simulator.py

# This module simulates incoming device activity.

# Each device is represented as a dictionary of behaviors/telemetry.



# Creates 5 simulated device sessions


# Each one includes:


# Number of failed logins


# Packet traffic level


# Whether data is clean


# Whether outbound connections are shady

# Whether suspicious processes (like reverse shell activity) were detected




# FieldName                             Meaning

# device_id                              Unique name or identifier for each
simulated device

# failed_logins                          Number of login failures during
this session or time window

# packets_per_second                     How much data this device is
sending (basic DoS detection hint)

# data_integrity_flag                    This is what you asked about — see
full explanation below

# unknown_outbound_connection            True if device connected to a
suspicious, unapproved IP (reverse shell behavior)

# suspicious_process_detected            True if known reverse shell
processes (e.g. bash, nc, powershell) were spawned

# activity_type                          General description like login,
upload, idle etc.



# What Is data_integrity_flag?

# This field means: "Was the data sent by the device clean, valid, and
expected?"


# True → data looks legit (structured, unaltered, verified)


# False → data is malformed, suspicious, or tampered with
# Real-World Meaning

# If you were building a real system, this would be detected by:


# Method    What It Does

# Checksums / Hash validation   Is the file/data received the same as the
sender’s copy? (detects tampering)

# Signature verification    If digital signature is missing or invalid,
flag as tampered

# Parsing logic If a sensor sends a JSON payload, but it’s missing fields,
corrupted, or unexpected → it’s not clean

# Anti-malware filters  Detects known malware patterns in uploaded files
or packets

# Protocol compliance   e.g., a TCP packet missing headers or flags →
malformed packet





# In my  Simulation: We’re manually setting data_integrity_flag = False on
certain devices like this:



# {

#     "device_id": "dev_03",

#     ...

#     "data_integrity_flag": False,  ==  Malformed or corrupt data

#     ...

# }

# In your rule-based logic, you can then say:
```

```
# python
# Copy
# Edit
# if not device["data_integrity_flag"]:
#     score -= 1   == Penalize
# So it's a trust signal.


# def get_simulated_devices():
#     return [
#         {
#             "device_id": "dev_01",
#             "failed_logins": 0,
#             "packets_per_second": 5,
#             "data_integrity_flag": True,
#             "unknown_outbound_connection": False,
#             "suspicious_process_detected": False,
#             "activity_type": "data_upload"
#         },
#         {
#             "device_id": "dev_02",  # REVERSE SHELL candidate
#             "failed_logins": 1,
#             "packets_per_second": 2,
#             "data_integrity_flag": True,
#             "unknown_outbound_connection": True,
#             "suspicious_process_detected": True,
#             "activity_type": "idle"
#         },
#         {
#             "device_id": "dev_03",  # Classic misbehavior
#             "failed_logins": 3,
#             "packets_per_second": 15,
#             "data_integrity_flag": False,
#             "unknown_outbound_connection": False,
#             "suspicious_process_detected": False,
#             "activity_type": "login_attempt"
#         },
#         {
#             "device_id": "dev_04",  # Clean, trustworthy device
#             "failed_logins": 0,
#             "packets_per_second": 9,
#             "data_integrity_flag": True,
#             "unknown_outbound_connection": False,
#             "suspicious_process_detected": False,
#             "activity_type": "data_upload"
#         },
#         {
#             "device_id": "dev_05",  # Zero-day mimic: stealthy but suspicious
#             "failed_logins": 0,
#             "packets_per_second": 1,
#             "data_integrity_flag": True,
#             "unknown_outbound_connection": True,
#             "suspicious_process_detected": True,
#             "activity_type": "idle"
#         }
#     ]




# Randomly assign values for:

# failed_logins

# packets_per_second

# data_integrity_flag

# unknown_outbound_connection

# suspicious_process_detected

# Use Python's random module

# Simulate, say, 5 devices per run

# Leon Jacob 13178938 WSU info3016
```

```python
# simulator.py - Random Device Behavior Generator

import random

def generate_device(device_id):
    """
    Randomly generates simulated behavior for a single device.
    """
    return {
        "device_id": device_id,
        "failed_logins": random.randint(0, 5),  # 0 is clean, 5 is brute force
        "packets_per_second": random.randint(1, 25),  # >10 is suspicious
        "data_integrity_flag": random.choice([True, True, False]),  # Mostly clean, some bad
        "unknown_outbound_connection": random.choice([False, True]),  # True may indicate reverse shell
        "suspicious_process_detected": random.choice([False, True]),  # Malware-like behavior
        "activity_type": random.choice(["idle", "data_upload", "login_attempt", "heartbeat"])
    }

def get_simulated_devices(n=5):
    """
    Returns a list of n devices with randomly generated behaviors.
    """
    print("🔄 Generating simulated devices...")
    return [generate_device(f"dev_{i+1:02}") for i in range(n)]
```

trust_log.json

```json
[
    {
        "device_id": "dev_01",
        "rule_score": 0,
        "rl_score": 2,
        "final_score": 2,
        "binary_trust": 1
    },
    {
        "device_id": "dev_02",
        "rule_score": -4,
        "rl_score": -1,
        "final_score": -5,
        "binary_trust": 0
    },
    {
        "device_id": "dev_03",
        "rule_score": -3,
        "rl_score": 0,
        "final_score": -3,
        "binary_trust": 0
    },
    {
        "device_id": "dev_04",
        "rule_score": 0,
        "rl_score": 2,
        "final_score": 2,
        "binary_trust": 1
    },
    {
        "device_id": "dev_05",
        "rule_score": -4,
        "rl_score": -1,
        "final_score": -5,
        "binary_trust": 0
    }
]
```