Executive Summary and App Presentation

INFORMATION SECURITY MANAGEMENT

https://github.com/tbrays/AttackTreeVisualisation

Contents

Executive Summary 2

Introduction 2

Overview of Risk Comparison 2

Visualising Threats 2

Summary 3

Application Presentation 4

Data Storage 4

Visualisation 4

Node Input and Aggregation 5

Testing 6

Automated Testing (with pytest) 6

Code Quality (with pylint) 6

Manual Testing 8

Code and Structure 14

Evaluation 14

Success Criteria 14

Future Development 14

References 15

Bibliography 16

Appendix A – Code Structure 17

Appendix B – Code 18

Executive Summary and App Presentation

Executive Summary

Introduction

Pampered Pets, a small independent retailer, is going digital, embracing e-commerce, cloud inventory technology and online marketing. Digitalisation brings new security threats.

This summary describes an interactive Python based application that was developed to visualise and assess risks before and after digitalisation. The application enables users to browse attack trees, adjust threat probabilities and observe the overall impact in real time through visual feedback.

Overview of Risk Comparison

The application models risk exposure using a structured attack tree approach. The table below summarises the key changes in threat posture, building on the prior risk identification work completed for Pampered Pets, incorporating threat modelling via attack trees to complement the STRIDE and DREAD analysis previously conducted (Brayshaw, 2025).

Aspect	Before (Manual System)	After (Digital System)
Risk Focus	Physical and staff based	Identity, data, and systems
Highest Risk Node	Staff dependency (50%)	Social engineering (65%)
Avg. Node Likelihood	31.4%	42.5%
Visibility of Risks	Low	High (via visual model)
Mitigation Strategy	Process and policy based	Technical and training based

Visualising Threats

The application generates visual attack trees, allowing users to see how risks are structured and propagate through the system.

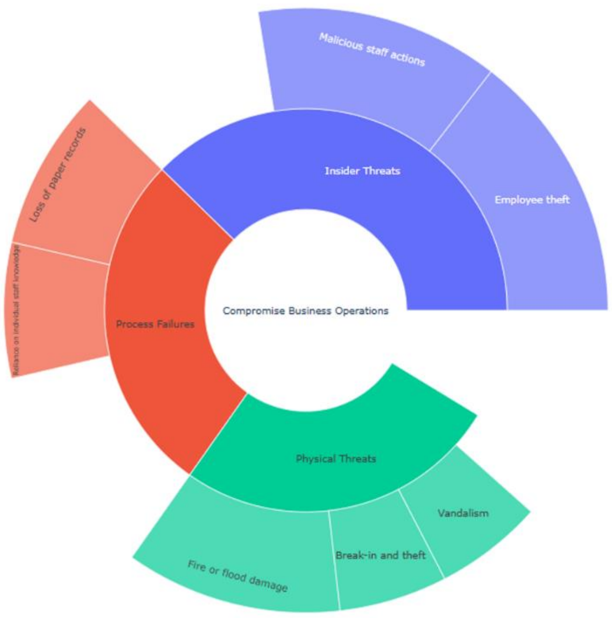


Figure 1 Pre-digitisation Attack Tree

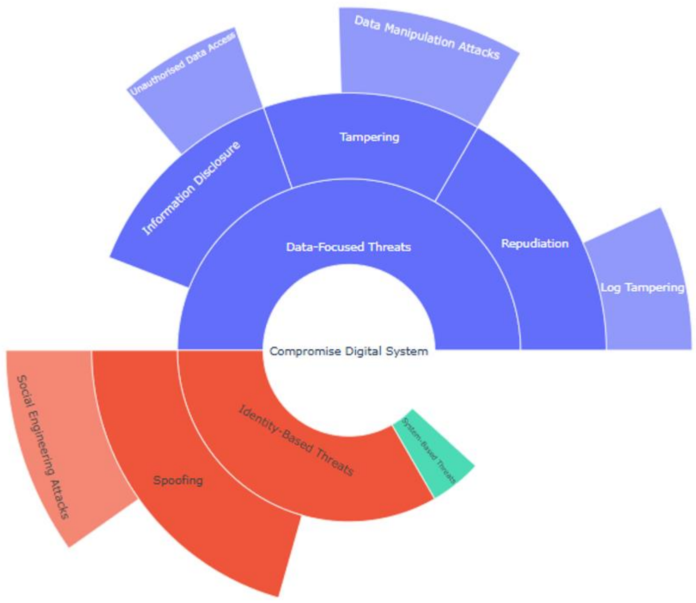


Figure 2 Post-digitisation Attack Tree

Each leaf node's likelihood can be adjusted interactively, with changes reflected instantly in the overall system's risk level. This provides a practical means for management to explore "what if" scenarios.

Summary

The application highlights the following key findings:

Risk Type Shift: Transition from physical/process risk to technical and human centric cyber threats.

Improved Clarity: Digital model provides greater visibility into threat paths and areas needing control.

Actionable Output: Outputs such as average risk, branch risk summaries and most vulnerable node help guide mitigation.

The tool complements the original STRIDE and DREAD analysis by offering an interactive view of threats.

Application Presentation

nicegui	plotly	pylint / pytest
json	copy	collections

Data Storage

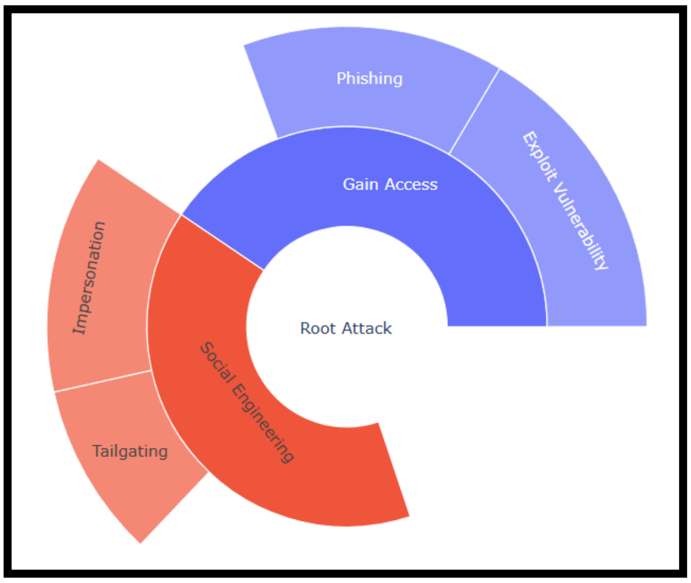
The application uses JSON (JavaScript Object Notation) to define the attack tree structure. Each node includes a label, parent, optional gate (AND/OR) and a likelihood value. This format was chosen for its simplicity, readability and strong support in Python through the json module (Basset, 2015).

JSON is ideal for modelling hierarchical data, such as attack trees, as it allows easy representation of parent-child relationships. Compared to XML or YAML, JSON is more concise and easier to read and write, making it a natural fit for the NiceGUI framework (CelerData, no date).

Three separate JSON files were created, the pre-digital, post-digital and demonstration models.

```
[
  {
    "label": "Gain Access",
    "parent": "Root Attack",
    "gate": "AND"
  },
  {
    "label": "Phishing",
    "parent": "Gain Access",
    "likelihood": 60
  }
]
```

Visualisation



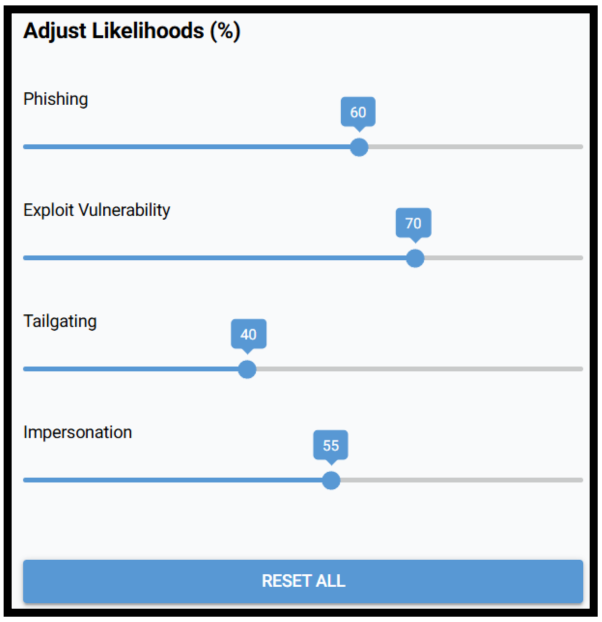
The application uses a Plotly sunburst chart to visualise the attack tree. Nodes are arranged in a radial layout, with colours and segment sizes reflecting their likelihood values. This format provides a clear visual summary of the structure.

Plotly was selected for its ability to generate interactive, browser-based charts with minimal setup and Python integration (Plotly, no date). The *generate_figure()* function in visualisation.py builds the sunburst from node labels, parents and likelihood values.

The chart is embedded using NiceGUI, a modern Python framework that allows developers to build rich web interfaces without needing HTML or JavaScript. It supports real time interaction and smooth integration with Plotly, suitable for a lightweight, Python first dashboard (NiceGUI, no date).

This combination enables an intuitive and interactive experience.

Node Input and Aggregation


Adjust Likelihoods (%)

Phishing 60

Exploit Vulnerability 70

Tailgating 40

Impersonation 55

Users update threat likelihoods via sliders (0–100%).

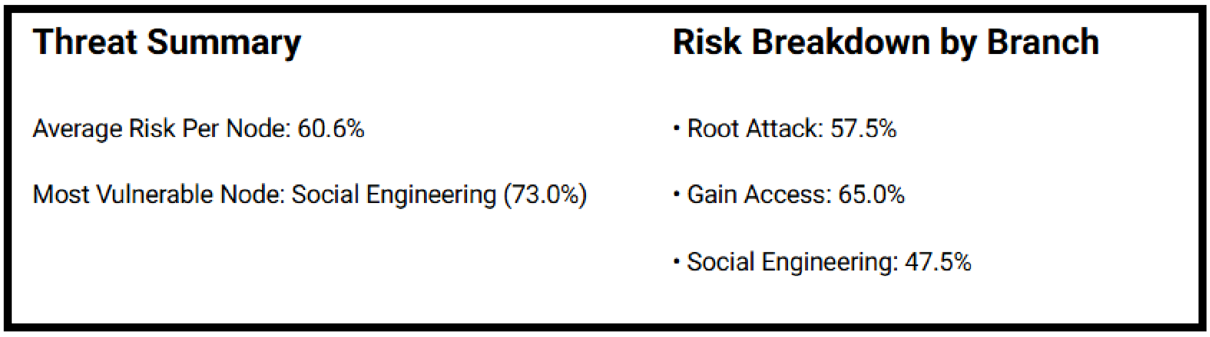
The logic is handled by *compute_likelihood()* in computation.py, which uses recursive calculations based on the logic gates defined in the tree:

- AND gate: All child threats must occur for the parent to occur.
- OR gate: Only one child threat must occur.

This enables realistic propagation of risk across the tree structure. Adjustments instantly trigger updates to both the sunburst visualisation and a summary panel that shows:

- Average risk per node
- Most vulnerable node
- Risk breakdown by category
- High-risk node warnings (≥75%)

This approach supports "what if" analysis, enabling simulation of different risk scenarios.

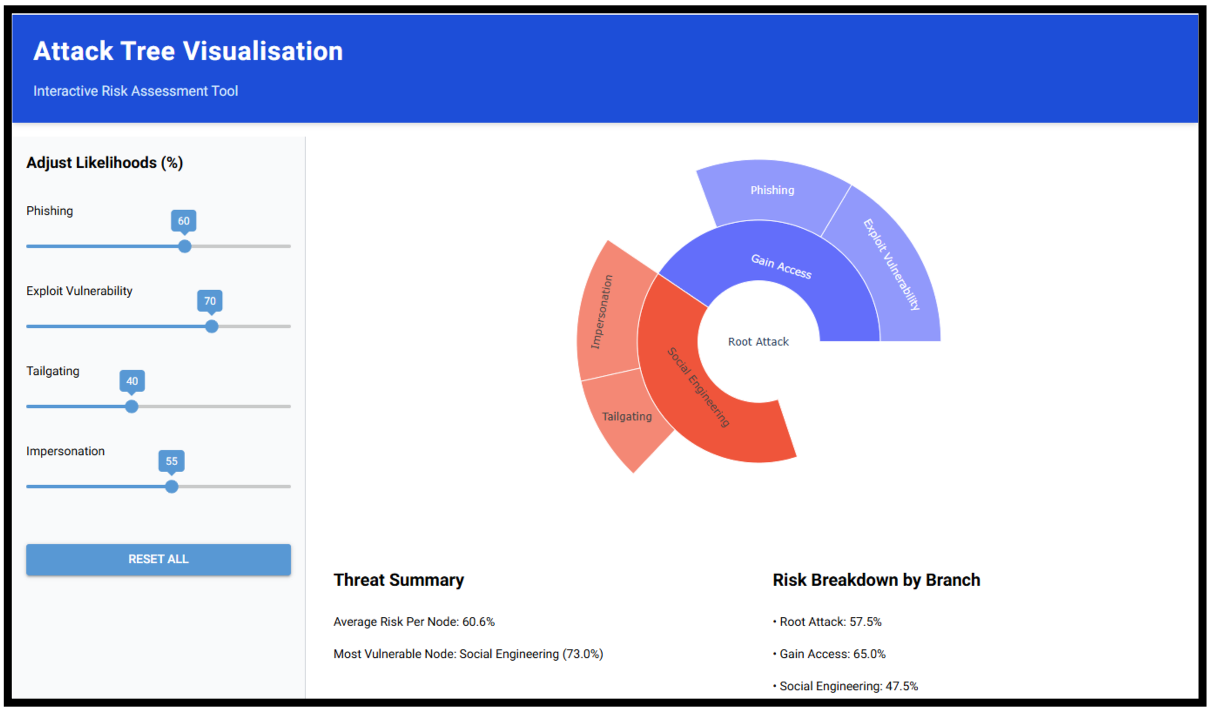

Threat Summary

Average Risk Per Node: 60.6%

Most Vulnerable Node: Social Engineering (73.0%)

Risk Breakdown by Branch

- Root Attack: 57.5%
- Gain Access: 65.0%
- Social Engineering: 47.5%


Attack Tree Visualisation

Interactive Risk Assessment Tool

Adjust Likelihoods (%)

Phishing 60

Exploit Vulnerability 70

Tailgating 40

Impersonation 55

Threat Summary

Average Risk Per Node: 60.6%

Most Vulnerable Node: Social Engineering (73.0%)

Risk Breakdown by Branch

- Root Attack: 57.5%
- Gain Access: 65.0%
- Social Engineering: 47.5%

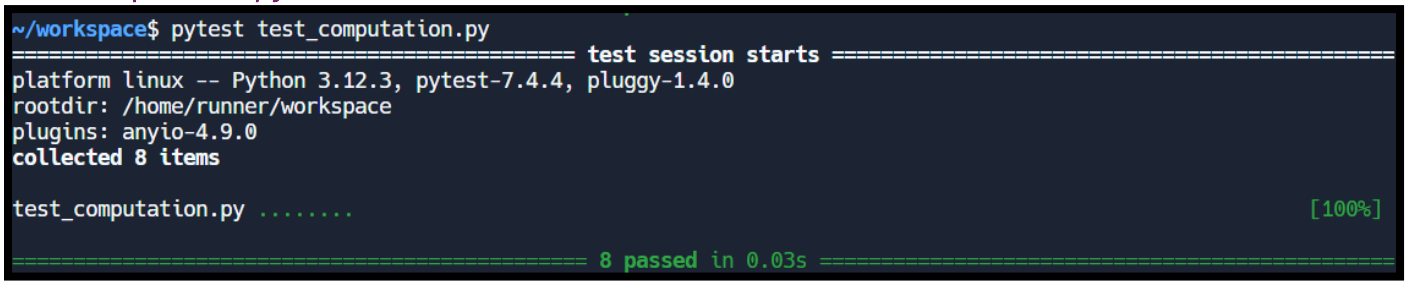
Testing

The application was tested manually and automatically to ensure accuracy and stability.

Automated Testing (with pytest)

Functional logic was tested using pytest, a modern testing framework known for its concise syntax and rich feature set (pytest, no date). It was chosen over alternatives like unittest and doctest due to its improved readability, faster setup and familiarity from prior experience.

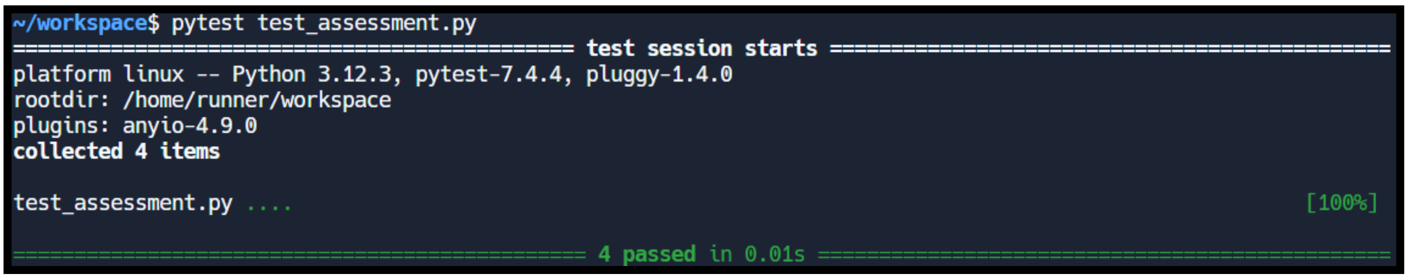
test_computation.py



```
~/workspace$ pytest test_computation.py
============================================== test session starts ==============================================
platform linux -- Python 3.12.3, pytest-7.4.4, pluggy-1.4.0
rootdir: /home/runner/workspace
plugins: anyio-4.9.0
collected 8 items

test_computation.py ........                                                                              [100%]

=============================================== 8 passed in 0.03s ===============================================
```

test_assessment.py

The summary logic from assessment.py was extracted into a helper function within the test file to enable unit testing without modifying the original module.



```
~/workspace$ pytest test_assessment.py
============================================== test session starts ==============================================
platform linux -- Python 3.12.3, pytest-7.4.4, pluggy-1.4.0
rootdir: /home/runner/workspace
plugins: anyio-4.9.0
collected 4 items

test_assessment.py ....                                                                                   [100%]

=============================================== 4 passed in 0.01s ===============================================
```

The state_handlers.py module was excluded from unit testing due to its tight coupling with GUI elements and reliance on external UI updates. Logic within this module was verified through manual testing.

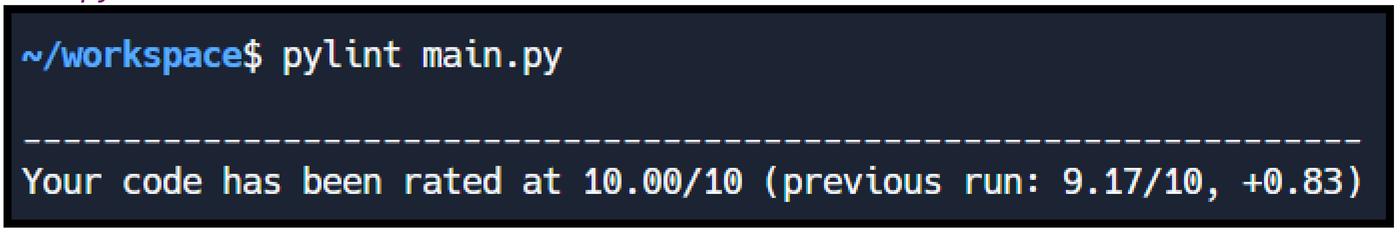
Code Quality (with pylint)

Code style and structure were evaluated using pylint, which provides detailed analysis including:

- Adherence to PEP8 and PEP257
- Detection of unused variables

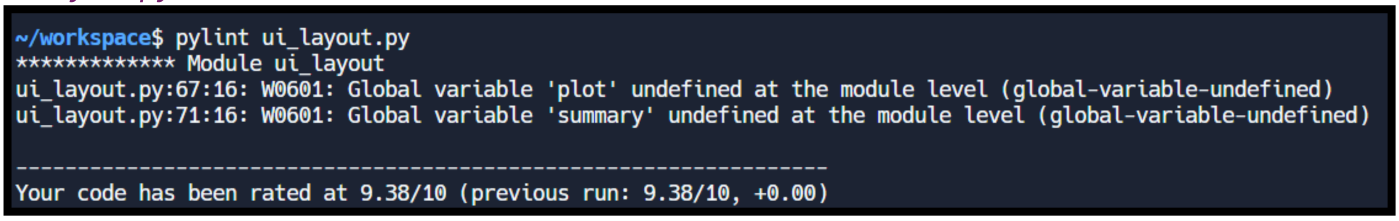
Alternatives like flake8 were considered. flake8 is lightweight but provides less contextual feedback. pylint was chosen for its thoroughness familiarity from prior experience (Pylint, no date).

main.py

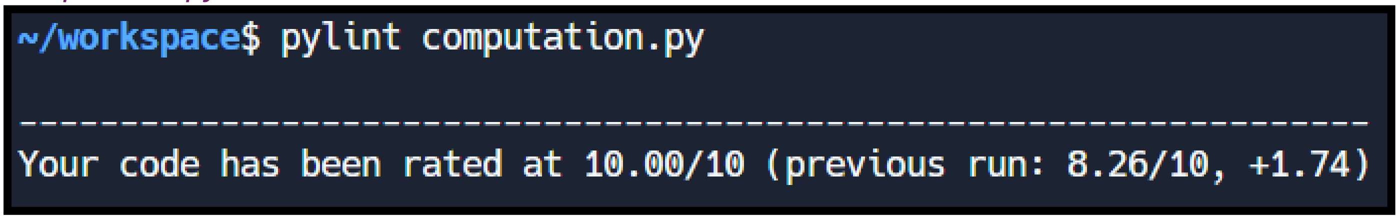


```
~/workspace$ pylint main.py

-------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 9.17/10, +0.83)
```

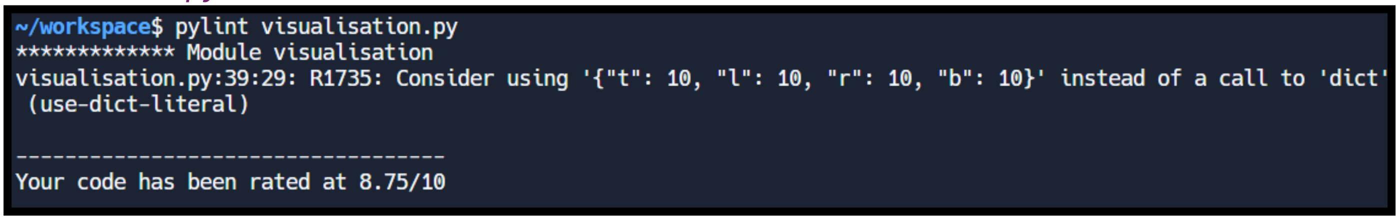
ui_layout.py


```
~/workspace$ pylint ui_layout.py
************* Module ui_layout
ui_layout.py:67:16: W0601: Global variable 'plot' undefined at the module level (global-variable-undefined)
ui_layout.py:71:16: W0601: Global variable 'summary' undefined at the module level (global-variable-undefined)

------------------------------------------------------------------
Your code has been rated at 9.38/10 (previous run: 9.38/10, +0.00)
```

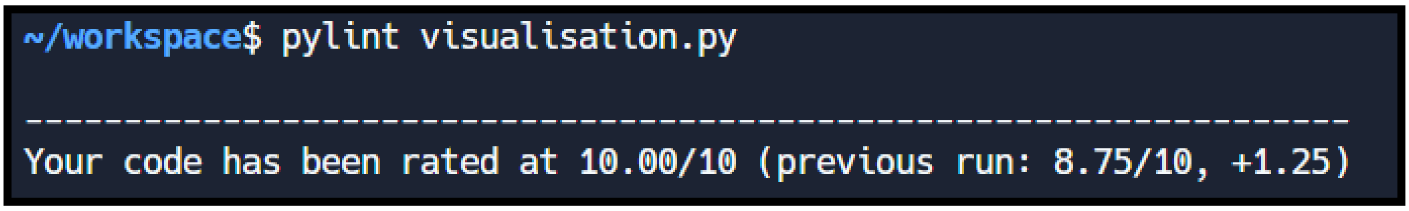
computation.py


```
~/workspace$ pylint computation.py

-------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 8.26/10, +1.74)
```

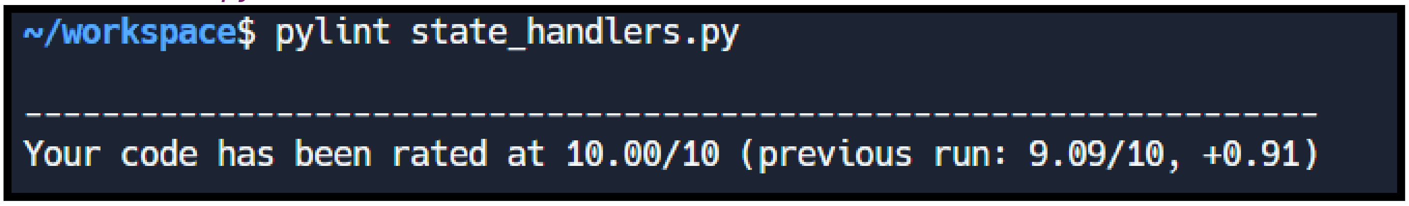
visualisation.py


```
~/workspace$ pylint visualisation.py
************* Module visualisation
visualisation.py:39:29: R1735: Consider using '{"t": 10, "l": 10, "r": 10, "b": 10}' instead of a call to 'dict'
 (use-dict-literal)

-----------------------------------
Your code has been rated at 8.75/10
```

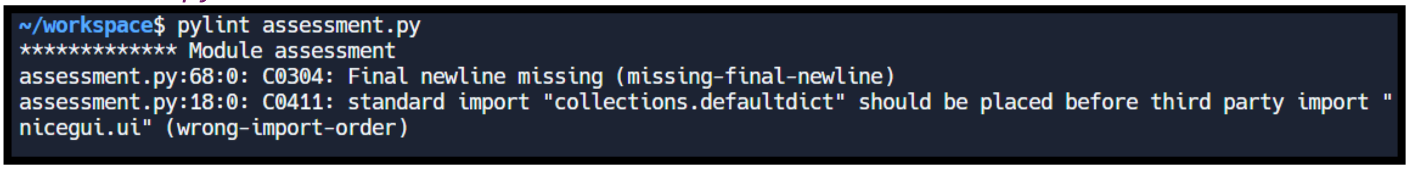
The linter recommended replacing dict() with literal syntax {...} as it is more concise, aligns with standard Python style and is slightly more efficient.


```
~/workspace$ pylint visualisation.py

-------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 8.75/10, +1.25)
```

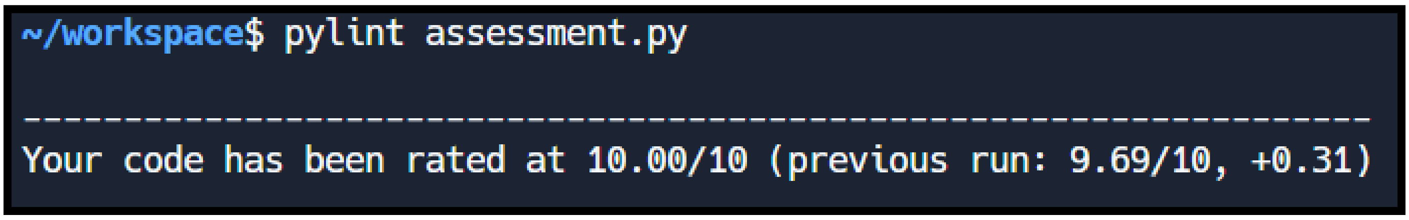
state_handlers.py


```
~/workspace$ pylint state_handlers.py

-------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 9.09/10, +0.91)
```

assessment.py


```
~/workspace$ pylint assessment.py
************* Module assessment
assessment.py:68:0: C0304: Final newline missing (missing-final-newline)
assessment.py:18:0: C0411: standard import "collections.defaultdict" should be placed before third party import "
nicegui.ui" (wrong-import-order)
```

Added new line at the end of document and changed order of import.


```
~/workspace$ pylint assessment.py

-------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 9.69/10, +0.31)
```

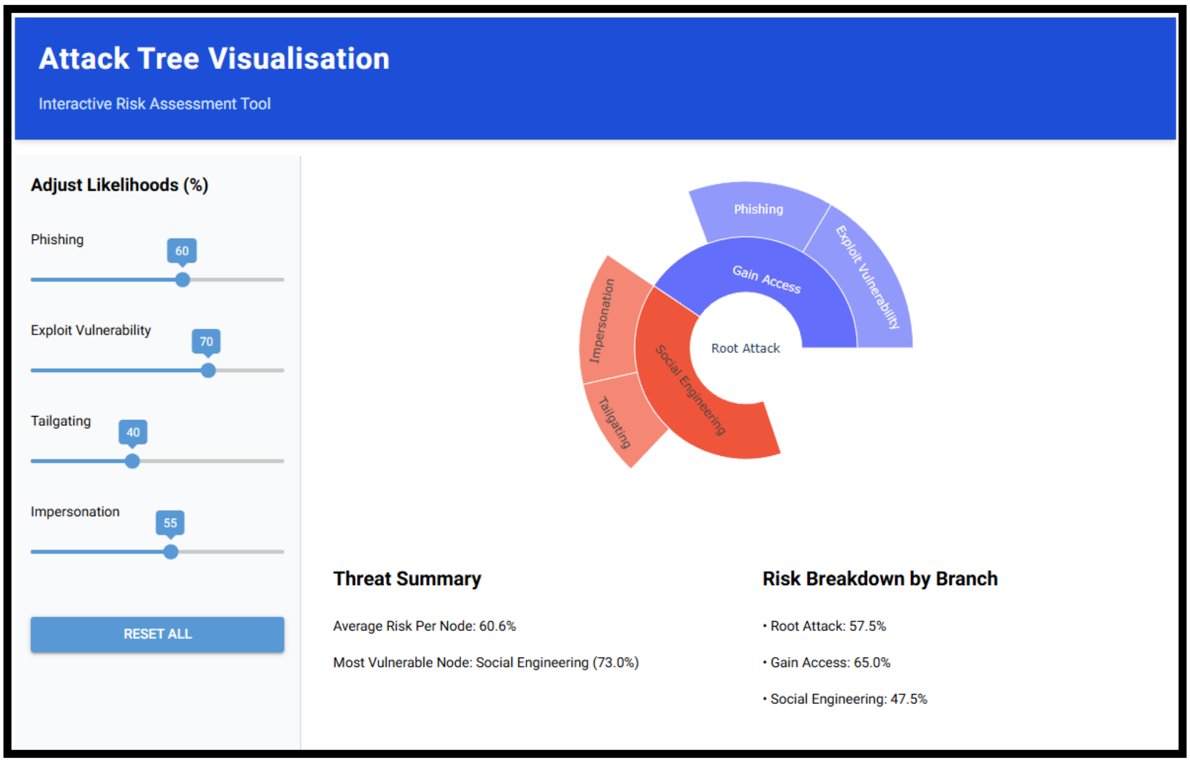
Manual Testing

Manual tests confirmed UI responsiveness, JSON handling, slider accuracy and real time summary updates. Edge cases were handled without errors.

Test No.	Feature to Test	Test Description	Expected Outcome
1	Application Launch	Run the app using python main.py	GUI loads with sliders, chart and summary panel
2	Tree Visualisation	Observe the sunburst chart when the app starts	Nodes and their relationships are shown clearly
3	Adjust Slider	Move a leaf node slider to a 75%	Chart and summary panel update in real time
4	Summary Updates	Observe the summary panel after slider change	Average risk and highest risk node reflect the new values
5	Reset Button	After changing one or more sliders, click "Reset All"	All sliders return to original values. Chart and summary update accordingly
6	High Risk Warning	Increase a slider to 90%	Node appears in “High Risk Nodes” warning section
7	UI Responsiveness	Move multiple sliders rapidly	No freezing or lag. Chart responds smoothly
8	Likelihood Edge Case: 0%	Set a leaf node to 0%	Risk propagates properly. Parent and root nodes update
9	Likelihood Edge Case: 100%	Set a leaf node to 100%	Risk propagates properly through chart and summary
10	Visual Feedback	Hover over chart segments	Tooltip displays label and likelihood

Test One

Test No.	Feature to Test	Test Description	Expected Outcome	Actual Outcome
1	Application Launch	Run the app using python main.py	GUI loads with sliders, chart and summary panel	Success



Threat Summary

Average Risk Per Node: 60.6%

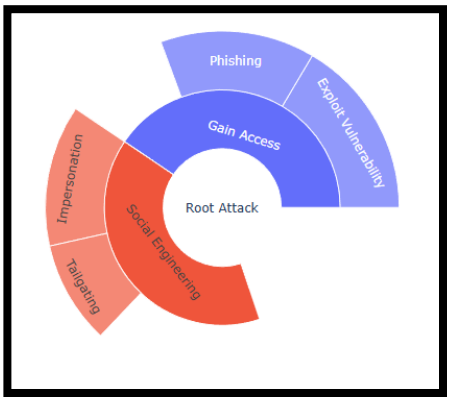
Most Vulnerable Node: Social Engineering (73.0%)

Risk Breakdown by Branch

- Root Attack: 57.5%
- Gain Access: 65.0%
- Social Engineering: 47.5%

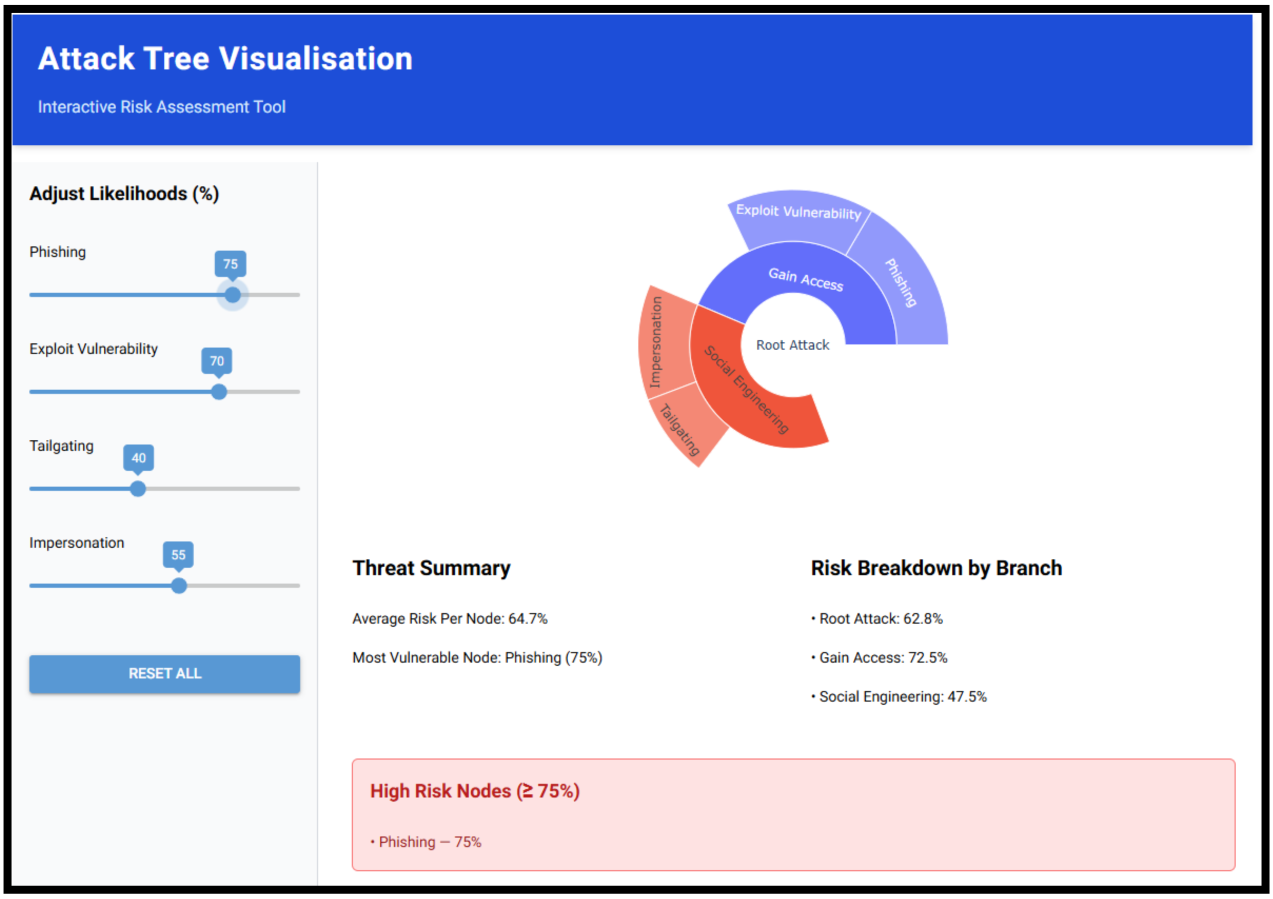
Test Two

Test No.	Feature to Test	Test Description	Expected Outcome	Actual Outcome
2	Tree Visualisation	Observe the sunburst chart when the app starts	Nodes and their relationships are shown clearly	Visible and clear



Test Three

Test No.	Feature to Test	Test Description	Expected Outcome	Actual Outcome
3	Adjust Slider	Move a leaf node slider to a 75%	Chart and summary panel update in real time	Works as expected



Attack Tree Visualisation

Interactive Risk Assessment Tool

Adjust Likelihoods (%)

Phishing	75
Exploit Vulnerability	70
Tailgating	40
Impersonation	55

Threat Summary

Average Risk Per Node: 64.7%

Most Vulnerable Node: Phishing (75%)

Risk Breakdown by Branch

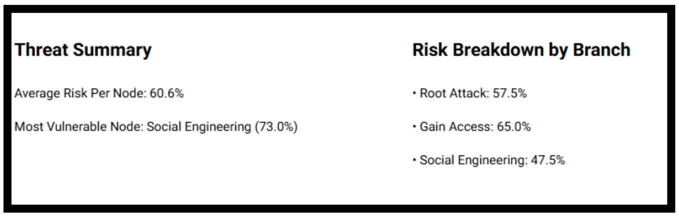
- Root Attack: 62.8%
- Gain Access: 72.5%
- Social Engineering: 47.5%

High Risk Nodes (≥ 75%)

- Phishing – 75%

Test Four

Test No.	Feature to Test	Test Description	Expected Outcome	Actual Outcome
4	Summary Updates	Observe the summary panel after slider change	Average risk and highest risk node reflect the new values	Correct

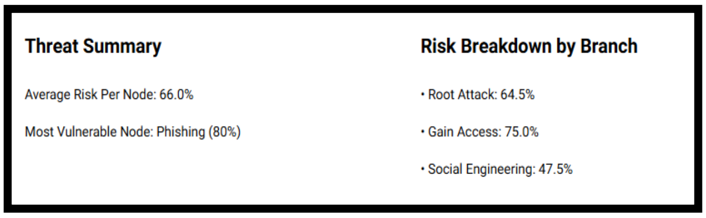

Threat Summary

Average Risk Per Node: 60.6%

Most Vulnerable Node: Social Engineering (73.0%)

Risk Breakdown by Branch

- Root Attack: 57.5%
- Gain Access: 65.0%
- Social Engineering: 47.5%


Threat Summary

Average Risk Per Node: 66.0%

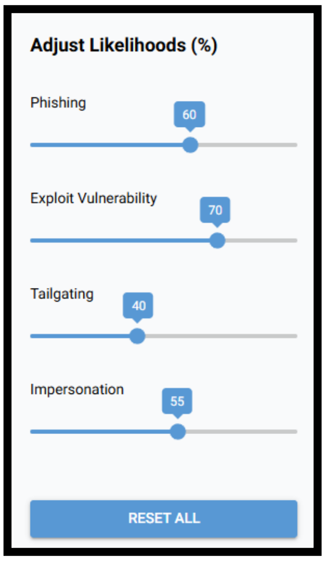
Most Vulnerable Node: Phishing (80%)

Risk Breakdown by Branch

- Root Attack: 64.5%
- Gain Access: 75.0%
- Social Engineering: 47.5%

Test Five

Test No.	Feature to Test	Test Description	Expected Outcome	Actual Outcome
5	Reset Button	After changing one or more sliders, click "Reset All"	All sliders return to original values. Chart and summary update accordingly	Resets correctly

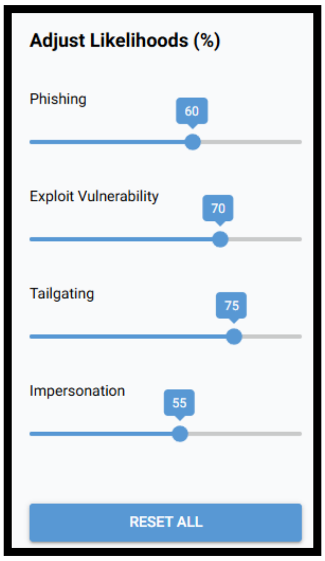

Adjust Likelihoods (%)

Phishing 60

Exploit Vulnerability 70

Tailgating 40

Impersonation 55

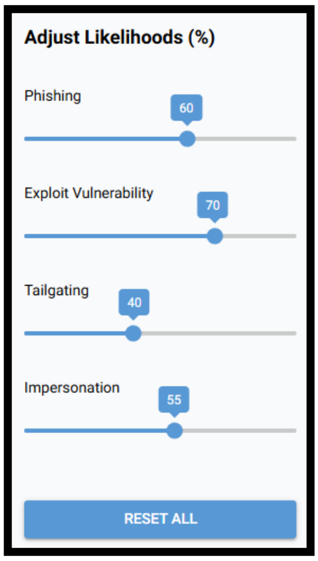

Adjust Likelihoods (%)

Phishing 60

Exploit Vulnerability 70

Tailgating 75

Impersonation 55


Adjust Likelihoods (%)

Phishing 60

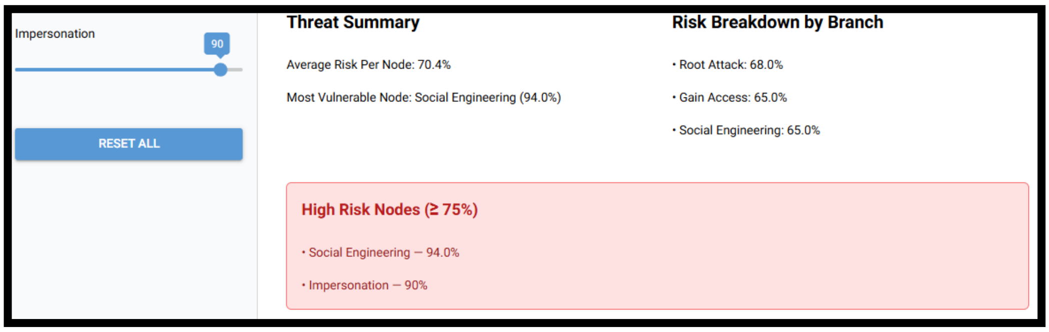
Exploit Vulnerability 70

Tailgating 40

Impersonation 55

Test Six

Test No.	Feature to Test	Test Description	Expected Outcome	Actual Outcome
6	High Risk Warning	Increase a slider to 90%	Node appears in "High Risk Nodes" warning section	Displays correctly


Impersonation 90

Threat Summary

Average Risk Per Node: 70.4%

Most Vulnerable Node: Social Engineering (94.0%)

Risk Breakdown by Branch

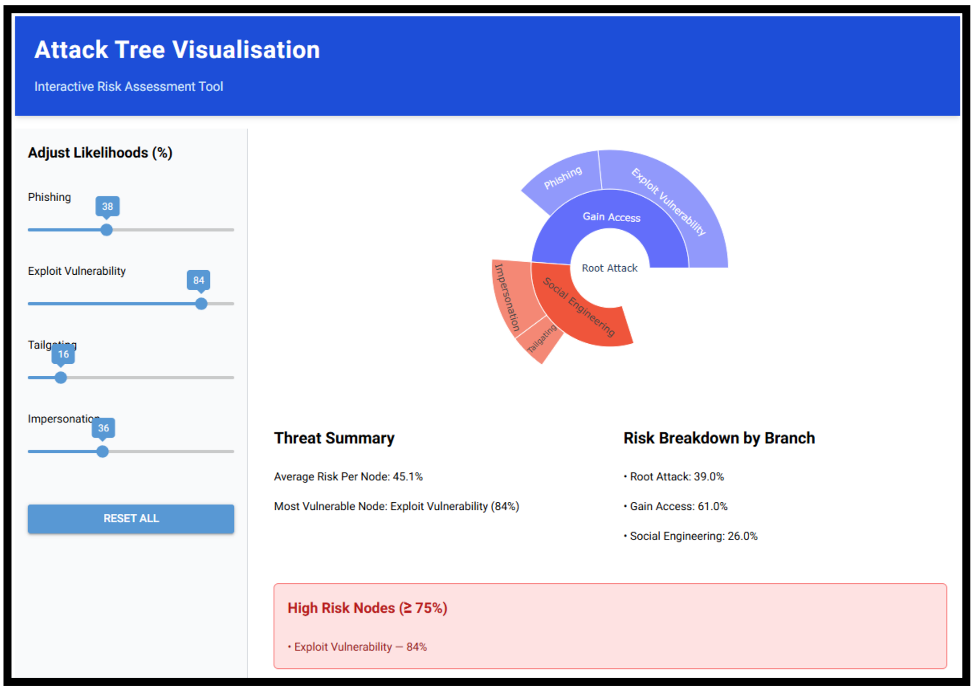
- Root Attack: 68.0%
- Gain Access: 65.0%
- Social Engineering: 65.0%

High Risk Nodes (≥ 75%)

- Social Engineering — 94.0%
- Impersonation — 90%

Test Seven

Test No.	Feature to Test	Test Description	Expected Outcome	Actual Outcome
7	UI Responsiveness	Move multiple sliders rapidly	No freezing or lag. Chart responds smoothly	Responsive



Attack Tree Visualisation
Interactive Risk Assessment Tool

Adjust Likelihoods (%)

Phishing 38
Exploit Vulnerability 84
Tailgating 16
Impersonation 36

RESET ALL

Threat Summary

Average Risk Per Node: 45.1%
Most Vulnerable Node: Exploit Vulnerability (84%)

Risk Breakdown by Branch

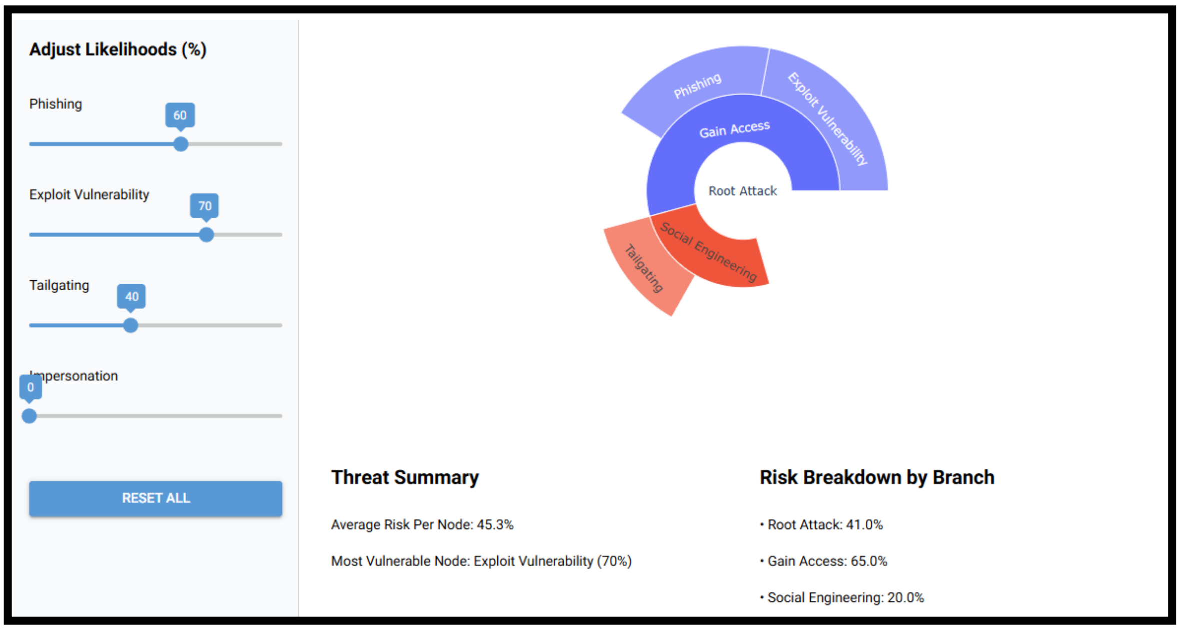
- Root Attack: 39.0%
- Gain Access: 61.0%
- Social Engineering: 26.0%

High Risk Nodes (≥ 75%)

- Exploit Vulnerability — 84%

Test Eight

Test No.	Feature to Test	Test Description	Expected Outcome	Actual Outcome
8	Likelihood Edge Case: 0%	Set a leaf node to 0%	Risk propagates properly. Parent and root nodes update	Handled correctly



Adjust Likelihoods (%)

Phishing 60
Exploit Vulnerability 70
Tailgating 40
Impersonation 0

RESET ALL

Threat Summary

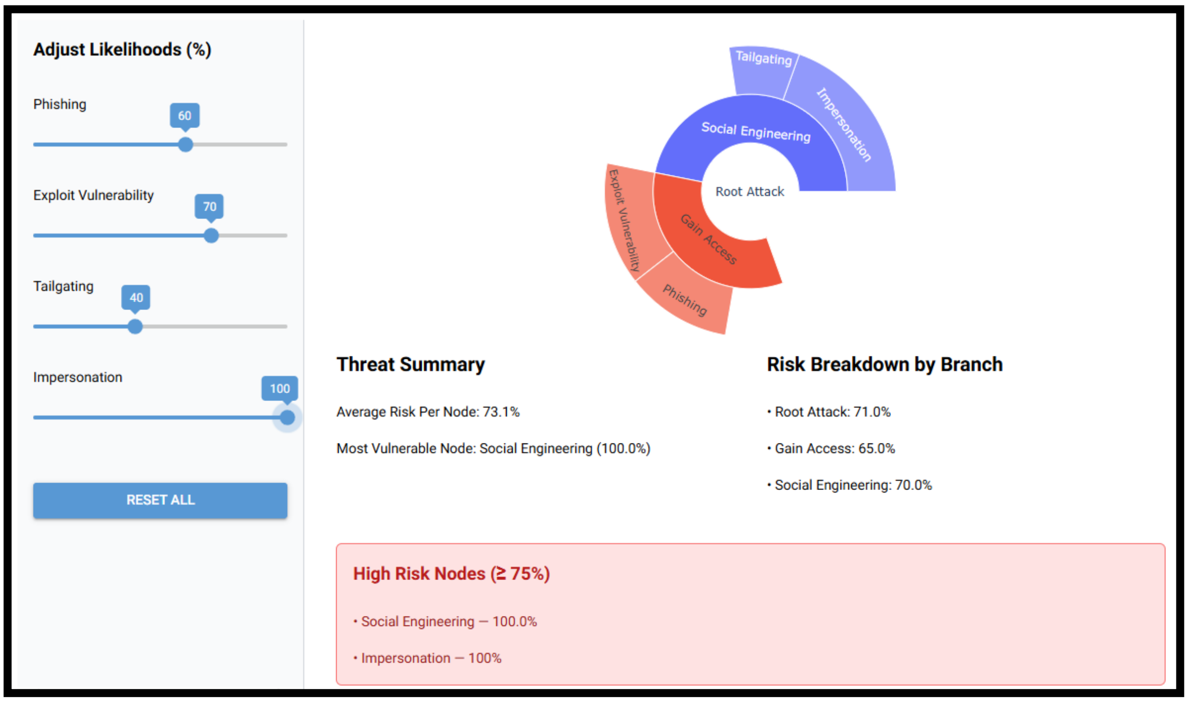
Average Risk Per Node: 45.3%
Most Vulnerable Node: Exploit Vulnerability (70%)

Risk Breakdown by Branch

- Root Attack: 41.0%
- Gain Access: 65.0%
- Social Engineering: 20.0%

Test Nine

Test No.	Feature to Test	Test Description	Expected Outcome	Actual Outcome
9	Likelihood Edge Case: 100%	Set a leaf node to 100%	Risk propagates properly through chart and summary	Handled correctly



Threat Summary

Average Risk Per Node: 73.1%

Most Vulnerable Node: Social Engineering (100.0%)

Risk Breakdown by Branch

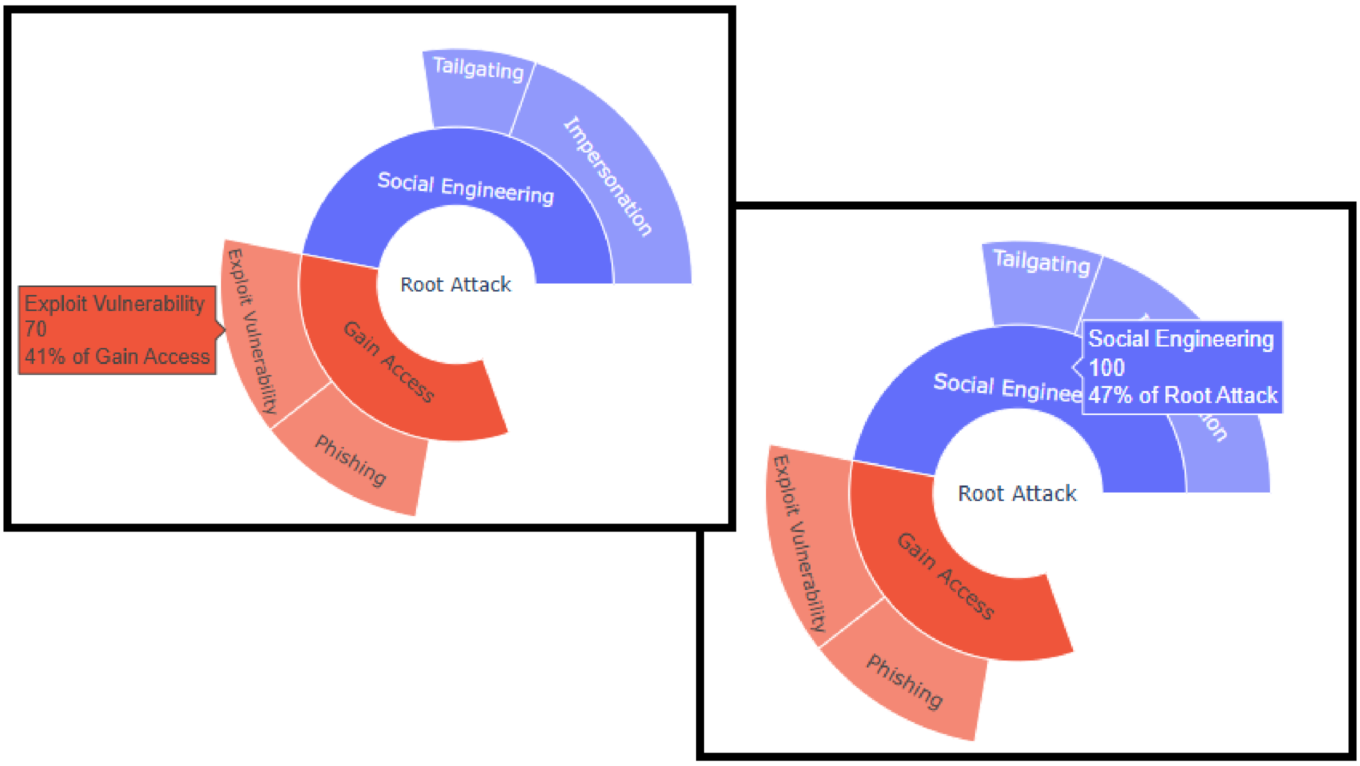
- Root Attack: 71.0%
- Gain Access: 65.0%
- Social Engineering: 70.0%

High Risk Nodes (≥ 75%)

- Social Engineering – 100.0%
- Impersonation – 100%

Test Ten

Test No.	Feature to Test	Test Description	Expected Outcome	Actual Outcome
10	Visual Feedback	Hover over chart segments	Tooltip displays label and likelihood	Tooltips appear



Code and Structure

The application follows a modular design to promote readability, maintainability and ease of extension. Each Python file handles a distinct function:

- main.py: Loads the attack tree and starts the UI.
- ui_layout.py: Builds the interface using NiceGUI components.
- computation.py: Contains recursive logic for calculating threat likelihoods.
- visualisation.py: Generates the Plotly sunburst chart.
- state_handlers.py: Responds to UI events and user input.
- assessment.py: Computes risk summaries and renders the results.

Code was written in line with PEP 8, encouraging consistent naming, spacing and structure (van Rossum, Warsaw and Coghlan, 2001). Functions and modules are documented using PEP 257 standards for docstrings (Goodger and van Rossum, 2001), supporting future maintenance.

Evaluation

Success Criteria

The application meets all the key requirements outlined in the brief:

- It accepts threat trees in a standard JSON format.
- Visualises the structure using an interactive Plotly sunburst chart.
- Allows user input of likelihood values on leaf nodes.
- Provides a summary panel with average risk, branch breakdowns and high-risk nodes.

It also fulfils the objective of comparing the pre-digital and post-digital threat models, enabling clear, data driven evaluation of risk before and after digitalisation.

Future Development

- Toggle between probability based and monetary based risk scoring.
- Support for importing trees in XML or YAML formats.
- Save/load capability to switch between different tree models.
- Report export options (PDF or CSV).

This application enhances Pampered Pets' ability to model and understand cyber risks associated with digitalisation. By combining visual modelling with interactive features, it supports informed, data driven decision making that aligns with the business's evolving digital needs.

References

Bassett, L. (2015) *Introduction to JavaScript Object Notation: A To-the-Point Guide to JSON*. Sebastopol, CA: O'Reilly Media.

Brayshaw, T. (2025) *Risk Identification Report*. ISM: Information Security Management. Report submitted to the University of Essex Online.

CelerData (no date) *YAML, JSON, and XML: A Practical Guide to Choosing the Right Format*. Available at: https://celerdata.com/glossary/yaml-json-and-xml-a-practical-guide-to-choosing-the-right-format (Accessed: 12 April 2025).

Goodger, D. and van Rossum, G. (2001) *PEP 257 – Docstring Conventions*. Available at: https://peps.python.org/pep-0257/ (Accessed: 18 April 2025).

NiceGUI (no date) *Documentation*. Available at: https://nicegui.io/documentation (Accessed: 12 April 2025).

Plotly (no date) *Plotly for Python*. Available at: https://plotly.com/python/ (Accessed: 12 April 2025).

Pylint (no date) *Pylint documentation*. Available at: https://pylint.readthedocs.io/en/stable/ (Accessed: 15 April 2025).

pytest (no date) *pytest Documentation*. Available at: https://docs.pytest.org/en/stable/ (Accessed: 15 April 2025).

van Rossum, G., Warsaw, B. and Coghlan, N. (2001) *PEP 8 – Style Guide for Python Code*. Available at: https://peps.python.org/pep-0008/ (Accessed: 18 April 2025).

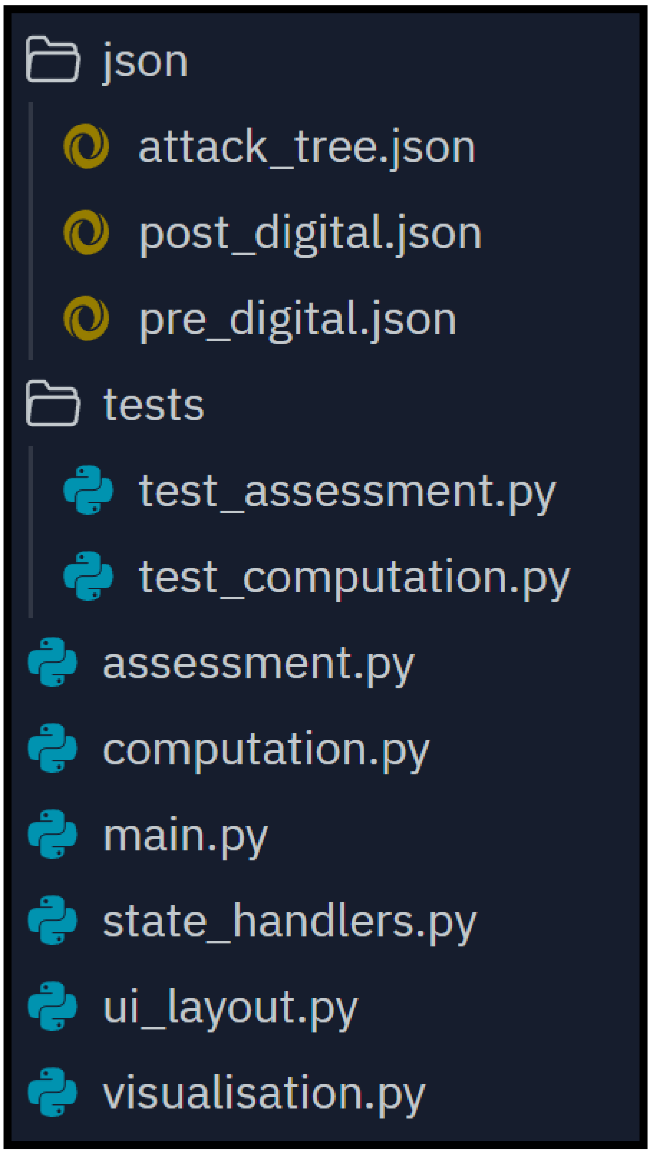
Bibliography

Intellipaat (2023) *How To Create A Sunburst Chart With Plotly*. Available at: https://www.youtube.com/watch?v=RdaV9dvC6sc (Accessed: 12 April 2025).

JCharisTech (2024) *NiceGUI Python Crash Course*. Available at: https://www.youtube.com/watch?v=B-45Ps3ZGzc (Accessed: 12 April 2025).

zauberzeug (2025) *NiceGUI*. Available at: https://github.com/zauberzeug/nicegui (Accessed: 15 April 2025).

Appendix A – Code Structure



```
json
    attack_tree.json
    post_digital.json
    pre_digital.json
tests
    test_assessment.py
    test_computation.py
assessment.py
computation.py
main.py
state_handlers.py
ui_layout.py
visualisation.py
```

Appendix B – Code

main.py

```
"""
main.py

Entry point for the threat modelling application.

This script loads an attack tree from a JSON file and starts a NiceGUI interface,
allowing users to visualise risks, adjust node likelihoods and see live risk summaries.
"""

import json
import copy
from nicegui import ui
from ui_layout import build_ui

def main():
    """
    Launches the threat modelling application.

    Loads an attack tree from a JSON file and starts a NiceGUI interface where users
can:
    - Visualise the attack tree using a Plotly sunburst chart.
    - Adjust likelihoods of leaf nodes via sliders.
    - View real time risk summaries.

    Run this file to start the app:
        python main.py
    """
    # Load the attack tree from a JSON file, switch to pre/post trees as needed
    with open('attack_tree.json', 'r', encoding='utf-8') as f:
        attack_tree = json.load(f)

    # Preserve the original tree for reset functionality
    original_tree = copy.deepcopy(attack_tree)

    # Build the UI and run the app
    build_ui(attack_tree, original_tree)
    ui.run()


if __name__ in {"__main__", "__mp_main__"}:
    main()
```

ui_layout.py

```
"""
ui_layout.py

Defines the user interface layout for the threat modelling application using NiceGUI.

This module builds the full UI, including:
- A sidebar with sliders for adjusting likelihoods of leaf nodes.
- A main area displaying an interactive Plotly sunburst chart.
- A summary panel showing calculated risk metrics.
- A reset button to restore original values.

Functions:
- build_ui(attack_tree, original_tree)
"""

from nicegui import ui
from computation import is_leaf, compute_likelihood
from visualisation import generate_figure
from assessment import update_assessment
from state_handlers import update_value_by_label, reset_all


def build_ui(attack_tree, original_tree):
   """
   Construct the main UI layout for the attack tree visualisation.

   Builds the following UI structure:
      - Top banner (title and subtitle).
      - Left sidebar (sliders for adjusting leaf node likelihoods).
      - Right content (Plotly sunburst chart and summary panel).
      - Footer (branding information).
   """
   slider_refs = {}

   with ui.column().classes("w-full h-screen"):

      # Header
      with ui.row().classes("w-full bg-blue-700 text-white p-6 shadow-md"):
         with ui.column().classes("w-full"):
            ui.label("Attack Tree Visualisation").classes("text-3xl font-bold tracking-wide")
            ui.label("Interactive Risk Assessment Tool").classes("text-base text-blue-100")

      with ui.row().classes("w-full flex-1 items-stretch no-wrap"):

         # Left Sidebar
         with ui.column().classes("w-1/4 p-4 bg-gray-50 overflow-auto border-r border-gray-300"):
            ui.label("Adjust Likelihoods (%)").classes("text-lg font-semibold mb-4")

            # Placeholders to bind handlers later
            plot = None
            summary = None

            for node in attack_tree:
               root_label = next((n["label"] for n in attack_tree if n["parent"] == ""), None)
               if node["label"] != root_label and is_leaf(node["label"], attack_tree):
                  label = node["label"]
                  with ui.row().classes("items-center w-full mb-3"):
                     ui.label(label).classes("text-sm whitespace-nowrap mr-2")
                     slider = ui.slider(
                        min=0,
                        max=100,
                        value=node["likelihood"],
                        step=1
                     ).props('label-always instant').classes("w-full")

                     slider_refs[label] = slider
                     slider.on("update:model-value", lambda e, label=label: update_value_by_label(
                        label, e.value, attack_tree, plot, summary
                     ))

            reset_button = ui.button("Reset All").classes("mt-6 bg-blue-600 text-white w-full")

         # Right Content Panel
         with ui.column().classes("w-3/4 p-4 h-full overflow-auto"):
            compute_likelihood("Root Attack", attack_tree)
            plot = ui.plotly(generate_figure(attack_tree)).classes("w-full h-[60%]")

            summary = ui.row().classes("w-full mt-6 gap-8")
```

computation.py

```python
"""
computation.py

Provides core logic for calculating threat likelihoods within an attack tree.

This module supports recursive aggregation of risk values from leaf nodes upward,
based on logical gates defined at each parent node.

Functions:
- is_leaf(label, attack_tree): Returns True if the node has no children.
- get_children(label, attack_tree): Retrieves immediate child nodes of a given parent.
- compute_likelihood(label, attack_tree): Recursively calculates the likelihood of a node
  based on its children and logic gate (AND/OR).
"""

def is_leaf(label, attack_tree):
    """
    Determine if a node has no children in the attack tree.

    Args:
        label (str): The label of the node.
        attack_tree (list[dict]): The tree structure.

    Returns:
        bool: True if node has no children, False otherwise.
    """
    return all(n['parent'] != label for n in attack_tree)


def get_children(label, attack_tree):
    """
    Get immediate child nodes of a given parent.

    Args:
        label (str): Parent node label.
        attack_tree (list[dict]): The tree structure.

    Returns:
        list[dict]: Child node dictionaries.
    """
    return [n for n in attack_tree if n['parent'] == label]


def compute_likelihood(label, attack_tree):
    """
    Compute likelihood of a node recursively using AND/OR logic.

    Args:
        label (str): The label of the node.
        attack_tree (list[dict]): The tree structure.

    Returns:
        float: Calculated likelihood (0-100).
    """
    node = next((n for n in attack_tree if n['label'] == label), None)
    if node is None:
        return 0

    if is_leaf(label, attack_tree):
        return node.get("likelihood", 0)

    children = get_children(label, attack_tree)
    child_probs = [compute_likelihood(child["label"], attack_tree) / 100 for child in children]

    gate = node.get("gate", "OR").upper()
    if gate == "AND":
        prob = 1
        for p in child_probs:
            prob *= p
    else:  # OR logic
        prob = 1
        for p in child_probs:
            prob *= (1 - p)
        prob = 1 - prob

    node["likelihood"] = round(prob * 100, 1)
    return node["likelihood"]
```

visualisation.py

```
"""
visualisation.py

Generates visual representations of the attack tree using Plotly.

This module provides functionality to create an interactive sunburst chart
based on the hierarchical structure and likelihood values of the threat model.

Functions:
- generate_figure(attack_tree): Returns a Plotly sunburst chart visualising
  node relationships and likelihood values from the attack tree.
"""

import plotly.graph_objs as go

def generate_figure(attack_tree):
    """
    Generate Plotly sunburst chart of attack tree.

    Args:
        attack_tree (list[dict]): The current attack tree data.

    Returns:
        plotly.graph_objs.Figure: Sunburst chart object.
    """
    labels = [n["label"] for n in attack_tree]
    parents = [n["parent"] for n in attack_tree]
    values = [n["likelihood"] for n in attack_tree if "likelihood" in n]


    fig = go.Figure(go.Sunburst(
        labels=labels,
        parents=parents,
        values=values,
        branchvalues='remainder',
        hoverinfo='label+value+percent parent',
        maxdepth=-1
    ))

    fig.update_layout(margin={"t": 10, "l": 10, "r": 10, "b": 10})
    return fig
```

state_handlers.py

```python
"""
state_handlers.py

Provides functions to manage user interactions and update application state.

This module handles UI-triggered events such as:
- Updating likelihood values from slider input
- Recomputing the attack tree and refreshing the chart
- Resetting all values to their original state

Functions:
- update_value_by_label: Updates a leaf node’s value and refreshes the visual.
- update_chart: Recomputes and redraws the chart and summary.
- reset_all: Restores original likelihood values and slider positions.
"""

from computation import compute_likelihood, is_leaf
from visualisation import generate_figure
from assessment import update_assessment

def update_value_by_label(label, new_value, attack_tree, plot, summary):
    """
    Update likelihood value of a leaf node and refresh UI.

    Args:
        label (str): Node label.
        new_value (float): New likelihood (0-100).
        attack_tree (list[dict]): The tree data.
        plot (ui.plotly): The plotly visual object.
        summary (ui.element): The summary container.
    """
    for node in attack_tree:
        if node["label"] == label:
            node["likelihood"] = new_value
            break
    update_chart(attack_tree, plot, summary)


def update_chart(attack_tree, plot, summary):
    """
    Recompute and update chart and summary panel.

    Args:
        attack_tree (list[dict])
        plot (ui.plotly)
        summary (ui.element)
    """
    compute_likelihood("Root Attack", attack_tree)
    plot.update_figure(generate_figure(attack_tree))
    update_assessment(attack_tree, summary)


def reset_all(attack_tree, original_tree, slider_refs, plot, summary):
    """
    Reset all leaf node values and sliders to original values.

    Args:
        attack_tree (list[dict])
        original_tree (list[dict])
        slider_refs (dict): Map of label → slider.
        plot (ui.plotly)
        summary (ui.element)
    """
    for orig in original_tree:
        for node in attack_tree:
            if node["label"] == orig["label"] and is_leaf(node["label"], attack_tree):
                node["likelihood"] = orig["likelihood"]
                if node["label"] in slider_refs:
                    slider_refs[node["label"]].value = orig["likelihood"]

    compute_likelihood("Root Attack", attack_tree)
    update_chart(attack_tree, plot, summary)
```

assessment.py

```
"""
assessment.py

Generates summary statistics and visual elements for the risk assessment panel.

This module provides a function to update the summary UI based on current threat
likelihoods within an attack tree. It displays key metrics such as:
- Average likelihood across non-root nodes
- The highest-risk node
- Risk breakdown by parent category
- A warning panel for high-risk nodes (likelihood ≥ 75%)

Functions:
- update_assessment(attack_tree, summary): Updates the UI with the latest risk metrics.
"""

from collections import defaultdict
from nicegui import ui

def update_assessment(attack_tree, summary):
    """
    Update the UI assessment panel with risk summaries.

    Displays:
        - Average likelihood across all non-root nodes.
        - The most vulnerable node.
        - Risk breakdown per parent branch.
        - A highlighted list of high-risk nodes (likelihood ≥ 75%).
    """
    summary.clear()
    non_root = [n for n in attack_tree if n["parent"] != ""]
    values = [n["likelihood"] for n in attack_tree if "likelihood" in n]

    avg_risk = sum(values) / len(values) if values else 0
    likelihood_nodes = [n for n in non_root if "likelihood" in n]
    highest = max(likelihood_nodes, key=lambda n: n["likelihood"], default=None)
    high_risk_nodes = [n for n in non_root if n.get("likelihood", 0) >= 75]

    with summary:
        with ui.row().classes("w-full justify-between gap-8"):
            with ui.column().classes("flex-1"):
                ui.label("Threat Summary").classes("text-xl font-semibold mb-2")
                ui.label(f"Average Risk Per Node: {avg_risk:.1f}%")
                if highest:
                    ui.label(f"Most Vulnerable Node: {highest['label']}
({highest['likelihood']}%)")

            with ui.column().classes("flex-1"):
                ui.label("Risk Breakdown by Branch").classes("text-xl font-semibold mb-2")
                grouped = defaultdict(list)
                for node in non_root:
                    if "likelihood" in node:
                        grouped[node["parent"]].append(node["likelihood"])
                for parent, risks in grouped.items():
                    avg = sum(risks) / len(risks)
                    ui.label(f"• {parent}: {avg:.1f}%")

        if high_risk_nodes:
            with ui.row().classes(
                "w-full bg-red-100 border border-red-400 rounded-md p-4 mt-4"
            ):
                with ui.column().classes("w-full"):
                    ui.label(
                        "High Risk Nodes (≥ 75%)"
                    ).classes("text-lg font-semibold text-red-700 mb-2")
                    for node in high_risk_nodes:
                        ui.label(
                            f"• {node['label']} — {node['likelihood']}%"
                        ).classes("text-red-800")
```

Appendix C – JSON Files

pre_digital.json

```json
[
    {
        "label": "Compromise Business Operations",
        "parent": "",
        "gate": "OR"
    },
    {
        "label": "Physical Threats",
        "parent": "Compromise Business Operations",
        "gate": "OR"
    },
    {
        "label": "Break-in and theft",
        "parent": "Physical Threats",
        "likelihood": 30
    },
    {
        "label": "Fire or flood damage",
        "parent": "Physical Threats",
        "likelihood": 10
    },
    {
        "label": "Vandalism",
        "parent": "Physical Threats",
        "likelihood": 20
    },
    {
        "label": "Insider Threats",
        "parent": "Compromise Business Operations",
        "gate": "OR"
    },
    {
        "label": "Employee theft",
        "parent": "Insider Threats",
        "likelihood": 40
    },
    {
        "label": "Malicious staff actions",
        "parent": "Insider Threats",
        "likelihood": 20
    },
    {
        "label": "Process Failures",
        "parent": "Compromise Business Operations",
        "gate": "OR"
    },
    {
        "label": "Loss of paper records",
        "parent": "Process Failures",
        "likelihood": 35
    },
    {
        "label": "Reliance on individual staff knowledge",
        "parent": "Process Failures",
        "likelihood": 50
    },
    {
        "label": "Supply Chain Disruptions",
        "parent": "Compromise Business Operations",
        "gate": "OR"
    },
    {
        "label": "Local supplier delays",
        "parent": "Supply Chain Disruptions",
        "likelihood": 45
    },
    {
        "label": "Product shortages",
        "parent": "Supply Chain Disruptions",
        "likelihood": 40
    },
    {
        "label": "Customer Service Issues",
        "parent": "Compromise Business Operations",
        "gate": "OR"
    },
    {
        "label": "Manual stock errors",
        "parent": "Customer Service Issues",
        "likelihood": 30
    },
    {
        "label": "Missed special orders",
        "parent": "Customer Service Issues",
        "likelihood": 25
    }
]
```

post_digital.json

```
[
    {
        "label": "Compromise Digital System",
        "parent": "",
        "gate": "OR"
    },
    {
        "label": "Identity-Based Threats",
        "parent": "Compromise Digital System",
        "gate": "OR"
    },
    {
        "label": "Spoofing",
        "parent": "Identity-Based Threats",
        "likelihood": 60
    },
    {
        "label": "Social Engineering Attacks",
        "parent": "Spoofing",
        "likelihood": 65
    },
    {
        "label": "Data-Focused Threats",
        "parent": "Compromise Digital System",
        "gate": "OR"
    },
    {
        "label": "Tampering",
        "parent": "Data-Focused Threats",
        "likelihood": 55
    },
    {
        "label": "Data Manipulation Attacks",
        "parent": "Tampering",
        "likelihood": 50
    },
    {
        "label": "Repudiation",
        "parent": "Data-Focused Threats",
        "likelihood": 30
    },
    {
        "label": "Log Tampering",
        "parent": "Repudiation",
        "likelihood": 25
    },
    {
        "label": "Information Disclosure",
        "parent": "Data-Focused Threats",
        "likelihood": 45
    },
    {
        "label": "Unauthorised Data Access",
        "parent": "Information Disclosure",
        "likelihood": 50
    },
    {
        "label": "System-Based Threats",
        "parent": "Compromise Digital System",
        "gate": "OR"
    },
    {
        "label": "Denial of Service",
        "parent": "System-Based Threats",
        "likelihood": 35
    },
    {
        "label": "Service Disruption",
        "parent": "Denial of Service",
        "likelihood": 40
    },
    {
        "label": "Elevation of Privilege",
        "parent": "System-Based Threats",
        "likelihood": 30
    },
    {
        "label": "Privilege Escalation Attacks",
        "parent": "Elevation of Privilege",
        "likelihood": 25
    }
]
```