Summative Assessment 2: Executive Summary and App Presentation

Information Security Management June 2024

Word count: 1261

GitHub repository:

https://github.com/busilas/ISM_UoE/tree/main/Unit06/pampered_pets_app

Executive Summary and App Presentation

Executive Summary

Introduction

Pampered Pets, a well-established bricks-and-mortar pet store in the suburb of Hashington-on-the-Water, is considering digital transformation to enhance internal processes and expand its business reach. This report evaluates the potential threats associated with both the current and proposed digitalized systems using the STRIDE threat model, provides risk assessment methodologies, and outlines mitigation strategies to ensure secure operations.

Objectives

1. The primary objectives of this report are to:

2. Investigate potential threats to Pampered Pets' existing and proposed digital systems.
3. Assess the potential for business growth through an online presence.
4. Evaluate cost reduction through an international supply chain.
5. Determine the risk of customer loss if online features are not provided.

Business Challenges

1. **Growth Potential with Online Presence:**
    - An online platform could expand customer reach beyond the local community, potentially increasing sales by up to 50%.
    - Digital marketing and e-commerce integration can attract new customers and retain existing ones by offering convenience and accessibility.
2. **Cost Reduction with International Supply Chains:**
    - Transitioning to an international supply chain could reduce costs by up to 24%, though it comes with risks such as supply chain disruptions and quality control issues.
3. **Customer Retention:**
    - Without online features, the business risks losing up to 33% of its customers to competitors who offer more convenience through digital channels.

Threat Analysis and Risk Assessment

To assess the risks associated with digitalization, we employed the NIST SP 800-30 risk assessment methodology and the STRIDE threat model. Here’s a detailed analysis of the threats and vulnerabilities:

Table 1. STRIDE Threat Model (Current System).

Threat	**Description**
Unauthorized access to wireless network	Intruders could exploit weak wireless security to gain network access.
Fake emails from customers	Phishing attacks via fake emails can lead to data breaches or fraud.
Manipulation of digital sales records	Tampering with sales records can result in financial discrepancies.
Alteration of warehouse inventory data	Unauthorized changes in inventory data can cause stock management issues.
Denial of transaction by customers	Customers could falsely claim they didn't make a transaction.
Data breaches exposing customer information	Personal data theft can damage customer trust and lead to legal repercussions.
Leakage of proprietary recipes	Exposure of proprietary recipes can compromise competitive advantage.
Wireless network outages	Network downtime can disrupt sales and warehouse operations.
Unauthorized escalation of access privileges	Employees might gain unauthorized access to sensitive information.

Table 2. STRIDE Threat Model (Post-Digitalization).

Threat	**Description**
Unauthorized access to e-commerce platform	Hackers could exploit vulnerabilities to access customer data.
Alteration of online orders and inventory data	Unauthorized changes can disrupt order fulfilment and inventory management.
Customers denying online transactions	Fraudulent denial of transactions can lead to financial losses.
Data breaches exposing customer and business data	Sensitive data theft can damage reputation and incur compliance penalties.
DDoS attacks on the e-commerce website	Distributed Denial of Service attacks can disrupt online services.
Unauthorized access to ERP and CRM systems	Compromised systems can lead to data manipulation and loss.

Risk Assessment Methodology

Risk assessment involves identifying, evaluating, and prioritizing potential threats. This report uses the STRIDE model to categorize threats and assess their likelihood and impact:

- **Likelihood:** The probability of a threat occurring, rated from 1 (low) to 5 (high). It is potential damage or loss caused by the threat.
- **Impact:** The severity of the consequences if the threat is realized, rated from 1 (low) to 5 (high). It is the probability of threat occurring.
- **Risk Rating:** Calculated by multiplying likelihood and impact (Risk Rating = Impact × Likelihood), resulting in a value between 1 and 25.
- The risk factor for each threat is calculated using the formula:
- Impact and likelihood values are assigned based on qualitative or quantitative assessments, and the product gives a numerical risk factor, which helps prioritize mitigation efforts.

Table 3. Risk rating table.

Risk rating (1-25)	**Risk level**	**Explanation**
-5	Low	Low likelihood and low impact; threats that are unlikely to significantly affect operations.
6-10	Medium	Moderate likelihood or impact; threats that could have noticeable consequences if realized.
11-15	Medium-High	Medium to high likelihood or impact; threats that pose a significant risk if not properly managed.
16-20	High	High likelihood and high impact; threats that could severely impact operations and reputation.
21-25	Critical	Very high likelihood and impact; threats that could have catastrophic consequences if exploited.

Threats and Risk Assessment

The following table summarizes the threats and their risk assessments for Pampered Pets:

Table 4. Threats summary.

Threat Description	Likelihood (1-5)	Impact (1-5)	Risk Rating (1-25)	Risk Level	Explanation
Unauthorized access to wireless network	3 (Medium)	3 (Medium)	9	Medium	Medium likelihood due to reliance on wireless network; medium impact could lead to data breaches or disruptions.
Fake Emails from customers placing orders	1 (Low)	3 (Medium)	3	Low	Low likelihood of occurrence if email security measures are robust; medium impact on operations if successful.
Manipulation of digital sales records	1 (Low)	4 (High)	4	Low	Low likelihood if access controls are strong; high impact on financial records and compliance.
Alteration of warehouse inventory data	2 (Medium)	3 (Medium)	6	Medium	Medium likelihood due to manual data entry; medium impact on inventory management and customer service.
Denial of transaction by customers	1 (Low)	3 (Medium)	3	Low	Occasional likelihood; medium impact on customer satisfaction and revenue.
Employees denying actions performed on the system	1 (Low)	2 (Low)	2	Low	Low likelihood but potential reputational impact if miscommunication occurs.
Data breaches exposing customer information	4 (High)	4 (High)	16	High	High likelihood due to handling sensitive data; high impact on customer trust and regulatory compliance.
Leakage of proprietary recipes and supplier information	2 (Medium)	4 (High)	8	Medium	Potential for competitive disadvantage; high impact on operational secrecy and supplier relations.
Wireless network outages impacting	3 (Medium)	4 (High)	12	Medium-High	Medium likelihood but high impact on sales,

sales and operations					customer service, and operational efficiency.
Unauthorized escalation of access privileges by employees	2 (Medium)	3 (Medium)	6	Medium	Low to medium likelihood; potential for insider threat; medium impact on data security and access controls.

Potential benefits of digital transformation

1. **Business Growth through Online Presence:** An online presence could grow Pampered Pets' business by up to 50% by reaching a wider audience and providing convenient online shopping options.
2. **Cost Reduction through International Supply Chain:** Transitioning to an international supply chain could reduce costs by up to 24%, providing access to a broader range of suppliers and competitive pricing.
3. **Risk of Customer Loss without Online Features:** Without offering online features, the business risks losing up to 33% of its existing customers who may prefer the convenience of online shopping.

Recommendations

1. **Implement Strong Security Measures:**
   - Use advanced encryption and multi-factor authentication for all online systems.
   - Regularly update and patch software to protect against known vulnerabilities.
2. **Enhance Employee Training:**
   - Conduct regular security awareness training to educate employees on identifying and preventing cyber threats.

3. **Develop Robust Incident Response Plan:**
    - Create a comprehensive incident response plan to quickly address and mitigate security incidents.
4. **Adopt Cloud Solutions:**
    - Utilize cloud-based services for scalability, reliability, and enhanced security.
5. **Regular Audits and Monitoring:**
    - Perform regular security audits and continuous monitoring to detect and respond to threats promptly.

Conclusion

Digital transformation offers significant growth opportunities for Pampered Pets but also introduces new threats that must be managed effectively. By implementing robust security measures, continuously monitoring risks, and prioritizing mitigation efforts, Pampered Pets can successfully transition to a digital business model while safeguarding its operations and customer trust.

Python Application for Threat Visualization and Impact Assessment

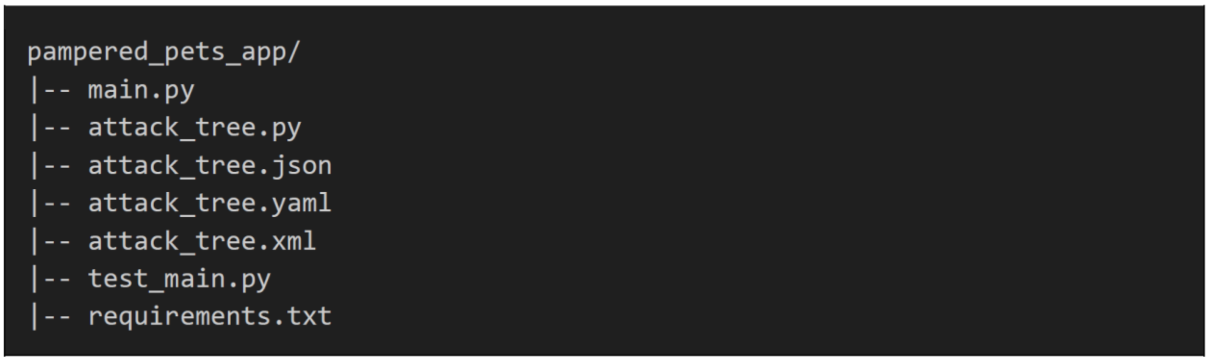
To aid Cathy in visualizing and assessing the impact of threats, we have developed a Python application that creates graphical representations of attack trees. This application allows users to add values to nodes, which are then aggregated to form an overall threat assessment rating.

Application Features

- Attack Tree Visualization: Generate graphical representations of attack trees from XML, YAML, or JSON specifications.
- Value Addition: Users can input monetary amounts or probabilities to leaf nodes.
- Impact Aggregation: The application aggregates values to provide an overall threat impact assessment.
- Risk Factor Evaluation: Evaluate risk factors based on JSON data input.
- Export Functionality: Export the visual representation of the attack tree as PNG or JPG files.

Application Structure

Main.py - the main script to run the application, attack_tree.py contains the logic for creating and managing attack trees and unit tests ensure the reliability and accuracy of the application.



```
pampered_pets_app/
|-- main.py
|-- attack_tree.py
|-- attack_tree.json
|-- attack_tree.yaml
|-- attack_tree.xml
|-- test_main.py
|-- requirements.txt
```

Figure 1. Application structure.

Justification for Choosing NetworkX, Matplotlib, and PyYAML

NetworkX, Matplotlib and PyYAML libraries were chosen for their powerful features, ease of use, and strong support for their respective functionalities, making them suitable for implementing and visualizing attack trees in a Python application.

Table 5. Justification of libraries.

Library	**Justification**	**Advantages**
NetworkX	Ideal for creating and manipulating graph-based data structures NetworkX, 2024; Zisko & Hyra, 2023).	Ease of use, graph visualization, extensive documentation, flexibility in graph types (Educative, 2023; Gao et al., 2023;).
Matplotlib	Comprehensive library for creating visualizations (Karl, 2024).	Versatility, integration with NetworkX, customizability, maturity and robustness (GeeksforGeeks, 2024; (Gupta, N. D.).
PyYAML	YAML parser and emitter suitable for configuration and data serialization (Ramuglia , 2024).	Readability of YAML files, ease of use, flexibility, compatibility with other Python libraries (PythonLand, 2023; Dilhara et al., 2021).

Explanation of risk calculation methodology

The risk calculation in the attack tree is performed by aggregating the values assigned to each node. Here's how it works:

1. Initial Values: Each node in the tree can have an initial value, representing the risk associated with that specific threat. The default value is set to 0.

2. Adding Values: Users can provide a dictionary of values, where each key is the node name and the value is the associated risk value. This allows updating the values dynamically based on new information or assessments.
3. Total Value Calculation: The total risk value of the tree is calculated by summing up the values of all nodes, starting from the root node and recursively adding the values of child nodes. This provides an overall risk assessment for the entire system.

Ranges for Risk Rating

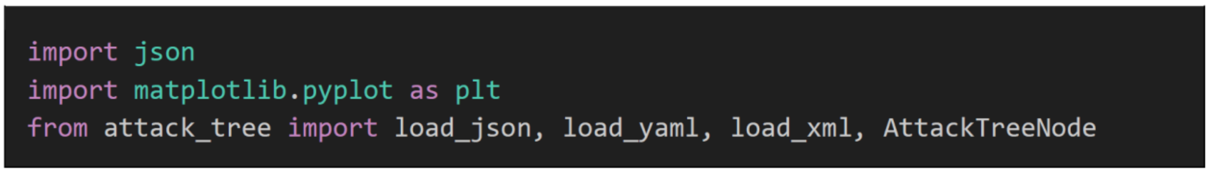
To categorize the risk levels, you can define specific ranges for the total risk value:

0-20	21-40	41-60	61-100
Low Risk	Moderate Risk	High Risk	Critical Risk

These ranges can be adjusted based on the specific requirements and risk tolerance of the organization.

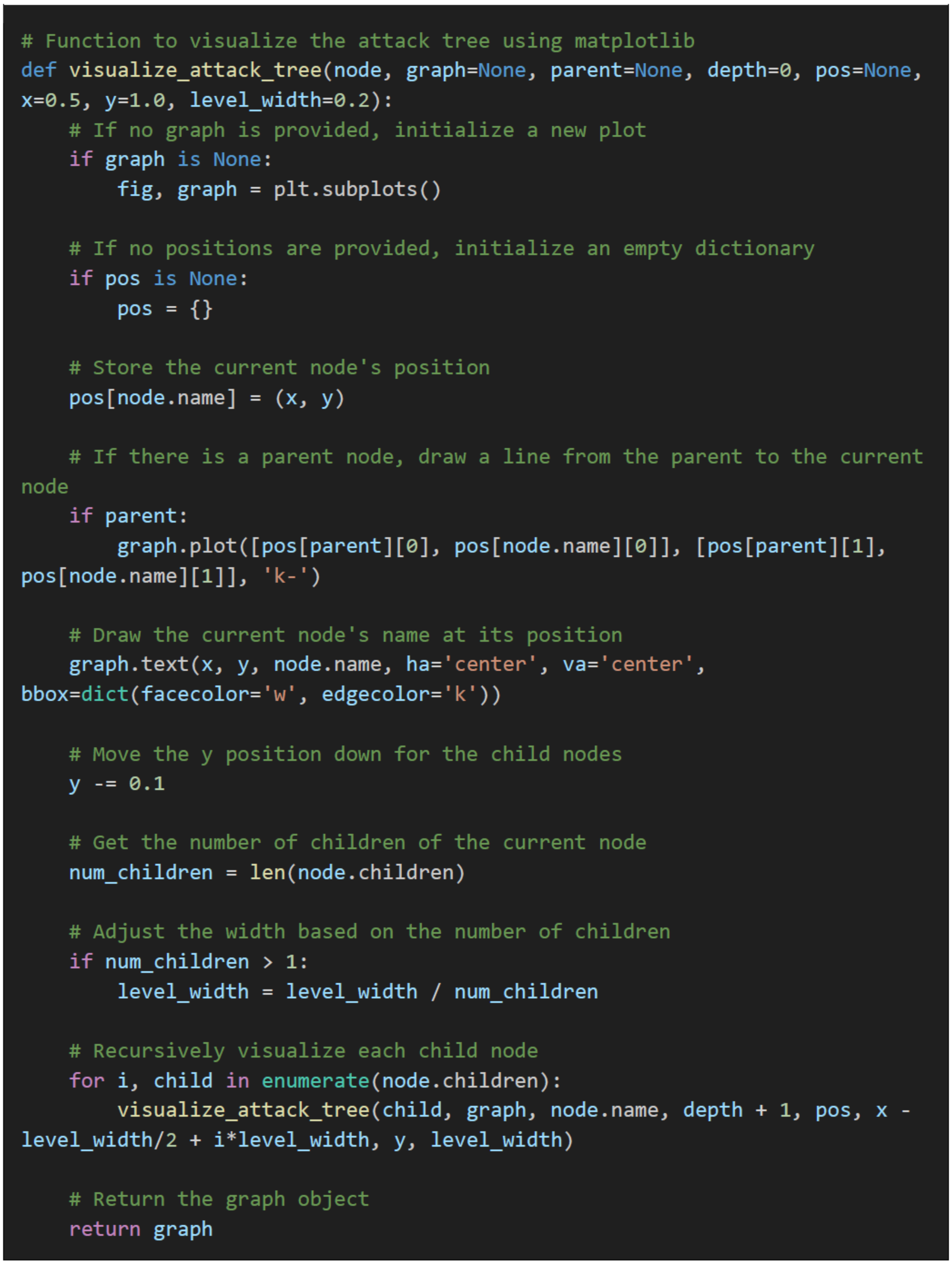
Explanation main.py of application code

Bellow provided the braked down code step by step to introduce its functionality of apps. This script is designed to load, visualize, and interact with attack trees. It supports loading attack tree data from JSON, YAML, or XML files, visualizing the tree using matplotlib, adding values to the nodes, and calculating the total threat value. The main application function provides a menu-driven interface for user interaction.



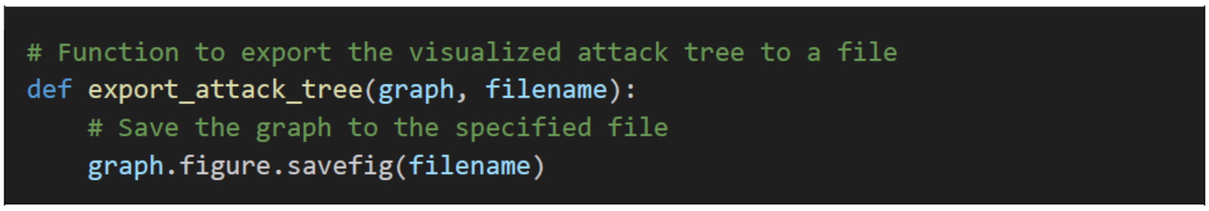
```python
import json
import matplotlib.pyplot as plt
from attack_tree import load_json, load_yaml, load_xml, AttackTreeNode
```

Figure 2. Import statements.



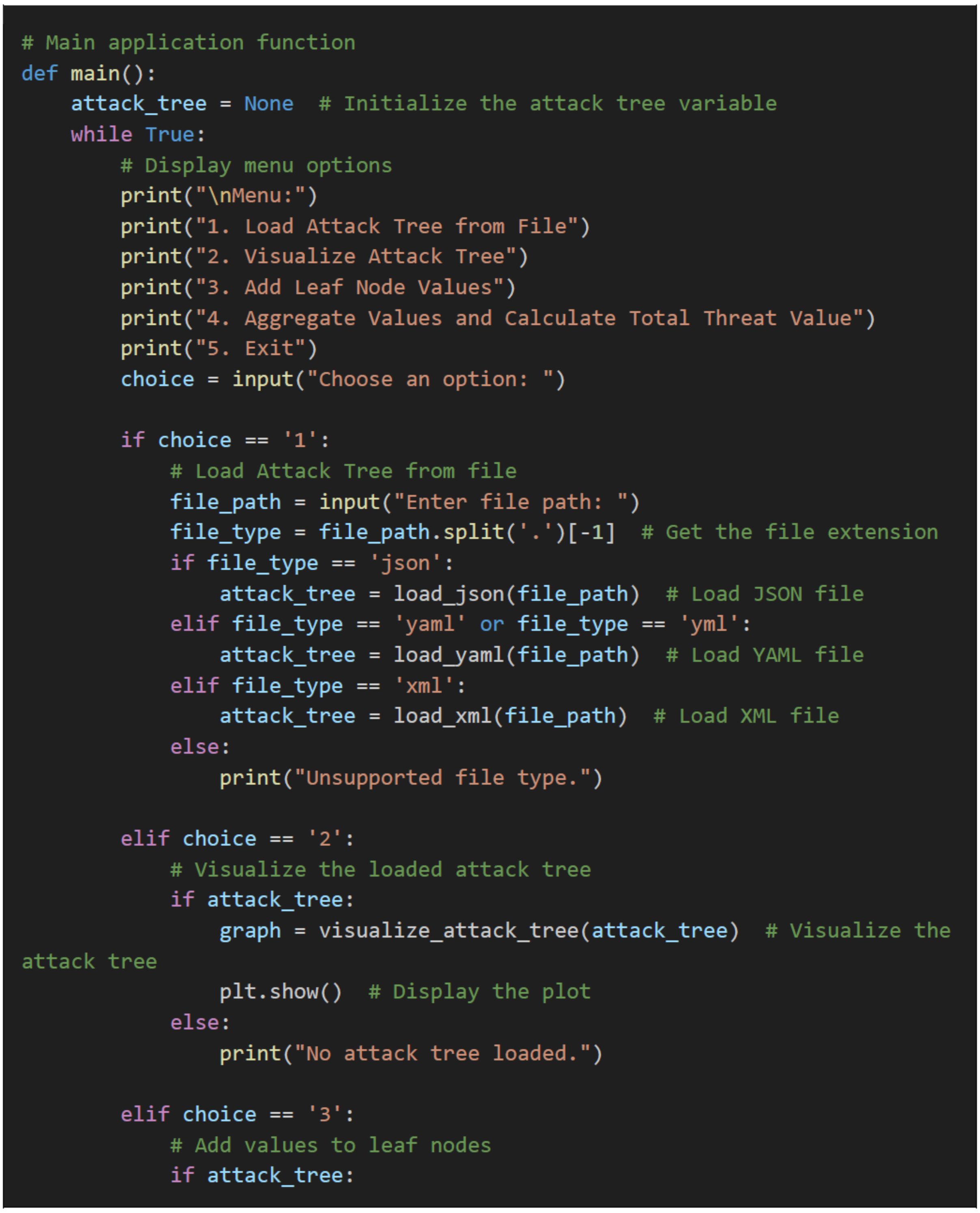
```python
# Function to visualize the attack tree using matplotlib
def visualize_attack_tree(node, graph=None, parent=None, depth=0, pos=None,
x=0.5, y=1.0, level_width=0.2):
    # If no graph is provided, initialize a new plot
    if graph is None:
        fig, graph = plt.subplots()

    # If no positions are provided, initialize an empty dictionary
    if pos is None:
        pos = {}

    # Store the current node's position
    pos[node.name] = (x, y)

    # If there is a parent node, draw a line from the parent to the current
node
    if parent:
        graph.plot([pos[parent][0], pos[node.name][0]], [pos[parent][1],
pos[node.name][1]], 'k-')

    # Draw the current node's name at its position
    graph.text(x, y, node.name, ha='center', va='center',
bbox=dict(facecolor='w', edgecolor='k'))

    # Move the y position down for the child nodes
    y -= 0.1

    # Get the number of children of the current node
    num_children = len(node.children)

    # Adjust the width based on the number of children
    if num_children > 1:
        level_width = level_width / num_children

    # Recursively visualize each child node
    for i, child in enumerate(node.children):
        visualize_attack_tree(child, graph, node.name, depth + 1, pos, x -
level_width/2 + i*level_width, y, level_width)

    # Return the graph object
    return graph
```

Figure 3. Function to visualize the attack tree.

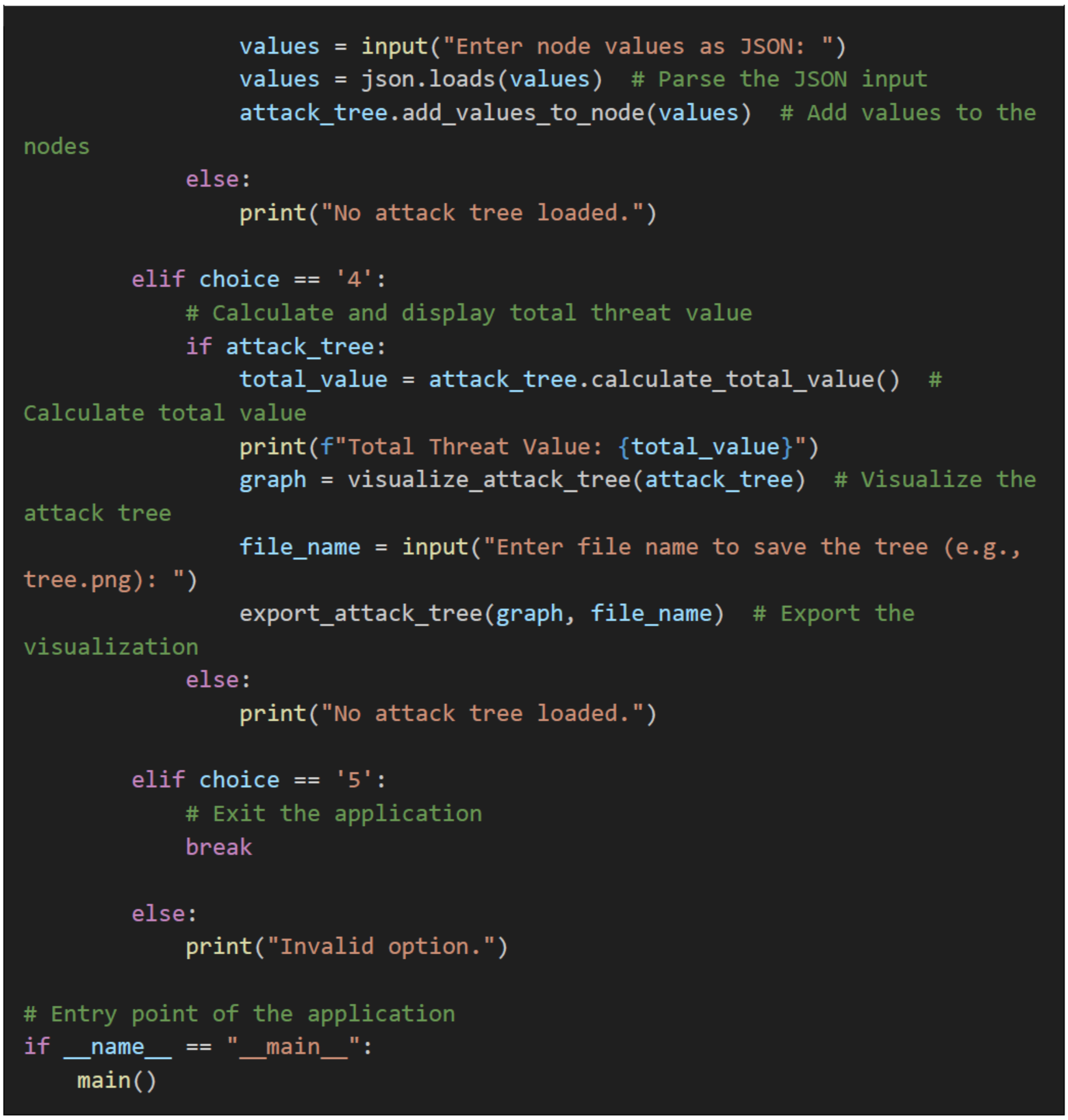


```
# Function to export the visualized attack tree to a file
def export_attack_tree(graph, filename):
    # Save the graph to the specified file
    graph.figure.savefig(filename)
```

Figure 4. Function to export the visualized attack tree.



```
# Main application function
def main():
    attack_tree = None  # Initialize the attack tree variable
    while True:
        # Display menu options
        print("\nMenu:")
        print("1. Load Attack Tree from File")
        print("2. Visualize Attack Tree")
        print("3. Add Leaf Node Values")
        print("4. Aggregate Values and Calculate Total Threat Value")
        print("5. Exit")
        choice = input("Choose an option: ")

        if choice == '1':
            # Load Attack Tree from file
            file_path = input("Enter file path: ")
            file_type = file_path.split('.')[-1]  # Get the file extension
            if file_type == 'json':
                attack_tree = load_json(file_path)  # Load JSON file
            elif file_type == 'yaml' or file_type == 'yml':
                attack_tree = load_yaml(file_path)  # Load YAML file
            elif file_type == 'xml':
                attack_tree = load_xml(file_path)  # Load XML file
            else:
                print("Unsupported file type.")

        elif choice == '2':
            # Visualize the loaded attack tree
            if attack_tree:
                graph = visualize_attack_tree(attack_tree)  # Visualize the
attack tree
                plt.show()  # Display the plot
            else:
                print("No attack tree loaded.")

        elif choice == '3':
            # Add values to leaf nodes
            if attack_tree:
```

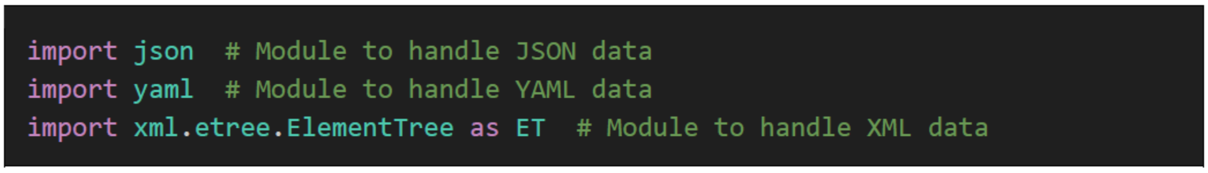


```
                values = input("Enter node values as JSON: ")
                values = json.loads(values)  # Parse the JSON input
                attack_tree.add_values_to_node(values)  # Add values to the
nodes
            else:
                print("No attack tree loaded.")

        elif choice == '4':
            # Calculate and display total threat value
            if attack_tree:
                total_value = attack_tree.calculate_total_value()  #
Calculate total value
                print(f"Total Threat Value: {total_value}")
                graph = visualize_attack_tree(attack_tree)  # Visualize the
attack tree
                file_name = input("Enter file name to save the tree (e.g.,
tree.png): ")
                export_attack_tree(graph, file_name)  # Export the
visualization
            else:
                print("No attack tree loaded.")

        elif choice == '5':
            # Exit the application
            break

        else:
            print("Invalid option.")

# Entry point of the application
if __name__ == "__main__":
    main()
```

Figure 5. Main application function.

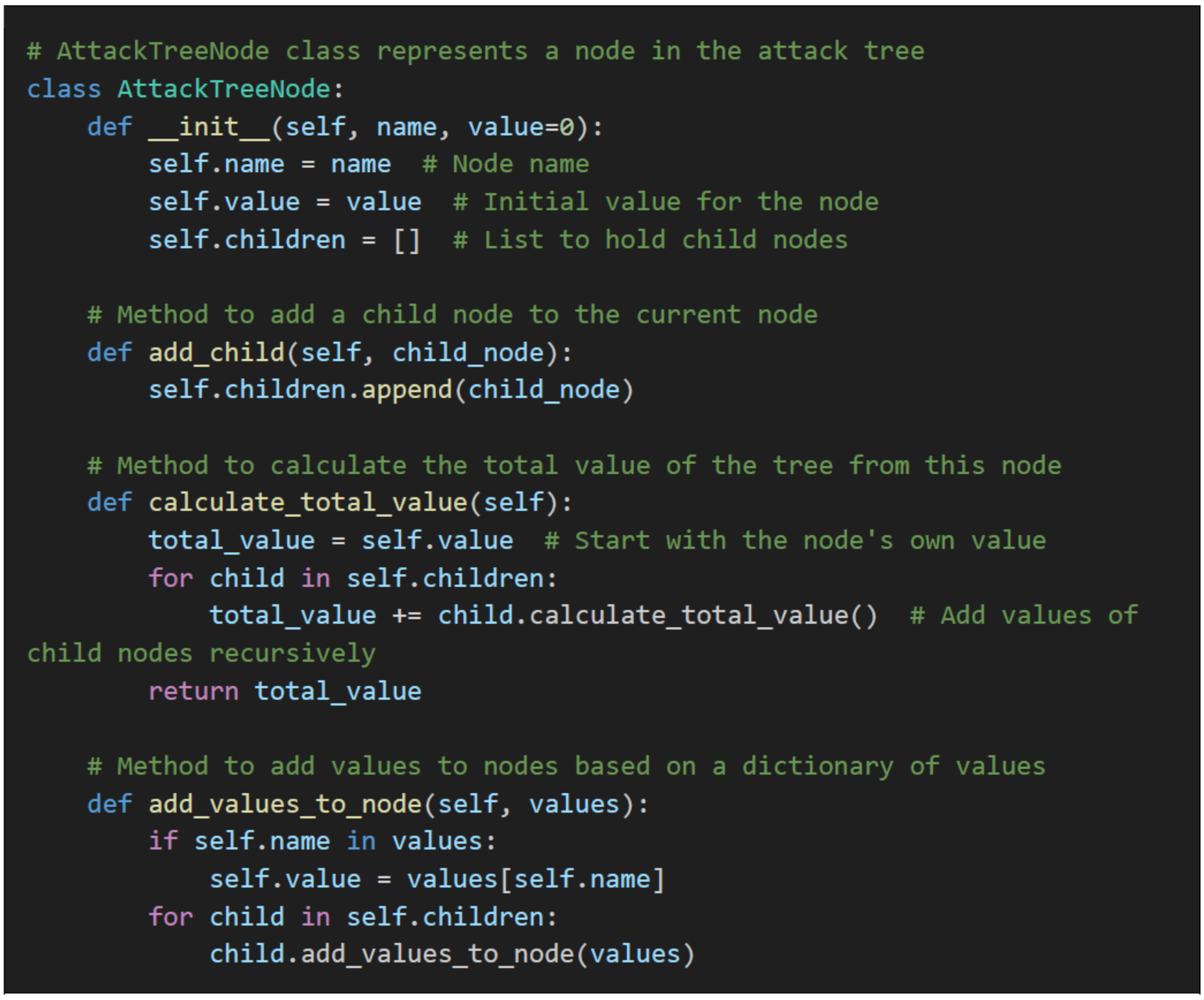
Explanation attack_tree.py of application code

The code of attach_tree.py defines a class to represent nodes in an attack tree and includes functions to build and load attack trees from various data formats (JSON, YAML, XML). The AttackTreeNode class provides methods for managing node values and calculating the total value of the tree. The helper functions build_tree and build_tree_xml construct trees from dictionary and XML data, respectively, while the load_* functions handle file loading and parsing.



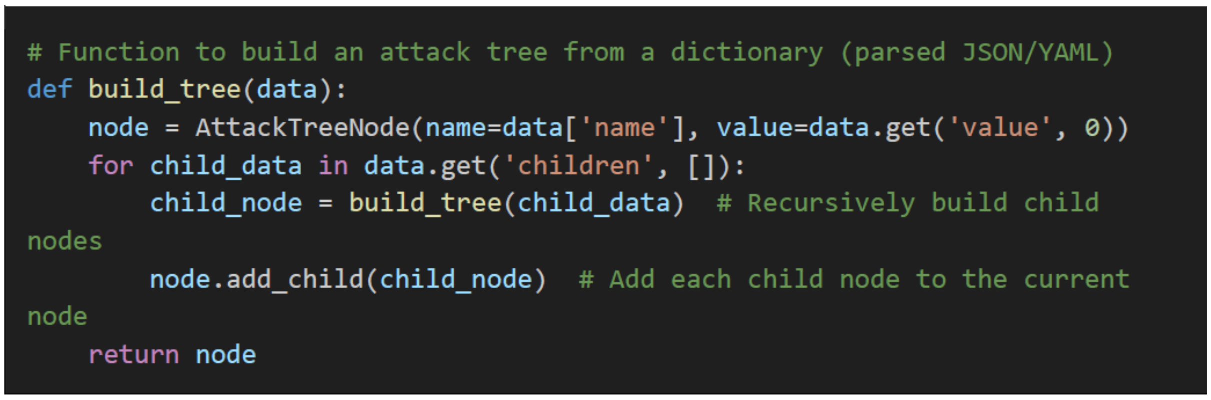
```
import json  # Module to handle JSON data
import yaml  # Module to handle YAML data
import xml.etree.ElementTree as ET  # Module to handle XML data
```

Figure 6. Import Statements.



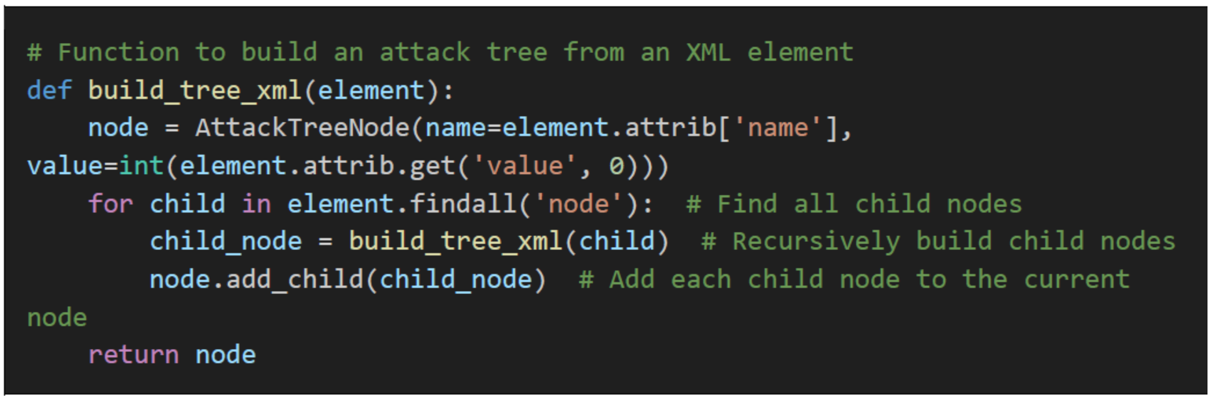
```
# AttackTreeNode class represents a node in the attack tree
class AttackTreeNode:
    def __init__(self, name, value=0):
        self.name = name  # Node name
        self.value = value  # Initial value for the node
        self.children = []  # List to hold child nodes

    # Method to add a child node to the current node
    def add_child(self, child_node):
        self.children.append(child_node)

    # Method to calculate the total value of the tree from this node
    def calculate_total_value(self):
        total_value = self.value  # Start with the node's own value
        for child in self.children:
            total_value += child.calculate_total_value()  # Add values of
child nodes recursively
        return total_value

    # Method to add values to nodes based on a dictionary of values
    def add_values_to_node(self, values):
        if self.name in values:
            self.value = values[self.name]
        for child in self.children:
            child.add_values_to_node(values)
```

Figure 7. AttackTreeNode class.



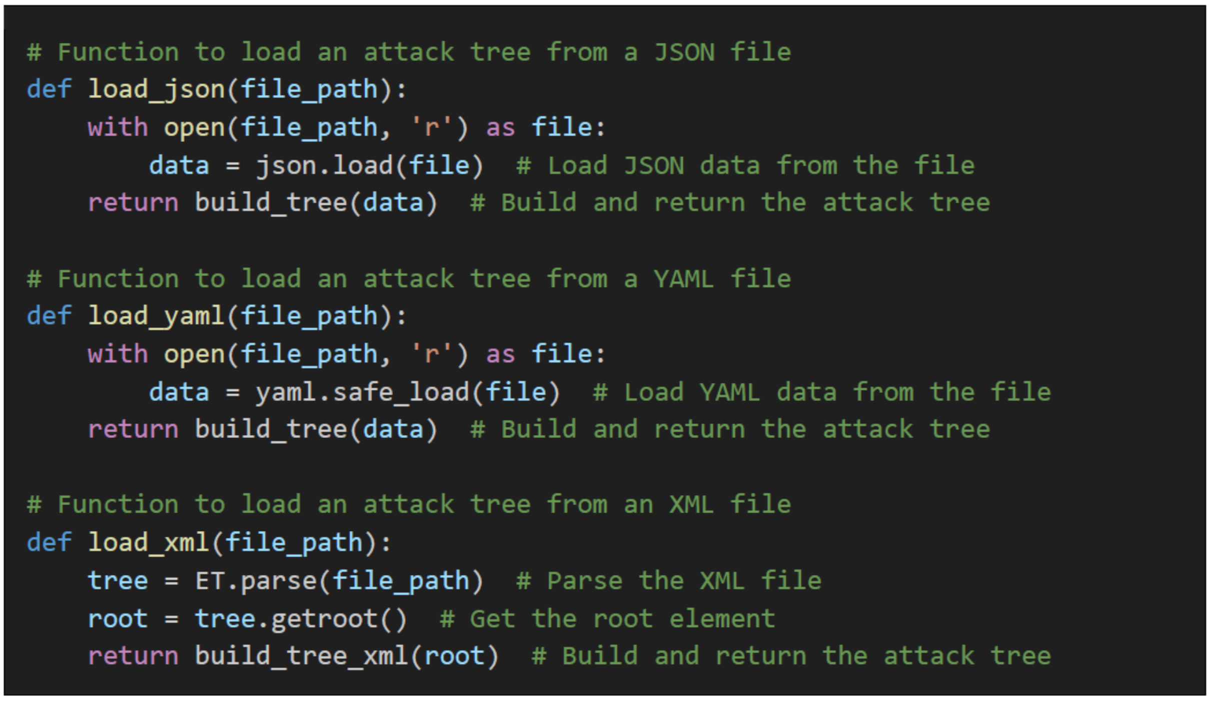
```python
# Function to build an attack tree from a dictionary (parsed JSON/YAML)
def build_tree(data):
    node = AttackTreeNode(name=data['name'], value=data.get('value', 0))
    for child_data in data.get('children', []):
        child_node = build_tree(child_data)  # Recursively build child
nodes
        node.add_child(child_node)  # Add each child node to the current
node
    return node
```

Figure 8. Function to build an attack tree from a dictionary.



```python
# Function to build an attack tree from an XML element
def build_tree_xml(element):
    node = AttackTreeNode(name=element.attrib['name'],
value=int(element.attrib.get('value', 0)))
    for child in element.findall('node'):  # Find all child nodes
        child_node = build_tree_xml(child)  # Recursively build child nodes
        node.add_child(child_node)  # Add each child node to the current
node
    return node
```

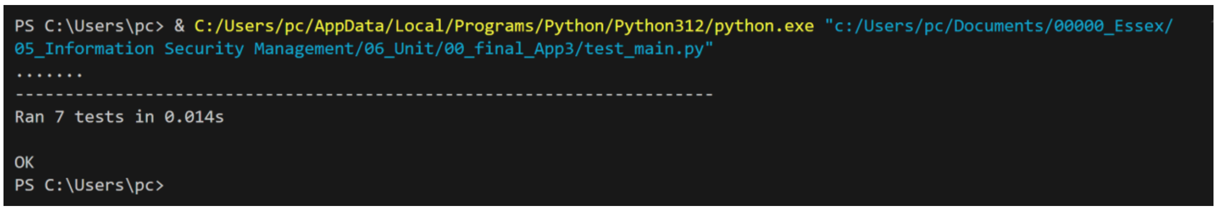
Figure 9. Function to build an attack tree from an XML element.



```python
# Function to load an attack tree from a JSON file
def load_json(file_path):
    with open(file_path, 'r') as file:
        data = json.load(file)  # Load JSON data from the file
    return build_tree(data)  # Build and return the attack tree

# Function to load an attack tree from a YAML file
def load_yaml(file_path):
    with open(file_path, 'r') as file:
        data = yaml.safe_load(file)  # Load YAML data from the file
    return build_tree(data)  # Build and return the attack tree

# Function to load an attack tree from an XML file
def load_xml(file_path):
    tree = ET.parse(file_path)  # Parse the XML file
    root = tree.getroot()  # Get the root element
    return build_tree_xml(root)  # Build and return the attack tree
```

Figure 10. Functions to load attack trees from files.

Unit Test result



```
PS C:\Users\pc> & C:/Users/pc/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/pc/Documents/00000_Essex/
05_Information Security Management/06_Unit/00_final_App3/test_main.py"
.......
----------------------------------------------------------------------
Ran 7 tests in 0.014s

OK
PS C:\Users\pc>
```

Figure 11. Unit Test results.

Reference list

Caralli, R. A., Stevens, J. F., Young, L. R., & Wilson, W. R. (2007). Introducing OCTAVE Allegro: Improving the Information Security Risk Assessment Process. Software Engineering Institute, Carnegie Mellon University.

Dilhara, M., Ketkar, A. and Dig, D. (2021) ‘Understanding software-2.0’, ACM Transactions on Software Engineering and Methodology, 30(4), pp. 1–42. doi:10.1145/3453478.

Educative (2023) NetworkX library in Python, Educative. Available at: https://www.educative.io/answers/networkx-library-in-python [Accessed: 06 July 2024].

Gai, K., Qiu, M., & Sun, X. (2018). A survey on FinTech. Journal of Network and Computer Applications, 103, 262-273.

Gallagher, P. (2009). Recommended Security Controls for Federal Information Systems and Organizations. NIST.

Gao, M., Li, Z., Li, R., Cui, Ch., Chen, X., Ye, B., Li, Y., Gu, W., Gong, Q., Wang, X. & Chen, Y. (2023) ‘EasyGraph: A multifunctional, cross-platform, and effective library for interdisciplinary network analysis’, Patterns, 4(10), p. 100839. doi:10.1016/j.patter.2023.100839.

GeeksforGeeks (2024) Introduction to matplotlib, GeeksforGeeks. Available at: https://www.geeksforgeeks.org/python-introduction-matplotlib/ [Accessed: 05 July 2024].

Gupta, S. (N. D.) Matplotlib library in python, enjoyalgorithms. Available at: https://www.enjoyalgorithms.com/blog/introduction-to-matplotlib [Accessed: 08 July 2024].

Karl, T. (2024) How to choose between Seaborn vs. matplotlib, New Horizons. Available at: https://www.newhorizons.com/resources/blog/how-to-choose-between-seaborn-vs-matplotlib [Accessed: 06 July 2024].

National Institute of Standards and Technology (NIST). (2012). NIST SP 800-30: Guide for Conducting Risk Assessments.

NetworkX (2024) NetworkX documentation, NetworkX. Available at: https://networkx.org/ [Accessed: 07 July 2024].

PythonLand (2023) Python YAML: How to load, read, and write YAML • python land tutorial, Python Land. Available at: https://python.land/data-processing/python-yaml [Accessed: 04 July 2024].

Ramuglia , G. (2024) Python YAML parser guide: Pyyaml, ruamel.yaml and more, Linux Dedicated Server Blog. Available at: https://ioflood.com/blog/python-yaml-parser/#:~:text=One%20of%20the%20main%20advantages,YAML%20files%20you’ll%20encounter. [Accessed: 05 July 2024].

Stoneburner, G., Goguen, A., & Feringa, A. (2002). Risk Management Guide for Information Technology Systems. NIST.

The OCTAVE Method. CERT Division, Software Engineering Institute, Carnegie Mellon University.

Zisko, F. & Hyra, A. (2023) ‘Security testing with python scripts’, Smart Cities and Regional Development (SCRD) Journal, 7(2), pp. 77–84. doi:10.25019/m4c99y31.