**Python Journal Template**

**Directions:** Follow the directions for each part of the journal template. Include in your response all the elements listed under the Requirements section. Prompts in the Inspiration section are not required; however, they may help you to fully think through your response.

Remember to review the Touchstone page for entry requirements, examples, and grading specifics.

**Name: Michael Bean**

**Date: 10/29/2023**

**Final Replit Program Share Link:**

Complete the following template. Fill out all entries using complete sentences.

## PART 1: Defining Your Problem

|  |
| --- |
| **Task**  State the problem you are planning to solve.  **Requirements**   * Describe the problem you are trying to solve for. * Describe any input data you expect to use. * Describe what the program will do to solve the problem. * Describe any outputs or results the program will provide.   **Inspiration**  When writing your entry below ask yourself the following questions:   * Why do you want to solve this particular problem? * What source(s) of data do you believe you will need? Will the user need to supply that data, or will you get it from an external file or another source? * Will you need to interact with the user throughout the program? Will users continually need to enter data in and see something to continue? * What are your expected results or what will be the end product? What will you need to tell a user of your program when it is complete? |
| Intrusion Detection System  The Problem? Because cybersecurity threats are ever-evolving, it's critical to have systems in place that can recognize and react to suspect network activity. To create an Intrusion Detection System (IDS) using Python is the issue we're trying to address. An IDS is a security tool that notifies managers when it detects malicious activity or policy violations in a network or system.  The input. We use tools like Wireshark or tcpdump, which can be used to record network traffic data. It includes details about packets, including their source, destination, protocol, and other metadata.  Signatures of known threats a repository of well-known attack signatures and patterns, like those offered by Snort or Suricata.  System logs: These include authentication logs, firewall logs, and other logs produced by network hardware and servers.  Data collection: The program will gather system logs and information about network traffic from numerous sources.  Data Preprocessing: The data will be preprocessed, including network packet parsing, feature extraction, cleaning, and log formatting.  What is will solve? The computer will learn the typical network behavior and spot abnormalities or departures from this norm using machine learning techniques (such as clustering, classification, or deep learning).  Signature-based detection: It will look for known attacks by comparing network traffic and logs to known threat signatures.  The Output: The application will produce warnings with details about the danger detected, the impacted system, and the severity level, when suspicious actions or attacks are discovered.  Alerts: When it discovers suspicious activity or known attacks, the application will send out immediate alerts. These warnings will provide information on the type of attack, source and destination IP addresses, and timestamps.  Logs: For subsequent analysis and forensics, it will record all network and system actions, including regular and suspicious ones.  Reports: The application has the option to produce reports that list the dangers it has found and their effects. |

## PART 2: Working Through Specific Examples

|  |
| --- |
| **Task**  Write down clear and specific steps to solve a simple version of your problem you identified in Part 1.  **Requirements**  Complete the three steps below **for at least two distinct examples/scenarios**.   * State any necessary input data for your simplified problem. * Write clear and specific steps in English (not Python) detailing what the program will do to solve the problem. * Describe the specific result of your example/scenario.   **Inspiration**  When writing your entry below ask yourself the following questions:   * Are there any steps that you don’t fully understand? These are places to spend more time working out the details. Consider adding additional smaller steps in these spots. * Remember that a computer program is very literal. Are there any steps that are unclear? Try giving the steps of your example/scenario to a friend or family member to read through and ask you questions about parts they don’t understand. Rewrite these parts as clearly as you can. * Are there interesting edge cases for your program? Try to start one of your examples/scenarios with input that matches this edge case. How does it change how your program might work? |
| We will be working to solve the problem with a version of an Intrusion Detection System (IDS) with two distinct scenarios: anomaly detection and signature-based detection.  **Scenario 1: Anomaly Detection**  **Input Data**:  Network traffic data, or packets including source, destination, and protocol information, is the input data.  A predetermined cutoff point (such as the maximum number of packets per minute) for anomaly identification.  **Steps**:   1. Data Collection: Gather data on network traffic continuously from multiple sources. 2. Data Preprocessing: Interpret incoming packets from the network. Features such as protocol, source IP, and destination IP should be extracted. 3. Anomaly Detection: Determine how many packets arrive each minute. This rate should be compared to the specified threshold. 4. Alert Generation: Send out an alert if the rate of receiving packets surpasses the predetermined level. Add information to the alert about the anomaly, such as the originating IP address and time.   **Result:**  An alert signaling a possible anomaly in network traffic is generated by the software if it notices an unusual rate of incoming packets.  **Scenario 2: Signature-Based Detection**  **Input Data**:  Data about network traffic, comprising packets with source, destination, and protocol details.  Signatures of recognized threats (e.g., patterns linked to known attacks).  **Steps**:   1. Data Collection: Gather information about network traffic continuously from multiple sources. 2. Data Preprocessing: Interpret incoming packets from the network. Features such as protocol, source IP, and destination IP should be extracted. 3. Signature Matching: Examine each incoming packet for signatures associated with known threats. Examine the packet to see if any parts match a known signature. 4. Alert Generation: Send out an alert in the event that a packet's signature matches one of known threats. Provide information in the warning regarding the identified threat, such as the signature name, source IP address, and time.   **Result**:  The program issues an alert indicating the existence of a particular known attack if it finds a network packet that matches a known threat signature.  **Potential Edge Cases**:  An intriguing edge case for either scenario could be a deliberate increase in network traffic (e.g., a large-scale Distributed Denial of Service (DDoS) attack or an abrupt spike in legitimate user activity). In this scenario, the software ought to correctly discern between typical and unusual traffic and issue alerts as needed. |

## PART 3: Generalizing Into Pseudocode

|  |
| --- |
| **Task**  Write out the general sequence your program will use, including all specific examples/scenarios you provided in Part 2.  **Requirements**   * Write pseudocode for the program in English but refer to Python program elements where they are appropriate. The pseudocode should represent the full functionality of the program, not just a simplified version. Pseudocode is broken down enough that the details of the program are no longer in any paragraph form. One statement per line is ideal.   **Help with writing pseudocode**   * Here are a few links that can help you write pseudocode with examples. Remember to check out part 3 of the Example Journal Template Submission if you have not already. Note: everyone will write pseudocode differently. There is no right or wrong way to write it other than to make sure you write it clearly and in as much detail as you can so that it should be easy to convert it to code later.   + <https://www.geeksforgeeks.org/how-to-write-a-pseudo-code/>   + <https://www.wikihow.com/Write-Pseudocode>   **Inspiration**  When writing your entry below ask yourself the following questions:   * Do you see common program elements and patterns in your specific examples/scenarios in Part 2, like variables, conditionals, functions, loops, and classes? These should be part of your pseudocode for the general sequence as well. * Are there places where the steps for your examples/scenarios in Part 2 diverged? These may be places where errors may occur later in the project. Make note of them. * When you are finished with your pseudocode, does it make sense, even to a person that does not know Python? Aim for the clearest description of the steps, as this will make it easier to convert into program code later. |
| <Write your pseudocode here>  # Initialize variables and data structures  threshold\_anomaly = predefined\_threshold\_for\_anomaly  known\_threat\_signatures = load\_known\_threat\_signatures()  # Define functions for common tasks  function collect\_network\_traffic\_data():  # Continuously collect network traffic data  # Return network packets  function preprocess\_packet(packet):  # Parse incoming network packet  # Extract features like source IP, destination IP, and protocol  # Return parsed packet data  function calculate\_packet\_rate(packets, time\_interval):  # Calculate the rate of incoming packets per minute  # Return packet rate  function match\_signature(packet, known\_signatures):  # Compare packet to known threat signatures  # Check if any part of the packet matches a known signature  # Return True if a match is found, False otherwise  function generate\_alert(alert\_type, details):  # Generate an alert with specified type and details  # Output the alert  # Main program loop  while True:  # Collect network traffic data  packets = collect\_network\_traffic\_data()    # Process each packet  for packet in packets:  parsed\_packet = preprocess\_packet(packet)    # Scenario 1: Anomaly Detection  packet\_rate = calculate\_packet\_rate(packets, time\_interval)  if packet\_rate > threshold\_anomaly:  generate\_alert("Anomaly", "Abnormal packet rate detected.")    # Scenario 2: Signature-Based Detection  if match\_signature(parsed\_packet, known\_threat\_signatures):  generate\_alert("Signature", "Known threat signature detected.") |

## PART 4: Testing Your Program

|  |
| --- |
| **Task**  While writing and testing your program code, describe your tests, record any errors, and state your approach to fixing the errors.  **Requirements**   * For at least one of your test cases, describe how your choices for the test helped you understand whether the program was running correctly or not.   For each error that occurs while writing and testing your code:   * Record the details of the error from Replit. A screenshot or copy-and-paste of the text into the journal entry is acceptable. * Describe what you attempted in order to fix the error. Clearly identify what approach was the one that worked.   **Inspiration**  When writing your entry below ask yourself the following questions:   * Have you tested edge cases and special cases for the inputs of your program code? Often these unexpected values can cause errors in the operation of your program. * Have you tested opportunities for user error? If a user is asked to provide an input, what happens when they give the wrong type of input, like a letter instead of a number, or vice versa? * Did the outcome look the way you expected? Was it formatted correctly? * Does your output align with the solution to the problem you coded for? |
| <Record your errors and fixes here>  In this test case, I want to make sure that the program generates alerts and correctly identifies packets that contain known threat signatures. I will offer packets from network traffic that are known to match the threat signatures that have been loaded.  During my testing, I encountered an error during the test where it failed to generate alerts when packets containing known threat signatures were expected to be detected.    **Approach to Fixing the Error)**:   1. **Error Analysis**: Examine the Python code in charge of signature-based detection, paying particular attention to the logic used to compare packets with known threat signatures. 2. **Code Review**: Go through the Python code that matches signatures with great care. Make sure you use conditionals, loops, data structures, and Python syntax correctly. 3. **Logging and Debugging**: To trace the values of Python variables, packet signatures, and the matching process logic, insert print statements into the Python code.      1. **Unit Testing (Python's unittest or pytest)**: Create unit tests in Python for the function or code segment that handles signature-based detection only. Make sure your test cases include packets with known threat signatures in them. 2. **Debugging Tools (Python's pdb)**: Use interactive debugging with Python's pdb module if you think there might be a problem with the logic. Use pdb.set\_trace() to insert breakpoints into your Python code so that you can step through the execution of the code interactively and inspect variables. 3. **Third-Party Debugging Tools**: Integrated development environments (IDEs) and third-party Python debugging tools that provide features like variable inspection, error tracking, and step-by-step debugging should be taken into consideration. Strong debugging features are offered by well-known Python IDEs like PyCharm, Visual Studio Code, and Jupyter Notebook. |

## PART 5: Commenting Your Program

|  |
| --- |
| **Task**  Submit your full program code, including thorough comments describing what each portion of the program should do when working correctly.  **Requirements**   * The purpose of the program and each of its parts should be clear to a reader that does not know the Python programming language.   **Inspiration**  When writing your entry, you are encouraged to consider the following:   * Is each section or sub-section of your code commented to describe what the code is doing? * Give your code with comments to a friend or family member to review. Add additional comments to spots that confuse them to make it clearer. |
| <Copy your full program code here, including comments>  # Variables and data structures  threshold\_anomaly = 100 # Predefined threshold for anomaly detection  known\_threat\_signatures = {  "SQL Injection": ["DROP", "UNION", "SELECT"],  "Cross-Site Scripting (XSS)": ["<script>", "onload", "alert"]  }  # Common tasks  def collect\_network\_traffic\_data():  # Collecting network traffic (simple)  return ["Normal packet", "SQL Injection packet", "Normal packet", "XSS packet"]  def preprocess\_packet(packet):  # Packet preprocessing (simple)  return packet  def calculate\_packet\_rate(packets, time\_interval):  # Calculation of incoming packets (simple)  return len(packets) / time\_interval  def match\_signature(packet, known\_signatures):  # Comparison to known threat signatures  for threat, signatures in known\_signatures.items():  for signature in signatures:  if signature in packet:  return True, threat  return False, None  def generate\_alert(alert\_type, details):  # Generating an alert (simple)  print(f"Alert: {alert\_type} - {details}")  # M Program loop (simulated)  while True:  # Network traffic data (simulate)  packets = collect\_network\_traffic\_data()  # Each packet  for packet in packets:  parsed\_packet = preprocess\_packet(packet)  # Anomaly Detection  packet\_rate = calculate\_packet\_rate(packets, time\_interval=1) # Assuming 1 minute intervals  if packet\_rate > threshold\_anomaly:  generate\_alert("Anomaly", "Abnormal packet rate detected.")  # Signature-Based Detection  matched, threat = match\_signature(parsed\_packet, known\_threat\_signatures)  if matched:  generate\_alert("Signature", f"Known threat signature detected: {threat}")  #Insert print statements for debugging  print("Packet:", packet)  print("Matching against known signatures:", known\_threat\_signatures) |

## PART 6: Your Completed Program

|  |
| --- |
| **Task**  Provide the Replit link to your full program code.  **Requirements**   * The program must work correctly with all the comments included in the program.   **Inspiration**   * Check before submitting your touchstone that your final version of the program is running successfully. |
| <Provide the link to your program here>  https://replit.com/join/hazlagdook-analystbean |