Cyber-Attack Detection in a Smart Home System

Last Modified: Apr 19, 2025

Objective

Implement a set of functions to identify and differentiate cyber-attack attempts from authorized requests by analyzing "attack signatures" (e.g., multiple failed login attempts, abnormal command frequency, out-of-bound power consumption values).

Background & Scope

In this exercise, you will design a lightweight, Google Analytics-style “snippet(s)” to add to key spots in a pre-existing Smart Building codebase. Each snippet calls your detection functions to log, flag, and alert on abnormal or malicious behavior—no machine learning required.

You don’t need to actually invoke your functions via these code snippets in a pre-existing codebase, as you don’t have access to such a system. You only need to show that these code snippets will invoke your detection functions, and the functions will correctly detect attacks by using unit tests.

Requirements

1. **Hot-Spot Instrumentation**
   Students should write functions that are called at predefined "hot spots" in the existing smart building codebase (similar to Google Analytics snippets). These functions will receive parameters such as request type, user role, timestamp, and source identifier (like IP or device ID). Hot spots can be Authentication entry points, Device/Space controller APIs, Power readings, etc. Example code snippet with parameters (only an example):

```
instrument(
  String eventName,        // e.g. "login_attempt", "toggle_device"
  String userRole,         // "ADMIN", "MANAGER", "USER"
  String userId,           // unique user identifier
  String sourceId,         // IP address or device ID
  Instant timestamp,       // java.time.Instant.now()
  Map<String, Object> context // e.g. { "success": false, "value": 120.5 }
);
```

2. **Detection Algorithm**
   The functions must:
   - Analyze incoming requests for patterns that indicate attacks.
   - Detect anomalies, such as:
     - Rate checks
       - A high number of failed login attempts within a short time.
         - Ex: Count >5 within 1 minute -> flag
       - Abnormal frequency of control commands *(toggle spams)* (e.g., turning devices on/off repeatedly).
         - Ex: Count >10 on/off commands in 30 seconds -> flag
     - Value checks
       - Power consumption values that are significantly out-of-range *(power spikes or out of possible range)*.
         - Ex: Reading > 150% of historical average -> flag
         - Ex: Negative or zero where it is not allowed
     - Role-aware filtering
       - Ignore normal bursts by ADMIN or MANAGER if within known business hours.
     - Decide and implement two more anomaly detection functions on your own that are not specified above.
   - Differentiate between authorized requests (even if high frequency) and potential attacks by checking context parameters like active sessions or role privileges.

     Example code snippet:

```
public class AttackDetector {
  public static void instrument(
    String eventName,
    String userRole,
    String userId,
    String sourceId,
    Instant timestamp,
    Map<String,Object> context
  ) {
    // TODO : Complete method with attack detection logic
  }
}
```

   - Log and alert possible attacks by:
     - Logging all suspicious events in a database or a JSON file with sufficient details (timestamp, request type, user info, etc.).
     - Triggering an alert (just a flag/mark in the log) when an abnormal event is detected.

3. **Test Harness**
    - Write a simple main() function or unit-test test cases that:
        - Simulates normal usage (e.g., USER toggles lights twice).
        - Simulates attacks (e.g., 6 failed logins in 30s according to the above example)
    - Demonstrate correct flagging and non-flagging.

Assignment Deliverables

1. **Code Implementation**
    - Functions that detect and log potential attacks.
    - Sample calls simulating both normal operations and attack scenarios.
2. **Documentation**
    - A clear explanation of the detection thresholds and logic.
    - An outline of how the functions would integrate into the smart building system.
3. **Test Cases**
    - Include test cases demonstrating detection of both legitimate high-frequency events and attack signatures covering all the attack/anomaly detections you have implemented.

Hint: Instead of machine learning, students can use rule-based algorithms such as threshold checking, statistical deviation, and frequency analysis to detect anomalies and abnormal activities. This involves setting specific limits for behavior (e.g., too many failed requests in a given period or unexpected command patterns) and flagging events that fall outside these boundaries. This approach keeps the assignment simple while still teaching valuable concepts in anomaly detection and security monitoring without requiring advanced ML knowledge.

Evaluation Criteria

- Hot-Spot Instrumentation
- Detection Logic (rules)
- Logging & Alert Quality
- Test Coverage
- Documentation Clarity