Department of Electronic & Telecommunication Engineering
University of Moratuwa

EN4720 - Security in Cyber-Physical Systems

Project - Milestone IV

Cyber-Attack Detection in a Smart Home System

1st June 2025

Amarasekara A.T.P. 200023C
Bandara D.M.D.V 200061N
Samarasekera A.M.P.S. 200558U
Wijetunga W.L.N.K. 200733D

Contents

1 Introduction **2**

2 Methodology **2**

2.1 Anomaly Detection 2

2.1.1 Failed login detection 2

2.1.2 Toggle spam and device overload detection 2

2.1.3 Power anomaly detection 3

2.1.4 Session hijacking and duration detection 3

2.1.5 Check for previously flagged user or IP 3

2.2 Testing and Simulation 4

2.2.1 Failed login detection 4

2.2.2 Toggle spam and device overload detection 4

2.2.3 Power anomaly detection 5

2.2.4 Session hijacking and duration detection 6

2.2.5 Check for previously flagged user or IP 6

3 Conclusion **7**

4 Appendix **8**

1 Introduction

This report presents a rule-based methodology to detect cyber-attacks in a smart home system. The objective is to implement detection functions that can identify malicious activity based on predefined behaviours and assigned rules. These detection mechanisms are designed to be invoked via instrumentation snippets integrated at key points within a smart building system. The report focuses on the methodology used for designing the detection functions along with the statistical methods used for determining parameters, the code and functionality along with the testing methods utilised.

2 Methodology

The functions for both the detection of anomalies and simulation for testing detection were implemented on a python notebook available in the Github repository. The functions are also available in the Appendix section.

2.1 Anomaly Detection

The following anomaly detection functions were implemented. This includes functions to detect failed logins, toggle spam and device overload, and power anomaly detection. In addition to this, session hijacking and duration detection, and flagged user detection was implemented. For all the functions, an ALERT is printed in addition to the information being logged and flagged users being saved to memory.

2.1.1 Failed login detection

This function is used to monitor login attempts to detect suspicious behaviour based on two main criteria, repeated failed logins and geographically impossible logins. For failed login detection, it observes unsuccessful login attempts by checking if the login is successful or not. If a login fails, the attempt is logged under the user's ID. The system then filters for failed attempts that occurred within the last 60 seconds. If more than 5 such failures are detected within this time window, the user and the source IP are flagged, and an alert is sent.

The function also checks whether logins are originating from unusual geographical locations. It tracks the countries associated with a user's login IP addresses over the past five minutes. If more than two countries are detected in that period, the system compares timestamps between logins from different countries. Using a predefined dictionary of minimum realistic travel times between the locations, it flags users if the time between two logins is shorter than physically possible.

2.1.2 Toggle spam and device overload detection

This function is used to detect abnormal behaviour related to device toggling, spamming, device overload, and inconsistent toggle states. It begins by logging each device-toggle event under the corresponding user and then filters for toggle actions performed within the past 30 seconds. If more than 10 such actions are detected in that short window, and the user is not an admin or manager—or the activity occurs outside business hours, then the system flags the behaviour as an anomaly.

The next check focuses on device overload. For each device, the system keeps track of who accessed it in the last 10 seconds. If the number of unique users exceeds a defined threshold of more than 5 users for critical devices and more than 10 for non-critical ones, then the system raises a flag.

Lastly, the function checks the sequence of toggle states recorded in that 10-second window to detect conflicting actions. If at least five toggles have occurred and the most recent state contradicts the one immediately before it, this may suggest conflicting control commands, possibly from compromised or unauthorised users. An alert is generated in such cases, and the users involved are flagged. This is used in scenarios where multiple parties may be issuing contradictory commands, or the same party spamming the system.

2.1.3 Power anomaly detection

This function monitors incoming power readings to detect anomalies by comparing each new reading against recent history. Every time a power reading occurs, the code retrieves the reading and appends it, with its timestamp, to a rolling list of power history. To maintain a manageable window of comparison, the list is limited to 20 entries. The code then calculates the average of all stored values, which serves as the baseline for what constitutes normal power usage.

If a reading is zero or negative, the code treats this as an invalid or impossible measurement and immediately flags it. Furthermore, for unusually large readings, the code checks if the current ‘value‘ exceeds 150% of the recent average. If so, it interprets this as a power spike. However, to avoid false positives during legitimate high-usage periods, it considers user roles and business hours. Otherwise, it is flagged, an alert is raised.

2.1.4 Session hijacking and duration detection

This function detects potential session hijacking and flags unusually long or idle user sessions based on session metadata and recent activity. When a session starts, it records the session ID, the initiating IP address, and the session type. It logs the session's creation and activity history, along with start time and the most recent interaction. To detect hijacking, it checks whether the same session has been accessed from more than one unique IP address within the last 60 seconds. If so, it raises an alert.

The second section handles session ending. When a session is closed, the system calculates both the total session duration and the idle time since the last recorded activity. It then evaluates these against defined thresholds, which vary by session type where for interactive sessions 24 hours and for others 48 hours. If a session exceeds its expected lifetime and has been idle for over 12 hours, it is considered anomalous and a flag is generated.

2.1.5 Check for previously flagged user or IP

This function monitors user activity by checking whether the user ID or the source IP address has been previously flagged for suspicious behaviour. It does this by testing if either user ID exists in a list of flagged users, or if source IP address appears in a set of flagged IPs. If either value is found in the flagged lists, the system generates an alert message noting the flagged user or IP.

2.2 Testing and Simulation

Each detection function was tested against the following simulated code.
This function simulates a typical sequence of user activity that will be considered normal system usage. It begins with a successful login, followed by a short pause and then the user then toggles a non-critical device, submits a normal power reading, and initiates a session. After a brief duration, the session is ended. Each action spaced out with slight random delays for realistic behaviour and thus no alert is triggered.

```
Event: login_attempt, Role: USER, User: normal_user, Time: 2025-05-31 17:21:23.872093
Event: toggle_device, Role: USER, User: normal_user, Time: 2025-05-31 17:21:25.318852
Event: power_reading, Role: USER, User: normal_user, Time: 2025-05-31 17:21:26.992119
Event: session_start, Role: USER, User: normal_user, Time: 2025-05-31 17:21:28.109250
Event: session_end, Role: USER, User: normal_user, Time: 2025-05-31 17:21:29.921610
```

Figure 1: Normal behaviour simulation with no alert flagged

2.2.1 Failed login detection

We use two functions for failed login detection,
The first function simulates repeated failed login attempts from the same user and IP address. It generates a series of unsuccessful login events spaced by a short, random delay to reflect a brute force login attempt. After 5 unsuccessful login events the detection algorithm triggers an alert.

```
Event: login_attempt, Role: USER, User: malicious, Time: 2025-06-01 18:32:40.570701
Event: login_attempt, Role: USER, User: malicious, Time: 2025-06-01 18:32:42.345373
Event: login_attempt, Role: USER, User: malicious, Time: 2025-06-01 18:32:43.974911
Event: login_attempt, Role: USER, User: malicious, Time: 2025-06-01 18:32:45.937692
Event: login_attempt, Role: USER, User: malicious, Time: 2025-06-01 18:32:47.328952
Event: login_attempt, Role: USER, User: malicious, Time: 2025-06-01 18:32:48.064427
ALERT: Too many failed login attempts for malicious
```

Figure 2: Login spam simulation results

There is also a function that simulates successful logins from the same user but from multiple different IP addresses in a short time gap. This activity mimics a geographical anomaly, where a single user appears to log in from multiple regions. Here once the user changes the IP from US to UK and UK to China in a short period, an alert is triggered.

```
Event: login_attempt, Role: USER, User: geo_user, Time: 2025-06-01 18:33:51.669999
Event: login_attempt, Role: USER, User: geo_user, Time: 2025-06-01 18:33:52.781265
Event: login_attempt, Role: USER, User: geo_user, Time: 2025-06-01 18:33:53.503861
ALERT: Unrealistic login locations for geo_user: US to UK in 0.00 hours
ALERT: Unrealistic login locations for geo_user: UK to CN in 0.00 hours
```

Figure 3: Unrealistic location simulation results

2.2.2 Toggle spam and device overload detection

This function simulates a user excessively toggling a non-critical device in quick succession. It sends a series of toggle events—eleven by default—with slight, random delays between each

action to mimic rapid interaction. The figure shows that once the user triggers the device more than 10 instances, an alert is sent.

```
Event: toggle_device, Role: USER, User: spammer, Time: 2025-06-01 18:29:16.645934
Event: toggle_device, Role: USER, User: spammer, Time: 2025-06-01 18:29:18.028253
Event: toggle_device, Role: USER, User: spammer, Time: 2025-06-01 18:29:19.621936
Event: toggle_device, Role: USER, User: spammer, Time: 2025-06-01 18:29:20.733699
Event: toggle_device, Role: USER, User: spammer, Time: 2025-06-01 18:29:22.551620
Event: toggle_device, Role: USER, User: spammer, Time: 2025-06-01 18:29:24.230015
Event: toggle_device, Role: USER, User: spammer, Time: 2025-06-01 18:29:25.007939
Event: toggle_device, Role: USER, User: spammer, Time: 2025-06-01 18:29:26.274133
Event: toggle_device, Role: USER, User: spammer, Time: 2025-06-01 18:29:27.318599
Event: toggle_device, Role: USER, User: spammer, Time: 2025-06-01 18:29:28.686623
Event: toggle_device, Role: USER, User: spammer, Time: 2025-06-01 18:29:30.306705
ALERT: Device toggled too frequently by spammer
```

Figure 4: Simulation of toggle spam with alert

This function tests for role privilege consideration. It simulates an admin spamming a system with over 10 device triggers during and after work hours, with an alert being sent only for outside work hours.

```
Event: toggle_device, Role: ADMIN, User: admin_real, Time: 2025-06-01 14:00:00.241958
Event: toggle_device, Role: ADMIN, User: admin_real, Time: 2025-06-01 14:00:00.241958
Event: toggle_device, Role: ADMIN, User: admin_real, Time: 2025-06-01 14:00:00.241958
Event: toggle_device, Role: ADMIN, User: admin_real, Time: 2025-06-01 14:00:00.241958
Event: toggle_device, Role: ADMIN, User: admin_real, Time: 2025-06-01 14:00:00.241958
Event: toggle_device, Role: ADMIN, User: admin_real, Time: 2025-06-01 14:00:00.241958
Event: toggle_device, Role: ADMIN, User: admin_real, Time: 2025-06-01 14:00:00.241958
Event: toggle_device, Role: ADMIN, User: admin_real, Time: 2025-06-01 14:00:00.241958
Event: toggle_device, Role: ADMIN, User: admin_real, Time: 2025-06-01 14:00:00.241958
Event: toggle_device, Role: ADMIN, User: admin_real, Time: 2025-06-01 14:00:00.241958
Event: toggle_device, Role: ADMIN, User: admin_real, Time: 2025-06-01 14:00:00.241958
Event: power_reading, Role: MANAGER, User: admin_real, Time: 2025-06-01 14:00:00.241958
Event: toggle_device, Role: ADMIN, User: admin_malicious, Time: 2025-06-01 22:00:00.042658
Event: toggle_device, Role: ADMIN, User: admin_malicious, Time: 2025-06-01 22:00:00.042658
Event: toggle_device, Role: ADMIN, User: admin_malicious, Time: 2025-06-01 22:00:00.042658
Event: toggle_device, Role: ADMIN, User: admin_malicious, Time: 2025-06-01 22:00:00.042658
Event: toggle_device, Role: ADMIN, User: admin_malicious, Time: 2025-06-01 22:00:00.042658
Event: toggle_device, Role: ADMIN, User: admin_malicious, Time: 2025-06-01 22:00:00.042658
Event: toggle_device, Role: ADMIN, User: admin_malicious, Time: 2025-06-01 22:00:00.042658
Event: toggle_device, Role: ADMIN, User: admin_malicious, Time: 2025-06-01 22:00:00.042658
Event: toggle_device, Role: ADMIN, User: admin_malicious, Time: 2025-06-01 22:00:00.042658
Event: toggle_device, Role: ADMIN, User: admin_malicious, Time: 2025-06-01 22:00:00.042658
Event: toggle_device, Role: ADMIN, User: admin_malicious, Time: 2025-06-01 22:00:00.042658
ALERT: Device toggled too frequently by admin_malicious
Event: power_reading, Role: MANAGER, User: admin_malicious, Time: 2025-06-01 22:00:00.042658
ALERT: Activity from previously flagged source: admin_malicious / 155.23.65.1
ALERT: Power spike (3000) exceeds 150% of avg (1550.00)
```

Figure 5: Malicious admin behaviour simulation

2.2.3 Power anomaly detection

This function simulates a situation where a user produces a series of normal power readings followed by a sudden, abnormal spike. It sends several consistent readings at a base value to establish a typical usage pattern. After a short pause between each, a final reading is submitted with a significantly higher value. This last value is intentionally selected to exceed 150% of the average of the preceding readings, which then triggers an alert.

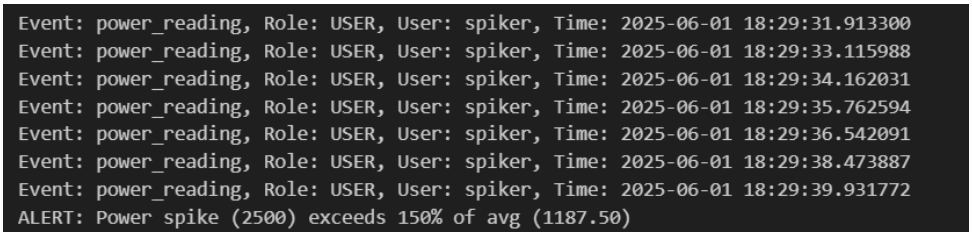

Figure 6: Simulation of power spike alert

2.2.4 Session hijacking and duration detection

This function simulates a session hijacking scenario in which the same user session is initiated from two different IP addresses within a short time frame. It begins by starting a session from one IP, then shortly afterwards initiates the same session ID from a different IP address. The session is later ended from the second location to which an alert is flagged.

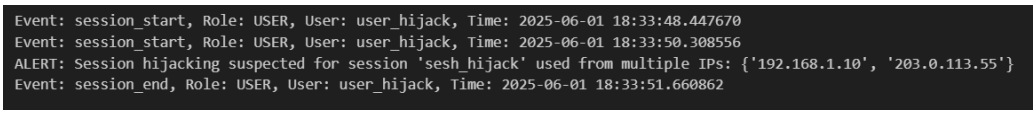

Figure 7: Session hijack simulation and alert

2.2.5 Check for previously flagged user or IP

This function simulates activity from a user who becomes flagged due to suspicious behaviour. It begins with multiple consecutive failed login attempts to tigger an alert and marking the user and IP as flagged. Afterwards, when the user tries to conduct a device toggle (or any action), it is detected and an alert is triggered.

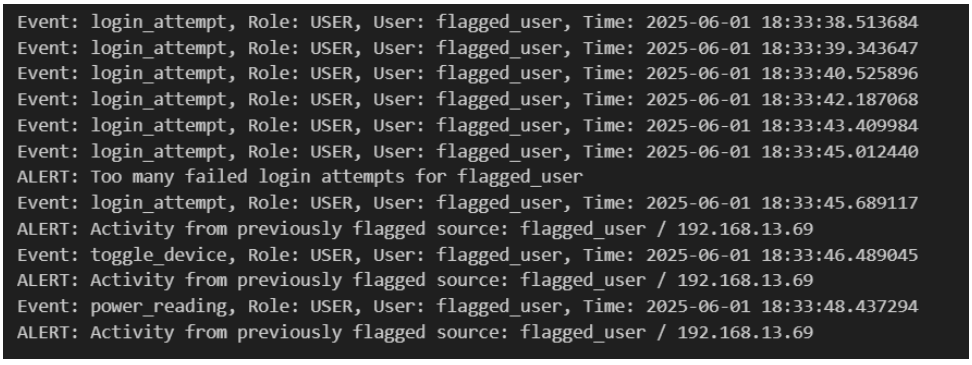

Figure 8: Simulation of flagged source detection and alert.

Following are the logs written to the text file for the simulations.

Detection, Date, Time, User, Source, Message
1, 2025/06/01, 18:32:48, malicious, 192.122.36.1, Too many failed login attempts for malicious
2, 2025/06/01, 18:33:02, spammer, 192.111.23.6, Device toggled too frequently by spammer

3, 2025/06/01, 18:33:11, spiker, 192.168.2.2, Power spike (2500) exceeds 150% of avg (1187.50)
4, 2025/06/01, 18:33:37, admin_malicious, 155.23.65.1, Device toggled too frequently by admin_malicious
5, 2025/06/01, 18:33:38, admin_malicious, 155.23.65.1, Activity from previously flagged source: admin_malicious / 155.23.65.1
6, 2025/06/01, 18:33:38, admin_malicious, 155.23.65.1, Power spike (3000) exceeds 150% of avg (1550.00)
7, 2025/06/01, 18:33:45, flagged_user, 192.168.13.69, Too many failed login attempts for flagged_user
8, 2025/06/01, 18:33:45, flagged_user, 192.168.13.69, Activity from previously flagged source: flagged_user / 192.168.13.69
9, 2025/06/01, 18:33:46, flagged_user, 192.168.13.69, Activity from previously flagged source: flagged_user / 192.168.13.69
10, 2025/06/01, 18:33:48, flagged_user, 192.168.13.69, Activity from previously flagged source: flagged_user / 192.168.13.69
11, 2025/06/01, 18:33:50, user_hijack, 203.0.113.55, Session hijacking suspected for session 'sesh_hijack' used from multiple IPs: '192.168.1.10', '203.0.113.55'
12, 2025/06/01, 18:33:53, geo_user, 172.16.1.1, Unrealistic login locations for geo_user: US to UK in 0.00 hours
13, 2025/06/01, 18:33:53, geo_user, 172.16.1.1, Unrealistic login locations for geo_user: UK to CN in 0.00 hours
14, 2025/06/01, 18:33:57, user3, 192.168.5.3, Device overload on device_5 (critical) by users: 'user1', 'user3', 'user2'
15, 2025/06/01, 18:33:58, user4, 192.168.5.4, Device overload on device_5 (critical) by users: 'user4', 'user1', 'user3', 'user2'

3 Conclusion

The report contains functions used to program rule-based methods designed to detect anomalous user behaviour and system activity within a smart home system. The detection logic focused on indicators such as failed login attempts, geographically inconsistent access patterns, excessive device toggling, power spikes, and session hijacking. Thresholds were defined and statistical methods were implemented for the detection of these anomalies.

For testing purposes, several simulation functions were developed to emulate normal and malicious behaviour. These functions covered all the detection methods implemented. The simulated inputs effectively triggered the intended alerts and finally wrote them onto the logs.

4 Appendix

Anomaly detection

```
def instrument(event_name, user_role, user_id, source_id, timestamp, context):
    print(f'Event: {event_name}, Role: {user_role}, User: {user_id}, Time: {timestamp}')

    # Update event counts for visualisation
    event_counts[event_name].setdefault(user_id, 0)
    event_counts[event_name][user_id] += 1

    # Update last active timestamp for session
    session_id = context.get('session_id')
    if session_id and session_id in session_durations:
        session_durations[session_id]['last_active'] = timestamp

    # Check for previously flagged user or IP
    if user_id in flagged_users or source_id in flagged_ips:
        alert_msg = f'Activity from previously flagged source: {user_id} / {source_id}'
        print(f"ALERT: {alert_msg}")
        log_anomaly(user_id, source_id, alert_msg)

    # Failed login detection
    if event_name == 'login_attempt':
        success = context.get('success', True)
        if not success:
            failed_logins.setdefault(user_id, []).append(timestamp)
            recent = [t for t in failed_logins[user_id] if (timestamp - t).total_seconds() <= 60]
            failed_logins[user_id] = recent
            if len(recent) > 5:
                flagged_users.add(user_id)
                flagged_ips.add(source_id)
                alert_msg = f'Too many failed login attempts for {user_id}'
                print(f"ALERT: {alert_msg}")
                log_anomaly(user_id, source_id, alert_msg)

        # Unusual login location detection
        country = ip_to_country.get(source_id, 'UNKNOWN')
        login_locations.setdefault(user_id, []).append((source_id, timestamp, country))
        recent_locations = [(ip, t, c) for ip, t, c in login_locations[user_id] if (timestamp - t).total_seconds() <= 300]
        login_locations[user_id] = recent_locations
        countries = set(c for _, _, c in recent_locations)

```

```
        if len(countries) > 2:
            # Check travel time feasibility
            for i, (_, t1, c1) in enumerate(recent_locations):
                for _, t2, c2 in recent_locations[i+1:]:
                    if c1 != c2:
                        hours = (t2 - t1).total_seconds() / 3600
                        min_travel = travel_times.get((min(c1, c2),
                            max(c1, c2)), 5)
                        if hours < min_travel:
                            alert_msg = f'Unrealistic login locations
                                for {user_id}: {c1} to {c2} in
                                {hours:.2f} hours'
                            print(f"ALERT: {alert_msg}")
                            log_anomaly(user_id, source_id, alert_msg)
                            flagged_users.add(user_id)
                            flagged_ips.update(ip for ip, _, _ in
                                recent_locations)
                            break

    # Toggle spam and device overload detection
    if event_name == 'toggle_device':
        toggle_events.setdefault(user_id, []).append(timestamp)
        recent = [t for t in toggle_events[user_id] if (timestamp -
            t).total_seconds() <= 30]
        toggle_events[user_id] = recent
        device_id = context.get('device_id', 'unknown')
        device_type = context.get('device_type', 'non-critical')
        toggle_state = context.get('state', 'on')
        device_access_log.setdefault(device_id, []).append((user_id,
            timestamp, toggle_state))
        if len(recent) > 10:
            if user_role not in ('ADMIN', 'MANAGER') or not
                is_business_hours(timestamp):
                alert_msg = f'Device toggled too frequently by
                    {user_id}'
                print(f"ALERT: {alert_msg}")
                log_anomaly(user_id, source_id, alert_msg)
                flagged_users.add(user_id)
                flagged_ips.add(source_id)

        # Device overload detection
        recent_access = [(u, t, s) for u, t, s in
            device_access_log[device_id] if (timestamp -
            t).total_seconds() <= 10]
        device_access_log[device_id] = recent_access
        unique_users = set(u for u, _, _ in recent_access)
        threshold = 2 if device_type == 'critical' else 4
        if len(unique_users) > threshold:
            alert_msg = f'Device overload on {device_id}
                ({device_type}) by users: {unique_users}'
```

```
            print(f"ALERT: {alert_msg}")
            log_anomaly(user_id, source_id, alert_msg)
            flagged_users.update(unique_users)

        # Conflicting toggle state detection
        states = [s for _, _, s in recent_access]
        if len(states) >= 5 and states[-1] != states[-2]:
            alert_msg = f'Conflicting toggle states on {device_id}: {states[-2]} to {states[-1]}'
            print(f"ALERT: {alert_msg}")
            log_anomaly(user_id, source_id, alert_msg)
            flagged_users.update(unique_users)

    # Power anomaly detection
    if event_name == 'power_reading':
        value = context.get('value', 0)
        power_history.append((timestamp, value))
        if len(power_history) > 20:
            power_history.pop(0)
        avg = sum(v for _, v in power_history) / len(power_history)
        if value <= 0:
            alert_msg = f'Invalid power reading ({value}) from {source_id}'
            print(f"ALERT: {alert_msg}")
            log_anomaly(user_id, source_id, alert_msg)
            flagged_users.add(user_id)
            flagged_ips.add(source_id)

        elif value > 1.5 * avg:
            if user_role not in ('ADMIN', 'MANAGER') or not is_business_hours(timestamp):
                alert_msg = f'Power spike ({value}) exceeds 150% of avg ({avg:.2f})'
                print(f"ALERT: {alert_msg}")
                log_anomaly(user_id, source_id, alert_msg)
                flagged_users.add(user_id)
                flagged_ips.add(source_id)

    # Session hijacking and duration detection
    if event_name == 'session_start':
        session_id = context.get('session_id')
        ip = source_id
        session_type = context.get('session_type', 'interactive')
        if not session_id:
            return
        session_creations.setdefault(session_id, []).append((user_id, timestamp))
        session_activity.setdefault(session_id, []).append((ip, timestamp))
        session_durations[session_id] = {'start': timestamp, 'end':
```

```
            None, 'last_active': timestamp, 'type': session_type}
        recent = [(ip_old, ts_old) for ip_old, ts_old in
            session_activity[session_id] if (timestamp -
            ts_old).total_seconds() <= 60]
        session_activity[session_id] = recent
        unique_ips = set(ip_old for ip_old, _ in recent)
        if len(unique_ips) > 1:
            alert_msg = f'Session hijacking suspected for session
                \'{session_id}\' used from multiple IPs: {unique_ips}'
            print(f"ALERT: {alert_msg}")
            log_anomaly(user_id, source_id, alert_msg)

    if event_name == 'session_end':
        session_id = context.get('session_id')
        if not session_id or session_id not in session_durations:
            return
        session_durations[session_id]['end'] = timestamp
        duration = (timestamp -
            session_durations[session_id]['start']).total_seconds() /
            3600
        idle_time = (timestamp -
            session_durations[session_id]['last_active']).total_seconds()
            / 3600
        threshold = 24 if session_durations[session_id]['type'] ==
            'interactive' else 48
        if duration > threshold and idle_time > 12:
            alert_msg = f'Excessive idle session duration for
                {session_id}({session_durations[session_id]}):
                {duration:.2f} hours, idle for {idle_time:.2f} hours'
            print(f"ALERT: {alert_msg}")
            log_anomaly(user_id, source_id, alert_msg)
            flagged_users.add(user_id)
            flagged_ips.add(source_id)
```

Normal user simulation

```

def sim_normal_use(user_id, user_ip, timestamp=None):
    if timestamp is None:
        timestamp = datetime.now()
    instrument('login_attempt', 'USER', user_id, user_ip, timestamp,
        {'success': True})
    time.sleep(random.uniform(0.5, 2.0))
    instrument('toggle_device', 'USER', user_id, user_ip,
        datetime.now(), {'device_id': 'device_1', 'device_type':
        'non-critical', 'state': 'on'})
    time.sleep(random.uniform(0.5, 2.0))
    instrument('power_reading', 'USER', user_id, user_ip,
        datetime.now(), {'value': 1000})
    time.sleep(random.uniform(0.5, 2.0))
```

```
    instrument('session_start', 'USER', user_id, user_ip,
       datetime.now(), {'session_id': 'normal_session',
       'session_type': 'interactive'})
    time.sleep(random.uniform(0.5, 2.0))
    instrument('session_end', 'USER', user_id, user_ip,
       datetime.now(), {'session_id': 'normal_session'})
```

Failed login simulation

```
def sim_failed_login(user_id, user_ip, count=6):
    for _ in range(count):
        instrument('login_attempt', 'USER', user_id, user_ip,
           datetime.now(), {'success': False})
        time.sleep(random.uniform(0.5, 2.0))

def sim_location_anomaly(user_id, ip_list):
    for ip in ip_list:
        instrument('login_attempt', 'USER', user_id, ip,
           datetime.now(), {'success': True})
        time.sleep(random.uniform(0.5, 2.0))
```

Code for toggle spam and device overload simulation

```
def sim_toggle_spam(user_id, role, user_ip, count=11):
    states = ['on', 'off']
    for i in range(count):
        instrument('toggle_device', role, user_id, user_ip,
           datetime.now(), {'device_id': 'device_2', 'device_type':
           'non-critical', 'state': states[i % 2]})
        time.sleep(random.uniform(0.5, 2.0))
```

Power anomaly simulation

```
def sim_power_spike(user_id, user_ip, base=1000, spike=2000):
    for _ in range(6):
        instrument('power_reading', 'USER', user_id, user_ip,
           datetime.now(), {'value': base})
        time.sleep(random.uniform(0.5, 2.0))
    instrument('power_reading', 'USER', user_id, user_ip,
       datetime.now(), {'value': spike})
```

Admin usage simulation

```
def sim_admin_behavior(admin_id, admin_ip, hour=10):
    ts = datetime.now().replace(hour=hour, minute=0, second=0)
    states = ['on', 'off']
    for i in range(11):
        instrument('toggle_device', 'ADMIN', admin_id, admin_ip, ts,
           {'device_id': 'device_3', 'device_type': 'critical',
           'state': states[i % 2]})
```

```
        time.sleep(random.uniform(0.5, 2.0))
    instrument('power_reading', 'MANAGER', admin_id, admin_ip, ts,
        {'value': 3000})
```

Session hijacking simulation

```
def sim_session_hijack(session_id):
    instrument('session_start', 'USER', 'user_hijack',
        '192.168.1.10', datetime.now(), {'session_id': session_id,
        'session_type': 'interactive'})
    time.sleep(random.uniform(0.5, 2.0))
    instrument('session_start', 'USER', 'user_hijack',
        '203.0.113.55', datetime.now(), {'session_id': session_id,
        'session_type': 'interactive'})
    time.sleep(random.uniform(0.5, 2.0))
    instrument('session_end', 'USER', 'user_hijack', '203.0.113.55',
        datetime.now(), {'session_id': session_id})
```

Flagged user simulation

```
def sim_flagged_source(user_id, user_ip):
    for _ in range(6):
        instrument('login_attempt', 'USER', user_id, user_ip,
            datetime.now(), {'success': False})
        time.sleep(random.uniform(0.5, 2.0))
    instrument('login_attempt', 'USER', user_id, user_ip,
        datetime.now(), {'success': True})
    time.sleep(random.uniform(0.5, 2.0))
    instrument('toggle_device', 'USER', user_id, user_ip,
        datetime.now(), {'device_id': 'device_4', 'device_type':
        'non-critical', 'state': 'on'})
    time.sleep(random.uniform(0.5, 2.0))
    instrument('power_reading', 'USER', user_id, user_ip,
        datetime.now(), {'value': 950})
```