IoTMonitor: A Hidden Markov Model-based Security System to Identify Crucial Attack Nodes in Trigger-action IoT Platforms

Md Morshed Alam, Md Sajidul Islam Sajid, Weichao Wang, Jinpeng Wei
Department of Software and Information Systems, University of North Carolina at Charlotte, Charlotte, USA
{malam3, msajid, wwang22, jwei8}@uncc.edu

***Abstract*—With the emergence and fast development of trigger-action platforms in IoT settings, security vulnerabilities caused by the interactions among IoT devices become more prevalent. The event occurrence at one device triggers an action in another device, which may eventually contribute to the creation of a chain of events in a network. Adversaries exploit the chain effect to compromise IoT devices and trigger actions of interest remotely just by injecting malicious events into the chain. To address security vulnerabilities caused by trigger-action scenarios, existing research efforts focus on validation of the security properties of devices, or verification of the occurrence of certain events based on their physical fingerprints on a device. We propose IoTMonitor, a security analysis system that discerns the underlying chain of event occurrences with the highest probability by observing a chain of physical evidence collected by sensors. We use the Baum-Welch algorithm to estimate transition and emission probabilities and the Viterbi algorithm to discern the event sequence. We can then identify the crucial nodes in the trigger-action sequence whose compromise allows attackers to reach their final goals. The experiment results of our designed system upon the PEEVES datasets show that we can rebuild the event occurrence sequence with high accuracy from the observations and identify the crucial nodes on the attack paths.**

***Index Terms*—Internet of Things, Hidden Markov Model, Trigger-action Platform, Smart Home**

I. Introduction

Due to the advances of trigger-action platforms (e.g. IFTTT [1]) in IoT domain, IoT networks become more vulnerable towards malicious event injection attacks. Since IoT devices create a chain of interactions maintaining functional dependencies between entities and actions [2] [3], it is possible for adversaries to remotely inject malicious events somewhere in the interaction chain using a ghost device and activate a critical action through the exploitation of autonomous trigger-action scenario. For instance, an adversary can inject a fake thermometer reading of 110°F into the chain to initiate a critical *window opening* action.

There are a number of research efforts in the existing literature that attempt to solve the vulnerabilities caused by the trigger-actions in an IoT network. Most of them are designed to validate security properties by identifying the unsafe or insecure state transitions in the network [2] [4] [5] [6]. There is another line of research attempts where policy violations are addressed by checking sensitive user actions that may violate security policies [5]. The research that is closest to our proposition is PEEVES [7], where physical fingerprints of the devices are extracted using machine learning techniques to verify whether or not a certain event actually occurs.

In this paper, we propose IoTMonitor, a security system that adopts a Hidden Markov Model based approach to determine the optimal attack path an attacker may follow to implement a trigger-action based attack, thus providing suggestions for subsequent patching and security measures. Our system examines the physical changes happening in an IoT environment due to the event occurrences, discovers the probabilistic relation between physical evidence and underlying events using the Baum-Welch algorithm [8] [9], and discerns the optimal attack path using the Viterbi algorithm [10]. When the optimal attack path is determined, IoTMonitor identifies the crucial nodes in the path that the attacker must compromise to carry out the attack. Such information can be used for prioritizing security measures for IoT platforms.

The contributions of the paper can be summarized as follows:

- We propose IoTMonitor, a Hidden Markov model based system that identifies the optimal attack path in a trigger-action IoT environment based on the probabilistic relation between actual IoT events and corresponding physical evidence;
- We implement the Baum-Welch algorithm to estimate transition and emission probabilities, and Viterbi algorithm to discern the attack path;
- We propose an algorithm to detect the crucial nodes in an extracted optimal attack path, thus providing guidelines for subsequent security measures;
- We thoroughly evaluate the performance of IoTMonitor in detecting the optimal attack path and achieve high accuracy scores.

The rest of the paper is organized into four sections. In Section II, we define the attack landscape, discuss an attack scenario, and present the threat model. In Section III, we present IoTMonitor and discuss each component of it in detail. Later in Section IV, we present the evaluation results of our approach. Finally, in Section V, we conclude the paper by summarizing the methodology and outputs of our experiments, and presenting future extensions.

II. Attack Landscape

A. A Sample Attack Scenario

Assume that Alice has a limited number of trigger-action enabled IoT devices including Smart Lock, Motion Detector, Accelerometer, Smart Light, Coffee Machine, and Smart Window. Alice controls each device through a mobile application from her cell phone. The devices communicate with each other through a hub. Since the platform supports trigger-action functionality, a device has the capability to trigger an action of another device.

Alice sets up the trigger events as follows. When she unlocks the smart lock of the front door and walks in, the motion sensor in the living room detects the motion and activates "home-mode". The home-mode activation event automatically turns on the smart light. When the light is turned on, the events associated with coffee grinding and window opening are triggered. When coffee is ready, Alice takes the coffee and enters into her bedroom by opening and subsequently closing the door. The vibration generated by the opening and closing operations of the door is measured by an accelerometer. Thus, a chain of events are triggered by the initial action.

Now, Eve, an attacker, wants to compromise the smart window remotely when Alice is not at home and the front door is locked. Her objective is to inject malicious events into the network to create a chain of interactions that eventually trigger the events associated with the window.

B. Threat Model

We assume that the attacker knows the locations of the IoT devices in the target system but she does not have physical access to the home. She can eavesdrop on wireless communication taking place between devices and the hub. Her goal is to perform a trigger-action attack by injecting fake events into the IoT network through ghost applications. The ghost applications impersonate target devices just by mimicking their characteristics and functionalities. Therefore, she does not need to deploy any real IoT devices to conduct the attack.

III. The IoTMonitor System

Since the attacker exploits trigger-action functionality of IoT network to generate a chain of interactions by injecting fake events, we can thwart a trigger-action attack effectively if we can identify the optimal attack path the attacker may follow and perform security hardening on the crucial nodes in the attack path. In this research work, we propose *IoTMonitor*, a system that discerns the optimal attack paths by analyzing physical evidence generated during the attack cycle, which are probabilistically correlated to the actual underlying events. IoTMonitor formulates the attack as a Hidden Markov Model (HMM) problem and solves it to determine the most likely sequence of events occur during an attack cycle and further identifies the crucial nodes in that sequence. Hence, in this paper, a *node* represents an event occurring at a particular device.

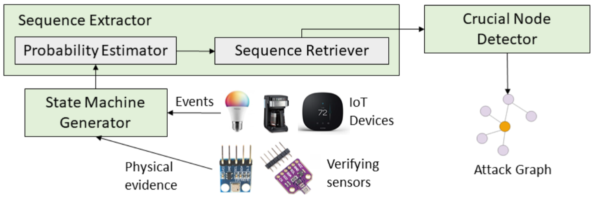

Fig. 1. IoTMonitor System

A. Our Assumption

We assume that a configured trigger-action sequence contains N events: $d_1, d_2, ..., d_N$. The attacker injects fake events $\{d_i\}$ in the chain to achieve her final goal. Note that the attacker does not necessarily have to inject d_1 since she can wait for the occurrence of some real events to trigger the automatic chain occurrence of the rest of the events required to implement the attack. When an event is triggered, it causes some physical changes in the environment, which can be perceived as corresponding physical evidence $\{ph_i\}$ captured by an array of sensors and harnessed to verify the occurrence of that specific event. Note that some event may trigger non observable evidence, but others may trigger more than one evidence.

Given this assumption, IoTMonitor models the trigger action scenario as a HMM problem, where physical evidence are visible to the analysis agent, but the actual events remain hidden. The tasks of the agent are to determine the probabilistic relation between events and evidence, and employ it to figure out the optimal attack path and diagnose the crucial nodes in that path.

B. IoTMonitor

The proposed IoTMonitor comprises three main components: 1) state machine generator, 2) sequence extractor, and 3) crucial node detector. Fig 1 shows the architecture of IoTMonitor. We discuss the components below in detail.

1) ***State Machine Generator****:* When events are triggered in the environment and the deployed sensors capture corresponding evidence per event occurrence, this component will construct a *state machine* to represent how state changes in the environment due to the exploitation of trigger-action functionalities across a series of time instances $t = 1, 2, ..., T$. Hence, *states* delineate useful information regarding the occurrence of different events d_i and corresponding evidence $\{ph_i\}$.

The state machine accommodates two types of states: 1) *true states*, which correspond to the actual event occurrences, and 2) *observation states*, which represent the physical evidence. Hence, the true states remain hidden, but the analysis agent leverages the observation states to infer the hidden true state sequence.

We define our state space as follows:

- true state, x_i : state responding to the occurrence of d_i
- observation state, y_j : a subset of the physical evidence $\{ph_1, ph_2, ..., ph_M\}$, which are emitted when the environment makes transition to a new state

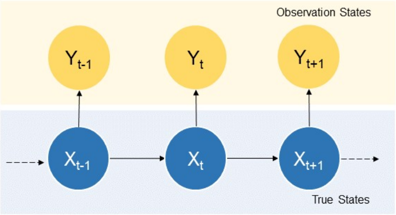

Fig. 2. A Sample State Machine

Hence, we assume that there are N true states $X = \{x_1, x_2, ..., x_N\}$, and T observation states $Y = \{y_1, y_2, ..., y_T\}$ in the state machine, where X_t and Y_t, respectively, denote the true state and observation state at time t. Here, each y_j contains a subset of the physical evidence $\{ph_1, ph_2, ..., ph_M\}$, where the total number of evidence is M. Note that each observation state Y_t in our experiment is determined with the help of a *sliding window* function, which is discussed in detail in Section IV.

When the environment is in x_i at time instance t and makes a transition to any $x_j \in X$ at time instance $t+1$, it changes its true state with a *transition probability* $q_{ij} \in Q$, which can be defined as:

$$q_{ij} = Pr(X_{t+1} = x_j | X_t = x_i), \quad 1 \leq i, j \leq N \tag{1}$$

Suppose, because of this state transition, the environment emits a new observation $y_k \in Y$ with an *emission probability* $\mu_j(y_k) \in E$, which can be defined as:

$$\mu_j(y_k) = Pr(Y_{t+1} = y_k | X_{t+1} = x_j), \quad \begin{array}{l} 1 \leq j \leq N \\ 1 \leq k \leq T \end{array} \tag{2}$$

In the equation (1), $Q = \{q_{ij}\}$ is termed as *state transition probability distribution*, while $E = \{\mu_j(y_k)\}$ in the equation (2) is termed as *emission probability distribution*.

To model the attack as HMM, we need to generate an *initial state distribution* $\sigma = \{\sigma_i\}$, such as:

$$\sigma_i = Pr(X_1 = x_i), \quad 1 \leq i \leq N \tag{3}$$

Hence, σ_i is the initial state distribution at time instance $t = 1$.

Combining all the five aforementioned tuples, IoTMonitor models the trigger-action attack as an HMM problem $\langle N, M, Q, E, \sigma \rangle$ and solves it to determine the optimal attack path given a sequence of observation states. IoTMonitor also creates a parameter $\theta = (\sigma, Q, E)$, which is called the *current model* of HMM.

Figure 2 shows a sample state machine where *blue* circles represent the true states and *yellow* circles represent the observation states.

Note: For the rest of the paper, we call *observation state* as only *observation* sometimes and use the terms *true state* and *state* interchangeably to mean the same thing.

*2) **Sequence Extractor**:* Once the trigger action sequence is modeled as an HMM problem, IoTMonitor attempts to estimate the probability values and retrieve the optimal hidden state sequence from the observations. First, it starts with estimating the converged state distributions, transmission probabilities, and emission probabilities. Then, it seeks to figure out the underlying state sequence that maximizes the probability of getting a certain observation sequence. To accomplish both tasks, the *sequence extractor* employs the following two subcomponents: a) probability estimator, and b) sequence retriever. The details of both subcomponents are described below.

a) **Probability Estimator**: Given a complete observation sequence $\langle Y_1, Y_2, ..., Y_T \rangle$, the goal of this component is to determine the following:

$$\theta^* = \underset{\theta}{argmax}\ Pr(Y_1, Y_2, ..., Y_T | \theta) \tag{4}$$

We use the Baum-Welch algorithm [8] [9] to iteratively update the current model θ and solve equation (4). It uses a *forward-backward procedure* to find the maximum likelihood estimate of θ given a certain set of observations. We assume that each observation Y_t is emitted by the environment at one discrete time instance $t = 1, 2, ..., T$.

Forward-backward Procedure: Let $\alpha_t(i)$ and $\beta_t(i)$ are the probabilities of getting the observation sequences $\langle Y_1, Y_2, ..., Y_t \rangle$ and $\langle Y_{t+1}, Y_{t+2}, ..., Y_T \rangle$, respectively, while the system is being in the true state x_i at time t. So,

$$\begin{aligned} \alpha_t(i) &= Pr(Y_1, Y_2, ..., Y_t, X_t = x_i | \theta) \\ \beta_t(i) &= Pr(Y_{t+1}, Y_{t+2}, ..., Y_T | X_t = x_i, \theta) \end{aligned} \tag{5}$$

We can compute $\alpha_t(i)$ and $\beta_t(i)$ using the following steps:

1. Initialization

$$\begin{aligned} &\alpha_1(i) = \sigma_i \mu_i(y_1), \quad 1 \leq i \leq N \\ &\beta_T(i) = 1, \quad 1 \leq i \leq N \end{aligned} \tag{6}$$

2. Induction

$$\alpha_{t+1}(j) = \mu_j(y_{t+1}) \sum_{i=1}^{N} \alpha_t(i) q_{ij}, \quad 1 \leq t \leq T-1, \quad 1 \leq j \leq N$$
$$\beta_t(i) = \sum_{j=1}^{N} q_{ij} \mu_j(y_{t+1}) \beta_{t+1}(j), \ t = T-1, ..., 2, 1, \quad 1 \leq i \leq N \tag{7}$$

These two steps combined is called the *forward-backward procedure*, and $\alpha_t(i)$ and $\beta_t(i)$ are termed as *forward variable* and *backward variable*, respectively.

Now, suppose $\delta_t(i)$ is the probability of the system being in the true state x_i at time instance t given the complete observation sequence $\langle Y_1, Y_2, ..., Y_T \rangle$ and the current model θ. We can define this probability in terms of the forward and backward variables $\alpha_t(i)$ and $\beta_t(i)$, i.e.,

$$\begin{aligned} \delta_t(i) &= Pr(X_t = x_i | Y_1, Y_2, ..., Y_T, \theta) \\ &= \frac{Pr(X_t = x_i, Y_1, Y_2, ..., Y_T | \theta)}{Pr(Y_1, Y_2, ..., Y_T | \theta)} \\ &= \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)} \end{aligned} \tag{8}$$

Again, given the complete observation sequence $\langle Y_1, Y_2, ..., Y_T \rangle$ and the current model θ, suppose, $\xi_t(i, j)$ is the probability of the system being in the true states x_i and x_j at time instances t and $t+1$, respectively. So,

$$\begin{aligned} \xi_t(i, j) &= Pr(X_t = x_i, X_{t+1} = x_j | Y_1, Y_2, ..., Y_T, \theta) \\ &= \frac{Pr(X_t = x_i, X_{t+1} = x_j, Y_1, Y_2, ..., Y_T | \theta)}{Pr(Y_1, Y_2, ..., Y_T | \theta)} \\ &= \frac{\alpha_t(i) q_{ij} \beta_{t+1}(j) \mu_j(y_{t+1})}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) q_{ij} \beta_{t+1}(j) \mu_j(y_{t+1})} \end{aligned} \tag{9}$$

Now, we can update the initial state distribution $\bar{\sigma}_i$, transition probability $\bar{q}_{ij}$, and emission probability $\bar{\mu}_j(y_k)$ using these two parameters $\delta_t(i)$ and $\xi_t(i,j)$. The state distribution can be updated as:

$$\bar{\sigma}_i = \delta_1(i) \tag{10}$$

where, $\delta_1(i)$ is the expected number of times the system is in the true state x_i at time instance $t = 1$.

To update the transition probabilities, we have to compute the ratio of *the expected number of state transitions from x_i to only x_j* (the numerator of the equation (11)) and *the expected number of transitions from x_i to all other true states* (the denominator of the equation (11)).

$$\bar{q}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \delta_t(i)} \tag{11}$$

And to update the emission probabilities, we have to take the ratio of two other quantities: *the expected number of times being in state x_j and observing the observation y_k* (the numerator of the equation (12)), and *the expected number of times being in state x_j* (the denominator of the equation (12)).

$$\bar{\mu}_j(k) = \frac{\sum_{t=1}^{T} 1_{(Y_t=y_k)} \delta_t(j)}{\sum_{t=1}^{T} \delta_t(j)} \tag{12}$$

where,

$$1_{(Y_t=y_k)} = \begin{cases} 1, & \text{if } Y_t = y_k \\ 0, & \text{Otherwise} \end{cases} \tag{13}$$

The updated parameters $\bar{\sigma} = \{\bar{\sigma}_i\}$, $\bar{Q} = \{\bar{q}_{ij}\}$, and $\bar{E} = \{\bar{\mu}_j(y_k)\}$ now constitute the new model $\bar{\theta} = (\bar{\sigma}, \bar{Q}, \bar{E})$. We need to iterate the equations (10) (11), and (12) until we find $\bar{\theta} \approx \theta$. This convergence is guaranteed in [9] by Baum et al., where it is ensured that either 1) the initial model θ defines a critical point in the likelihood function where $\bar{\theta} = \theta$, or 2) $\bar{\theta}$ explains the observation sequence $\langle Y_1, Y_2, ..., Y_T \rangle$ more suitably than θ, i.e. $Pr(Y_1, Y_2, ..., Y_T | \bar{\theta}) > Pr(Y_1, Y_2, ..., Y_T | \theta)$ [11].

b) **Sequence Retriever**: Once the probability estimator determines the converged HMM model θ^*, now, it is job for the *Sequence Retriever* to extract the optimal sequence of hidden events using Viterbi algorithm [10]. Given a particular observation sequence $\langle Y_1, Y_2, ..., Y_t \rangle$ at time instance t and $Y_t = y_k$, the goal here is to determine the following:

$$\begin{aligned} \omega_t(i) &= \underset{x_1,\ldots,x_{i-1}}{max} \Big\{ Pr(X_1 = x_1, ..., X_t = x_i, Y_1, ..., Y_t = y_k | \theta) \Big\} \\ &= \underset{x_1,x_2,\ldots,x_{i-2}}{max} \left\{ \underset{x_{i-1}}{max} \Big\{ \omega_{t-1}(i-1) q_{(i-1)(i)} \Big\} \mu_t(y_k) \right\}, \\ &\qquad\qquad 2 \leq t \leq T,\ 1 \leq i \leq N \end{aligned} \tag{14}$$

Hence, $\omega_t(i)$ represents the maximum probability of the occurrence of a particular state sequence $\langle x_1, x_2, ..., x_i \rangle$ at time instance t that corresponds to the aforementioned observation sequence $\langle Y_1, Y_2, ..., Y_t \rangle$.

The equation (14) can be solved recursively to determine the highest probability of the occurrence of a complete state sequence $\langle x_1, x_2, ..., x_N \rangle$ for the time instance $2 \leq t \leq T$ given that $\omega_1(i) = \sigma_i \mu_i(y_1)$. The recursion stops after computing $\omega_T(i)$ such as:

$$\omega_T^* = \underset{1 \leq i \leq N}{max}\ \omega_T(i) \tag{15}$$

But to obtain the optimal hidden sequence, we must trace the arguments that maximize the equation (14) during each recursion. To achieve that, we introduce a variable χ to hold all the traces such as:

$$\chi_t(i) = \underset{1 \leq i \leq N}{argmax} \Big\{ \omega_{t-1}(i-1) q_{(i-1)(i)} \Big\}, 2 \leq t \leq T, 1 \leq i \leq N \tag{16}$$

Note that $\chi_1(i) = 0$ for $t = 1$ because we start tracing the states for the very first time at time instance $t = 2$ once we have at least one previous state.

Once we have $\chi_T(i)$, all we need is backtracking through the traces to discern the optimal hidden sequence such as:

$$\psi_t^* = \chi_{t+1}(\psi_{t+1}^*),\ t = T-1, ..., 2, 1 \tag{17}$$

Hence, $\psi_T^*(i) = \chi_T(i)$, and $\Upsilon = \{\psi_1^*, \psi_2^*, ..., \psi_T^*\}$ is the extracted optimal sequence. Note that each $\psi_t^* \in \Upsilon$ represents a true state in X.

3) ***Crucial Node Detector****:* After the *sequence retriever* extracts the hidden optimal sequence $\Upsilon = \{\psi_1^*, \psi_2^*, ..., \psi_T^*\}$, the component *crucial node detector* applies Algorithm 1 to detect the crucial events in the attack chain the attacker must compromise to successfully implement the attack. Hence, the most frequently triggered events are defined as *crucial events*.

If there are p number of different extracted sequences $\Upsilon_1, \Upsilon_2, ..., \Upsilon_p$ for p different attempts, the Algorithm 1 first determines the *longest common subsequence S_i* between each Υ_i and the original sequence $X = \{x_1, x_2, ..., x_N\}$. Later, it computes the *SCORE* value for each pair of states in the subsequence such as:

$$\begin{aligned} &SCORE\Big[S_i[j], S_i[j+1]\Big] = \text{number of times a pair} \\ &\Big\{S_i[j], S_i[j+1]\Big\} \text{ is present in the subsequence} \end{aligned} \tag{18}$$

Algorithm 1: Crucial node detection algorithm

```
Input: $X, \Upsilon_1, \Upsilon_2, ...., \Upsilon_p$
Output: Pairs of true states responding to the most frequently
        triggered events
$i \leftarrow 1$
while $i \leq p$ do
    $S_i \leftarrow$ LCS between $X$ and $\Upsilon_i$ // LCS = Longest Common
    Subsequence
    for $j \leftarrow 1$ to $(|S_i| - 1)$ do
        $E[i,j] \leftarrow \{S_i[j], S_i[j+1]\}$
        if $E[i,j]$ not in $SCORE.Keys()$ then
            $SCORE[E[i,j]] \leftarrow 1$
        else
            $SCORE[E[i,j]] \leftarrow SCORE[E[i,j]] + 1$
return $\underset{E[i,j]}{argmax}\ (SCORE[E[i,j]])$
```

Finally, the algorithm updates the *SCORE* values based on the presence of pairs in all subsequences and retrieves the pairs with the maximum *SCORE* value. It may output a number of pairs of states, such as $\{x_{c_i}, x_{c_j}\}$, where there is a crucial state transition in the state machine from x_{c_i} to x_{c_j}. Our goal is to identify the events (we call them *nodes*) associated with such transitions that are exploited by the attackers to compromise the chain.

A Simple Example

Suppose, there is a sequence of states (responding to some triggered events): {**door-opened, light-on, camera-on,**

fan-on, window-opened}. And after making three separate attempts, the sequence retriever returns the following three sequences:

Sequence-1: {door-opened, light-on, light-on, camera-on, fan-on}, Sequence-2: {fan-on, light-on, camera-on, fan-on, window-opened}, Sequence-3: {door-opened, light-on, camera-on, window-opened, fan-on}.

Now, if we apply Algorithm 1 on this scenario, we find that the pair {**light-on, camera-on**} obtains the highest score. Consequently, we can conclude that the transition from the state **light-on** to **camera-on** is the most vital one in the state machine, and the nodes associated with those states are the most crucial ones in the chain. IoTMonitor identifies these crucial nodes so that we can perform security hardening to minimize the attacker's chance of compromising an IoT network. The security hardening part is out of the scope of this paper, and we plan to incorporate such capability in the extended version of IoTMonitor in recent future.

IV. Results and Evaluation

To evaluate the performance of IoTMonitor, we utilize the PEEVES dataset [7] that records IoT event occurrences from 12 different IoT devices and sensors measurements from 48 deployed sensors to verify those events. We use 24-hours data for our experiment, and our experiment executes on a 16 GB RAM and 4 CPU core system.

A. Dataset Processing

Our experiment mainly deals with three types of data: 1) event data (used as true states) 2) sensor measurements (used as observations), and 3) timestamps. We concentrate only on those event occurrences which can be verified by the sensor measurements. Since sensor measurements here capture the physical changes that have happened in the environment due to the event occurrences, they can be used to crosscheck whether a certain event has occurred. We conceptualize the function *sliding window* to determine whether an event is verifiable. Hence, the function provides us with a time window (in milliseconds) w_i that starts at the timestamp of a particular event occurrence. After an event occurrence is recorded at time instance t_i, if we find the necessary sensor measurements to verify that occurrence within the time instance $t_i + w_i$, we consider that event verifiable and keep it in the sequence of events occurred. Otherwise, we discard it from the sequence. In our experiment, we consider 20 such sliding windows with the size between 105 milliseconds and 200 milliseconds with an increase of 5 milliseconds.

B. Experiment Setting

At the beginning of our experiment, we choose Gaussian distribution to randomly assign the transition probabilities and initial state probabilities for each true state. On the other hand, we use Dirichlet distribution to assign the emission probabilities. We use the same seed value for each execution.

C. Probability Estimation Time

Probability estimation time represents the time required to estimate the converged transition probability distribution Q and the emission probability distribution E. Figure 3(a) presents the estimation time for four different sequences of events of different lengths (5, 10, 15, and 20) against a range of sliding windows. In the figure we show the average estimation time after 10 executions.

As we can see from Figure 3(a), the longest estimation time is < 4 seconds for the sequence length of 20, while in most cases, it is < 0.5 seconds. As the window size increases, the estimation time starts to decrease and stabilize. There are a few exceptional cases where the estimation time increases sharply for a increase in window size. For example, when the window size increases from 105 to 110 for the sequence of length 20, we see a sudden spike. We examine the source data and find that this spike is caused by the appearance of two new events that were not present earlier. Since the number of unique events increases and repetition of same events decreases in the sequence, the initial state distribution and transition probabilities are needed to be adjusted which costs adversely to the total estimation time. However, this type of exception is transient, and the graph stabilizes eventually. We do not present the estimation time for the sequences of lengths > 20 in the Figure 3(a) since we observe very little change in pattern for those sequences.

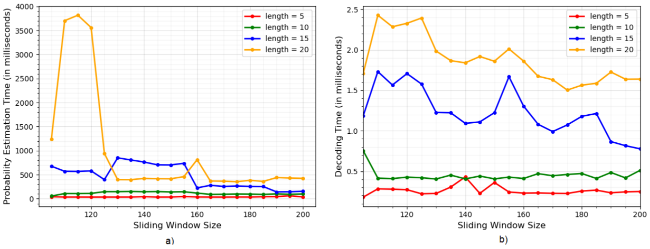

Fig. 3. a) Probability estimation time with respect to sliding window size and length of the event sequence; b) Decoding time with respect to sliding window size and length of the event sequence

D. Decoding Time

Decoding time represents the time required to extract the hidden sequence when we have the converged θ^*. Similar to probability estimation time, we take average decoding time after 10 executions. The Figure 3(b) presents the decoding time for four different sequences of events with lengths 5, 10, 15, and 20 against a range of sliding windows.

If we look at the graph at Figure 3(b), we see that the decoding time decreases when the window size increases. The longest decoding time we get is < 2.5 milliseconds which is very fast for the retrieval of hidden event sequences. Although we see few little temporary spikes for the length 15 after sliding window 150, we still achieve < 2.0 milliseconds as the decoding time.

E. Computational Overhead

Since our experiment dedicates most of the computation time to estimate the probabilities, we measure *computational*

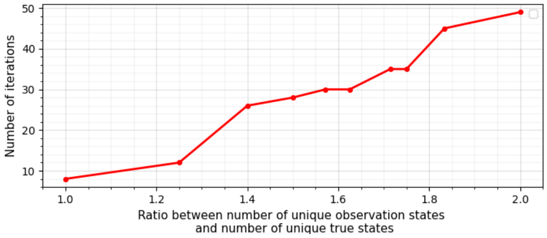

Fig. 4. Number of iterations to estimate the converged transition probabilities and emission probabilities with respect to the ratio between number of observation states and number of true states

overhead as the total number of iterations of the *forward-backward procedure* required to reach the convergence of transition probabilities and emission probabilities. In Figure 4, we present the required total number of iterations (in y-axis) with respect to the ratio between *the total number of unique observation states* and *the total number of unique true states* (in x-axis). We can see that, the computational overhead increases roughly linearly with the ratio.

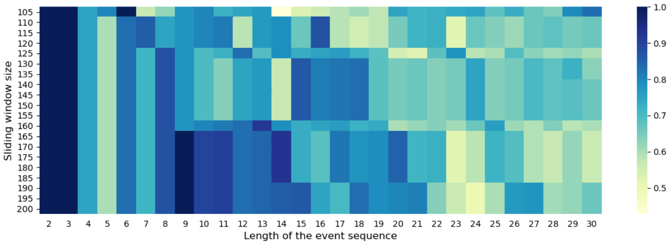

Fig. 5. Accuracy score vs Sliding window size vs Length of the event sequence

F. Accuracy Score

To determine how accurately the extracted hidden sequence of events represent the real events, we compute f-score for 29 different sequence of events starting with the length 2 and ending at length 30. We do not consider the sequence with length 1 because it does not offer any uncertainty in terms of transition and emission probability. We present a heatmap to visually show the correlation among accuracy score, sliding window size and length of the event sequence. In Figure 5, the accuracy scores are presented as colors.

As we can see, when the length of event sequence is < 15, the increase in window size after 160 assures a very high accuracy score. We even get the accuracy score of 1.0 in some occasions. There is only one exception for the sequence of length 5. Hence, we see a decrease in accuracy score after the window size 105, and that's because we see a completely new sequence for the window sizes 110 to 200. Similar pattern also arises, although to a less extent, for the sequence of length 7. But it is quite evident that the increase in window size for the smaller lengths ensures higher accuracy score (equals or close to 1.0). When the length increases to a considerable extent, we start to see the impact of sliding windows on the accuracy score diminishing slowly. Since our system emphasizes on the functional dependencies (in terms of transition probability) of the events to extract the hidden sequence, the longer the sequence becomes, the looser are the dependencies.

V. Conclusion

In this research work, we propose IoTMonitor that focuses on the extraction of the underlying event sequence using HMM approach given a set of physical evidence emitted during a trigger-action based attack in an IoT environment. We use the Baum Welch algorithm to estimate transition and emission probabilities, and Viterbi algorithm to extract the underlying event sequence. Our experiments show that both probability estimation and sequence extraction operations converge reasonably fast. In terms of accuracy score, IoTMonitor achieves 100% in multiple cases and $\geq$ 90% in a number of cases. We draw a heatmap to visually show the correlation among accuracy score, sliding windows, and length of the event sequences. We also present an algorithm to identify the crucial events in the extracted sequence which the attackers wish to compromise to implement a trigger-action attack.

Immediate extensions to our approach include the following efforts. First, we currently focus on the crucial nodes that appear in multiple attack paths. If we extend our research to an attack graph, we can identify crucial node pairs on different attack paths. Second, the physical evidence collected by sensors could contain noises or even inaccurate data. We will improve our algorithm to provide more robust attack detection capability for IoT platforms.

References

[1] "Ifttt: Every thing works better together," https://ifttt.com/, accessed: 2020-08-21.

[2] Z. B. Celik, G. Tan, and P. Mcdaniel, "IOTGUARD : Dynamic Enforcement of Security and Safety Policy in Commodity IoT," no. February, 2019.

[3] M. M. Alam and W. Wang, "A comprehensive survey on data provenance: State-of-the-art approaches and their deployments for iot security enforcement," *Journal of Computer Security*, vol. 29, pp. 423–446, 06 2021.

[4] D. T. Nguyen, C. Song, Z. Qian, and S. V. Krishnamurthy, "IotSan: Fortifying the Safety of IoT Systems Dang," *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*, pp. 387–400, 2018.

[5] L. Babun, A. K. Sikder, A. Acar, and A. S. Uluagac, "Iotdots: A digital forensics framework for smart environments," *CoRR*, vol. abs/1809.00745, 2018. [Online]. Available: http://arxiv.org/abs/1809.00745

[6] Z. B. Celik, P. Mcdaniel, and G. Tan, "Soteria: Automated iot safety and security analysis," in *USENIX Annual Technical Conference*, 2018.

[7] S. Birnbach, S. Eberz, and I. Martinovic, "Peeves: Physical event verification in smart homes," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2019.

[8] L. B. Baum and J. A. Eagon, "An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology," *Bulletin of the American Mathematical Society*, vol. 73, no. 3, pp. 360–363, 1967.

[9] L. E. Baum and G. R. Sell, "Growth transformations for functions on manifolds," *Pacific Journal of Mathematics*, vol. 27, no. 2, pp. 211–227, 1968.

[10] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[11] L. R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.