SAPIENZA
UNIVERSITÀ DI ROMA

Using Sensor and Process Noise Fingerprint to Detect Cyber Attacks in CPS

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica

Corso di Laurea in Informatica

Candidato

Andrei Laurentiu Lepadat
Matricola 1677093

Relatore

Prof. Enrico Tronci

Anno Accademico 2020/2021

Tesi non ancora discussa

Using Sensor and Process Noise Fingerprint to Detect Cyber Attacks in CPS
Tesi di Laurea. Sapienza – Università di Roma

Questa tesi è stata composta con LaTeX e la classe Sapthesis.

Versione: 24 novembre 2021

Email dell'autore: lepadat.1677093@studenti.uniroma1.it

Decidere se inserire. Ne vale la pena?

Indice

Sommario

1. Introduzione

1.1 Contesto

1.2 Motivazioni

1.3 Contributi

1.4 Stato dell'arte

1.5 Struttura

2. Background

Ogni sistema cyber-fisico che si rispetti è dotato di almeno un sensore che ha il compito di misurare una determinata "qualità" fisica di interesse per il sistema stesso. I dati che vengono rilevati dai sensori spesso vengono memorizzati localmente e/o in modo remoto e possono essere impiegati, come nel lavoro qui presentato, per fini paralleli o trasversali a quelli per cui sono stati installati. Una sequenza di dati estratti da sensori ordinata temporalmente viene chiamata *serie temporale* (*time-series* in inglese).

Comunemente i sensori sono imperfetti per costruzione e trasportano intrinsecamente un'incertezza (*process noise*) che influenza le misurazioni da essi compiute. Sia

$$\bar{y}_k = y_k + \delta_k \tag{2.1}$$

il valore misurato da un determinato sensore nell'istante di tempo k, composto da y_k, il valore effettivo in quell'istante della grandezza misurata, più δ_k, il rumore aggiunto.

In un determinato istante di tempo, il valore di ogni sensore del sistema costituisce lo *stato* del sistema. La sfida di estrarre il fingerprint dai sensori è data dal fatto che questi stati sono dinamici. Prendendo in considerazione, per esempio, un termometro, se la temperatura dell'ambiente che misura rimane costante nel tempo è facile estrarre il fingerprint del rumore e costruirne il profilo, ma in processi reali non è così semplice, gli stati cambiano continuamente, per esempio l'aumento di velocità di una macchina per via della pressione sul pedale dell'acceleratore. È importante catturare queste variazioni affinché le misurazioni dinamiche dei sensori possano essere stimate. In [1] questo problema viene affrontato definendo un modello analitico del sistema interessato, rappresentato tramite il modello *State-Space*. Vengono implementate le tecniche definite in [2], definendo così il modello lineare tempo inviariante (LTI) del sistema, rappresentato dal sistema di equazioni

$$\begin{cases} x_{k+1} = Ax_k + Bu_k + \vartheta_k, \\ y_k = Cx_k + \eta_k : \end{cases} \tag{2.2}$$

in cui $x_k \in \mathbb{R}^n$ rappresenta lo stato del sistema, $u_k \in \mathbb{R}^p$ l'input di controllo e ϑ_k il rumore al tempo k. $y_k \in \mathbb{R}_m$ e $\eta_k \in \mathbb{R}_m$ rappresentano, rispettivamente, la misurazione e il rumore del sensore al tempo k. A, B, C sono le matrici dello spazio di stato di dimensioni adeguate che rappresentato la dinamica del sistema.

Definito il precedente sistema, ci sono molti punti che un attaccante mal intenzionato potrebbe bersagliare. Nel lavoro presentato, così come in [1], vengono presi in considerazione *spoofing attack* ai sensori che potrebbero essere portati a termine

tramite uno schema *Man-in-The-Middle*. L'equazione lineare che rappresenta questa tipologia di attacchi è data da

$$\bar{y}_k = y_k + \delta_k = Cx_k + \eta_k + \delta_k,^1 \tag{2.3}$$

in cui $\delta_k \in \mathbb{R}_m$ rappresenta un attacco ai sensori.

In [1], dato l'output $\bar{y}_k$, viene adoperato il *filtro di Kalman* per stimare lo stato del sistema e il vettore dei *residui*, definito, in questo contesto, come la differenza tra la reale misurazine effettuata dal sensore e la stima della misurazione calcolata dal filtro nell'istante k:

$$r_k := \bar{y}_k - \hat{y}_k, \tag{2.4}$$

dove $\hat{y}_k$ è l'output del filtro di Kalman.

Detto ciò, per quantificare la bontà del modello del sistema, viene utilizzato l'*Errore Quadratico Medio (RMSE)*, definito come

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}. \tag{2.5}$$

Questa metrica rappresenta la distanza tra il valore stimato e quello misurato, ovvero quanto il primo è lontano dal secondo. Nella letteratura della teoria del controllo, modelli con un'accuratezza[2] superiore al 70% sono considerati accettabili approssimazioni della dinamica di sistemi reali.

Per ogni momento statistico (media, deviazione standard, ...) di una serie storica (ma non solo) si può definire un *intervallo di confidenza* che esprime la probabilità che il valore calcolato sugli N campioni della serie approssimi il valore effettivo del momento statistico. Questo intevallo, nel caso del valore medio, si definisce come

$$Pr\{\bar{x} - \epsilon \leq \mu \leq \bar{x} + \epsilon\} = 1 - \delta, \tag{2.6}$$

in cui μ e $\bar{x}$ sono, rispettivamente, la media effettiva e quella calcolata. ϵ e δ sono valori che dipendono da N, e mantenendo δ costante e incrementando N, anche ϵ cresce, allargando l'intervallo di confidenza. Tale intervallo può essere definito anche per momenti di ordine superiore.

Nel contesto del presente lavoro, come si vedrà, volendo giudicare la legittimità delle misurazioni di un determinato sensore, determinate proprietà statistiche delle nuove misurazioni (nuove nel contesto di normale funzionamento del sistema *aperto* ad attacchi) verranno confrontate con le stesse proprietà di misurazioni effettuate in condizioni *sicure* (questi valori sono chiamati valori di *reference*). Per le nuove misurazioni, prendendo ancora in esempio il valore medio e volendo avere un intervallo di confidenza il più piccolo possibile (quindi un ϵ il più piccolo possibile), bisogna essere attenti per via di valori di N non molto grandi, caratteristica preferibile in quanto non si vogliono campionare troppi valori in situazioni real-time (equivarrebbe ad aspettare di più per prendere una decisione, e quindi essere potenzialmente per più tempo sotto attacco). Scegliere un'intervallo di condifenza delle nuove

[1] Notare l'ugualianza con l'equazione 2.1: un attacco è considerato come un'introduzione di rumore nella misurazione fatta da un sensore.

[2] Se l'RMSE rappresenta un errore, l'accuratezza viene calcolata come 100 - RMSE.

Feature	Descrizione
Media	$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$
Varianza	$\sigma = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2$
Dev. Med. Ass.	$D_{\bar{x}} = \frac{1}{N}\sum_{i=1}^{N}\lvert x_i - \bar{x}\rvert$
Asimmetria	$\gamma = \frac{1}{N}\sum_{i=1}^{N}(\frac{x_i-\bar{x}}{\sigma})^3$
Curtosi	$\beta = \frac{1}{N}\sum_{i=1}^{N}(\frac{x_i-\bar{x}}{\sigma})^4 - 3$

Tabella 2.1. Lista delle feature utilizzate; x è la serie temporale di dimensione N proveniente dal sensore.

misurazioni troppo grande potrebbe portare ad una sovrapposizione tra il nuovo intervallo ed quello di reference[3]. Per avere una sensibilità migliore contro gli attacchi il problema viene affrontato avvalendosi dell'aiuto di un determintao modello di *Machine Learning*, che ha un buon comprtamento verso valori che non sono perfettamente discriminabili.

A tale scopo viene definito un problema di M.L. che ha come *feature* alcuni valori statistici (di cui si è parlato precedentemente) estratti dai vettori residui[4]. Queste feature sono mostrate nella Tabella 2.1.

Un problema di M.L. può essere definito come una funzione

$$f : X \to Y, \tag{2.7}$$

dato un insieme D (dataset) contenente informazioni riguardanti f. Fare il *learning* della funzione f significa trovare un'altra funzione $\hat{f}$ che approssima e ritorna valori più vicini possibile ad f, specialmente per elementi non presenti in D. Nel presente lavoro il problema viene definito come un problema di *classificazione*, ovvero in cui f è definita tale che

$$\begin{aligned} X &:= \mathbb{R}^m, \\ Y &:= \{C_1, C_2, \ldots, C_k\}; \end{aligned} \tag{2.8}$$

e *supervised*, cioé in cui il datasert è del tipo

$$D = \{(x_i, y_i)_{i=1}^{N}\}, x_i \in X, y_i \in Y \tag{2.9}$$

Quindi f associa ad ogni elemento di X, cioè un vettore di m reali, un elemento di Y, cioè la classe C_i di appartenenza.

[3]Il caso ideale sarebbe o di inclusione del primo nel secondo o di disgiunzione, rilevando nel primo caso un valore ammesso e nel secondo caso un attacco.

[4]Il vettore dei residui è quindi parte fondamentale per la definizione dei fingerprint dei sensori.

3. Metodi

Come anticipato nella precedente sezione, il prossimo passo è quello di definire correttamente il problema di M.L. ed utilizzare un algoritmo di learning adeguato. A questo scopo bisogna preparare i dati estratti dai sensori (e di conseguenza il vettore dei residui) per poter essere utilizzati efficacemente dall'algoritmo di learning. Il dataset conterrà coppie in cui il primo elemento x_i è un vettore le feature elencate nella Tabella 2.1, e il secondo elemento y_i è la classe di apparteneza che rappresenta il sensore che ha prodotto le misurazioni utilizzate per costruire x_i. Data la natura statistica delle feature utilizzate, sarebbe inadeguato estrarre le stesse basandosi solamente sulla misurazione avvenuto all'istante di tempo k. Detto ciò, bisogna partizionare ogni serie temporale in blocchetti (*chunk*) di dimensione d. Da ogni chunk verranno poi estratte le feature precedentemente menzionate ed etichettate con la classe (quindi sensore) di appartenenza. Scegliere la giusta dimensione d dei vari chunk è una scelta sensibile in quanto forzerà anche la dimensione dei chunk composti da residui di sensori che verranno calcolati durante il normale funzionamento del sistema. L'idea è di avere chunk di serie temporali abbastanza grandi da catturare la dinamica del processo e piccoli abbastanza da ridurre il tempo di attesa per discriminare nuove misurazioni.

Tutte le istanze di X che appartengono allo stesso sensore determinano il fingerprint del sensore e il primo compito della funzione f (Funzione 2.7) è quello di distinguere correttamente i sensori, quindi generare un fingerprint il più preciso e privo di ambiguità possibile. Nel Capitolo 5, l'esistenza del fingerprint verrà provata empiricamente.

L'algoritmo di M.L. utilizzato è il *Support Vector Machine* (SVM), impiegato come un classificatore a 2 classi (One vs. One) che etichetta i dati provenienti dal legittimo sensore (*ground truth*) come un *1* e come *0* i dati provenienti dagli altri sensori. In questo modo gli attacchi possono essere considerati come un tipo dato appartenente ad altre classi.

L'algoritmo SVM verrà allenato su un campione di dati che dipenderà dal sistema considerato e verrà validato usando la tecnica del *k-fold cross validation*. Successivamente, siccome nel dataset non sono presenti dati relativi ad attacchi, il classificatore a 2 classi potrebbe mancare alcuni degli attacchi.

Per affrontare questo problema viene utilizzato la modalità di SVM ad una classe (*one-class* SVM) per fare il learning del modello di M.L., basandosi su una classe di dati normali (appartenenti quindi ad un solo sensore). Nella fase di testing tutto ciò reputato estraneo viene considerato come un attacco.

4. Implementazione

Alla luce di quanto detto nel Capitolo 2 per quanto riguarda l'estrazione dei vettori dei residui, nel presente lavoro viene presa una strada diversa da quella presentata in [1].

Viene utilizzato un modello di sistema dinamico ideale (che non introduce alcun tipo di rumore nelle misurazioni dei sensori) e viene inserito il process noise "artificialmente", sotto forma di rumore gaussiano.

Sia y_k l'output di un sistema del tipo definito in 2.2 all'istante k e siano $\mathcal{X}_j \sim \mathcal{N}(0, \sigma_j^2)$, $j = 1, \ldots, m$[1], m (una per sensore) distribuzioni di probabilità gaussiane con media nulla e varianza σ_j^2. Il valore in output del sistema dopo l'introduzione di rumore è uguale a

$$\bar{y}_k = y_k + \delta_k, \tag{4.1}$$

dove δ_k è il vettore che contiene le m estrazioni secondo $\mathcal{X}_j$ all'istante k.

Il vettore dei residui viene quindi ottenuto eseguendo la sottrazione vettoriale $r_j = \bar{y}_j - y_j$, in cui y_j (la serie storica, ovvero la collezione, delle misurazioni relative al sensore j all'istante n) agisce come il valore stimato secondo le dinamiche del sistema. Dal momento che ogni r_j deve contenere valori di un sistema reale sottoposto a process noise, le varianze σ_j^2 devono essere tali che le distribuzioni $\mathcal{X}_j$ generino valori per $j = 1, \ldots, m$, dalla 2.5, tali che

$$100 - \sqrt{\frac{\sum_{k=1}^{n}(\bar{y}_{j,k} - y_{j,k})^2}{n}} \geq D_j, \forall j \in \{1, \ldots, n\}. \tag{4.2}$$

D_j è un valore sempre maggiore o uguale di 70 e, in base al ruolo del sensore all'interno del sistema, può assumere valori più o meno grandi. Sostituendo dall'Equazione 4.1, si nota che l'RMSE per ogni sensore j dipende banalmente dalla serie di valori estratti secondo $\mathcal{X}_j$.

Calcolati i vettori dei residui e preparati (divisi in chunk ed estratte le feature) per essere processati dall'algoritmo SVM, e fatto il learning del modello di M.L., quest'ultimo viene validato[2] tramite la k-fold cross validation con $k = 2, \ldots, 15$. Questo significa, per valori di k crescenti, il dataset D (per come è stato definito nel Capitolo 3) viene suddiviso in k partizioni e di queste, $k - 1$ vengono utilizzate per fare il learning dell'algoritmo e una per fare il testing. In questo modo si cerca

[1] m è la dimensione di y_k, uguale al numero di sensori del sistema.

[2] Questa operazione, con un k variabile, ha una duplice funzionalità: quella di validare il modello di M.L. trovato e quella di capire quanto il modello è sensibile allla quantità di dati utilizzati nella fase di learning.

di evitare problemi di sovradattamento, tipico della suddivisione del dataset in due partizioni. Per ogni k, l'accuratezza finale del modello è data dalla media delle accuratezze calcolate durante le k fasti di test. Dato che si vuole un modello di M.L. robusto nella dimensione del dataset, l'accuratezza di ognuna delle $k-1$ ($k = 2, \ldots, 15$) esecuzioni della k-fold cross validation viene confrontata con le altre, e minore la variazione maggiore la robustezza del modello.

5. Risultati sperimentali

Descritte le basi teoriche e metodologiche prese in considerazione, queste sono state implementate e testate per via software. In questo capitolo vengono delineati gli obiettivi della fase sperimentale, il software utilizzato per simulare i sistemi considerati e che quindi produce i valori di output che sono punto centrale di questo lavoro. Vengono presentati di conseguenza due modelli utilizzati come banco di prova per le tecniche descritte nel Capitolo 3.

5.1 Obiettivi

Gli obiettivi principali di questa fase sono molteplici.

- Provare l'esistenza del fingerprint relativo ai sensori e quindi della distinguibilità dei dati provenienti da sensori diversi.
- Trovare la giusta dimensione dei chunk in cui saranno divisi i vettori dei residui.
- Mostrare che la quantità di dati di sensori acquisita non incide particolarmente sul fingerprint.
- Definire la tipologia di attacchi che possono essere identificati e quindi valutare la precisione con cui vengono riconosciuti.

5.2 Configurazione

Il software che implementa le funzionalità qui presentate è stato scritto utilizzando diversi linguaggi di programmazione, tra cui principalmente Python e MATLAB. Il software utilizzato per ottenere dati provenienti da simulazioni di sistemi cyber-fisici è Simulink, un software per la simulazione e la modellazione di sistemi dinamici.

5.3 Casi di studio

5.4 Correttezza

5.5 Valutazione computazionale

5.6 Valutazione tecnica

6. Conclusioni