Machine Learning Enhanced Real-Time Intrusion Detection Using Timing Information

Hang Xu and Frank Mueller, North Carolina State University, USA, hxu9, fmuelle@ncsu.edu
Mithun Acharya and Alok Kucheria, ABB Inc., USA, mithun.acharya, alok.kucheria@us.abb.com

***Abstract*—Past work has investigated intrusion detection mechanisms for real-time control devices. This work contributes a novel framework of separating security monitoring and detection from real-time control, where the former is performed on Cloud edge devices while the latter is run on embedded devices attached to the system that is controlled. We contribute a security monitoring system that validates worst-case timing bounds of the target controller and also validates its control outputs by comparing it against model-based predictions, which are derived from machine learning.**

I. Introduction and Motivation

As industrial and governmental cyber-physical systems (CPS) as well as personal and residential Internet-of-Things (IoT) devices have become ubiquitous, cyber attacks of these systems have also been rising [6]. Such attacks and intrusions usually pose severe risks to the controlled subsystems and may result in home intrusion, dangerous road scenarios, destruction of power grid or other industrial devices that may cause disruptions of operations, bodily injury or even loss of life. The cyber vulnerabilities of CPS and IoT devices lie in the exposure to public networks as required by their functionality and the relatively limited computational capability to deploy heavy-weight traditional security methods, e.g., public key cryptography. These factors motivate our work to significantly increase cyber security across CPS and IoT computing devices with sensitive controls.

II. Past Work and Contribution

In previous work [10], timing the execution paths of and "fingerprinting" the computation tasks in a normal state has been proved effective to identify intrusions on embedded systems. This work assumes the intrusion detection system (IDS) is deployed on the same host as the target CPS or IoT device and is not intruded or can only be intruded after the target is, which may not be the case in practice. In [3], the system states are monitored by a third party FPGA device, which switches into a safe controller mode if the system states are outside of safe thresholds. Such a security system may still be compromised when an attacker tampers with the communication packets to forge a normal state. [1] detects intrusion by monitoring encrypted packets, which cannot be easily deciphered in the communication channel but such encryption may not be feasible on low-end devices. [8] detects intrusion using a physical model but may not be applicable since some control processes cannot be mathematically transformed into models.

We present a novel intrusion detection approach that relies on fine-grained timing information of CPS or IoT devices enhanced by real-time machine learning (ML). **Novel contributions:**

- We separate the IDS from the target embedded system to increase isolation and decrease the attack surface of the detection system. More specifically, the target control system maintains state data, which is transmitted with sensor data to the IDS for verification.
- We use ML models to detect the packets whose data has potentially been tampered with.
- We use ML models for control systems, where the physical system model cannot easily be derived.
- We enhance ML inferencing with tighter and predictable upper bounds on execution time to facilitate a prompt response to intrusions.
- We use highly accurate ML models on an embedded system, exposing it to real industry sensor data from the field, to demonstrates suitability and efficiency for intrusion detection under the constrained resources of embedded systems.

III. A Real-Time Intrusion Detection Framework

Fig. 1 depicts a conceptual overview of our IDS (green/right box), which resides on a different hardware board than the target control system (blue/left box). The IDS communicates with and monitors the streaming telemetries sent from the target control system and external third party sources.

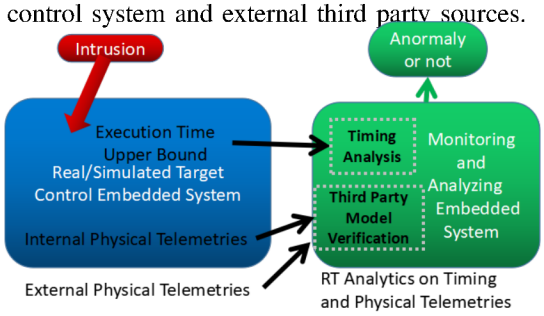

Fig. 1. System Diagram

Three categories of data packets are transmitted from the target control system to the detection system:

- *Execution Time Upper Bound:* We conduct timing analysis on the target control tasks to obtain tight upper bounds on the execution time for specific paths through the control code.
- *Internal Physical Telemetries:* We collect the sensed physical data from the controlled system, which could be the entire or a part of the target control system state.

• *External Physical Telemetries:* We collect the sensed physical data from a third party system with a sensor array and verify the validity of the target control system state.

To make tampering and omitting of the execution time data more difficult, we can anchor the execution time data management in OS kernel as one of the scheduler functionalities. Even if the OS kernel is compromised and the execution times plus system state are tempered with to circumvent detection, simply transmitting arbitrary system state values does not compromise the entire detection mechanism. Detection is not only based on the internal system states but also on their relation to and consistency with external physical states. For example, an intruder may record and replay the output and voltage current of an inverter, yet false power data can still be detected if it does not match the expected values given the current weather data at this geographic location.

Based on the above data, the IDS conducts two analyses, *Timing Analysis* and *Third Party Model Verification*. The former analysis checks the execution time against valid upper bounds obtained from prior experiments. If the error is larger than a certain threshold or the data packets are not received before their deadline, the system will report an anomaly. If the former analysis does not detect any anomaly, the latter analysis is conducted, which supplies the input physical telemetry data to a third party ML model and compares the measured physical telemetry with the expected outcome based on the ML model. The inputs to the model are selected based on the specific control application from internal and external physical telemetries. If the error is larger than a certain threshold, the system will report an anomaly. The pseudo code of intrusion detection is shown in Algorithm 1 using the model parameters of Table I.

The two analyses can be further fused to more effectively counter stealthy attack. A stealthy attack may evade the detection if it results in moderately suspicious execution times and moderately suspicious physical state telemetries. However, the IDS could still detect this attack by considering the execution times and telemetries together, and raising an alarm if both are moderately suspicious. Such fused detection will be based on weighting the thresholds of the timing analyses and the ML verification output and obtaining the summed overall detection threshold. How to weight and fuse the detection thresholds is beyond the scope of this paper. Such fused detection handles independent detection of one data sources as a special case, where other sources receive "zero" weights.

IV. Promptness

Our system addresses one of the most important requirements of IDS, *promptness*, as follows. One of our objectives is to guarantee prompt response to the intrusion through reducing the detection delay, i.e., the duration between the intrusion event and the detection of such intrusion. "Promptness" does not only imply a tight average timing bound of detection delay but also a tight upper timing bound, namely T_{detect}. The total detection delay is the aggregate of two terms:

TABLE I
Parameters of the Detection Algorithm

Symbol	*Description*
N	number of code snippets in the target control code
D_{comm}	deadline of communication delay
$WCET[N]$	worst-case execution time vector
TH_{ML}	ML model verification threshold
$socket$	controller socket file descriptor
$Data$	streaming data from controller
T_{dtc}	the data reception timestamp on the detector
T_{tgt}	the transmission timestamp on the controller
$T_{ctrl}[N]$	execution time vector on target control system
PHY_{in}	internal physical telemetries
PHY_{ex}	external physical telemetries
MSR_{in}	physical telemetries selected as measured inputs
MSR_{out}	physical telemetries selected as measured outputs
EXP_{out}	inference output of ML model

Algorithm 1 Intrusion Detection Algorithm

```
1: function DETECT_ANOMALY( socket )
2:     Data = read(socket);
3:     T_dtc = gettimeofday();
4:     if Data ≤ 0 then
5:         return True;                          ▷ packet not received
6:     else
7:         [T_tgt, T_ctrl[N], PHY_in, PHY_ex] = parse(Data);
8:         if T_dtc − T_tgt > D_comm then
9:             return True;          ▷ data packet not received in time
10:        else if ∃i, T_ctrl[i] > WCET[i], 0 ≤ i < N, i ∈ Z then
11:            return True;                ▷ execution time over bound
12:        else
13:            [MSR_in, MSR_out] = select_telemetry(PHY_in, PHY_ex)
14:            EXP_out = ML_Model(MSR_in)
15:            if ‖EXP_out − MSR_out‖ > TH_ML then
16:                return True;              ▷ ML verification failed
17:            else
18:                return False;                          ▷ No anomaly
19:            end if
20:        end if
21:    end if
22: end function
```

• The duration between the start of the intrusion and the time when the data packets are transmitted to and started to be processed by the detection system, namely T_{comm}.

• The duration between the start and the end of the analysis of the packet data on the detection system, namely T_{analy}.

Reducing the upper bound of the detection delay is not only important to guarantee prompt detection of an anomaly but is also essential to ensure fast data stream monitoring for the target control system.

The transmission interval for streaming data packets can be configured by the programmer to update the execution time bounds and system state to detect an anomaly in a timely manner. Ideally, such an interval is aligned to the control system sampling interval to constantly monitor all system timing bounds and physical states. In order to apply the ideal data transmission interval, the detection delay should be less than the ideal transmission interval. Since the communication delay is determined by the operation system and network medium, we have less margin to tune and optimize the communication delay compared to the analysis delay. Hence,

we use experiments to determine the communication delay and then focus on tightening the upper bound of analysis delays via code optimization. In future work, we plan to enhance the network stack and operating system kernels to further optimize on communication delay.

Communication Delay: Our system uses a communication delay deadline, D_{comm}, on the detection system to verify the validity of the first part of the detection delay, which is the aggregate of T_{proc}^{target} (network processing delay on the control system), T_{trans} (network transmission delay between control and detection systems), and T_{proc}^{detect} (system and network processing delay on the detection system. The network processing delay on both the target and detection systems can be bounded within 2ms [9]. Both systems are usually deployed in the same local area network (LAN) connected via Ethernet with a typical upper bound of delay $<$ 1ms for 100Mb wired Ethernet [2], which is better than any wireless delay [7].

We assume the system clocks are synchronized for the target control and detection systems using Network Time Protocol (NTP) or Precision Time Protocol (PTP) with a system clock error T_{error} bound of 18ms and tens of nanoseconds, respectively [4].

We ensure that the detector waits for and then timestamps the arrival of packets from the controller instead of leaving the arrived packet in the socket buffer and timestamping them later. By not buffering packets, we tighten the upper bound on the communication delay.

Assuming we use NTP and given the above timing bounds, we obtain a theoretical communication delay deadline $D_{comm} = \max T_{comm} + T_{error} = \max(T_{proc}^{target} + T_{trans} + T_{proc}^{detect}) + T_{error} = 2 + 1 + 2 + 18ms = 23ms$.

Analysis Delay: The analysis delay is the worst-case execution time (WCET) of the detection task, which is checking the validity of the execution time upper bound and the state of the control system using the ML model. In our experiment, we found that the WCET for the execution time checking code is significantly smaller than the WCET of the ML model checking code (see experimental section).

The WCET of the ML model checking code is dependent on the ML library deployed on the detection system. Since ML model verification is an inference task, we only consider the WCET of the inference API of the ML library. We select representative ML libraries to compare their suitability for a more predictable upper bound on the execution time of their inference APIs. We select Keras with a Tensorflow backend as the representative for an interpreter-based ML library and Caffe for a compilation-based library. In contrast to the Python interpreter-based Keras library, the inference code of Caffe is written in C++ and compiled into native code, which should in principle result in tighter upper bounds of execution time. Another significant advantage of Caffe over Keras is that it utilizes less memory than Keras and does not dynamically allocate/free any of it. Python's background garbage collector does not provide fine-grained real-time control and often perturbs the predictability of execution time of the ML tasks under Keras. The same is true for Python's reliance on an interpreter, which not only adds overhead for execution but also reduces predictability. (Notice that Python's libraries, such as numpy, often make calls to lower-level C or Cuda libraries for CPUs and GPUs, respectively, which results in better and more predictable performance on higher-end platforms, but not on embedded architectures such as the Raspberry Pi.)

Our experimental comparison shows that the average execution time of Keras's inference phase is about 4 times slower than that of the original Caffe code basis. However, the standard deviation of the execution time, which is directly related to execution time predictability, varies significantly for the original Caffe code distribution, i.e., it is occasionally two orders of magnitude larger and otherwise 4 times smaller than that of Keras. This somewhat surprising result shows that Keras outperforms the original Caffe code in performance and real-time predictability for the ML task of inferencing.

V. Accuracy

The accuracy of our system is evaluated by the confusion matrix, where the false negative (FN) rate indicates undetected anomalies and the false positive (FP) rate indicates normal state flagged as abnormal.

The overall system accuracy is affected by both the accuracy of execution time upper bound checking and the accuracy of ML model verification. We denote the FP rate and FN rate for the timing analysis and ML model verification as FP_{tm}, FP_{ML},FN_{tm} and FN_{ML} respectively. The derivation can be briefly described as follows. An FN detection of the overall system occurs when and only when an attack takes place to the controller but neither timing analysis nor ML verification detect such an intrusion. Thus, an FN event implies both timing analysis and ML verification failed, which is equivalent to multiplying the probabilities of these two independent detectors. In contrast, an FP occurs when there is no attack but either timing analysis or ML verification flag an anomaly, i.e., the union of FP events of the two detection methods. Since an anomaly flagged by timing analysis precedes ML verification, the FP event of the ML verification coincides with a true negative (TN) of timing analysis. Since $TN = 1 - FP$, the overall system detection FN, FN_{sys}, and false-positive, FP_{sys}, rates are:

$$FN_{sys} = FN_{tm} * FN_{ML} \quad (1)$$

$$FP_{sys} = FP_{tm} + (1 - FP_{tm}) * FP_{ML} \quad (2)$$

Here, we observe that FN_{sys} is reduced by a factor of $0 <= FN_{ML} <= 1$. Although an extra term, $(1-FP_{tm})*FP_{ML}$, is added to the overall system FP rate, detection still depends on FN and FP. In other words, a trade-off exits between the cost of reacting to false alarms and missing anomalies, where the latter exposes systems to greater risk. Such a study is beyond the scope of this paper. Instead, we focus on configuring the detection threshold of the timing analysis and the model verification for better overall accuracy of the detection system based on a pre-trained ML model.

VI. Experiment

We consider a practical industrial problem, where a green (solar) power generation source is secured. The core part of

a solar power system is the inverter, which controls power conversion and flow via an embedded micro-controller or DSP. We simulate the solar inverter's embedded system and the intrusion detector on two Raspberry pi B microcomputers, respectively, to assess accuracy and response time of the IDS.

The simulated inverter (controller) is a real-time application with two tasks, one of which reads data from a file to simulate sensing while the other sends packets containing the sensed data and execution time information to the detector. The detector is a real-time application with two tasks, one for reading and parsing the TCP message from the controller and the other for anomaly detection analysis described in pseudo code 1. To allow the anomaly detector to keep up with the data streaming rate of the controller and to consider the WCETs of all tasks on the controller and detector, we choose 25ms as the relative deadline for both tasks on the controller, and 40ms and 10ms for those of the data reading and anomaly detection tasks of the detector, respectively.

Via our partner, ABB Inc., we have access to field data collected from ABB's $UNO\_2.0\_2.5$ inverter and ABB's weather station $VSN800-14$ from 2014 to 2017 located at Kihei, Hawaii. This data set consists of electrical state data from the solar inverter and weather data from the weather station deployed $20km$ away from the solar inverter plant, both collected every five minutes. The electrical data reflects the inverter's output power generation, and the weather data includes the ambient temperature, the global horizontal irradiance (GHI) and plane of array irradiance (POAI) etc. at the weather station. In this experiment, the detector selects the weather information as ML model input, computes the predicted inverter output power as ML inference result and compares the result with the measured output power, which enhances anomaly detection beyond timing bounds checking. We ignore the data collection interval of 5mins and assume it aligns with the task period, 50ms, of the control system, since timing analysis and ML inference are performed offline.

Controller and detector are sharing the same wired LAN of a router to reduce the communication delay. The Raspberry Pis synchronize with the same NTP server. The worst-case communication delay was experimentally upper bounded at 2.7ms, aggregating network transmission delay, OS processing delay considering NTP synchronization error, which is notably much lower than $D_{comm} = 23ms$ (see Sect. V), likely due to different networking than used in previous work [4]. Due to clock drift, time-difference tables, dynamically updated as part of exchanged network packets, may be more suitable here [5]. Our target code snippet of the controller has a measured execution time of 10ms. We simulate additional execution due to malicious code via "sleeping" for by random intervals of $0-10$ms. For reproducibility, we set the seed of the random number generator. Based on the these parameters, we choose the detection threshold of the execution time to be 0.1%, 1%, and 10%, namely 0.01, 0.1 and 1ms. 80% of data samples are subject to intrusion. The other 20% feature timing deviation less than 0.01ms, which are not considered to be intrusions. We offline train our ANN model via the Caffe library with high inference accuracy in terms of variance, namely 0.891. Since the power output of the inverter is below 2kW, we configure the detection threshold for ML inference deviation to be 1%,5%, and 10% of 1kW, namely 10, 50, and 100W.

We conduct 9 groups of experiments with different detection thresholds of execution time and ML model output deviation, each group has 10 experiments with 2500 data samples per experiment. We average over the 10 groups and compare the confusion matrix of the overall detection system combining timing analysis and ML model anomaly detection with sole timing analysis checks (past work).

Fig. 2 shows that the system's FP rate decreases when detection thresholds of the WCET or the ML model increase. However, the FP rate of the overall system tends to increase and the accuracy (true positive plus true negative) tends to decrease when detection thresholds of the WCET or the ML model increase. This illustrates the trade-off between selecting appropriate detection thresholds as a means to optimize overall system efficacy considering the effects of FP and FN events for a specific application.

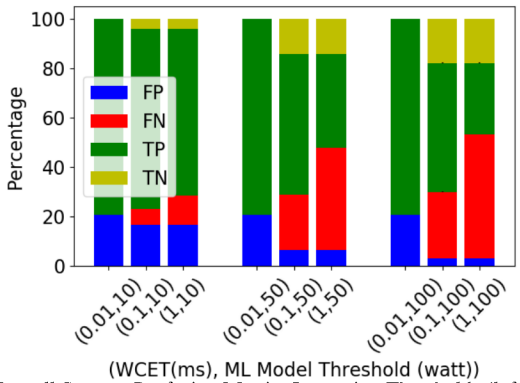

Fig. 2. Overall System Confusion Matrix, Increasing Thresholds (left to right)

Fig. 3 show that ML helps increase detection accuracy of the overall system by about 3%, especially when the execution time upper bound is not as tight (0.1 and 1ms).

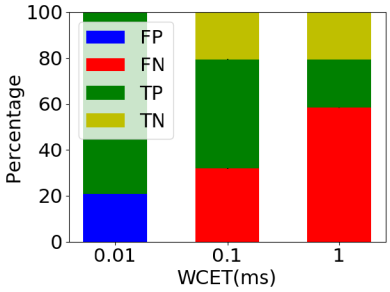

Fig. 3. Timing Analysis Stacked Bar Confusion Matrix

VII. Conclusion

We enhanced prior intrusion detection based on timing analysis via ML model verification and conducted experiments to demonstrate its effectiveness based on practical industrial data. We investigated the trade-off between FP and FN rates when selecting the detection thresholds of WCET and ML model output. Future work will capitalize on this trade-off and develop optimization techniques to further improve intrusion detection for CPS/IoT.

ACKNOWLEDGEMENT

This work was funded in part by NSF grants 1329780, 1813004.

REFERENCES

[1] Sachin P. Joglekar and Stephen R. Tate. Protomon: embedded monitors for cryptographic protocol intrusion detection and prevention. *International Conference on Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004.*, 1:81–88 Vol.1, 2004.

[2] Ming Li. Delay analysis of networked control systems based on 100 m switched ethernet. *TheScientificWorldJournal*, 2014:751491, 2014.

[3] Sibin Mohan, Stanley Bak, Emiliano Betti, Heechul Yun, Lui Sha, and Marco Caccamo. S3a: Secure system simplex architecture for enhanced security and robustness of cyber-physical systems. In *Proceedings of the 2Nd ACM International Conference on High Confidence Networked Systems*, HiCoNS '13, pages 65–74, New York, NY, USA, 2013. ACM.

[4] T. Neagoe, V. Cristea, and L. Banica. Ntp versus ptp in com puter networks clock synchronization. In *2006 IEEE International Symposium on Industrial Electronics*, volume 1, pages 317–362, July 2006.

[5] Tao Qian, Frank Mueller, and Yufeng Xin. Hybrid edf packet scheduling for real-time distributed systems. In *Euromicro Conference on Real-Time Systems*, pages 37–46, July 2015.

[6] A. Sadeghi, C. Wachsmann, and M. Waidner. Security and privacy challenges in industrial internet of things. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2015.

[7] Shweta Singh and Arun Tripathi. Analysis of delay and load factors in wired and wireless environments, 12 2015.

[8] Nils Svendsen and Stephen Wolthusen. Using physical models for anomaly detection in control systems, 03 2009.

[9] J. Xie and M. Xie. Delay bound analysis in real-time networks with priority scheduling using network calculus. In *2013 IEEE International Conference on Communications (ICC)*, pages 2469–2474, June 2013.

[10] C. Zimmer, B. Bhat, F. Mueller, and S. Mohan. Time-based intrusion dectection in cyber-physical systems. In *International Conference on Cyber-Physical Systems*, pages 109–118, April 2010.