Detecting Zero-day Controller Hijacking Attacks on the Power-Grid with Enhanced Deep Learning

Zecheng He
Princeton University
27 PUBLICATIONS **640** CITATIONS
SEE PROFILE

Aswin Raghavan
SRI International
28 PUBLICATIONS **168** CITATIONS
SEE PROFILE

Ruby B. Lee
Princeton University
266 PUBLICATIONS **10,125** CITATIONS
SEE PROFILE

Detecting Zero-day Controller Hijacking Attacks on the Power-Grid with Enhanced Deep Learning

Zecheng He
Princeton University
Princeton, NJ
zechengh@princeton.edu

Aswin Raghavan
SRI International
Princeton, NJ
aswin.raghavan@princeton.edu

Sek Chai
SRI International
Princeton, NJ
sek.chai@princeton.edu

Ruby Lee
Princeton University
Princeton, NJ
rblee@princeton.edu

ABSTRACT

Attacks against the control processor of a power-grid system, especially zero-day attacks, can be catastrophic. Earlier detection of the attacks can prevent further damage. However, detecting zero-day attacks can be challenging because they have no known code and have unknown behavior.

In order to address the zero-day attack problem, we propose a data-driven defense by training a temporal deep learning model, using only normal data from legitimate processes that run daily in these power-grid systems, to model the normal behavior of the power-grid controller. Then, we can quickly find malicious codes running on the processor, by estimating deviations from the normal behavior with a statistical test. Experimental results on a real power-grid controller show that we can detect anomalous behavior with over 99.9% accuracy and nearly zero false positives.

1 INTRODUCTION

The power-grid is a critical infrastructure, and attacks on it can cripple society, affecting national security, economic competitiveness and societal interactions. In 2015, Ukraine experienced cyber attacks against its power-grid system [2]. During this attack, 30 substations were switched off and 230 thousand people were left without electricity. The U.S. Government Accountability Office (GAO) questioned the current adequacy of defenses against power-grid attacks, and the North American Electric Reliability Corporation (NERC) has recognized these concerns [44]. With power-grid substations controllable through cyberspace transactions, the concern about cyber attacks on the physical power-grid systems is a real and serious threat, which is the focus of our paper. Although these attacks can have disastrous impact, they are still an unsolved problem.

Previous work on the cyber-security of power-grid systems focus on protecting the networks and data transmission. Wang [55] proposed a secure wireless communication architecture in the power-grid stations. Fouda [15] proposed an authentication scheme for communications in the context of a power-grid system. Sikdar [41] protected the anonymity of data to defend against the power-grid traffic analysis attacks. Zhang [58] used SVM at different levels of the power-grid system to detect network anomalies in a power-grid system. Each of these solved a specific loophole of secure communication in the power-grid system, but can hardly guarantee the detection of new attacks.

There have been other detection approaches proposed to protect the power-grid system by examining the physical sensors. Manandhar [31] used Kalman filters to model the voltage sensors and χ^2 test to detect physical anomalies in the power-grid system. However, their model assumed simple sine wave inputs which cannot be extended to a broader scope of sensor readings. In addition, the electrical sensor readings could be spoofed by the adversary [25]. Valenzuela [50] used Principle Component Analysis (PCA) and irregular-space analysis to detect anomalies in the power-grid. However, they may have oversimplified the problem by assuming the anomalies lie in a subspace of the input data space. Cardenas [9] suggested using linear models and threshold random walk (TRW) to deal with the unseen attacks. However, their proposed linear models are too simple to reflect the real complex physical processes. Zhao [59] proposed an expert system with Hidden Semi-Markov model. Their system requires the interaction between experts, thus it could be biased by the human experience of the known attacks.

Besides networks and sensors, protecting the control code running on the industrial programmable logic controllers (PLCs) is an essential and fundamental task in protecting the power-grid systems. No matter how the adversary spreads the malware or bad programs through system vulnerabilities, his ultimate goal is taking control of the power-grid infrastructures, destroying them or turning them off, and thus inducing huge physical impact. This attack strategy has been shown in the Stuxnet [26] and BlackEnergy [2] attacks. Earlier detection of the anomalous controller behavior leading to an attack can be helpful in preventing further severe impact or damage. Therefore, rather than physical attacks on the physical power-grid system, we consider cyber attacks that hijack the controller code that controls the actuators of the physical system.

However, it is challenging to protect the controller in the power-grid system because ① there is no prior-knowledge about the attacks, especially in the realistic "zero-day" attack settings. Labeled attack data are highly sensitive and not available for training. ② Various controllers, e.g. ABB, Mitsubishi, Siemens and Wago, and OS, e.g. Windows, Unix and proprietary OS (SIMATIC WinCC), are widely used in modern power-grid systems, exposing a large attack surface. ③ The power-grid systems usually implement weak anti-virus and integrity checking mechanisms on both executed code and transferred data, due to the computation capacity and hard-realtime constraints [22]. For example, runtime code integrity checking [39, 40] cannot be directly applied to power-grid systems, because of the high overhead in a real-time system. ④ Many concurrent threads are running on the power-grid controller simultaneously. Stealthy attack programs that have infected the controller can hide within these threads. Dolgikh [13] proposed customized normalcy profiles for programs in the power-grid systems, but they assumed that there are only limited applications running in the system.

In this paper, we propose a new data-driven methodology to detect zero-day controller hijacking attacks on power-grid systems. Our proposed defense provides comprehensive and reliable protection against unknown attacks. It does not look for specific attacks or triggers, but rather a holistic view of controller operation, thus it can catch a broad scope of anomalies. Specifically, our proposed approach first automatically learns the normal behavior of the controller using temporal deep learning, and generates a corresponding behavioral model of the controller. Then we use the generated model, enhanced by an effective statistical test to detect the deviation from processor normal behavior through *reconstruction error distribution.* Earlier detection of the anomalous controller behavior can help quick mitigation and recovery from the attack, for example, switching to a safe but not up-to-date version of the control code.

Our main contributions in this paper are:

- We observe that the controller behavior in the power-grid systems are predictable to some degree, because the hard-realtime control events come up in a periodic way. This observation is verified through a large-scale experiment on a real controller.
- We propose a new data-driven, learning-based, statistically-enhanced approach for detecting anomalies in the power-grid controllers. Intuitively, this defense predicts the future behavior of the controller through a well-trained deep temporal model, and investigates the reconstruction error distribution. No prior-knowledge of attacks is needed in either the training or inference, thus the proposed defense can be used to detect zero-day attacks.
- We evaluate our proposed defense on real power-grid controllers, by simulating the publicly reported attack functionalities [11, 26]. We achieve over 99.9% accuracy with nearly zero false positives. We also compare it with 11 conventional anomaly detection approaches to show the superiority of our proposed solution.

In Section 2, we provide background on the power-grid system, Program Logic Controllers (PLCs), hardware performance counters features and statistical test we use in our defense. In Section 3, we articulate our threat model. In Section 4, we present our detection methodology in detail. In Section 5, we present the experimental results and discussion. We discuss related work in Section 6 and conclude in Section 7.

2 BACKGROUND

2.1 Architecture and Vulnerability of Power-Grid Systems

Figure 1 shows a SCADA (Supervisory Control And Data Acquisition) system controlling a power-grid [3]. While the rest of the SCADA network and components are similar to general information processing systems, the power-plant substations are the cyber-physical systems we focus on. The key components in these substations are physical sensors (e.g. voltage, current), actuators (e.g. relays, valves) and programmable logic controllers (PLCs). The PLC obtains inputs from the sensors, and outputs control signals to the actuators according to its control logic or control code. The substations are connected through local networks.

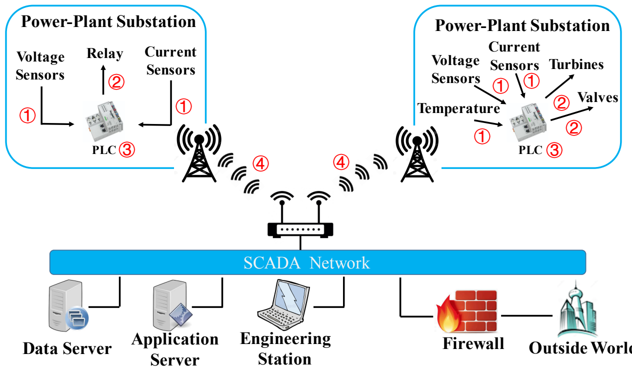

Figure 1: A diagram of a power-grid system [5]. Attacks can happen in ① sensors, ② actuators, ③ controllers and ④ networks.

Attacks against the power-grid system can happen in any of the components [23], i.e. sensors, actuators, controllers and the interconnected networks illustrated in Figure 1. Attacks against the physical actuators or sensors usually require physical accesses to the devices, and are not in the scope of this paper. The ultimate goal of the network attacks against the power-grid is launching the malware, compromising the controller and inducing physical damage. Hence, we focus on the detection of the controller's abnormal behavior in our study of cyber attacks on the power-grid.

2.2 Programmable Logic Controller (PLC)

A programmable logic controller (PLC) is a digital device, equipped with microprocessors, which is widely used as the controller in a power-grid subsystem. Figure 2 shows the block diagram of a PLC [1]. A PLC is composed of several subsystems, e.g. CPU, internal memory, power subsystem and digital-to-analog converter. A PLC can take physical sensor (and other) inputs from the target industrial devices, make decisions and send commands to control a large range of electrical devices and actuators.

The PLC used in our experiments is a Wago PLC with ARM Cortex A8 processors. A custom real-time Linux OS runs on this PLC. This PLC is widely used as the controller in a power-grid system. We would like to monitor and predict the PLC behavior through hardware performance counters to detect possible attacks against the power grid system.

2.3 Hardware Performance Counters

In this work, we use low-level hardware events to monitor the controller behavior. Controller hardware events, such as the number of executed instructions, cache operations and exceptions, are

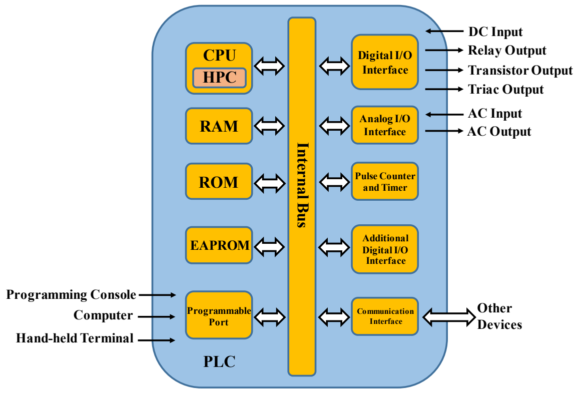

Figure 2: Programmable logic controller block diagram [1]

automatically and securely counted and stored in Hardware Performance Counters (HPC) – a set of special-purpose registers. HPCs are now widely available in commodity processors, e.g., Intel, ARM and PowerPC. Detailed discussion on the availability of HPCs on modern processors is given in Appendix A.

Hardware performance counters provide high tamper-resistance and a low-cost way to monitor the controller behavior. First, unlike anti-virus software, HPCs are hardware that cannot be directly accessed and modified by malware [52]. This characteristic of HPCs significantly increases the tamper-resistance property. Second, HPCs automatically record the hardware events, requiring no extra logging or recording. Reading from HPCs is more efficient than hashing the whole control code, making it low-cost with negligible overhead.

Previous work has revealed the feasibility of using HPCs for detecting malware [7, 12, 36, 37], firmware-modification [54] and kernel root-kit [43, 53]. The key assumption of these detection techniques is that the behavior of programs can be captured and characterized with the occurrences of certain hardware events. We present a temporal deep learning method to model the controller behavior differences through the low-level HPC micro-architectural events.

2.4 Kolmogorov-Smirnov Test

The Kolmogorov-Smirnov test (KS test) is a nonparametric test of one-dimensional probability distributions. It can be used to distinguish if two sets of samples are from the same distribution. The Kolmogorov-Smirnov statistic for two sets with n and m samples is:

$$D_{m,n} = sup_x |F_n(x) - F_m(x)| \tag{1}$$

where F_n and F_m are the empirical distribution functions of two sets of samples respectively, i.e. $F_n(t) = \frac{1}{n}\sum_{i=1}^{n} 1_{x_i \le t}$. sup is the supremum function. Figure 3 illustrates the statistic $D_{m,n}$ [6]. The red and blue lines are the two empirical distributions of the two sets. The black arrow is the two-sample KS test statistic $D_{m,n}$. The x-axis is the cumulative threshold and the y-axis is the cumulative probability. The null hypothesis that the two sets of samples are i.i.d. sampled from the same distribution, is rejected at level α if:

$$D_{n,m} > c(\alpha)\sqrt{\frac{n+m}{nm}} \tag{2}$$

where $c(\alpha)$ is a pre-calculated value and can be found in the standard KS test lookup table.

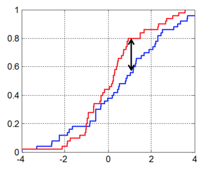

Figure 3: Illustration of Kolmogorov-Smirnov statistic $D_{m,n}$ [6]

3 THREAT MODEL

In this paper, we consider cyber attacks that hijack the controller code that controls the actuators of the physical system. We protect the integrity of the control code running on a power-grid controller. We do not explicitly consider the attacks against networks in power-grid systems, because the ultimate goal of the network attacks is corrupting the controller, which has been shown in the Stuxnet [26] and BlackEnergy [2] attacks. Thus these attacks can be detected if we can detect the controller's anomalous behavior. We highlight key points of our threat model in detail:

Our threat model specifically includes zero-day attacks. We assume that there is no prior-knowledge of attack code, and the way the adversary hijacks the controller. We only assume the goal of the attacker is to take control of the hijacked controller to run unauthorized code on the controller. Note that we do not make any assumptions about the type of the unauthorized code, i.e. it can be malicious control logic, worms, trojans or spyware, etc. Therefore, though not explicitly targeted, availability and confidentiality breaches are also included in our threat model. Furthermore, we assume the normal behavior of the controller can be collected in a clean environment and used for training a deep learning model. We assume that the collection of performance data from the normal operation of the PLC is readily available, e.g., through the built-in hardware performance counters.

Attacker Capabilities. We consider an active attacker who can hijack the controller code, add to, delete or modify the programs running on the controller, by exploiting system security design holes. Consequences of the attacks include controller failures, incorrect outputs to the controlled physical actuators and other subsystems. We assume the attacker can either access the victim system remotely (e.g. network or Botnet) or physically (e.g. USB). The attack code can be polymorphic and stealthy.

Target Systems. We consider a power-grid system in our threat model. First, we assume the power-grid system implements weak anti-virus and integrity checking mechanisms on both executed code and transferred data, due to the computation capacity and hard-realtime constraints [22]. Second, we assume the low-level hardware performance counters are accessible and trusted. This assumption is reasonable because hardware registers are harder to tamper with. Third, the target system is relatively stable, i.e. the code running on the controller changes infrequently. This assumption is reasonable because the control processes, once downloaded to the controller, usually are not updated very often [27].

4 OUR DETECTION METHODOLOGY

4.1 Overview

We consider the attack scenario in which the attacker breaches the integrity of the control code and causes the controller to function incorrectly in the power-grid system. Our key idea is, the normal behavior of the controller in a power-grid system is predictable using a temporal deep learning model and tamper-resistant HPC features. The deviation of the real monitored behavior from the predicted behavior indicates the anomalies in the controller. We further use a statistical test to determine the occurrence of such deviation, and consider that as abnormal controller behavior. We show an overview of the workflow of our proposed method in Figure 4. There are four steps in the hijacking detection. We use "baseline" and "normal", "real observed sequence" and "testing sequence" interchangeably.

(1) Training a deep sequence predictor to predict baseline controller behaviors in the future. The sequence predictors used in this paper are LSTM and Conditional-RBM.
(2) Calculating the baseline reconstruction error distribution D_1 as the reference distribution. In our experiments, we use the squared error of the predicted behavior v_1 and the observed behavior v_2, i.e. $|v_1 - v_2|^2$, as the reconstruction error.
(3) Using the sequence predictor online to predict the future behavior using the historical behavior. Calculating the reconstruction error distribution D_2 of the observed (testing) behavior which can be normal or abnormal.
(4) Applying the KS test on D_1 and D_2 to determine if they are from the same distribution.

We show our proposed system design in Figure 5. It consists of several components: a deep learning accelerator and an HPC interface [35] for predicting the normal behavior of the controller, an external RAM for storing the reference "gold values", a KS test module and an anomaly response module, e.g. a code version controller. The deep learning module reads the HPC measurements in real-time and computes the testing reconstruction errors. The KS test module receives the testing and reference reconstruction errors, from the deep learning module and the external RAM, respectively. The positive KS test result (two sets of reconstruction errors with different distributions) triggers the code version controller to switch to a "safe but not updated" version of the control code [34].

Next, we address the challenges and concerns in designing an effective protection for power-grid controllers.

Capturing high-level controller behavior with low-level HPC features. Although using HPC measurements benefits from the low-cost and high tamper-resistant properties, it is challenging to appropriately capture and characterize high-level controller behavior from the low-level hardware events. Conventional machine learning or clustering based anomaly detection approaches require carefully designed heuristic features, however, high-level semantic features are typically hard to handcraft from low-level measurements.

To address this concern, first, we observe that the functionality of control programs in a power-grid system are more periodic and stable than general-purpose systems. This feature eases the modeling of program behavior. Second, we use temporal deep learning to automatically characterize the behavior of the controller from low-level HPC measurements. Temporal deep learning models, e.g. Long Short-Term Memory (LSTM), have been shown to be able to capture the semantic features from low-level measurements in other fields, e.g natural language processing (NLP) [20, 51, 57], video [45, 60] and speech analysis [18, 32]. For this reason, we believe the deep learning model can also capture the behaviors in the power-grid systems, and thus is able to detect anomalous controller behavior.

Predictability of the controller behavior. A mild level of predictability of the controller is sufficient for our defense. The "accurate prediction" of the controller's behavior is not necessary, since we are looking for the *difference* between the prediction error distributions. We do not expect nearly zero prediction errors for normal behavior (accurate prediction), but we would like the prediction errors of normal/abnormal behavior to be different. We show that this mild level of predictability is achievable in the power-grid system in Section 4.2 through our experiments.

Concurrent threads. PLCs in a power-grid system can have concurrent threads with dependencies among the threads. Therefore, to provide comprehensive protection, we have to monitor all threads running on the controller. Typically, there are fewer threads (10-100) running on the power-grid controller than a general-purpose system, making it possible and low-cost to monitor all threads.

Hardware events selection. Hundreds of architectural and micro-architectural hardware events are available in mainstream processors. However, not all HPC measurements are equally good for power-grid controller anomaly detection. The properties of good hardware events include ① sensitivity to code changes, ② stability from one execution to another, ③ wide availability in all processors. We select four candidate HPCs for each thread based on these criteria: ① the number of cycles and ② the number of instructions executed for each thread, to provide the overall running status of the thread. ③ The number of L1 cache misses for each thread, to reflect the locality and temporal properties of data usage. ④ The number of branch instructions for each thread, to show the changes in the control flow of the threads.

Polymorphic and stealthy attacks. To bypass the static malware analysis, the adversary can write code variants that are functionally equivalent (polymorphic codes). However, our detection characterizes the program by its functionality and behavior. No matter how the adversary changes his code, the malicious functionalities of the malware always exist. A similar situation occurs for stealthy attacks, where the attacker tries to hide by prolonging the time-span or mimicking the normal behavior. Therefore, the accumulated suspiciousness in a long time-span can still be captured through temporal deep learning, because it can adjust its monitoring window automatically and capture the accumulated deviations. In addition, we comprehensively monitor all threads on the processor, significantly raising the bar for the adversary to create code mimicking all threads on the controller.

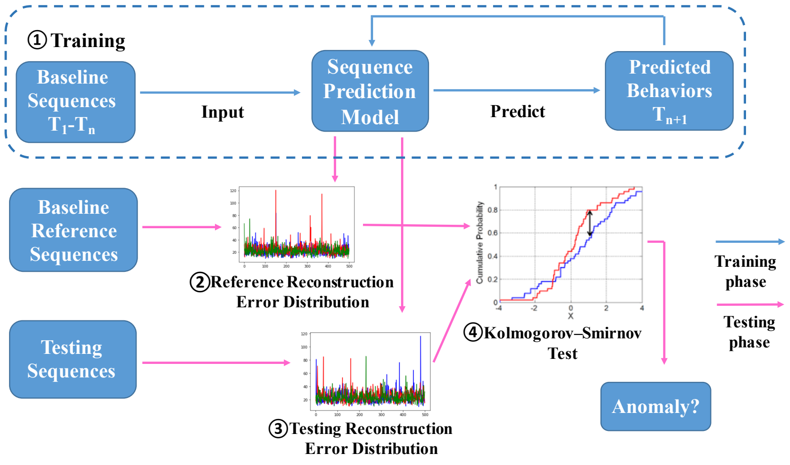

Figure 4: Workflow overview of our proposed deep temporal model and statistical test based method. Offline profiling: ① Train a deep learning model with only normal data, to predict normal behavior of the controller. ② Collect the prediction error distribution (Reconstruction Error Distribution) D_1 of the baseline behavior. Online detecting: ③ Monitor the real behavior of the controller. Compute the deep learning model reconstruction error distribution D_2. ④ Statistically test if D_1 and D_2 are the same distribution. If not, a deviation from the normal behavior is observed and anomalous behavior is detected.

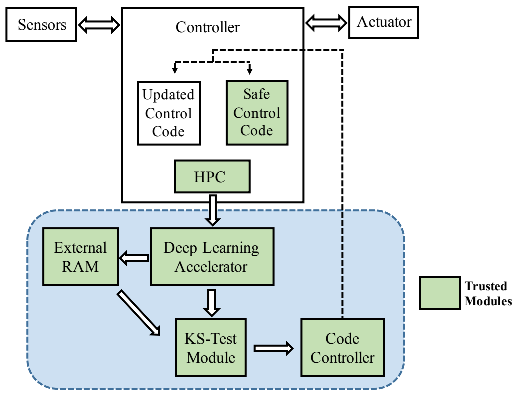

Figure 5: Overview architecture of our proposed detection system. It consists of an interface between HPC and the deep learning accelerator [35], an external RAM storing the reference "gold values", a KS test module and an anomaly response module, e.g. shown here as a code version controller.

4.2 Controller Behavior Prediction with Deep Temporal Models

Model training. The first step is training a temporal deep learning model to predict the controller's future behavior based on its historical behavior, illustrated as ① in Figure 4.

In the training, we first collect a set of sequences of the controller behavior measurement $\{S_i\}_{i=1}^N$, which are HPC measurements in our scenario, in the clean environment. N is the number of total sequences. Each sequence S_i consists of T continuous behavior measurements, i.e. $S_i = [S_i^1, S_i^2, ..., S_i^T]$. In our experiments, each behavior measurement S_i^t is a vector consisting of 4 HPC readings for 23 threads, i.e. a vector of dimension 4*23=92. At time t, the deep learning model predicts S_i^{t+1} using behavior history $[S_i^1, ..., S_i^t]$. We denote this prediction as O_i^{t+1}. The loss function is defined as the accumulated prediction errors, i.e.

$$loss = \frac{1}{N}\frac{1}{T-1}\sum_{i=1}^{N}\sum_{t=2}^{T}(S_i^t - O_i^t)^2 \quad (3)$$

Intuitively, since $\{S_i\}_{i=1}^N$ are normal behavior collected in the clean environment, the loss penalizes the incorrect prediction of normal behavior. We train this model to minimize the loss function with Stochastic Gradient Descent (SGD).

To justify our observation that the controller behavior in a power-grid system is predictable, we collect 300,000 samples from 4 HPC readings of 23 threads on a real power-grid system that runs every day. We find 40 HPC values for specific threads are always zeros, thus we exclude these HPC values and train an LSTM to predict the controller behavior regarding the remaining HPCs. We use Signal-to-Noise Ratio (SNR) as our metric of predictability. Without handcrafting features, we achieve **an average SNR = 12.68 dB**. Note that random guesses (if controller behavior is non-predictable) result in an SNR value very close to $-\infty$. This observation indicates that the controller behavior is predictable, with respect to the HPCs, to a mild degree, in the power-grid system.

Reconstruction error distribution profiling. After training the model, we profile the normal behavior in terms of *reconstruction error distribution*, illustrated as ② in Figure 4. First, a longer reference sequence of controller behavior measurement, $R = [R^1, R^2, ..., R^{T'}]$, is collected in the clean environment. Note that R is another sequence rather than previous S_i, consisting of T' time frames. Each time frame R^i is a vector of dimension 4*23=92 in our experiment. Second, at time frame t, we use the trained model to predict time frame $t+1$, $t+2$,...,$t+L$ using the corresponding history behavior.

We denote the prediction as $O^{t+1}, \ldots, O^{t+L}$ respectively. The reconstruction error is defined as the accumulated prediction errors, i.e.

$$E(t) = \sum_{i=t+1}^{t+L} (R^i - O^i)^2 \tag{4}$$

We define the distribution of E(1), E(2), E(3)... as the **reconstruction error distribution**. Using the reconstruction error distribution as the profile has several advantages. ① It is more robust to the noise in the measurement, because noise makes a single prediction error vary significantly but the distribution remains stable. ② It simplifies the model learning process. Since we profile the error distributions, we no longer need to train a "perfect" predictor. ③ It broadens the scope of target systems by decreasing the required level of predictability. By using reconstruction errors, we allow the existence of imprecisions in the prediction. Thus it reduces the required predictability of the system to a mild level.

In the profiling, collections of the reference reconstruction error distribution are required to be gathered in a clean environment. This can be done by collecting the HPC sequences immediately after a trusted version of control code is uploaded. The gathered HPC sequences are sent to the trained deep learning module to calculate the reference reconstruction errors. These offline reconstruction errors are then stored in a separate RAM partition, and used as the references in the online detection. To better represent the reconstruction error distribution by the empirical samples, multiple reference reconstruction errors can be generated and compared with the observed reconstruction errors in the online detection phase.

Model selection. Among the deep learning models, Recurrent Neural Network (RNN) and its variation Long Short-Term Memory (LSTM) [21] have become the most powerful ones in modeling sequential data. They achieve significant successes in sequential data processing [19, 47]. An LSTM cell has three gates that control information flow: the forget gate, the input gate and the output gate. The forget gate controls the amount of previous information remaining in the cell memory. The input gate controls the amount of new information flowing to the cell memory, and the output gate controls the amount of information flowing to the output. Thus, LSTM automatically determines what information to "remember" and "forget". Details of LSTM are given in Appendix B.

The Restricted Boltzmann Machine (RBM) is another generative neural network which can learn a probability distribution. RBMs are shallow, two-layer neural nets i.e. one visible layer and one invisible layer. Each visible node takes a low-level feature (e.g. HPC values in the context of a power-grid system) from a sample in the dataset to be learned. Each invisible node represents a high-level learned feature (e.g. controller behavior in the context of a power-grid system). However, the vanilla RBM does not capture the temporal information, therefore in this work, we leverage a variant of RBM which considers historical information, i.e. Conditional-RBM (CRBM). The detailed structures of RBM and CRBM are given in Appendix B.

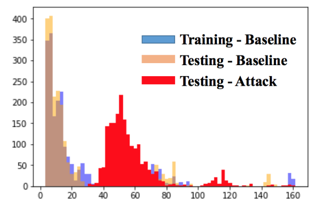

Figure 6: A real histogram example of the reconstruction error distributions of ① reference baseline behavior (blue), ② testing normal behavior (orange) and ③ testing abnormal behavior (red). We observe that the distributions of reference (blue) and testing normal (orange) behavior are very similar, but distinct from the distribution of abnormal (red) behavior.

We use either LSTM or Conditional-RBM as our predictors of the controller behavior, enhanced by the Kolmogorov-Smirnov statistical test (Section 4.4).

Model evolution. To accommodate the legitimate controller behavior drift, e.g. an update of the control code, the deep learning model needs to be retrained or fine-tuned online. Note that either in the initial training or online fine-tuning, the deep learning model must be trained with HPC measurements from a clean (no attack) environment. Otherwise, the trained model cannot predict the normal behavior correctly.

4.3 Online Hijacking Detection

The online hijacking detection is illustrated as step ③ and ④ in Figure 4. The controller behavior, in terms of HPCs, is dynamically monitored at runtime. Similar to the offline profiling phase, the gathered HPC sequences are sent to the deep learning module for prediction. The same form (Eq. 4) of reconstruction errors are calculated and sent to the KS test module, along with the reference reconstruction errors gathered in the offline phase.

We show a histogram example of the reconstruction error distributions in Figure 6. The blue histogram shows the reference reconstruction error distribution gathered in the offline profiling. The orange/red histograms show the testing normal/abnormal reconstruction error distribution, respectively. The blue distribution (reference) is very similar to the orange distribution (testing normal), but very distinct from the red distribution (testing abnormal). In addition, the blue and orange distributions mainly concentrate at the low-end (small errors), justifying our observation that the deep learning model can predict the normal behavior of the processor, even with small behavior deviations.

All the computation of deep learning prediction and reconstruction errors are performed outside the controller, on a trusted GPU or deep learning accelerator, because if the controller is hijacked, the results computed by it cannot be trusted.

Our detection system can be integrated with different mitigation approaches. For example, after a controller hijacking attack is detected, an alarm is sent out. Another mitigation response is switching to a "safe version" of controller code in a ROM, which may not be up-to-date but is free from attack. Other responses are possible, such as controlling the actuator settings, while checking that an attack has actually occurred rather than a false alarm.

4.4 Statistical Test for Reconstruction Error Distributions

We now address the problem: how to effectively determine anomalous behavior from the reconstruction errors?

Problems with Hard Thresholds and Gaussian Assumptions. Previous work [30] [29] assume a multivariate Gaussian distribution and set a hard threshold for the reconstruction errors, e.g. mean + 3 * standard deviations (or best on the validation set), to detect the anomalies. However, we find that the Gaussian assumption does not correctly reflect the reconstruction error distributions of the predicted processor behavior. Figure 6 illustrates the "one-sided long-tail" characteristic of error distribution, which clearly differs from the Gaussian distribution. Besides, we find that the threshold value significantly affects the accuracy and relies highly on the split of the validation set, making it not stable nor reliable.

Figure 7 illustrates the detection using a hard threshold, i.e. mean + 3 * std on the validation set. The top figure (a) shows the raw reconstruction error for normal behavior (left half, time frame t=0-12000) and attacks (right half, time frame t=12000-24000). Note that the separation of normal behavior and attacks is only for clearer illustration, real monitoring results can be an arbitrary random mixture of normal behavior and attacks. We observe that the reconstruction errors on the right (attacks) are generally larger than the reconstruction errors on the left (normal behavior). A hard threshold of mean + 3 * std on the validation set is shown as the red horizontal line in Figure 7 (a). Figure 7 (b) shows the detection results, where "1"s represent the time frames recognized as abnormal, and "0"s represent the time frames recognized as normal. We observe that such a hard threshold induces significant false positives and false negatives.

Kolmogorov-Smirnov Test for Anomaly Detection. Hence, we propose a new approach for detecting anomalies from the reconstruction errors, without the Gaussian assumptions. We propose using the Kolmogorov-Smirnov test to determine if the reconstructed error distribution is the same distribution as the reference error distribution (Section 4.2). Suppose the reference reconstruction error distribution is D_1 and the reconstruction error distribution on the questioned sequence is D_2. We sample n and m independent samples from the two distributions, respectively. We calculate the KS test statistic, i.e., $D_{m,n} = sup_x |F_n(x) - F_m(x)|$. Then we define a significance level p for this test, i.e. the probability of detecting a difference under the null hypothesis that samples are drawn from the same distribution. We reject the null hypothesis, i.e., recognize that the samples as anomalies, if the inequality in Eq (2) holds. The KS test eliminates previous unrealistic Gaussian assumptions and significantly improves the accuracy in our experiments.

4.5 Security Analysis

We now discuss the potential attacks and how they are defended by our proposed solution.

Case 1: The attacker hides and waits for a specific time to hijack the controller. If the attack code keeps silent, it does not do any harm to the system. As long as the attack becomes active and influences the behavior of the controller, our experiments show that we can detect it in a short time and eliminate the damage to the system. Note that our proposed method provides continuous and holistic protection of the controller.

Case 2: The attacker tampers with the prediction and KS test. If the deep learning prediction and KS test are computed on the hijacked controller, the attacker can also compromise the detection results. However, in our design, the prediction and KS test are done entirely outside the controller.

Case 3: The attacker steals the reference reconstruction samples. If the attacker obtains the reference samples, we claim there is still no way to construct a code sequence to cheat our system, because the deep learning module serves as a "black-box" oracle to the attacker: he cannot reverse the input (HPC sequence) from the output (reconstruction errors).

5 EXPERIMENTAL RESULTS

5.1 Data Collection Setup and Metrics

We implement a prototype of our proposed defense system. Our target system is a real power-grid controller sub-system, running the real deployed power-grid controlling code. The control codes are written in Structured Text (IEC 61131-3 standard), and run on the PLC as a highly multi-threaded program (e.g., 23 threads as I/O, controls, etc.). The 23 threads in the Wago PLC process being monitored correspond to the main thread and 22 threads that they spawn in the order described below. When the PLC powers on, 7 threads are created. When an application is loaded, another 15 threads will be killed and re-spawn with a different thread identifier (TID). The last thread (PLC_Task) is the one specifically running the loaded application (utilizing the other threads to do various tasks). We list the threads and their start phase in Table 1.

Table 1: Threads running on Programmable Logic Controller

Start Time	Thread(s)
PLC powers on (7)	spi1, codesys3, com_DBUS_worker, 0ms_Watch_Thread, CAAEventTast, SchedExeption, Schedule
When an application is loaded (15)	WagoAsyncRtHigh, WagoAsyncRtMed, WagoAsyncRtLow, WagoAsyncHigh, WagoAsyncMed, WagoAsyncLow, WagoAsyncBusCyc, WagoAsyncBusEvt, WagoAsyncBusEvt, WagoAsyncBusEvt, WagoAsyncBusEvt, ProcessorLoadWa, KBUS_CYCLE_TASK, ModbusSlaveTCP, PLC_Task
Main thread (1)	Main

The HPCs are monitored on a real PLC with an ARM Cortex A8 processor and custom real-time Linux distribution. Data from 4 HPCs was collected for each thread:

- Number of cycles executed, N_C
- Number of instructions executed, N_I
- Number of branches executed, N_B
- Number of L1 cache misses, N_L.

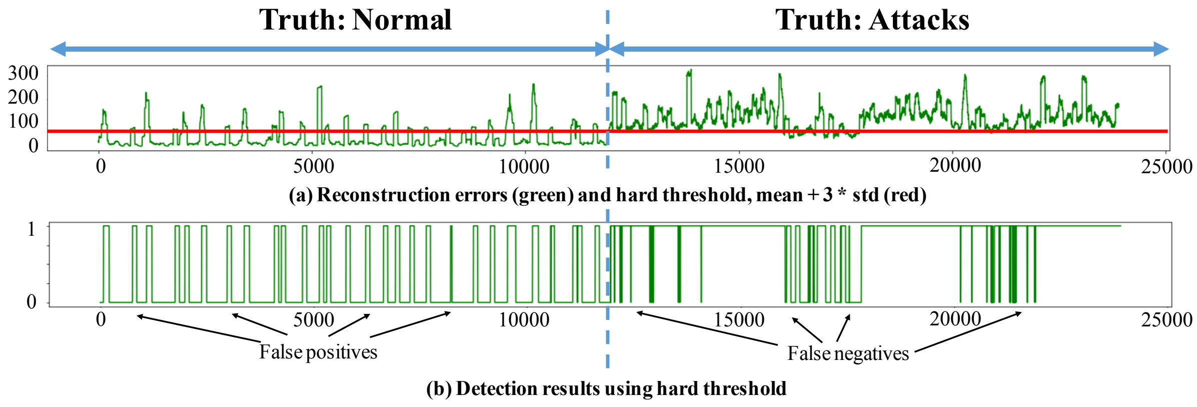

Figure 7: Detecting anomalies using hard threshold (mean + 3 * stddev). Time frame t=0-12000 show the reconstruction error of normal behavior (no attack), t=12000-24000 show the reconstruction error under attacks. (a) The raw reconstruction errors (green) and hard threshold, mean + 3 * stddev. (b) Detection results using hard threshold, "0" is predicting normal, "1" is predicting abnormal. We observe many false positives and false negatives with a hard threshold.

The total number of HPCs monitored is 4N if there are N threads. However, as the monitored ARM processor has only 2 read ports, only "N_C and N_L" or "N_B and N_I" are read simultaneously and then switch. The HPCs are sampled at 1 kHz for each of the 23 threads.

The system was tested with six different real attacks in Table 2, that are known to cause damage to the power-grid. The attacks include simulating the publicly reported malicious functionalities of Stuxnet [26] and BlackEnergy [2], i.e. halting the controller and replaying prerecorded inputs to the legitimate code. In addition, we consider the other attacks against the input, output and control logic of the processor, which are the key components of the control code. We sample 300,000 samples for each attack running.

We define 6 important performance metrics below that we will use in our experiments: accuracy, false positive rate, false negative rate, precision, recall and F1 score. Precision measures the ratio of true positives among all predicted positives. Recall measures the ratio of detected positives among all real positives. The F1 score is the harmonic mean of Precision and Recall, which balances these two metrics. We show the values of the metrics in Tables 3, 4 and 5. We will compare these results in detail in the following sections.

5.2 Effectiveness in Detecting Attacks

We first show that our enhanced deep learning approach is effective in amplifying small changes of control logic code (e.g., small code changes in Attacks 1-6) in the PLC to large KS statistics D. We show three representative examples, Testing Normal (row 1), Attack 1 (row 2) and Attack 2 (row 3), in Figure 8. ① The left column shows the reconstruction errors in the time domain. We observe that, in general, the reconstruction errors of Attacks (red in the 2nd and 3rd row) are slightly larger than the testing normal scenario (green in the 1st row). ② The middle column shows the histograms of the reconstruction error distribution. We observe that the reconstruction error distribution of testing normals (green, row 1 middle) is more similar to the reference distribution (blue, row 1 middle), than the distribution of attacks (red, rows 2 and 3). ③ The right column shows the KS test statistics D. We find that D is significantly larger under the attacks (rows 2 and 3) than testing normal (row 1). This gives us assurance that our enhanced deep learning method (LSTM + KS test) can indeed detect attacks which materialize as only small code changes in the PLC in power-grid systems.

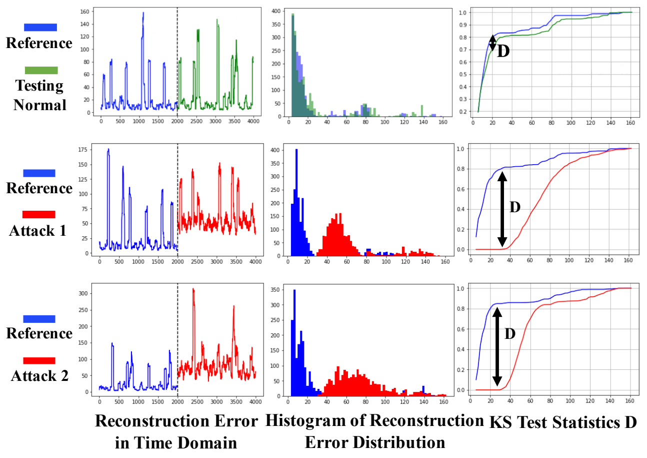

Figure 8: Effectiveness in detecting attacks. Left: LSTM reconstruction errors in the *time domain* for testing normal behavior (row 1), Attack 1 (row 2) and Attack 2 (row 3). Middle: Histogram of reconstruction error distribution for Normal behavior, Attacks 1 and 2. Right: CDF of reconstruction error distribution and KS statistic D for Normal behavior, Attacks 1 and 2. The KS statistic amplifies the differences in the time domain in cases of attacks (second and third rows), while remaining small in the case of normal behavior (row 1).

5.3 Results and Discussion

Anomaly Detection Comparison. We show the detection results of our proposed enhanced deep learning based approach,

Table 2: Baseline and attacks against a power-grid PLC

	Description	Hijack Type
Baseline (Testing Normal)	Baseline (a Proportional Integral Derivative (PID) controller)	None
Attack 1	Overwrite the input (an additional line of code to overwrite the value of the input)	Input hijack
Attack 2	Saturate the input (2 additional lines of code with IF condition on input value)	Control flow hijack
Attack 3	Disable the PID control code (the entire PID block is commented out)	Entire Code hijack
Attack 4	PID in "manual" mode (output is fixed)	Output hijack
Attack 5	2 PIDs Cascaded (input of a PID controller is sent through a second PID controller)	Control flow hijack
Attack 6	Overwrite the output (an additional line of code to overwrite the value of the output)	Output hijack

and compare with 11 different conventional anomaly detection approaches in Table 3. Note that all conventional anomaly detection approaches require heuristic features, while our approach takes raw data as input. This significantly reduces the tedious feature design and extraction process. We fine-tune the hyper-parameters and report the highest F1 score for the conventional anomaly detection methods.

Our proposed deep temporal model based detection methods (last 2 rows) achieve as high as 99.97% accuracy and 0.9997 F1 score with no false negatives (all anomalies are detected) for the LSTM + KS test approach. We observe that the deep temporal model based detection approaches, especially LSTM, are better than all conventional methods, in terms of accuracy and F1 score, in detecting power-grid controller anomalies. Some approaches achieve slightly lower false positive rates than CRBM+KS test, however, the false negative rates are much higher and vice versa. Our proposed LSTM and CRBM methods are better than conventional approaches because conventional detection methods are not able to automatically extract inherent but complex features which can be used to model the normal behaviors of the target power-grid controller. In addition, LSTM is superior to all conventional anomaly detection approaches and CRBM, as it can automatically adjust the monitoring window, while the conventional approaches and CRBM require a fixed window size which could be highly biased.

Deep Learning Models Comparison. We investigate how the types and architectures of the deep learning models affect the detection results. First, we show the results of LSTM with different numbers of hidden nodes and CRBM in Figure 9 (a). We observe that the LSTMs perform better than the CRBM, because the LSTM automatically adjusts the "memory window size" while CRBM uses a fixed size. We also find that an LSTM with a medium number (128) of nodes in the hidden layer performs better than an LSTM with a small (5) or large (256) number of nodes in the hidden layer, because too few nodes in the hidden layer significantly limit the capability of the LSTM. The model can hardly represent the controller behavior because of the low capacity. On the other hand, an LSTM with too many nodes in the hidden layer causes another problem – over-fitting. In this case, the LSTM model "memorizes" the training samples and lacks generalization ability.

Next, we compare the training mechanisms of LSTM with CRBM. We provide the accuracy and false positive (FP) rate versus training epochs in Figure 9 (b). We find that LSTM convergences faster than CRBM, and achieves better final detection results. LSTM also becomes stable at the end of the training, however, CRBM continues to oscillate as we continue to train it. CRBM introduces uncertainty in training because of its sampling step, making the training for CRBM less stable than LSTM.

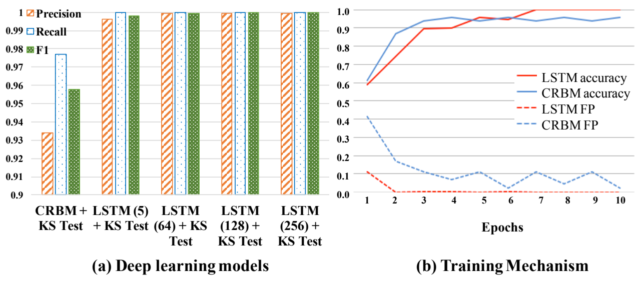

Figure 9: (a) Detection results of using different deep learning models as the predicting module. In general, LSTM performs better than CRBM, regarding precision, recall and F1 score. The number of hidden nodes in LSTM does not significantly affect the results, as long as not too few hidden nodes (>5). (b) Training mechanism for LSTM and CRBM. LSTM converges faster and achieves better ultimate accuracy than CRBM.

Hard Threshold and Statistical Test Comparison. We evaluate the effectiveness and necessity of our statistical test. In Table 4, we compare the detection results with and without KS test. The first 4 rows show the detection results of different LSTMs using hard thresholds, while the next 4 rows show the corresponding LSTM + KS test for comparison. We observe that the statistic test improves all evaluated LSTM models. We observe that, without KS test, the highest detection accuracy is 97.7% when 128 hidden nodes are used. The accuracy rises to 99.98% using KS test. In general, the KS test has three benefits: ① It gets rid of the Gaussian assumptions in the thresholding approaches (Elliptic Envelope), especially when this Gaussian assumption does not hold in our usage scenarios. ② It provides insensitivity to the deep learning architectures (LSTM in our experiment). ③ It is not sensitive to the p-values in the scenario of power-grid anomaly detection. The KS test gives us what we call an **enhanced deep learning method**.

Significance Levels. Although we find that the final accuracy is not sensitive to the p-value, it is still the one parameter we should deal with carefully. Thresholds are generally very arbitrary and highly dependent on the validation set and data distributions. However, significance level p, a.k.a. p-value, is explainable and indicates how incompatible the data are with the hypothesis. The most widely used value is $p = 0.05$, which trades off between the

Table 3: Anomaly detection methods on detecting zero-day attacks against a real ARM Cortex A8 PLC controlling a real power-grid. Our proposed deep learning + statistical test approach performs better than all evaluated methods.

Anomaly Detection Method	Key Hyper-parameters	Accuracy	False Positive	False Negative	Precision	Recall	F1
One-Class SVM	RBF Kernel, $\gamma = \frac{1}{N_{features}}$	0.849	0.104	0.621	0.267	0.379	0.236
KNN	$N_{neighbors} = 5$	0.930	0.000	0.766	1.000	0.235	0.380
PCA	$N_{components} = 10$	0.841	0.161	0.141	0.347	0.859	0.495
Exponential Moving Average	Smoothing Factor = 0.2, Smooth Window = 20	0.911	0.000	0.982	0.994	0.018	0.127
Elliptic Envelope Estimation [38]	Support Fraction = 1.0	0.867	0.146	0.000	0.406	1.000	0.578
Robust Elliptic Envelope estimation [42]	Support Fraction = $\frac{N_{samples}+N_{features}+1}{2}$	0.928	0.079	0.000	0.559	1.000	0.717
Local Outlier Factor [8]	$N_{neighbors} = 20$	0.929	0.000	0.783	1.000	0.218	0.357
Isolation Forrest [28]	$N_{trees} = 100$	0.949	0.056	0.000	0.640	1.000	0.780
Bitmap Encoding [56]	Sections = 4	0.911	0.001	0.968	0.799	0.032	0.061
HBOS [16]	$N_{bins} = 10$, regularizer $\alpha = 0.1$	0.899	0.111	0.001	0.473	0.999	0.642
Fast ABOS [24]	$N_{neighbors} = 5$	0.929	0.000	0.784	1.000	0.216	0.355
CRBM + KS test (Proposed)	$N_{hidden} = 50$	**0.957**	**0.062**	**0.023**	**0.934**	**0.977**	**0.958**
LSTM + KS test (Proposed)	$N_{hidden} = 256$	**0.9997**	**0.0005**	**0.000**	**0.9995**	**1.000**	**0.9997**

Table 4: Effectiveness of combining deep learning models and statistical KS test

	Accuracy	False Positive	False Negative	Precision	Recall	F1
LSTM(5)	0.935	0.028	0.103	0.970	0.897	0.932
LSTM(128)	0.977	0.028	0.017	0.972	0.983	0.977
LSTM(256)	0.976	0.041	0.007	0.960	0.993	0.976
LSTM(512)	0.976	0.041	0.007	0.960	0.993	0.976
LSTM(5) + KS	0.998	0.004	0.000	0.996	1.000	0.999
LSTM(128) + KS	0.9998	0.0004	0.000	0.9996	1.000	1.000
LSTM(256) + KS	0.9997	0.0005	0.000	0.9995	1.000	1.000
LSTM(512) + KS	0.9997	0.0005	0.000	0.9995	1.000	1.000

false positives and the false negatives. In general, a small p-value decreases false positives but increases false negatives, and vice versa. For a fair comparison, we report the results when $p = 0.05$ although we can achieve even better results with a smaller p.

From Table 3 and 5 we observe that our KS test based approaches are much better in accuracy than all conventional hard-thresholding approaches, except when the p-value=0.5, the accuracy is 0.947, which is slightly lower than Isolation Forrest (0.949). Also, we observe that KS test based approaches are more robust regarding p-values. For a broad range of p-values, the KS test gives good results, in contrast to thresholding based approaches. In addition, we find the KS test also makes the result not sensitive to different architectures. We have tested LSTMs with 5, 128, 256 and 512 hidden nodes.

Table 5: Influence of Confidence Value p

	Accuracy	False Positive	False Negative	Precision	Recall	F1
p = 0.5	0.947	0.105	0.000	0.905	1.000	0.950
p = 0.2	0.993	0.013	0.000	0.987	1.000	0.993
p = 0.1	0.998	0.003	0.000	0.997	1.000	0.998
p = 0.05	0.999	0.001	0.000	0.999	1.000	0.999
p = 0.01	1.000	0.000	0.000	1.000	1.000	1.000

5.4 Latency Evaluation

In our prototype, we implement the LSTM on an Nvidia 1080Ti GPU on a server. The server is equipped with 2 Intel Xeon E5-2667 2.90GHz CPUs, each with 6 cores. The level 1 instruction and data caches are 32 KB, the level 2 cache is 256 KB and the level 3 cache is 16MB. The OS we are using is Ubuntu 14.04.5 LTS.

We list the training and inference times obtained with the above setup in Table 6. Note that the training is typically done only once, and needs to be repeated only if the functionality of the control code for the PLC is modified. In each inference time period, we can infer multiple PLC sequences in parallel. We can parallelize 256 PLC sequences, the degree of parallelization depends on the GPU we are running on. For each time frame, we need to predict 2000 time stamps in the future and subsample it, to ensure the independence required by KS test samples. Thus, the time taken to detect an attack is 2 seconds (2000 / 1 kHz), i.e. if an attack happens at t seconds, it can be discovered at $t + 2$ seconds as anomalous PLC behavior. In a real implementation, a trusted deep learning accelerator can be used (Figure 5), which will give faster execution time and greater security through hardware isolation.

Table 6: Execution time for model training and inferencing, and KS testing, for detecting the controller hijacking attacks.

	Training	Inference	KS Test
LSTM	21.1 min	39.8 ms	8.63 ms
CRBM	3.5 min	265 ms	8.63 ms

6 RELATED WORK

Past work have also proposed anomaly detections for power-grid systems. Manandhar [31] used Kalman filters to model the voltage sensors and χ^2 test to detect physical anomalies in the power-grid system. Valenzuela [50] proposed Principle Component Analysis (PCA) based smart-grid attack detection approaches. Cardenas [9] suggested using linear models and threshold random walk (TRW) to deal with the unseen attacks. However, their proposed models may be too simple to represent realistic power-grid systems. Mitchell [33] set behavior rules for devices in the power-grid systems. Zhao [59] proposed an expert system with Hidden Semi-Markov model. Their system requires manually setting rules and the interaction between experts, respectively. Thus they could be biased by the human experience of the known attacks.

Previous defenses on cyber attacks on the power-grid systems focused on network and data communication security. Wang [55] protected the wireless communication in power-grid systems by proposing a secure wireless communication architecture. Fouda [15] proposed an authentication scheme for communications in the power-grid system. Sikdar [41] protected the anonymity of data to defend against the power-grid traffic analysis attacks. Each of these solved a specific loophole of secure communication in the power-grid system, but can hardly guarantee the detection of new attacks. Dolgikh [13] proposed customized normalcy profiles to protect the power-grid systems. However, they both assumed that there are only simple applications running in the system. Gonzalez [17] and Stone [46] used power and Radio Frequency (RF) side channel to detect the controller anomalies. However, the controller behavior can hardly be accurately monitored through the side-channels. To the best of our knowledge, we are the first to discuss using deep learning on the low-level hardware features to automatically learn the controller's normal behavior, and provide a low-cost, high tamper-resilience and comprehensive protection of the controller in the power-grid system.

7 CONCLUSION

In this paper, we propose a new data-driven methodology to detect zero-day controller hijacking attacks on power-grid systems. We observe the controller behavior in power-grid systems is predictable in terms of hardware performance counters, and verify this using a large-scale experiment on a real power-grid controller subsystem. Our method leverages deep temporal models, enhanced with the statistical test. A key advantage of our method is that only normal data is used, and no prior-knowledge about the attack data is needed, thus enabling detection of zero-day attacks. We evaluate our detection system on a real programmable logic controller with an ARM Cortex A8 processor used in power-grid substations. We show the significant improvement of using enhanced deep learning compared to 11 conventional anomaly detection approaches. Experimental results show that our proposed system achieves over 99.9% accuracy with nearly zero false positives.

Acknowledgement

We thank Prashanth Krishnamurthy, Farshad Khorrami and Ramesh Karri from NYU for giving us access to the raw data on the baseline behavior of the PLC. This material is based on work supported partially by NSF #1526399.

REFERENCES

[1] 2010. Programmable Logic Controller Block Diagram. https://www.youtube.com/watch?v=oOCW5muXNyo.

[2] 2015. December 2015 Ukraine power grid cyberattack. https://en.wikipedia.org/wiki/December_2015_Ukraine_power_grid_cyberattack.

[3] 2015. Power Grid Cyber Attacks Keep the Pentagon Up at Night. https://www.scientificamerican.com/article/power-grid-cyber-attacks-keep-the-pentagon-up-at-night/.

[4] 2015. Understanding LSTM Networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs.

[5] 2016. The Efficacy and Challenges of SCADA and Smart Grid Integration. https://www.csiac.org/journal-article/the-efficacy-and-challenges-of-scada-and-smart-grid-integration/.

[6] 2018. Kolmogorov-Smirnov Test. https://en.wikipedia.org/wiki/Kolmogorov_Smirnov_test.

[7] Mohammad Bagher Bahador, Mahdi Abadi, and Asghar Tajoddin. 2014. Hpcmalhunter: Behavioral malware detection using hardware performance counters and singular value decomposition. In *Computer and Knowledge Engineering (ICCKE), 2014 4th International eConference on*. IEEE, 703–708.

[8] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. 2000. LOF: identifying density-based local outliers. In *ACM sigmod record*, Vol. 29. ACM, 93–104.

[9] Alvaro A Cárdenas, Saurabh Amin, Zong-Syun Lin, Yu-Lun Huang, Chi-Yen Huang, and Shankar Sastry. 2011. Attacks against process control systems: risk assessment, detection, and response. In *Proceedings of the 6th ACM symposium on information, computer and communications security*. ACM, 355–366.

[10] Miguel A Carreira-Perpinan and Geoffrey E Hinton. 2005. On contrastive divergence learning.. In *Aistats*, Vol. 10. 33–40.

[11] Defense Use Case. 2016. Analysis of the cyber attack on the Ukrainian power grid. *Electricity Information Sharing and Analysis Center (E-ISAC)* (2016).

[12] John Demme, Matthew Maycock, Jared Schmitz, Adrian Tang, Adam Waksman, Simha Sethumadhavan, and Salvatore Stolfo. 2013. On the feasibility of online malware detection with performance counters. In *ACM SIGARCH Computer Architecture News*, Vol. 41. ACM, 559–570.

[13] Andrey Dolgikh, Tomas Nykodym, Victor Skormin, and Zachary Birnbaum. 2012. Using behavioral modeling and customized normalcy profiles as protection against targeted cyber-attacks. In *International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security*. Springer, 191–202.

[14] Asja Fischer and Christian Igel. 2012. An introduction to restricted Boltzmann machines. *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications* (2012), 14–36.

[15] Mostafa M Fouda, Zubair Md Fadlullah, Nei Kato, Rongxing Lu, and Xuemin Sherman Shen. 2011. A lightweight message authentication scheme for smart grid communications. *IEEE Transactions on Smart Grid* 2, 4 (2011), 675–685.

[16] Markus Goldstein and Andreas Dengel. 2012. Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm. *KI-2012: Poster and Demo Track* (2012), 59–63.

[17] Carlos Aguayo Gonzalez and Alan Hinton. 2014. Detecting malicious software execution in programmable logic controllers using power fingerprinting. In *International Conference on Critical Infrastructure Protection*. Springer, 15–27.

[18] Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. 2013. Hybrid speech recognition with deep bidirectional LSTM. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*. IEEE, 273–278.

[19] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, 6645–6649.

[20] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*. 1693–1701.

[21] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[22] Abdulmalik Humayed, Jingqiang Lin, Fengjun Li, and Bo Luo. 2017. Cyber-physical systems securityâĂŤA survey. *IEEE Internet of Things Journal* 4, 6 (2017), 1802–1831.

[23] Farshad Khorrami, Prashanth Krishnamurthy, and Ramesh Karri. 2016. Cybersecurity for control systems: A process-aware perspective. *IEEE Design & Test* 33, 5 (2016), 75–83.

[24] Hans-Peter Kriegel, Arthur Zimek, et al. 2008. Angle-based outlier detection in high-dimensional data. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 444–452.

[25] Denis Foo Kune, John Backes, Shane S Clark, Daniel Kramer, Matthew Reynolds, Kevin Fu, Yongdae Kim, and Wenyuan Xu. 2013. Ghost talk: Mitigating EMI signal injection attacks against analog sensors. In *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE, 145–159.

[26] Ralph Langner. 2011. Stuxnet: Dissecting a cyberwarfare weapon. *IEEE Security & Privacy* 9, 3 (2011), 49–51.

[27] Edward A Lee. 2015. The past, present and future of cyber-physical systems: A focus on models. *Sensors* 15, 3 (2015), 4837–4869.

[28] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. IEEE, 413–422.

[29] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. 2016. Lstm-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint arXiv:1607.00148* (2016).

[30] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. 2015. Long short term memory networks for anomaly detection in time series. In *Proceedings*. Presses universitaires de Louvain, 89.

[31] Kebina Manandhar, Xiaojun Cao, Fei Hu, and Yao Liu. 2014. Detection of faults and attacks including false data injection attack in smart grid using kalman filter. *IEEE transactions on control of network systems* 1, 4 (2014), 370–379.

[32] Yajie Miao, Mohammad Gowayyed, and Florian Metze. 2015. EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding. In *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*. IEEE, 167–174.

[33] Robert Mitchell and Ray Chen. 2013. Behavior-rule based intrusion detection systems for safety critical smart grid applications. *IEEE Transactions on Smart Grid* 4, 3 (2013), 1254–1263.

[34] Sibin Mohan, Stanley Bak, Emiliano Betti, Heechul Yun, Lui Sha, and Marco Caccamo. 2013. 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*. ACM, 65–74.

[35] Meltem Ozsoy, Caleb Donovick, Iakov Gorelik, Nael Abu-Ghazaleh, and Dmitry Ponomarev. 2015. Malware-aware processors: A framework for efficient online malware detection. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. IEEE, 651–661.

[36] Meltem Ozsoy, Khaled N Khasawneh, Caleb Donovick, Iakov Gorelik, Nael Abu-Ghazaleh, and Dmitry Ponomarev. 2016. Hardware-Based Malware Detection Using Low-Level Architectural Features. *IEEE Trans. Comput.* 65, 11 (2016), 3332–3344.

[37] Nisarg Patel, Avesta Sasan, and Houman Homayoun. 2017. Analyzing hardware based malware detectors. In *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 25.

[38] Peter J Rousseeuw and Katrien Van Driessen. 1999. A fast algorithm for the minimum covariance determinant estimator. *Technometrics* 41, 3 (1999), 212–223.

[39] Arvind Seshadri, Mark Luk, Ning Qu, and Adrian Perrig. 2007. SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes. In *ACM SIGOPS Operating Systems Review*, Vol. 41. ACM, 335–350.

[40] Arvind Seshadri, Mark Luk, Elaine Shi, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. 2005. Pioneer: verifying code integrity and enforcing untampered code execution on legacy systems. In *ACM SIGOPS Operating Systems Review*, Vol. 39. ACM, 1–16.

[41] Biplab Sikdar and Joe H Chow. 2011. Defending synchrophasor data networks against traffic analysis attacks. *IEEE Transactions on Smart Grid* 2, 4 (2011), 819–826.

[42] DG Simpson. 1997. Introduction to Rousseeuw (1984) Least Median of Squares Regression. In *Breakthroughs in Statistics*. Springer, 433–461.

[43] Baljit Singh, Dmitry Evtyushkin, Jesse Elwell, Ryan Riley, and Iliano Cervesato. 2017. On the detection of kernel-level rootkits using hardware performance counters. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM, 483–493.

[44] Siddharth Sridhar, Adam Hahn, and Manimaran Govindarasu. 2012. Cyber–physical system security for the electric power grid. *Proc. IEEE* 100, 1 (2012), 210–224.

[45] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. 2015. Unsupervised learning of video representations using lstms. In *International conference on machine learning*. 843–852.

[46] Samuel Stone and Michael Temple. 2012. Radio-frequency-based anomaly detection for programmable logic controllers in the critical infrastructure. *International Journal of Critical Infrastructure Protection* 5, 2 (2012), 66–73.

[47] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*. 3104–3112.

[48] Graham W Taylor, Geoffrey E Hinton, and Sam T Roweis. 2007. Modeling human motion using binary latent variables. In *Advances in neural information processing systems*. 1345–1352.

[49] Graham W Taylor, Geoffrey E Hinton, and Sam T Roweis. 2011. Two distributed-state models for generating high-dimensional time series. *Journal of Machine Learning Research* 12, Mar (2011), 1025–1068.

[50] Jorge Valenzuela, Jianhui Wang, and Nancy Bissinger. 2013. Real-time intrusion detection in power system operations. *IEEE Transactions on Power Systems* 28, 2 (2013), 1052–1062.

[51] Shuohang Wang and Jing Jiang. 2015. Learning natural language inference with LSTM. *arXiv preprint arXiv:1512.08849* (2015).

[52] Xueyang Wang, Sek Chai, Michael Isnardi, Sehoon Lim, and Ramesh Karri. 2016. Hardware Performance Counter-Based Malware Identification and Detection with Adaptive Compressive Sensing. *ACM Transactions on Architecture and Code Optimization (TACO)* 13, 1 (2016), 3.

[53] Xueyang Wang and Ramesh Karri. 2013. Numchecker: Detecting kernel control-flow modifying rootkits by using hardware performance counters. In *Proceedings of the 50th Annual Design Automation Conference*. ACM, 79.

[54] Xueyang Wang, Charalambos Konstantinou, Michail Maniatakos, and Ramesh Karri. 2015. Confirm: Detecting firmware modifications in embedded systems using hardware performance counters. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. IEEE Press, 544–551.

[55] Xudong Wang and Ping Yi. 2011. Security framework for wireless communications in smart distribution grid. *IEEE Transactions on Smart Grid* 2, 4 (2011), 809–818.

[56] Li Wei, Nitin Kumar, Venkata Nishanth Lolla, Eamonn J Keogh, Stefano Lonardi, and Chotirat (Ann) Ratanamahatana. 2005. Assumption-Free Anomaly Detection in Time Series.. In *SSDBM*, Vol. 5. 237–242.

[57] Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei-Hao Su, David Vandyke, and Steve Young. 2015. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. *arXiv preprint arXiv:1508.01745* (2015).

[58] Yichi Zhang, Lingfeng Wang, Weiqing Sun, Robert C Green II, and Mansoor Alam. 2011. Distributed intrusion detection system in a multi-layer network architecture of smart grids. *IEEE Transactions on Smart Grid* 2, 4 (2011), 796–808.

[59] Feng Zhao, Guannan Wang, Chunyu Deng, and Yue Zhao. 2014. A real-time intelligent abnormity diagnosis platform in electric power system. In *Advanced Communication Technology (ICACT), 2014 16th International Conference on*. IEEE, 83–87.

[60] Wentao Zhu, Cuiling Lan, Junliang Xing, Wenjun Zeng, Yanghao Li, Li Shen, Xiaohui Xie, et al. 2016. Co-Occurrence Feature Learning for Skeleton Based Action Recognition Using Regularized Deep LSTM Networks.. In *AAAI*, Vol. 2. 8.

APPENDIX

A. Availability of HPCs

HPCs have been widely implemented on most commodity processors, include Intel, ARM and PowerPC.

An Intel processor provides various CPU performance measurements in real-time in HPCs. It supports not only "core", but also "uncore", performance counters. Core refers to the processor, its general-purpose registers and its private caches. Uncore refers to the Level-2 (L2) or Level-3 (L3) cache shared between cores, the integrated memory controllers, and the Intel QuickPath Interconnect to the other cores and shared system components and the I/O hub. Examples of the core HPCs are the number of: instructions retired, elapsed core clock ticks, L2 cache hits and misses, L3 cache misses and hits, and the core frequency. Examples of the uncore HPCs are the number of: bytes read from memory controller(s), bytes written to the memory controller(s), and data traffic transferred by the Intel QuickPath Interconnect links.

There are also many performance event counters provided by an HPCs in an ARM processor. For example, the ARM Cortex-A series processor provides about 30 performance events, including the number of: Level-1 (L1) data cache refill events, or L1 instruction cache refill events, TLB (Translation Lookaside Buffer) refill events, all instructions executed, memory read/write instructions executed, exceptions taken, exceptions returned, software changes of PC, branches executed, branches mis-predicted, external memory accesses, data memory accesses, and instruction cache accesses. However, only a given number of events, e.g. only 4 for Cortex-A8, can be read through the performance monitoring units at the same time.

B. Details on LSTM and CRBM

We describe the detailed structures of LSTM, RBM and CRBM.

Figure 10 shows the inner structure of LSTM [4]. A basic LSTM cell has three gates that control information flow. The three gates are the forget gate f_t, input gate i_t and output gate o_t. f_t controls how much previous information remains in the memory. i_t controls how much new information is stored in the memory, and o_t controls how much information flows to the output. We define x_t as the input vector. There are also two state vectors, i.e., cell vector c_t and output vector h_t:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{5}$$
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{6}$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{7}$$
$$c_t = f_t .* c_{t-1} + i_t .* tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{8}$$
$$h_t = o_t .* tanh(c_t) \tag{9}$$

where .∗ is element-wise multiplication and σ is the sigmoid function.

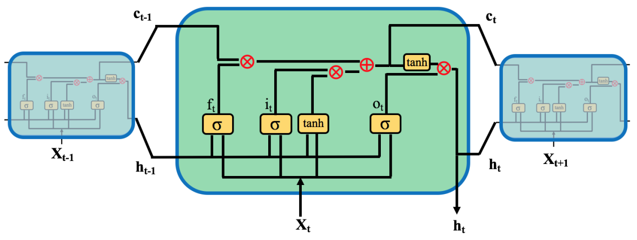

Figure 10: LSTM layers[4]

Restricted Boltzmann Machine (RBM) is another generative neural network which is able to learn a probability distribution. Figure 11 left shows the basic structure of an RBM with m hidden units and n visible units [14]. Both hidden and visible units can only take values of zeros and ones. The probability of taken zeros and ones are given by the following equations (H denotes hidden units and V denotes visible units):

$$p(H_i = 1|v) = \sigma(\sum_{j=1}^{m} w_{ij}v_j + c_i) \tag{10}$$

$$p(V_j = 1|h) = \sigma(\sum_{i=1}^{n} w_{ij}h_i + b_j) \tag{11}$$

where σ is a sigmoid function. For RBM which takes real value inputs, the model can be extended to:

$$p(H_i = 1|v) = \sigma(\sum_{j=1}^{m} w_{ij}v_j + c_i) \tag{12}$$

$$p(V_j|h) = N(b_i + \sum_{i=1}^{n} w_{ij}h_i, 1) \tag{13}$$

where $N(\mu, \sigma)$ is a Gaussian function. Contrastive Divergence (CD) is a widely used approximation algorithm for training the RBM [10].

However, the vanilla RBM does not consider any temporal information and thus is not feasible to be used as the predictor. Conditional RBM (CRBM) [48] is an extension of the original RBM by taking history information into consideration. Figure 11 right shows the structure of a CRBM.

$$a_{i,t} = a_i + \sum_k A_{ki}v_{k,<t} \tag{14}$$

$$b_{j,t} = b_j + \sum_k B_{kj}v_{k,<t} \tag{15}$$

The effect of the history information on hidden units (and visible units) can be viewed as a dynamic bias $b_{j,t}$ ($a_{i,t}$) instead of b_j (a_i) in the above equations [49].

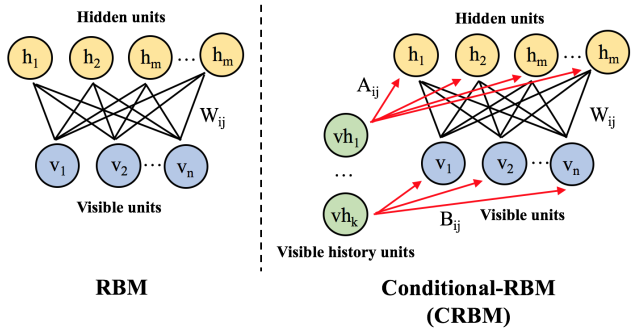

Figure 11: Left: The undirected graph of an RBM with m hidden and n visible variables [14]. Right: Conditional-RBM introduces history into RBM.