CORGIDS: A Correlation-based Generic Intrusion Detection System

Ekta Aggarwal
University of British Columbia
Vancouver, Canada
ektaa@ece.ubc.ca

Mehdi Karimibiuki
University of British Columbia
Vancouver, Canada
mkarimib@ece.ubc.ca

Karthik Pattabiraman
University of British Columbia
Vancouver, Canada
karthikp@ece.ubc.ca

André Ivanov
University of British Columbia
Vancouver, Canada
ivanov@ece.ubc.ca

ABSTRACT

Cyber-physical systems (CPS) consist of software and physical components which are knitted together and interact with each other continuously. CPS have been targets of security attacks due to their safety-critical nature and relative lack of protection. Specification based intrusion detection systems (IDS) using data, temporal, data temporal and time, and logical correlations have been proposed in the past. But none of the approaches except the ones using logical correlations take into account the main ingredient in the operation of CPS, namely the use of physical properties. On the other hand, IDS that use physical properties either require the developer to define invariants manually, or have designed their IDS for a specific CPS. This paper proposes CORGIDS, a generic IDS capable of detecting security attacks by inferring the logical correlations of the physical properties of a CPS, and checking if they adhere to the predefined framework. We build a CORGIDS-based prototype and demonstrate its use for detecting attacks in the two CPS. We find that CORGIDS achieves a precision of 95.70%, and a recall of 87.90%, with modest memory and performance overheads.

KEYWORDS

Intrusion Detection Systems; Internet-of-Things; Cyber-physical Systems; Security; Generic Intrusion Detection Model

ACM Reference Format:
Ekta Aggarwal, Mehdi Karimibiuki, Karthik Pattabiraman, and André Ivanov. 2018. CORGIDS: A Correlation-based Generic Intrusion Detection System. In *CPS-SPC '18: 2018 Workshop on Cyber-Physical Systems Security and PrivaCy, Oct. 19, 2018, Toronto, ON, Canada.* , 12 pages. https://doi.org/10.1145/3264888.3264893

1 INTRODUCTION

Cyber-physical Systems (CPS) interact closely with their physical environment, e.g., smart grids, smart automobiles. CPS have been targets of security attacks due to their safety-critical nature and relative lack of protection. The advent of interconnected CPS to the Internet (also known as the Internet of Things) has exacerbated their vulnerability as they obviate the need for attackers to have physical access to the CPS. Attacks on CPS such as the smart grid [32], smart cars [8] and smart medical devices [22, 30] have been demonstrated in the recent past. Therefore, there is a compelling need to protect CPS from security attacks.

Intrusion Detection Systems (IDS) have been used for protecting computer systems from security attacks, including CPS [6, 24, 27]. Traditional forms of IDS are signature-based, where signatures of known attacks are compared against the operations of the system to identify attacks. Unfortunately, signature-based IDS are a poor fit for CPS as the attacks are often tailored to each kind of CPS, and hence cannot be described by generic signatures. Further, due to the remote and often disconnected nature of their operation, the attack database in CPS cannot be updated frequently unlike traditional computer systems. Finally, a motivated attacker can launch hitherto unknown attacks against a CPS, thereby evading detection by signature-based schemes.

In contrast to signature-based IDS, anomaly based IDS extract a model of a system's behavior and detect any deviations from the extracted model as an attack. Such IDS do not need an attack database, and can hence detect hitherto unknown attacks. Because CPS have constrained behaviors, it is often straightforward to derive anomaly based IDS for them, making these systems a good match for CPS. Unfortunately, anomaly based systems exhibit high rates of false-positives in practice, as learning a stable model of the system is often challenging. Therefore, some researchers have proposed using physics-based models for anomaly detection models for intrusion detection in CPS [9, 10, 25, 26, 35]. The notion is that because CPS interact closely with their physical environments, they need to follow laws of physics, which can in turn be used as the detection model. Efforts have been made to use the physical properties of the power grid [10, 28], unmanned aircraft vehicles (UAVs) [25] and water treatment systems [1] to build a model which represents the expected behavior of the CPS. However, in the prior work, the IDS is designed specifically for a particular CPS. Therefore, the above solutions cannot be easily generalized to other CPS, as the process of finding an appropriate model is both time consuming and effort intensive for developers.

CPS-SPC '18, October 19, 2018, Toronto, ON, Canada

ACM ISBN 978-1-4503-5992-4/18/10…$15.00
https://doi.org/10.1145/3264888.3264893

In this paper, we consider the *logical correlations* among the physical properties of the CPS as the model for anomaly-based IDS. Our hypothesis is that physical properties exhibit deterministic and predictable correlations among themselves, as they have to adhere to the laws of physics. For example, consider the case of an Unmanned Aerial Vehicle (UAV), which needs to follow Newton's laws of motion during flight. Some physical properties of an UAV are: distance travelled, altitude, speed, battery life left and flight time. When an UAV is flying at a fixed altitude, it has a non-zero speed due to which the distance traveled and flight time increases, while the battery life left in the UAV decreases. These relationships encompass the logical correlations among the physical properties of the UAV. If during flight it is observed that the battery life left in the UAV is not decreasing while the speed of the UAV is non-zero, this would imply that there is some anomaly in the system, which potentially indicates an attack.

This paper proposes a generic intrusion detection system capable of detecting security attacks by inferring the logical correlations of the system and checking if they adhere to a predefined framework. We use HMMs to automatically infer the logical correlations among the physical properties of the system with no a priori knowledge of the physical laws adhered to by the system or any intervention by the programmer. The HMM identifies a state as malicious by detecting either an undesired data correlation or lack of an expected data correlation among its physical properties. We use HMMs as they are good at detecting outliers, and are typically used to model time-based systems (Section 3.2). Though other papers have used logical correlations to detect anomalies [9, 19, 19, 21, 21, 35], none of them have used HMMs as the core to build a generic IDS. *To the best of our knowledge, CORGIDS is the first generic intrusion detection system which uses HMMs to infer logical correlations exhibited by the system to determine if an intrusion has occurred.*

Our contributions are:

(a) We propose the use of logical correlations among the physical properties of a CPS to detect intrusions, and use HMMs to infer the logical correlations.
(b) We designed a CORrelation based Generic Intrusion Detection System (CORGIDS), using HMMs to detect intrusions. We demonstrate its use in two CPS, namely i) an UAV, and ii) a smart artificial pancreas platform.
(c) We evaluate the effectiveness of CORGIDS by performing five targeted attacks on the above mentioned CPS. We find that CORGIDS is successfully able to detect all five attacks, and has lower false-positive and false-negative rates than other intrusion detection techniques.

2 RELATED WORK

We begin by reviewing different techniques for creating anomaly-based IDS. Based on the models created, IDS can be categorized into three main classes, a) *data invariants*, which aims to use the values of data variables to generate the model; b) *temporal invariants*, which uses the sequence of events in a given system to create the model; and (c) *physical invariants*, which uses the physical properties to create a multi-dimensional model. We discuss these invariants below.

Data Invariants: Significant work [4, 5, 11, 12, 15] has been done to determine how to extract data invariants from a system. Ernst et al. [15] built Daikon to dynamically mine data invariants of a system, thus creating pre- and post-conditions which hold at every entry and exit of a method/function. Csallner et al. [12] propose DySy to extract data invariants by dynamically executing test cases and simultaneously performing symbolic execution of the program under study. In subsequent work, Csallner [11] designed DSD-Crasher to determine a program's intended behavior for automatically generating test cases and finding bugs. Baliga et al. [4, 5] proposed Gibraltar, for inferring and enforcing data invariants to detect rootkits in the operating system's kernel.

Temporal Invariants: Temporal invariants have been used to get a better understanding of a system, uncover bugs and to build IDS. Yang et al. [34], define their model, Perracotta, to take as input a program and dynamically output temporal invariants. Gabel and Su [17] built Javert which is configured with two basic predefined patterns of temporal invariants. In subsequent work [18] they built a tool OCD, which is capable of analyzing the trace continuously using a sliding window concept to generate invariants. Beschastnikh et al. [7] generate temporal invariants dynamically through the use of system logs (traces) and programmer-specified regular expressions. Lemieux et al. [23] dynamically generate all the instantiations of the invariants from a log file and the property types supplied in their tool called TEXADA.

Physical Invariants: Approaches from the prior work which use physical properties of the CPS, to generate invariants can be classified into those that manually define physical invariants of the CPS, and those that generate the invariants automatically from the CPS's behavior. The second approach is more useful than the first one, as it reduces the developer effort and time.

- **Manually defined physical invariants**: Mitchell and Chen [25] aim to secure an UAV by specifying the physical invariants for each sensor and actuator embedded inside the system. In subsequent work, they designed an adaptive specification based IDS [26] called BRUIDS, which could be adapted based on the attacker type and environment changes. Similarly, Choudhari et al. [10] manually describe scheduling invariants and physical invariants in the form of Lyapunov functions. Combining these invariants they produce cooperating invariants which specify and maintain the stability of the system. In another work, Paul et al. [28] represent a CPS with one system invariant which encapsulates all of its subsystems. Adepu and Mathur [1] design an IDS for a water treatment plant by manually describing the invariants for a particular sensor in terms of the water level changes between two consecutive readings.
- **Automatically generated physical invariants**: Chen et al. [9] dynamically generate the physical invariant which is a Support Vector Machine (SVM) model. This SVM model is then used to classify an activity as benign or malicious for a real-world water purification plant. However, as they use statistical model checking, they only provide probabilistic guarantees that the system is correct, leaving room for false positives and false negatives. Zohrevand et al. [35] dynamically generated the physical invariant which was a hidden

semi-Markov model for a water supply system. Though they based their approach on data collected from a real water supply system, their model was specialized for a specific CPS. In recent work, Aliabadi et al. [2] designed ARTINALI, which dynamically mined data, time and event invariants from an execution trace of the program. They used the invariants in an IDS, but did not consider physical properties other than time when generating the invariants. Krotofil et al. [21] used correlations to identify anomalies in the Tennessee Eastman (TE) process challenge (a realistic simulation of a chemical process). They used the Pearson correlation coefficient for deriving the cluster entropy, which is highly sensitive to outliers, and does not work well with non-linear data unlike HMMs. Also, they rely on physical placement of the sensors for the effectiveness of their approach. On the other hand, Iturbe et al. [19] use HMMs to distinguish between malicious attacks and natural disturbances for TE process challenge using logical correlations. In contrast, we focus on attack detection rather than diagnosis.

Summary: There has been significant prior work to use data and temporal invariants for intrusion detection. Unfortunately, the former class of systems incur high false-positives and false-negatives, thus implying unreliable detection. Physical invariants have the capacity to detect security attacks with low false positives and negatives, but current work either requires the invariants to be manually specified, which is time and effort intensive, or the systems have important gaps which inhibit their generalizability. In this paper, we propose an automated technique for capturing the logical correlations among physical variables in a generic CPS, and use such correlations for detecting intrusions.

3 APPROACH

We first introduce our approach for building a generic IDS. We begin by presenting the threat model. Then we introduce HMMs which we employ to find the correlation between the logical properties of the system to detect intrusions. Secondly, we present the work-flow of CORGIDS and an example to illustrate the work-flow.

3.1 Threat Model

Access: We use the term System Under Test (SUT) to represent the system on which we want to perform our analysis. We assume that the attacker has the capability to gain read and write access to the communication channel between the SUT and the controller. Using this access, the attacker can modify the contents or add data packets being transferred. This assumption is realistic as previous work [14] has shown that such access is rather easy to get.

Further, we assume that the attacker has the *root* access to the SUT, which means that the application code can be modified to suit the attacker's needs. However, as an attacker is likely to want to remain stealthy, we assume that they are more likely to make small changes to the program rather than large-scale changes such as replacing the entire program with their own. We also assume the attacker cannot modify the operating system kernel or the device firmware. This can be ensured by using code signing or trusted computing hardware if it is available.

Capabilities: We assume that the attacker, using access to the communication channel, can perform two types of attacks. The first one is spoofing, where the contents of the data packets can be modified, and the second one is flooding where the number of data packets being sent to the controller can be increased. The attacker can also perform physical attacks on the CPS, for example by rebooting it, at arbitrary points in time. For this paper, we are *not* considering network attacks such as Denial-of-Service (DoS) or message dropping attacks. Also, we only consider attacks that change the correlation between the logical properties. Attacks which do not create an impact on the logical properties are not considered.

3.2 Hidden Markov Models

We build an IDS using an HMM to find logical correlations among the physical variables in a system. HMMs are useful for systems which can be represented by sequences or time series. An HMM is a finite model that can be used to describe a probability distribution over an infinite number of possible sequences in a given system [13]. Unlike a simple Markov model, an HMM is composed of a number of hidden states. Each hidden state 'emits' symbols according to emission probabilities, and the states are interconnected by state-transition probabilities. Starting from an initial state, a sequence of states is generated by moving from state to state according to the state-transition probabilities until an end state is reached. Each state then emits symbols according to that state's emission probability distribution, creating an observable sequence of symbols. More formally, an HMM can be represented by (π, A, θ) where π represents the starting probability of the transitions between the hidden states, while the transition probability matrix is denoted by A and θ represents the emission probability of the hidden states.

HMMs are a good fit for problems in which i) the model parameters and observed data are present, and there is a need to estimate the sequence of hidden states; ii) the observed data is given and the model parameters are to be estimated, and iii) the information of model parameters and observed data is present while there is a need to find the likelihood of the data. In CORGIDS, we use HMMs for the third kind of problem, i.e., we measure the likelihood of the current observed data belonging to the predefined model's parameters. In order to do so, we feed the values of correlated physical properties of the system into an HMM, which then infers the correlations between them. These correlations are then used to determine the likelihood of the current observed data as stemming from the model and its parameters. Any deviation is signaled as an anomaly and a possible security attack.

We use HMMs as the core of intrusion detection module mainly because they are capable of finding data patterns in high dimensional, non-linear time series based data systems. Also, HMMs work by creating hidden states and then transitioning between them which is very similar to the operations of CPS system, which are typically modeled as state machines. Unlike techniques such as correlation coefficients, HMMs are also highly resilient to noise and outliers. For instance, Krotofil et al. [21] use Pearson Correlation Coefficient(PCC) to determine correlation for the cluster entropy. PCC measures linear correlation among the variables, therefore is not suitable for multidimensional non-linear data. Also the variables undergoing PCC must be either based on interval or ratio scale,

making this approach much less generic. Chen at el. [9] employ SVMs to detect an anomaly in a time series based system. Unfortunately, SVMs do not work well with time series data, because they work with a snapshot of the state and classify it into a class. However, by manipulating the input feature vector to the SVM in such a way that it encapsulates the time factor, authors use it for anomaly detection. On the other hand, Aliabadi et al. [2] use Frequent Item Set Mining algorithm which does not model the system, but mines the data under different events. Unfortunately, they do not consider the physical properties, except time, of the CPS. Iturbe et al. [19] use Principal Component Analysis (PCA) which infers correlation among the variables and is better suited for linear correlations, as it works by generating orthogonal projections. However, for non-linearly correlated data as in our case, PCA is not able to find correlations.

3.3 Work-flow of CORGIDS

CORGIDS is a generic intrusion detection system which exploits the correlation of the logical properties of the SUT to detect intrusions. Figure 1 shows the key components and the work flow of CORGIDS.

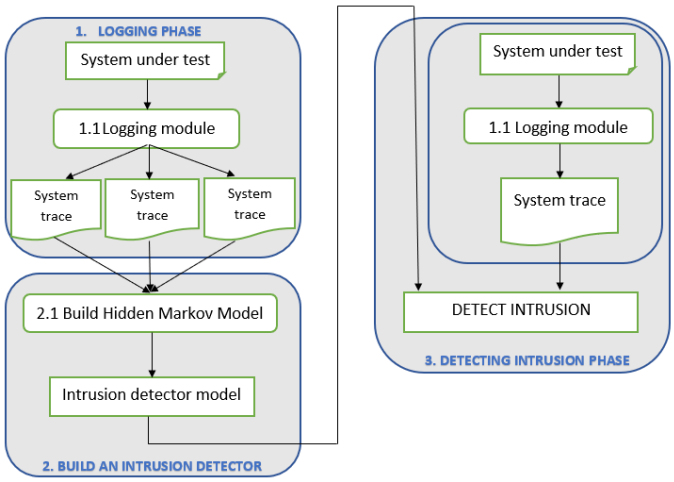


Figure 1: Workflow of CORGIDS

The CORGIDS workflow can be broken down into three main phases, namely, a) Logging Phase; b) Building an Intrusion Detector Phase; and c) Detecting Intrusion Phase. We explain each of the phases below.

(a) Logging Phase: The *1. Logging Phase* in Figure 1 is the starting point for building an intrusion detector and for deploying it on a system for which intrusion detection is desired. SUT is an input to this phase and is passed through the *1.1 Logging module* in which it is manually instrumented to collect the values of the correlated properties[1]. These properties are chosen by the user based on her general knowledge of the SUT. The Logging phase will ensure that the traces which contain the values of the properties while the system is running are collected. Also, we assume that the source code of the SUT is available and can be modified for instrumentation - this is reasonable as the developer of the system will deploy CORGIDS. At the end of this *logging phase*, the system traces containing the values of the logical properties of the SUT are collected.

(b) Building an Intrusion Detector: In this phase, the system traces collected from the *Logging Phase* are used to build an HMM which behaviorally represents the SUT. The pseudo code for the algorithm for building an intrusion detector is below. To build an intrusion detector the system traces are fed into the HMM model for its training in Line 2.

Algorithm Building an intrusion detector

```
1:  procedure BuildAnIntrusionDetector (logs):
2:      trainedModel = trainHMMModel (logs)
3:  for l in logs
4:      logLikelihood(i) = log(trainedModel(l))
5:      S = sum of all logLikelihood(i)'s
6:  M = mean of S
7:  return trainedModel, M;
8:  procedure trainHMMModel (logs):
9:  for hiddenStates = 2, hiddenStates++
10:     create an HMM model model(i);
11:     logLikelihood(i) = log(model(i))
12:     if δ (logLikelihood) < Threshold then
13:         return model(i);
```

We begin training an HMM in *procedure TRAINHMMMODEL* in Line 8, by varying the number of *hiddenStates*. The number of hidden states is a free parameter of an HMM which needs tuning in order to create a model which can be used for intrusion detection. We iterate and create HMM's *model(i)* with the starting value of two hidden states and keep increasing the number of hidden states by one (Line 9 - 10). The log likelihood of the *model(i)* is calculated which represents the goodness of the *model(i)* fit of the model to the data that was used for constructing it. The log likelihood is stored in *logLikelihood(i)* as shown in Line 11. The *threshold* in Line 12 represents the minimum difference between the current and previous HMM's log likelihood. Using threshold as a stopping criteria for HMM has also been used in prior work [16]. The best HMM is returned in Line 13 with its parameters as the model to be used for intrusion detection. At this point the *trainedHMMModel* is used to calculate the log likelihood for each of the training system traces (Line 3). As a by product, the mean of log likelihood is calculated (Line 5- 6) to get the estimated log likelihood value M (Line 6) for a training log. M is then used for comparison in the later steps to detect intrusion. Creating HMMs by increasing the number of hidden states uses a significant amount of memory and computational power. However as this phase needs to be carried out just once for a SUT, it is not a major bottleneck. Once the intrusion detector HMM is created, it can be used to detect an anomaly in the next phase, which is much more efficient.

[1]The approach of manually instrumenting the code to collect logs has been used by prior work [2, 9]

(c) Detecting Intrusion Phase: This phase starts with the *Logging Phase* which is used to collect the system trace from the SUT while it is running. In this phase, only the system trace corresponding to the running SUT is collected, rather than many different system traces. The trace generated is then used together with the intrusion detector built in the *Building an Intrusion Detector Phase*. Using the HMM model and its saved parameters, the log likelihood of the current system trace is calculated and compared with the mean log likelihood M calculated in the *Building an Intrusion Detector* in Line 6. If the log likelihood of the system trace is less than a specified range (δ) from M, it signifies that the system trace does not follow the behavior which was observed when the HMM was being trained. The specified range (δ) is found by running a sensitivity analysis (Section 6.1). Further, as the system traces used for building the HMM were assumed to be correct (i.e., not attacked), this implies that the current system trace represents a system under attack. Thus, we flag the current state of the SUT to be malicious.

3.4 Example

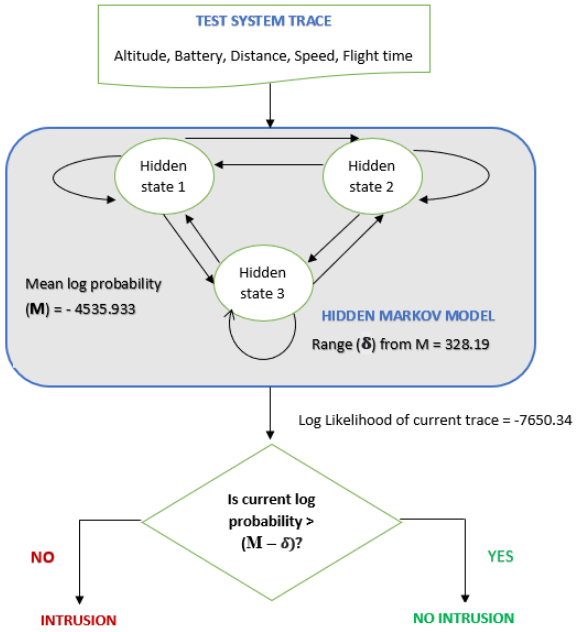


Figure 2: Approach of CORGIDS

We use our earlier example of an UAV from Section 1 to illustrate how CORGIDS can be used to detect intrusions in Figure 2. As described in Section 1, an UAV has physical properties such as the current altitude, battery percentage left, distance traveled, current speed and flight time. These physical properties are correlated to each other as per the laws of physics. We now elaborate how our approach will detect an intrusion in an UAV using the work-flow described in Figure 2. We use a distance spoofing scenario for the UAV as an example.

Table 1: Slice of a non-faulty system trace obtained while flying an UAV on a random route

Altitude (m)	Battery left (%)	Distance travelled (m)	Speed (m/s)	Flight time (s)
..	..	..	..	..
40	89	42.1445	1	38.32
40	89	44.2563	2	39.342
40	89	47.2397	3	40.356
40	89	51.0202	3	41.376
40	88	55.2434	4	42.345
40	88	59.5897	4	43.346
40	88	64.1632	4	44.335
41	88	68.8979	4	45.323
41	88	73.7389	4	46.351
41	87	78.6564	4	47.448
41	87	83.6196	4	48.551
41	87	88.6138	4	49.61
41	87	93.627	5	50.604
41	86	98.6659	5	51.507
..	..	..	..	..

Table 2: Slice of a faulty system trace obtained while an UAV was flying on a random route and infected by distance spoofing attack

Altitude (m)	Battery left (%)	Distance traveled (m)	Speed (m/s)	Flight time (s)
..	..	..	..	..
40	89	42.7868	1	38.206
40	89	45.2942	2	39.279
41	89	48.6934	3	40.272
42	89	42	4	41.261
42	88	57.0199	4	42.267
43	88	46	4	43.285
43	88	66.0254	4	44.357
44	88	70.7879	4	45.347
44	87	65	4	46.292
45	87	80.5709	4	47.37
46	87	85.5441	4	48.386
46	87	49	4	49.373
47	86	54	4	50.367
47	86	100.6006	4	51.402
..	..	..	..	..

First the *Logging Phase* starts, where the UAV is instrumented to collect the correlated properties such as current altitude, current battery percentage left, distance traveled, current speed and flight time. The above properties are collected at regular intervals of time to form the system traces. A section of the sample system trace collected is shown in Table 1. In the trace, we can observe that all the properties are correlated with each other, and that the correlations are fairly stable. For instance, if the *Speed* of the UAV increases, the *Distance traveled* will also increase. Further, the *Distance traveled* property can have values that are either increasing or stagnant. We run multiple iterations of the UAV by varying the routes it travels, to collect non-faulty system traces from it.

In the second phase, we *Build an Intrusion Detector* by using the system traces collected from *Logging Phase*. We start by generating

model(i) by varying the number of hidden states in line 9 to 10 in the given algorithm. Then for each of the *model(i)* generated, we calculate the *logLikelihood(i)* in line 11 to determine if the *model(i)* fits the data used for constructing it. To accomplish this, the difference in *logLikelihood(i)* is compared to the *Threshold* in line 12 and if the *Threshold* is met, *model(i)* is returned. We found that an HMM with 15 hidden states is the one which meets the threshold. As showing 15 hidden states in the Figure 2 will clutter it, we simplify the model by showing only 3 hidden states. Further, in lines 3 to 5, the sum of *logLikelihood's* for all the correlated logs S is calculated from which M (mean log likelihood) = -4535.933 is extracted.

To demonstrate how an attack will be detected by our approach, we consider an attack where the attacker decides to spoof the values of distance traveled found inside the data packets being transferred from the UAV to the Ground Control Station (GCS). UAV's periodically send the flight data to the GCS to keep it updated about its whereabouts. To intervene the working of UAV, the attacker gains access to the communication channel between the UAV and GCS. Now, an attacker can easily change the contents of the data packets being transferred. This attack is explained in detail in Section 5.

In the final phase, when the UAV is deployed in production, the *Detecting Intrusion Phase* activates in the GCS and uses the current system trace produced from the logging module along with the trained HMM. A slice of the faulty-system trace is shown in Table 2. As can be observed, the values of distance traveled are changing but do not follow the correlations observed in the earlier trace in Table 1. As only the distance traveled values have been tampered with, leaving other logical properties intact, we get a correlation which is different from the one that is expected by the trained HMM. This results in the difference between the mean and current log likelihood values being greater than the threshold value - say (δ). From Figure 2, the log likelihood of the current system trace is more than δ from the M (mean log likelihood). Therefore, CORGIDS flags the current state of the UAV to be *malicious*. The value of the threshold (δ) = 328.19 is determined experimentally (Section 6.1).

4 EXPERIMENTAL SETUP

This section first describes the details of the two testbeds on which CORGIDS is applied, then briefly describes the attacks to be planted. It then discusses the experimental procedures, and how we chose the experimental parameters. Finally, it presents the evaluation criteria which will be used in Section 6.

4.1 Testbeds

To demonstrate the generality of CORGRIDS, we choose two testbeds on which we carry out our experiments. These testbeds are two CPS which contain correlated properties and a predefined framework according to which the properties change their values.

(a) Unmanned Aerial Vehicle (UAV): An UAV, commonly known as a drone, is different from an aircraft mainly because it does not have a pilot aboard. UAV's periodically send the flight data to the GCS to keep it updated about its whereabouts. A UAV mainly consists of sensors, control logic and actuators forming a closed loop. The sensors sense the current state of the UAV and its environment and pass it on to the controller, which makes the decision about the next step to be taken. The decision taken is then sent to the actuator - this loop runs infinitely often while the UAV is flying. We use ArduPilot's Software in the Loop (SITL) [3] to perform our experiments. ArduPilot is an open-source autopilot software and is vastly deployed on various vehicle systems. We use SITL as a simulator on a local machine as we did not have access to a real UAV.

(b) Smart Artificial Pancreas (SAP): SAP is a device used by the diabetic patients to automatically analyze the insulin to be injected to the patient based on the blood glucose samples collected. SAP helps in reducing human error and analyzes the current blood glucose levels regularly at fixed intervals of time. A SAP consists mainly of i) a blood glucose monitor, which reads the blood glucose levels of the patients at regular interval of time, ii) a controller, which based on the blood glucose values decides the insulin that needs to be injected, iii) an insulin pump, which based on the value generated by the controller, injects a specific amount of insulin into the patient. We use Open Artificial Pancreas System (OpenAPS) an open source SAP to evaluate CORGIDS. OpenAPS implements the controller part of the SAP, and has been used in prior studies [2]. We did not have access to a real patient, due to which we use the simulated values from blood glucose monitor and the insulin pump for our experiments. The values of blood glucose instead from the blood glucose monitor, were taken from the test cases provided by OpenAPS. These values were then served as input to the OpenAPS to get the insulin required by the patient. We install the OpenAPS controller on a Raspberry Pi 3 microprocessor to evaluate the memory and performance overhead of CORGIDS.

4.2 Experimental Procedure

To evaluate CORGIDS's efficacy, we partition the process of attack detection into two phases, namely training phase and testing phase. The system traces obtained from SUT are randomly divided into training and testing batches. The training phase is the one in which we train our intrusion detector from the non-faulty system traces that are randomly assigned. Sensitivity analysis is also performed to analyze the parameters which have the most impact on the performance of the intrusion detector. For instance, for the UAV testbed, we randomly generated routes which the UAV used as the flight plan. Therefore, after having the UAV fly on all the randomly generated routes, we obtain the non-faulty system traces. These logs were then randomly distributed for training and testing. In the testing phase, we test the intrusion detector to find out if an intrusion was correctly detected. The results are then used to gauge the performance of CORGIDS based on the evaluation criteria. In order to reduce variability, we ran five-fold cross validation for each of the attacks described above.

5 ATTACKS DESCRIPTION AND DETECTION

In this subsection we discuss the attacks we emulated on each testbed, and their implications. The attacks which we discuss here are targeted attacks, which means that they specifically target physical properties of the two CPS platform. Note that the attacks designed for both the testbeds are intentionally stealthy, i.e. we expect

that the attacker wants to remain undetected while introducing some malicious content to accomplish his goal. Also, we use attack trees for planting attacks on the testbeds. These attack trees are based on prior attacks on very similar systems, thus making them realistic and appropriate for testing CORGIDS.

5.1 Attacks on UAV

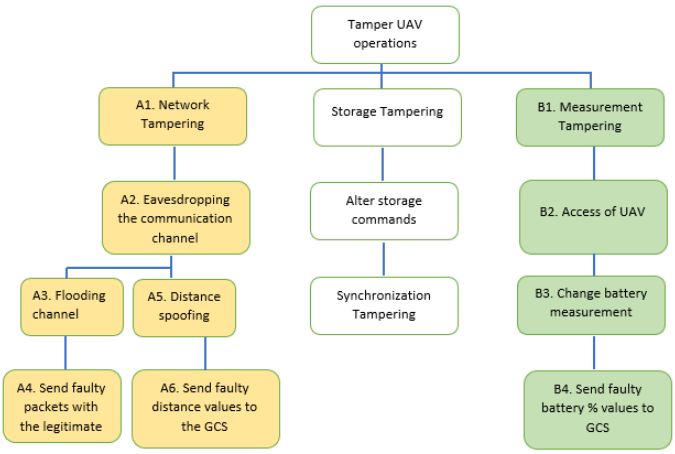


Figure 3: Attack tree for UAV

As discussed above, an UAV regularly transmits flight data to the GCS, so that it can be tracked throughout its flight. The GCS based on the flight data received interprets if the UAV is following the instructed guidelines or has drifted from it. We formulate an attack tree for faulty UAV operations (shown in Figure 3), based on attacks introduced in previous work [20, 25]. There are three branches in this tree, namely, i) Network Tampering; ii) Storage Tampering; and iii) Measurement Tampering. We used two branches to develop attacks which are discussed below.

- **Battery Tampering Attack (Block B1-B4)**: This attack occurs when an attacker is able to tamper with the control logic of the UAV by hacking it. By changing the control logic, the attacker can change the decisions that are made based on the input physical properties from the sensors. Obtaining the access of the UAV is not an unreasonable condition mainly due to the availability of tools capable of achieving the same [29, 31]. In this attack, the attacker can change the part of the code where the percentage of battery left in the UAV is being sent to the GCS. The original value of percentage battery left in the UAV can be substituted with a value greater than the current value, to lead the GCS into the false understanding that the UAV has plenty of battery left in it. Specifically, if the attacker through eavesdropping the communication channel, knows that the battery decreases at a particular rate, he can then send faulty values to make the GCS believe that battery is depleting at a decreased rate to accomplish his motive. Thus eventually, reaching to a point where the UAV crashes on the ground due to battery drainage, and leads to the possession of sensitive data by the attacker. As we did not have access to a real UAV, we performed our experiments on ArduPilot (a real time simulator for UAV) running on a local machine. Therefore, we had access to the UAV and modified its control logic to plant this attack in the code.
- **Flooding Attack (Block A1-A4)**: The flooding attack occurs when the communication channel between the UAV and GCS is compromised. In this scenario, an attacker can mount the attack by flooding the communication channel by the sending the extra packets along with the ones destined to be received by the GCS [29]. The motive of this attack could be populating the channel so that the GCS is unable to infer the correct whereabouts of the UAV thereby, leading the attacker to control and use the UAV as desired. The extra packets being sent can contain physical properties which are different from the legitimate ones. However, we assume that the attacker is stealthy and chooses values close to the real ones to avoid detection. To achieve this attack we injected faulty data packets into the communication channel between UAV and GCS.
- **Distance Spoofing Attack (Block A(1,2,5,6))**: By sending a different value of the distance traveled rather than the original value, an attacker can falsely portray the current route or the current position of the UAV to the GCS. This attack can take place when an attacker eavesdrops on the communication channel to know the format of data being transmitted. This knowledge then can be used to spoof the value of the distance covered in the data packets being sent to the GCS. The motivation behind this attack can be that the attacker wants to fool the GCS by leading it to believe that the UAV is following a different schedule/route than the planned one. This attack is mounted by spoofing false distance traveled data into the communication channel between the GCS and UAV. Similar to flooding attack, we intercepted the communication channel to send spoofed values for the distance traveled property to the GCS.

5.2 Detection of attacks on UAV

- **Battery Tampering Attack**: As detailed in the attack description, the attacker changes only the battery values in a data packet which also contains other correlated properties such as distance traveled, altitude, speed, and flight time. When this data is received by the GCS with CORGIDS enabled on it, the trained HMM model in the intrusion detector module detects an abnormal activity. A malicious activity is detected because the correlation expected by the HMM is not the same as received by it, mainly due to the difference in the relationship of battery with the other properties in the data packet. As a result, the log likelihood of the current system trace comes out to be less than the intrusion detector, which makes it faulty.
- **Flooding Attack**: To detect this attack, the data packets that are received by the GCS are fed into the intrusion detector module of CORGIDS. A key point to note here is that, if the UAV sends one data packet per second to the GCS, the data packets received at the GCS end will be greater because of the flooding attack. The trained HMM model will detect a

malicious activity as the number of data packets which are used for decision making are greater than the case when there is no flooding attack. This will lead to a lesser log likelihood of the current data packets than the trained HMM, thus flagging the current state as anomalous.

- **Distance Spoofing Attack**: When spoofed messages reach the GCS, they are given to the trained HMM model to find out discrepancy, if any. An important thing to note here is that the number of data packets sent by the UAV and received by the GCS are same. However, in some packets the distance traveled by UAV is spoofed to falsely portray that it is following a different route or may be the sensors are returning some faulty values. However, the correlation between the distance traveled and other flight data parameters from faulty and non-faulty packets, is not what is expected by the trained HMM. Thus, the intrusion detector flags the current state to be anomalous as the log likelihood of the data packets fed into it is lesser than expected.

5.3 Attacks on SAP

The correct execution of SAP is of vital importance as the life of the patient depends on it. As discussed above, SAP consists of three components: blood glucose monitor, controller and insulin pump forming a closed loop. The attacks we derive for SAP are discussed below and take advantage of the communication channel and the access of the code for the controller [2]. We build an attack tree shown in Figure 4 using the attacks demonstrated in prior work [2, 30]. We base our attacks on the two scenarios described in it.

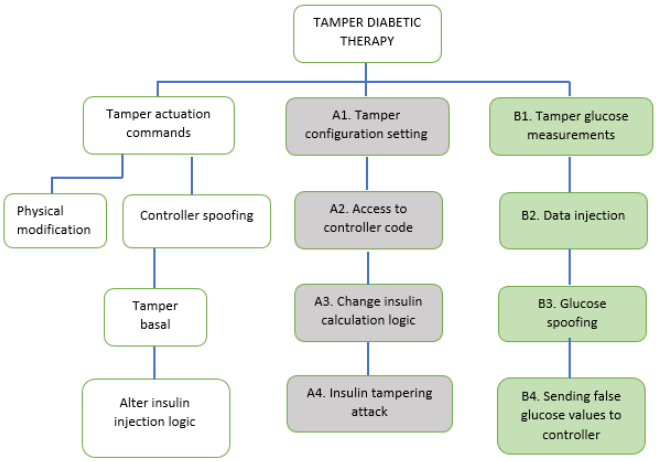


Figure 4: Attack tree for SAP

- **Insulin Tampering Attack (Block A1-A4)**: Similar to battery tampering attack, this attack also occurs when the attacker can hack the controller i.e., OpenAPS [30]. After hacking, the attacker can modify the logic where the rate of insulin is calculated, based on the input blood glucose values sampled from the patient. This will lead to injection of faulty insulin dosage into the patients body which can prove fatal. As we used the Raspberry Pi 3 for our SAP experiments, we remotely connected to it and then changed the control logic to reflect this attack. After the attack had been planted, the insulin dosage command sent out by the controller was faulty as expected.
- **Glucose Spoofing Attack (Block B1-B4)**: The glucose spoofing attack modifies the value of the blood glucose contained in the data packets being sent from the patient. The incorrect value which will be substituted can be either greater than or less than the current blood glucose value. This change in the real value will lead the controller to calculate an incorrect value of insulin (though the logic through which insulin dose calculated is untouched), which will have harmful effects on the patient. This attack was mounted by injecting false data into the communication channel between the blood glucose monitor and the controller.

5.4 Detection of attacks on SAP

- **Insulin Tampering Attack**: As the attacker modifies only the insulin dosage while keeping the other properties the same, the intrusion detector is able to detect the attack, as the current correlation is not what it expects after its training phase. Similar to above attacks the log likelihood of the current system trace is less than that of the trained HMM, thus arousing suspicion.
- **Glucose Spoofing Attack**: The intrusion detector module of CORGIDS receives the correlated properties which contains both faulty and non-faulty values of the blood glucose in it. Thus, based on this input data, the log likelihood generated by the current log differs from that expected by the trained HMM. This indicates that there is an intrusion in the current state of the SAP.

Although, the attacks that are demonstrated in this paper break the logical correlation directly, CORGIDS is also capable of detecting an anomaly which is generated through indirect attacks. For example, instead of changing the physical property like battery % left in the battery tampering attack (this is a attack in which correlations are broken directly), we could change either the value of some variable (other than the physical variable) used in the UAV or alter other logic which does not directly effect the physical property. These changes will propagate in the program and ultimately reach the receiving end of the CPS (an actuator). If they do not, then the attack is likely to be harmless as the attacker cannot change the physical behavior of the CPS without modifying its outputs.

6 EVALUATION

In this section, we present the results from the sensitivity analysis and attacks seeded in section 5. We also compare CORGIDS to closely related work in terms of the metrics in Section 6.2.

6.1 Sensitivity Analysis

Before evaluating CORGRIDS, we performed a sensitivity analysis to find out the values of the three experimental parameters which

[2]The attacks on SAP may seem similar to the ones proposed by [2], mainly because of similar attack names, but the attack and its detection methodology is totally different when compared to ours.

have the highest values of Precision and Recall (Section 6.2). The three experimental parameters are as follows.

- Window size (w): A window size is defined as the time duration which is under consideration for detecting any intrusion [35] in a SUT. A large w means that greater historical data is required by the HMM to decide of a malicious activity.
- Acceptable range (δ): An acceptable range defines a range within which the testing system trace's likelihood can vary from the mean log likelihood from the trained HMM. A log with the value within range from the specified mean will be marked to be similar to the training logs. If the value of δ is chosen to be large, then we are enforcing loose control and allowing system traces with substantial variation from the trained HMM to be considered benign.
- Threshold of consecutive decisions (λ): We perform a stateful test [33] by maintaining the historical decisions and generating alert only if it goes above the threshold. The intuition behind using the λ is to look at the historical decisions of the intrusion detector to see if there is really an anomaly or if it is just one time spike in the system. Greater value of λ enforces more number of consecutive historical intrusion decisions to generate an alert.

The results from sensitivity analysis are shown in Figure 5. w is measured in minutes while δ in standard deviations. A key point to note here is that more the value of precision and recall for a set of experimental parameters, higher is the rate of detection. We carry out our sensitivity analysis by varying one parameter at a time and keeping others constant. For instance, Graph *a* denotes the scenario where the w is varied from 2 to 4 minutes while keeping δ = 1 and λ = 2. Similar to graph *a*, we now sweep the constant parameters, that is, δ and λ from their lowest to the highest values. This forms the first row, Graph *a* - *d* in Figure 5. Similar to first row, we conduct experiments by varying δ in second row and λ in last row. This sensitivity analysis represents the data collected from the distance spoofing attack on the UAV testbed. Though similar analysis was performed for the other attacks on the UAV and SAP, due to space limitations they are not included here.

From the graphs, the precision and recall are increasing as the w is increasing from 2 to 4 minutes, while they are decreasing when the δ and λ are increasing from 1 to 3 standard deviations and 2 to 4 decisions respectively. From this trend we can infer that the precision and recall are largest when we have a large window size (w) with small threshold of consecutive decisions (λ) and acceptable range (δ). Similar trend was observed for other attacks on the two testbeds. The reason for the trend observed is that a HMM requires substantial historical data to determine if there is some anomaly in the system. With less history (smaller window size), it is unable to correctly infer the state of the current state of the system. Thus, when we provide greater window size (w) of 4 minutes, it is able to create a more realistic model of the system. As the HMM is now more confident abut the system after having a large w, it can now make decisions confidently, thus giving the best results for the least value of δ and λ.

An important point to note here is that though CORGIDS is able to detect attacks even with less favorable values of w, λ and δ, it achieves less precision and recall in doing so. On the other hand, if we use the results obtained from sensitivity analysis we are able to have higher values of precision and recall. Thus, either we can choose the lesser favorable parameters and obtain results quickly at the cost of accuracy, or we can work with the most favorable parameters while incurring some latency but obtain the fewer FNs.

Table 3: FP and FN obtained for CORGIDS on the two testbeds

Testbed	Targeted Attack	FP (%)	FN(%)
UAV	Battery Tampering	0.0	12.20
	Flooding	0.0	11.30
	Distance Spoofing	0.0	12.80
SAP	Insulin Tampering	5.60	4.20
	Glucose Spoofing	2.80	8.40

Table 4: Comparison of Precision and Recall for OpenAPS platform

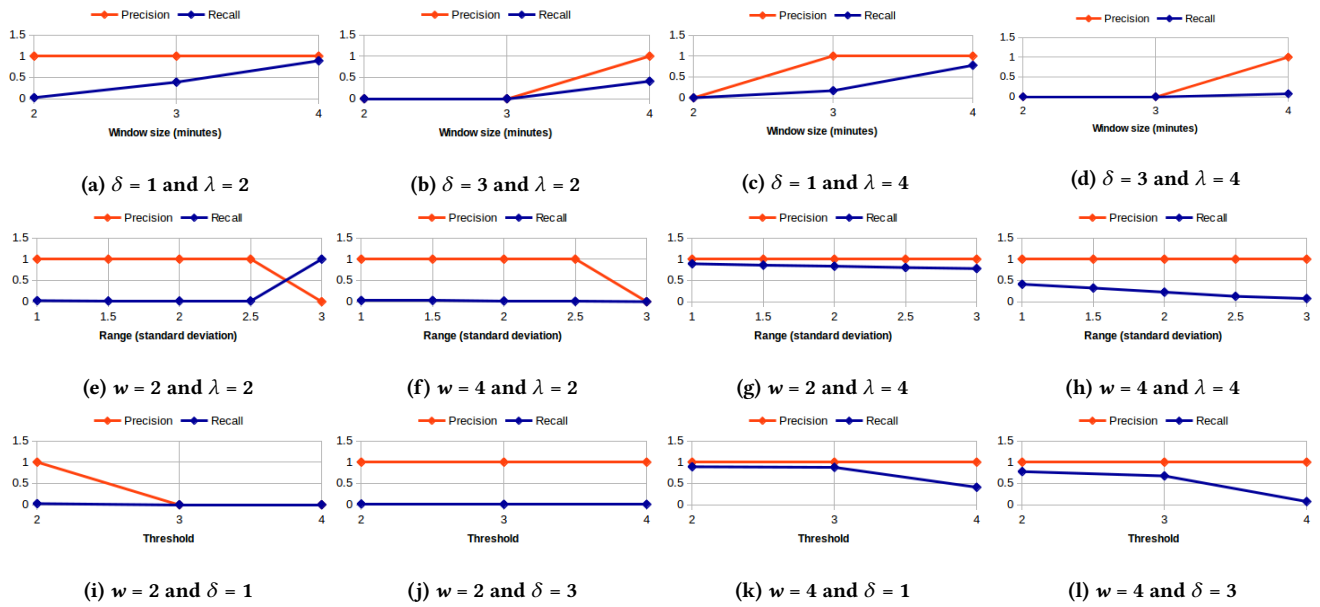
Methodology	Testbed	FP(%)	FN(%)	Precision(%)	Recall(%)
ARTINALI	SEGMeter	12	2.3	89.06	97.7
	OpenAPS	13.5	2	87.89	98
Zohrevand et al. [35]	Water Treatment System	-	-	78.87	81.4
Chen et al. [9]	Water Purification Plant	-	15	-	-
CORGIDS	UAV	0.00	12.10	100	87.90
	SAP	4.20	6.30	95.70	93.70

6.2 Evaluation Criteria

We use Precision, Recall, performance overheads and memory overheads to evaluate CORGIDS.

- Precision: For a malicious execution of SUT, when an intrusion detector correctly detects an intrusion, is called Precision. For an intrusion detector, the higher precision the better.
- Recall: While Recall is the percentage when the SUT execution was malicious and the intrusion detector correctly identified it among all the malicious SUT executions. For an intrusion detector, the higher recall the better.
- Performance Overhead: Performance overhead reflects the additional time taken, when CORGIDS is deployed on the SUT. It helps to determine if the time taken by the IDS to detect intrusion is greater than the cycle, in which case it is not very helpful to use an IDS.
- Memory Overhead: As the devices in which CORGIDS will be used will be memory constrained, it is essential to calculate its memory overhead. Memory occupied by CORGIDS on SUT will be used to determine this overhead.
  As we don't have access to a real UAV and use a simulator for the experiments, we do not calculate memory and performance overheads for UAV testbed.

For evaluating CORGIDS, we choose the value obtained for each of the three variables (w, λ and δ) for which the sensitivity analysis



(a) δ = 1 and λ = 2 (b) δ = 3 and λ = 2 (c) δ = 1 and λ = 4 (d) δ = 3 and λ = 4

(e) w = 2 and λ = 2 (f) w = 4 and λ = 2 (g) w = 2 and λ = 4 (h) w = 4 and λ = 4

(i) w = 2 and δ = 1 (j) w = 2 and δ = 3 (k) w = 4 and δ = 1 (l) w = 4 and δ = 3

Figure 5: Sensitivity Analysis: Variables are w, δ and λ. The vertical axes in all figures are the values of precision and recall calculated after averaging 5 fold cross validation of test system traces.

was performed. Table 3 contains the results for False Positives (FP) and False Negatives (FN) for the two testbeds, namely, an UAV and SAP on which CORGIDS was deployed. Table 4 compares our results to only those related papers [2, 9, 35] which dynamically generate physical invariants [3]. We acknowledge that Table 4 does not provide an apples-to-apples comparison, but we include it here to provide better context about CORGIDS' performance. Krotofil et. al. and Iturbe et. al. [19, 21] do not measure the performance of their methodology, and hence we could not compare with them. We additionally calculate precision and recall for our and the papers mentioned in table 4. To calculate the FP and FN values for CORGIDS which will be used to generate precision and recall, we average over FP and FN values from Table 3. We cannot compare the precision value of CORGIDS with Chen et. al. [9], as the later did not provide it in their paper.

6.3 RQ1. Precision

In this subsection, we discuss the precision achieved by seeding the attacks on the SUT and using CORGIDS to detect an intrusion. Also, we compare our precision with prior work in Table 4. As can be observed, CORGIDS achieves precision of 100% and 95.70% for the UAV and OpenAPS platform respectively. In comparison, no other intrusion detector has a precision greater than 90%. Specifically, CORGIDS provides an 21.33% improvement in precision over Zohravend et al. [35] and approximately 8.88% over Aliabadi et al. [2] for the OpenAPS platform.

The reason behind the higher precision percentage for CORGIDS is the use of correlations exhibited by the two CPS. CORGIRDS detects attacks by using an HMM to infer if the current system traces exhibits the same trend with which it was trained. This is the reason that especially for the UAV platform, the HMM recognizes an anomaly with almost 100% precision. The reason for comparatively low precision value for OpenAPS platform is the lack of training examples. We maintained a 70:30 ratio for training and testing traces. However the lack of availability of patient's diabetic therapy data led to a lower number of training samples. This, in turn, negatively affected the training of the HMM used by CORGIDS.

[3]Note: For the research papers with which we compare our work for the UAV and SAP testbed, we directly use the FPs, FNs, precision and recall evaluation metrics provided by them. We manually calculated Precision and Recall for [2] from the FP and FN values provided in their paper

6.4 RQ2. Recall

Here we discuss the recall factor of CORGIDS and compare it to the related work mentioned in tables 3 and 4 respectively. From Table 4, CORGIDS receives a high recall percentage among all the related work. Though CORGIDS does not have the highest recall, it is quite close to ARTINALI with 93.70% for the OpenAPS platform. CORGIDS improves the recall by 15.11% when compared to Zohravend et al. [35]. On the other hand, CORGIDS achieves 11.14% lower recall than ARTINALI when both of them are compared with their lowest recall factors.

CORGIDS achieves lesser recall than ARTINALI [2] mainly because the behavior of the SUT under attack was stealthy and did not deviate much from the normal trend. As the deviation was less, the logical correlation between the properties seemed very similar to the one expected, thus the HMM did not mark the state as anomalous. As Chen et al. [9] do not provide recall factor, but give the value of FN for their approach, we compare the FN value to CORGIDS, which is 15%. This is higher than CORGID's FN values or 12.10% and 6.30% for the two platforms. Chen et al. use SVMs for intrusion detection, while CORGIDS uses HMMs. HMMs are

better able to capture the sequence of states and their transitions in a CPS, and hence CORGIDS achieves lower FN values.

6.5 RQ3. Memory overhead

We measured the memory overhead of CORGIDS running on the OpenAPS platform. The experiments were performed on a Raspberry Pi 3 with approximately 1 GB of RAM. We found that CORGIDS consumes 36.15 MB when detecting intrusion. The reason behind this memory overhead is that CORGIDS uses HMM for intrusion detection. The trained HMM model when loaded into memory along with the libraries required for it to generate a decision, requires more space. However, as CORGIDS is used by controller to detect intrusion in the SUT, and the controllers are not memory constrained as compared to the SUT. For instance, in Raspberry Pi 3, it took only a fraction (36.15 MB) of memory from the 1 GB available RAM. Thus, we surmise that the memory overhead incurred by CORGIDS is acceptable.

6.6 RQ4. Performance overhead

Like memory overhead, performance overhead measurements were also taken from the Raspberry Pi 3 platform. We consider the average of 10 executions for the overhead. Ideally, the time taken to deduce a decision should be less that the execution cycle of the SUT, in order for the intrusion detector to keep up with the system. It takes approximately 1.25 seconds for CORGIDS to generate a decision based on the input correlated logs. This is negligible compared to the time taken by a single execution cycle of OpenAPS, which is about 5 minutes.

Scalability of CORGIDS: To understand the scalability of the overheads with HMM size, we varied the HMMs used for intrusion detection. Thus, we created multiple HMMs by varying the tuning parameter of a HMM, which is the number of *hiddenStates*. The number of *hiddenStates* were varied among 2, 5, 10, 15, 20. We observed that the memory and performance overhead of CORGIDS remains the same regardless of the number of *hiddenStates* in the HMM. This is because the libraries which are loaded along with the HMM is the dominant factor in the time, and this does not depend on the number of *hiddenStates* in the HMM.

7 DISCUSSION

We first discuss the threats to validity followed by the generic applicability and how CORGIDS can be circumvented.

7.1 Threats to validity

We consider three threats to validity, namely, i) Internal, ii) External and iii) Construct. An *Internal threat* to our work is that we have considered only five attacks. For instance, we have not experimented with other types of targeted attacks such as dropping attacks or arbitrary attacks like data mutation, branch flipping or artificial delay insertion [2]. Instead, we attempted to mitigate this threat by using attacks which are very different in nature and exploit different domains of the testbeds. Another internal threat is the use of simulations to gauge the effectiveness and performance of CORGIDS. Though this threat is substantial, we have attempted to mitigate it by keeping the simulations as unbiased as possible. For instance, for each flight of an UAV, we randomized the number of way-points, latitude and longitude of each way-point and altitude. However, for future work, we will also evaluate CORGIDS on a real testbed. An *External threat* consists of the use of only two testbeds from the CPS domain to prove that our approach is effective and general. However, finding testbeds which are publicly available (open-source) and also are security critical is a difficult task. We attempted to mitigate this threat by choosing two testbeds which are entirely different in behavior and utility. An UAV is used for flight operations and uses physical laws of motion for operation, while the SAP is a medical device and uses biological properties of the human body to calculate the appropriate amount of insulin to be injected. Finally, the *Construct threat* to validity is use of only FP, FN and Precision, Recall for the evaluation of CORGIDS. However, these metrics are also used substantially by prior work in this area and therefore are valid for comparison purposes.

7.2 Generalization

Building a generic IDS for systems exhibiting correlation is one of the key contributions of this paper. Our approach utilizes the correlation exhibited by logical properties, for instance use of logical properties such as speed, distance traveled, altitude, battery in the UAV and flight time. These values are dependent on each other and change according to only a predefined framework, for example, the laws of physics for an UAV. However, CORGIDS cannot be applied to those systems which do not exhibit such correlations. For instance, systems except CPS, financial systems, in which no correlation can be found between its variables/properties are not the candidates for using CORGIDS.

7.3 Circumventing CORGIDS

As discussed in Section 3.1, we assumed that the attacker has capabilities which can be used to plant an attack on the SUT. An attacker who knows about the internals of the system can circumvent the intrusion detection done by CORGIDS. The scenario is the attacker hacks the SUT and changes the logging module of CORGIDS to send the correct correlated values of physical properties irrespective of them being faulty at that point of time. If the attacker were to continue this operation throughout the UAV's flight, CORGIDS would not be able to detect intrusion, because correctly correlated values will be received by GCS. However, updating all the correlated values at every second during the entire flight is constrained by power consumption, time and effort [21]. So, the case would likely be that the attacker would not be able to forge the values throughout the entire duration, thus leading to some discrepancy in values of logical properties which would be flagged by CORGIDS.

Another point to note is that we demonstrated the effect of varying a single correlated property for intrusion detection. Varying multiple properties in the system will have a similar effect and will lead to an unbalanced correlation which will be spotted by the HMM. Also, by varying the rate of increase or decrease of the anomalous correlated property, variations in the log probability of the current system state will arise which will be marked malicious. However, if log probability of the current malicious state is very close to the benign state's log probability, it is likely that CORGIDS

would not be able to distinguish between these two states, and thus the attack would not be detected.

8 CONCLUSIONS AND FUTURE WORK

CPS systems exhibit correlation between their logical properties as they need to interact with the physical environment, which are subject to the laws of physics. Lately, attackers have targeted CPS owing to their loose security control measures. Though, the use of physical properties (logical properties) of the CPS to detect an intrusion has gained prominence lately, all these solutions either use manually defined physical rules, or dynamically build the invariants but only for a specific CPS. In this paper, we propose a generic IDS, CORGIDS designed for systems which exhibit correlations, that uses Hidden Markov Models (HMMs) for extracting the correlations. HMMs are much more resilient to outliers and noise compared to other techniques, and do not presuppose a distribution of the properties, making them generic. We demonstrate the use of CORGIDS on two diverse CPS. We find that CORGIDS is able to detect intrusion with significantly less FPs and FNs and with more precision and recall when compared with other IDS.

ACKNOWLEDGEMENT

The authors are thankful to Magnus Almgren and the anonymous reviewers of the CPS-SPC workshop for their helpful comments. This research was partially supported by a research grant from the Natural Sciences and Engineering Research Council of Canada (NSERC), and a research gift from Intel.

REFERENCES

[1] Sridhar Adepu and Aditya Mathur. 2016. Using process invariants to detect cyber attacks on a water treatment system. In *IFIP International Information Security and Privacy Conference*. Springer, 91–104.

[2] Maryam Raiyat Aliabadi, Amita Ajith Kamath, Julien Gascon-Samson, and Karthik Pattabiraman. 2017. ARTINALI: Dynamic Invariant Detection for Cyber-Physical System Security. (2017).

[3] Ardupilot SITL [n. d.]. Ardupilot Software in the Loop. http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html. Accessed: May 13, 2018.

[4] Arati Baliga, Vinod Ganapathy, and Liviu Iftode. 2008. Automatic inference and enforcement of kernel data structure invariants. In *Computer Security Applications Conference, 2008. ACSAC 2008. Annual*. IEEE, 77–86.

[5] Arati Baliga, Vinod Ganapathy, and Liviu Iftode. 2011. Detecting kernel-level rootkits using data structure invariants. *IEEE Transactions on Dependable and Secure Computing* 8, 5 (2011), 670–684.

[6] Giuseppe Bernieri, Fabio Del Moro, Luca Faramondi, and Federica Pascucci. 2016. A testbed for integrated fault diagnosis and cyber security investigation. In *Control, Decision and Information Technologies (CoDIT), 2016 International Conference on*. IEEE, 454–459.

[7] Ivan Beschastnikh, Yuriy Brun, Sigurd Schneider, Michael Sloan, and Michael D Ernst. 2011. Leveraging existing instrumentation to automatically infer invariant-constrained models. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, 267–277.

[8] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, Tadayoshi Kohno, et al. 2011. Comprehensive Experimental Analyses of Automotive Attack Surfaces.. In *USENIX Security Symposium*. San Francisco.

[9] Yuqi Chen, Christopher M Poskitt, and Jun Sun. 2018. Learning from Mutants: Using Code Mutation to Learn and Monitor Invariants of a Cyber-Physical System. *arXiv preprint arXiv:1801.00903* (2018).

[10] Ashish Choudhari, Harini Ramaprasad, Tamal Paul, Jonathan W Kimball, Maciej Zawodniok, Bruce McMillin, and Sriram Chellappan. 2013. Stability of a cyber-physical smart grid system using cooperating invariants. In *Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual*. IEEE, 760–769.

[11] Christoph Csallner, Yannis Smaragdakis, and Tao Xie. 2008. DSD-Crasher: A hybrid analysis tool for bug finding. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 17, 2 (2008), 8.

[12] Christoph Csallner, Nikolai Tillmann, and Yannis Smaragdakis. 2008. DySy: Dynamic symbolic execution for invariant inference. In *Proceedings of the 30th international conference on Software engineering*. ACM, 281–290.

[13] Sean R Eddy. 1996. Hidden markov models. *Current opinion in structural biology* 6, 3 (1996), 361–365.

[14] Göran N Ericsson. 2010. Cyber security and power system communication—Essential parts of a smart grid infrastructure. *IEEE Transactions on Power Delivery* 25, 3 (2010), 1501–1507.

[15] Michael D Ernst, Jeff H Perkins, Philip J Guo, Stephen McCamant, Carlos Pacheco, Matthew S Tschantz, and Chen Xiao. 2007. The Daikon system for dynamic detection of likely invariants. *Science of Computer Programming* 69, 1 (2007), 35–45.

[16] MA Ferrer, IG Alonso, and CM Travieso. 2000. Influence of initialisation and stop criteria on HMM based recognisers. *Electronics Letters* 36, 13 (2000), 1165–1166.

[17] Mark Gabel and Zhendong Su. 2008. Javert: fully automatic mining of general temporal properties from dynamic traces. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. ACM, 339–349.

[18] Mark Gabel and Zhendong Su. 2010. Online inference and enforcement of temporal properties. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. ACM, 15–24.

[19] Mikel Iturbe, José Camacho, Inaki Garitano, Urko Zurutuza, and Roberto Uribeetxeberria. 2017. On the feasibility of distinguishing between process disturbances and intrusions in process control systems using multivariate statistical process control. *arXiv preprint arXiv:1706.01679* (2017).

[20] Ahmad Y Javaid, Weiqing Sun, Vijay K Devabhaktuni, and Mansoor Alam. 2012. Cyber security threat analysis and modeling of an unmanned aerial vehicle system. In *Homeland Security (HST), 2012 IEEE Conference on Technologies for*. IEEE, 585–590.

[21] Marina Krotofil, Jason Larsen, and Dieter Gollmann. 2015. The process matters: Ensuring data veracity in cyber-physical systems. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*. ACM, 133–144.

[22] Neal Leavitt. 2010. Researchers fight to keep implanted medical devices safe from hackers. *Computer* 43, 8 (2010), 11–14.

[23] Caroline Lemieux, Dennis Park, and Ivan Beschastnikh. 2015. General ltl specification mining (t). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*. IEEE, 81–92.

[24] Tianbo Lu, Jinyang Zhao, Lingling Zhao, Yang Li, and Xiaoyan Zhang. 2015. Towards a framework for assuring cyber physical system security. *Int. J. Security Appl.* 9, 3 (2015), 25–40.

[25] Robert Mitchell and Ing-Ray Chen. 2012. Specification based intrusion detection for unmanned aircraft systems. In *Proceedings of the first ACM MobiHoc workshop on Airborne Networks and Communications*. ACM, 31–36.

[26] Robert Mitchell and Ing-Ray Chen. 2014. Adaptive intrusion detection of malicious unmanned air vehicles using behavior rule specifications. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 44, 5 (2014), 593–604.

[27] Robert Mitchell and Ing-Ray Chen. 2015. Behavior rule specification-based intrusion detection for safety critical medical cyber physical systems. *IEEE Transactions on Dependable and Secure Computing* 12, 1 (2015), 16–30.

[28] Tamal Paul, Jonathan W Kimball, Maciej Zawodniok, Thomas P Roth, Bruce McMillin, and Sriram Chellappan. 2014. Unified invariants for cyber-physical switched system stability. *IEEE Transactions on Smart Grid* 5, 1 (2014), 112–120.

[29] Johann-Sebastian Pleban, Ricardo Band, and Reiner Creutzburg. 2014. Hacking and securing the AR. Drone 2.0 quadcopter: investigations for improving the security of a toy. In *Mobile Devices and Multimedia: Enabling Technologies, Algorithms, and Applications 2014*, Vol. 9030. International Society for Optics and Photonics, 90300L.

[30] Jerome Radcliffe. 2011. Hacking medical devices for fun and insulin: Breaking the human SCADA system. In *Black Hat Conference presentation slides*, Vol. 2011.

[31] Nils Miro Rodday, Ricardo de O Schmidt, and Aiko Pras. 2016. Exploring security vulnerabilities of unmanned aerial vehicles. In *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*. IEEE, 993–994.

[32] Florian Skopik, Zhendong Ma, Thomas Bleier, and Helmut Grüneis. 2012. A survey on threats and vulnerabilities in smart metering infrastructures. *International Journal of Smart Grid and Clean Energy* 1, 1 (2012), 22–28.

[33] David I Urbina, Jairo A Giraldo, Alvaro A Cardenas, Nils Ole Tippenhauer, Junia Valente, Mustafa Faisal, Justin Ruths, Richard Candell, and Henrik Sandberg. 2016. Limiting the impact of stealthy attacks on industrial control systems. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1092–1105.

[34] Jinlin Yang, David Evans, Deepali Bhardwaj, Thirumalesh Bhat, and Manuvir Das. 2006. Perracotta: mining temporal API rules from imperfect traces. In *Proceedings of the 28th international conference on Software engineering*. ACM, 282–291.

[35] Zahra Zohrevand, Uwe Glasser, Hamed Yaghoubi Shahir, Mohammad A Tayebi, and Robert Costanzo. 2016. Hidden Markov based anomaly detection for water supply systems. In *Big Data (Big Data), 2016 IEEE International Conference on*. IEEE, 1551–1560.