Mini-Me, You Complete Me! Data-Driven Drone Security via DNN-based Approximate Computing

Aolin Ding
Rutgers University
aolin.ding@rutgers.edu

Praveen Murthy
Swirlds Inc.
murthy@cal.berkeley.edu

Luis Garcia
University of Southern California ISI
lgarcia@isi.edu

Pengfei Sun
F5 Networks
p.sun@f5.com

Matthew Chan
Rutgers University
matthew.chan@rutgers.edu

Saman Zonouz
Rutgers University
saman.zonouz@rutgers.edu

ABSTRACT

The safe operation of robotic aerial vehicles (RAV) requires effective security protection of their controllers against cyber-physical attacks. The frequency and sophistication of past attacks against such embedded platforms highlight the need for better defense mechanisms. Existing estimation-based control monitors have tradeoffs, with lightweight linear state estimators lacking sufficient coverage, and heavier data-driven learned models facing implementation and accuracy issues on a constrained real-time RAV. We present Mini-Me, a data-driven online monitoring framework that models the program-level control state dynamics to detect runtime data-oriented attacks against RAVs. Mini-Me leverages the internal dataflow information and control variable dependencies of RAV controller functions to train a neural network-based approximate model as the lightweight replica of the original controller programs. Mini-Me runs the minimal approximate model and detects malicious control state deviation by comparing the estimated outputs with those outputs calculated by the original controller program. We demonstrate Mini-Me on a widely adopted RAV physical model as well as popular RAV virtual models based on open-source firmware, ArduPilot and PX4, and show its effectiveness in detecting five types of attack cases with an average 0.34% space overhead and 2.6% runtime overhead.

CCS CONCEPTS

• **Security and privacy → Intrusion/anomaly detection and malware mitigation**; • **Computer systems organization → Embedded and cyber-physical systems**.

KEYWORDS

Robotic Vehicle, Cyber-physical System Security, Intrusion Detection, Control Semantics, Neural Networks, Control Estimation

ACM Reference Format:
Aolin Ding, Praveen Murthy, Luis Garcia, Pengfei Sun, Matthew Chan, and Saman Zonouz. 2021. Mini-Me, You Complete Me! Data-Driven Drone Security via DNN-based Approximate Computing. In *24th International Symposium on Research in Attacks, Intrusions and Defenses (RAID '21), October 6–8, 2021, San Sebastian, Spain.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3471621.3471869

RAID '21, October 6–8, 2021, San Sebastian, Spain

ACM ISBN 978-1-4503-9058-3/21/10…$15.00
https://doi.org/10.1145/3471621.3471869

1 INTRODUCTION

Robotic aerial vehicles (RAVs), such as drones, are a type of autonomous cyber-physical system (CPS). RAVs are typically an interconnected integration of sensing, actuation, and control algorithms. The sensor components perceive the physical environment state, and controllers carry out system-wide automatic operations relying on processor-based algorithms and control logic. The calculated output values are sent to underlying actuators, e.g., the motors of a drone. RAVs have been adopted in a wide range of public and civilian applications such as package delivery services [10, 18] and urban planning [41]. Meanwhile, the widely available RAVs face substantive emergent threats stemming from cyber-physical vulnerabilities. The most common types of vulnerabilities are categorized as either *control-flow* attacks or *data-oriented* attacks.

Control-flow attacks originate from a software vulnerability and enable attackers to substitute or subvert the program's logic by hijacking its control flow. The traditional *control-flow* attacks including code reuse attacks [62, 73] and control flow hijacking [22, 70] are well studied and defended by classic security solutions such as control flow integrity [15, 30, 76] and memory protection [17, 28, 47, 51, 58, 81]. *Data-oriented*[1] attacks exploit the RAV's attack surfaces existing on the external sensor readings and remote configuration interfaces [48, 49]. GPS and sensor spoofing attacks [31, 46, 68, 74, 79] demonstrate that vulnerable sensor systems could lead to unsafe controller misbehaviors in consumer-grade RAVs.

Since the physical attacks and control semantic bugs are exposed as new challenging security threats, research efforts [25, 27, 37, 49, 59] have been made to counter the *data-oriented* threats involving both physical components and control parameters. Generally, defenses aim to detect any model trajectory discrepancies by modeling the drone's operational behaviors. The models are either formally derived or data-driven. Classical control modeling approaches [16, 27, 37] are used to estimate a vehicle's characteristics and control algorithms, but only using a linear control model to estimate a limited number of state variables lacks sufficient completeness. Meanwhile, formal verification works [35, 57, 84] have

[1] We mainly focus on *data-oriented* attacks as the *external* physical attacks and *internal* control parameter attacks against RAV cyber-physical components that will eventually impact the physical operation of the RAV, leading to unsafe flight modes.

leveraged formal software methods to verify controller semantics before executing a safety-critical control program. However, offline verification approaches often suffer from the state-space (e.g., execution path) explosion problem as complexity scales and their non-negligible overheads often outweigh the potential protection for RAVs–especially when considering real-time constraints.

Recently, more works have leveraged advancements in machine learning (ML) to capture the dynamic, non-linear behavior of complex cyber-physical systems and detect abnormalities [14, 24, 40, 44, 60, 71]. However, these systems typically treat the cyber-physical dynamics as a blackbox, i.e., the associated deep learning models aim to capture the input-to-output relationships of the entire CPS. First, such models require large amounts of training data to capture all the control states with all possible contingencies. More critically, the prior ML-based estimation works only consider the sensor-measured control dynamics and configurable control parameters without exploring any intermediate controller semantics. Thus they can only detect discrepancies at a macro-scale for the overall system, but they cannot pinpoint the compromised cyber-physical module that leads to this failure. Moreover, these solutions barely pay attention to real-time constraints when performing on-board intrusion detection especially on embedded systems like RAVs.

In this paper, we propose Mini-Me, a runtime solution to monitor end-to-end control logic execution through changes in state variable (i.e., from control state inputs to calculated actuation outputs). Mini-Me can detect *data-oriented* exploits, which includes the *external* physical attacks (e.g., sensor/signal spoofing attacks) and *internal* control parameter manipulations (e.g., deciding control variables in essential control logic) leading to unsafe physical states. Mini-Me protects the RAV against these attacks by running a machine learning-based approximate computing model to compare the expected control state outputs and perceived outputs. In particular, Mini-Me is able to learn and approximate more advanced control algorithms (e.g., high-order and non-linear portion), which are usually represented as linear components in control state estimation works [27, 37] without adequate attention to its comprehensiveness examination. Mini-Me is designed to create a *lightweight* replica of a concrete control software snippet with respect to the transparent model construction, which is underemphasized in previous learning-based intrusion detectors. Meanwhile, the model construction in Mini-Me successfully responds to the "data hunger" demands of training an accurate and converged learning model.

Mini-Me first identifies the control algorithms associated with vehicle dynamics. Then Mini-Me utilizes the dataflow analysis to determine the state variable changes and their mutual dependencies in the targeted controller software segment. Based on the identified controller variables, Mini-Me collects the training dataset in benign experiments and trains an adaptive neural network (NN) model to approximate the logic behaviors of the target controller snippet, which owns identical semantics but in different representations. Mini-Me simplifies the trained model and deploys it in a RAV for online monitoring. Mini-Me detects the RAV's operational abnormality by validating the deviation of the expected state outputs from the approximate model and the perceived outputs from the running control logic.

The contributions of the paper are as follows:

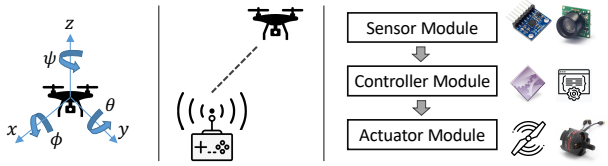


Figure 1: RAV vehicle dynamics and main control loop

- We present a novel, on-board intrusion detector utilizing program-level dataflow analysis to define and construct semantically similar approximate models for the RAV's linear and nonlinear controller programs.
- We design a lightweight neural network model with explainable architecture that can be optimized for practical RAV deployments.
- We validate our proposed online monitoring framework against different attack scenarios amidst multiple RAVs' runtime flight missions.

The structure of this paper is as follows: Section 2 provides background on the RAV control model and program-level dataflow analysis. Section 3 presents the limitations of previous works and challenges which motivate our overall design and the threat model of our work. Section 4 describes the prototype design of our solution and technical details. Section 5 presents the experiment setup and practical attack cases on multiple physical and virtual RAVs [11, 12]. Section 7 gives a overview of related work in the RAV security domain. We summarize our works and conclude in Section 8.

2 BACKGROUND

In this section, We briefly present the generic RAV control models (e.g, quad-copter) and program-level dataflow analysis.

2.1 RAV Control Model

The RAV control model is an integration of vehicle dynamics and control algorithms. The vehicle dynamics reflects the attitude and movement of a RAV, which encompass six degrees of freedom (6DoF) including the three axes (x, y, z) and the three rotation directions (roll: ϕ, pitch: θ and yaw: ψ) as shown in Fig 1. Linear and nonlinear controllers are utilized to control the RAV's position, attitude, velocity and movements for each corresponding axis. For instance, the cascaded PID controllers adjust the RAV's attitude by tuning the control inputs in a closed loop. In a typical implementation of the RAV control model, the main control loop receives the sensor measurements (e.g., accelerator, gyroscope) and calculates the next control outputs in the controller module (e.g., proportional-integral-derivative: PID) based on the observed current state. Then the actual signals will be sent to actuators (e.g., motors) to control the vehicle's operations. In a flight mission (e.g., path following), these modules execute periodically until the completeness of the mission (e.g., arrive at the last waypoint).

2.2 Dataflow Analysis

To investigate the internal variable changes and dependencies in RAV controller algorithms, the dataflow analysis is used to determine the *inputs* and *outputs* of specific code snippets by collecting

semantic information with reverse engineering tools. Dataflow analysis covers all paths through the program including infeasible paths. To carry out dataflow analysis [19, 32, 65], the first step is to generate a control flow graph (CFG), a directed graph in which nodes correspond to statements and edges represent possible execution paths. Considering traversal directions between the basic blocks, dataflow analysis can be naturally classified into *forward* analysis and *backward* analysis. Reaching-Definition Analysis is a typical forward dataflow analysis schema, in which dataflow has the same direction as control flow. The basic dataflow information at each program point includes the *define* and the *use* of variables. An assignment operation or value modification *defines* a variable at that point. Meanwhile, an operation making a *read* reference to a variable *uses* it. If one variable is *defined* first then *used* after previous definition, this *use-after-define* indicates it could be a intermediate variable (e.g., temporary variable in value propagation). For a practical dataflow analysis, it usually starts with every instruction and computes its *use* and *define* sets at that each program point. In a basic block consisting of multiple instructions, it can merge all the dataflow information in an iterative updating framework and eventually determine the broader scope of the *input* and *output* sets for that entire basic block. By using such dataflow analysis, we can precisely track the value changes of those control state variables (e.g., roll angle, control gains) in RAVs, which provides the surface to learn and approximate their operational behaviors.

3 MOTIVATION AND THREAT MODEL

Limitations of Existing Detection Methods. The safe and robust operation of RAVs relies on the various linear and nonlinear control subsystems. An obvious manner to check controller states is adding redundancy or duplication of the controller to provide a failsafe backup. However, in many cases, and especially in a RAV, duplication of the entire controller is infeasible in such resource-constrained embedded systems. Therefore, a state estimator is commonly used to conduct the control state checking in the controller program binary. To simplify the state estimation, the previous estimation works [27, 37] extract the nonlinear components using a linear state space template. In the practical deployment of RAVs, the assumptions of a linear system coverage do not always hold. For instance, the nano quadcopter Crazyflie 2.0 [12] uses the quaternion instead of Euler angles to calculate its rotations, which involves the non-linear factors (e.g., sine, cosine). In addition, a high-level state-space representation of control invariants may not cover some indirect but safety-associated control state variables (e.g., ratio of PID gains) and thus neglect the blind spots in the internal control flow of controller software.

For the learning-based detection solutions [14, 24, 40, 44], the key of their success is to train a classifier to distinguish the malicious behaviors from benign behaviors. The classification objects could be trends of sensor measurements [14], mutations [24], temporal behaviors [40, 44], etc. However, most of these works are learning a high-level mapping relation between sensor measurements and system states, which is lack of transparency and explainability for regarding those complicated and interconnected controller algorithms as a big black-box. Specifically for RAVs, these works do not pay adequate attention to the restrictions in such resource-constrained embedded systems with limited computing power and storage capability, which cause deployment difficulties (e.g., memory usage, performance overhead) in practical RAV applications.

Technical Challenges. Our insights in the aforementioned discussions motivate us to address the following technical challenges when practically deploying a machine learning monitoring framework on RAVs: (1) build a transparent model, that is, the inputs and outputs of the model should be extracted in a clear definition and learn from both the vehicle dynamics and controller programs including linear and non-linear target control logic; (2) low performance overhead, in other words, the model should be lightweight (e.g., consume a relatively small amount of computing resources) and run fast so that it can meet the soft real-time deadlines and stop the unsafe operations before they actually harm the RAV system.

Threat Model. Mini-Me can detect *data-oriented* attacks on *external* physical surfaces[2] and *internal* control parameter spaces[3] of RAVs. We assume that the attackers are capable to exploit existing vulnerabilities in the external communication channels and perform spoofing attacks (e.g., GPS [46], acoustic noises [68], magnetic fields [67]). Additionally, we assume the attackers know existing control semantic bugs and their triggering conditions. Therefore, the attacker may modify the vehicle parameters to generate the bug-triggering conditions as presented in [48, 49] without the requirement of code injection or control flow hijacking. For instance, the attackers might exploit the control-semantic bugs (e.g., input validation bugs reported by [49]) to compromise controller states, which eventually harm the RV's safe operations (e.g., steering the drone towards a ground crash). In the following, we state the assumptions regarding the requirements to ensure Mini-Me's correct functionality.

We assume that the control logic behaves securely as intended based on benign operations before establishing the Mini-Me approximate model. This allows Mini-Me to learn from a legitimate control logic by intercepting the embedded controller's inputs (e.g., sensors and vehicle dynamics) and train a neural network-based replica. We also assume the control flow integrity of RAV controller software, that is, there are some trusted execution paths for us to deploy the monitoring framework. Hence the attacker can not bypass or corrupt the Mini-Me checking code. In this paper, the attacks triggered by generic software vulnerabilities (e.g., buffer overflows) or traditional program bugs (e.g., return-oriented programming) are out of scope, as they have been effectively handled by existing code and memory security efforts [30, 47, 51, 55, 58].

4 MINI-ME DESIGN

In this section, we present the approach of Mini-Me, a learning-based online monitoring framework. We utilize the dataflow analysis to establish the control state inputs and outputs for RAV controller algorithms. Then we construct and train a lightweight neural network model to monitor and detect the *data-oriented* attacks against the RAV system at runtime.

[2]The *external* sensor attacks seek to manipulate the sensor readings or inject erroneous signals into actuators/sensors through external channels [46, 67, 68].
[3]Our focus of *internal* parameter attack is on the bug-triggering of control parameters during the RAV's execution using control-semantic bugs [49] against the controller itself (as opposed to sensor attacks).

4.1 Overview

Facing the sophisticated threats on the controller software and physical components, we focus on protecting RAV systems against *data-oriented* attacks. We analyze the control program and build an NN-based approximate model as a safe reference, with which we can validate the performed actuation by comparing output vectors of the original control logic with those of the approximate model. Therefore, regardless of the attack vector used to compromise the controller software, Mini-Me can detect the final effect of such attacks on their maliciously calculated actuation outputs. As a clarification example, the controller software could be compromised through either sensor spoofing attacks or input validation bugs before the actuation commands are directly or indirectly corrupted by the adversaries. Mini-Me monitors and detects the final output corruption no matter how the controller software got compromised initially.

As shown in Figure 2, the main procedures of Mini-Me can be described as follows: (1) We reverse engineer the RAV's controller binary to obtain its control flow graph and perform dataflow analysis on critical controller functions. (2) We construct the learning model with the identified inputs and outputs for specific controller segments and train an approximate model using the training dataset from benign experiments. (3) We deploy the learning model as an online monitor and detect attacks by checking the estimation error of expected outputs and perceived outputs.

We utilize a learning model to approximate linear and non-linear controller functions since it can concretize the complicated numerical relations between sensor inputs, controller state variables and actuator outputs in a semantically-similar way [34] (a.k.a. **Challenge 1**). While execution speed and model accuracy are basically trade-offs for neural networks, we only learn those critical controller segments and accordingly train a well-defined neural network model, which can be compressed to accelerate its inference (a.k.a. **Challenge 2**). Meanwhile, we leverage the program-level dataflow analysis [19, 32, 65] to improve the transparency and explainability of our model instead of learning the entire control program as a black-box. Note that our DNN-based model is more explainable [4] since we force it to reason about intermediate concepts, as is done in concept bottleneck methods [52].

4.2 Control Program Instrumentation and Dataflow Analysis

Identifying The Critical Controller Segments. To construct the learning-based approximate model, Mini-Me needs to define the input and output vectors instead of blindly training the model with sensors and actuators. The monitoring function also needs to be inserted into the main control loop of the RAV's control program. With the binary executable of the target RAV controller, we identify the frequently executing controller functions such as PID controller and extended Kalman filter (EKF). The mathematical functions span the linear and non-linear controllers, which take the sensor measurements and vehicle dynamics as inputs to calculate the outputs for actuators. We reverse engineer the RAV's controller firmware

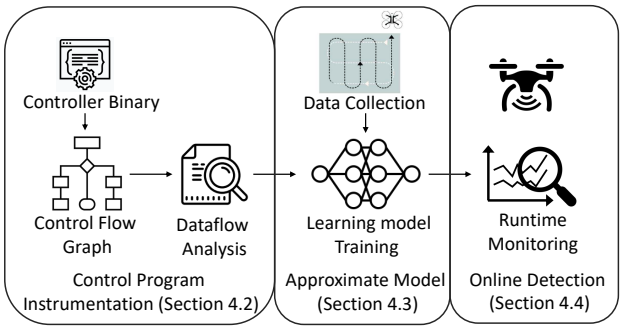


Figure 2: High-level architecture of Mini-Me

by lifting the binary executable to LLVM intermediate representative (IR). We obtain the memory-map I/O locations and resolve the binary-lifting imprecision using memory layout information (e.g., type of microcontroller unit (MCU) chip) and in-application debugging tools [2, 6]. We apply symbolic taint analysis [66] on its control flow graph (CFG) to trace all the functions calls that correlate to the sensors and actuators.

Based on our empirical analysis during the drone's actual flight operations, the core controller functions often execute frequently and thus take up the most significant share of the total time consumption in the main control loop, especially those functions that contain intensive numerical calculations (e.g., Jacobian matrix). Therefore, we identify those critical controller functions by checking their execution frequency and the arithmetical features (e.g., addition, multiplication operations of floating-point). Specifically, we first locate the interested mathematical functions in the main control loop through the CFG. We apply the time profiling and collect their calling frequency information using the dynamic binary instrumentation framework (e.g.,Valgrind [13]) as shown in [27]. Meanwhile, we recursively traverse their inner basic blocks to fetch the arithmetical statistics (e.g., number of arithmetical instructions). Through these analyses, we collect the metadata of interested controller functions.

Pinpoint State Variables via Dataflow Analysis. In order to create an approximate model of the controller functions we extracted from previous steps, we need to determine the state variables including critical control parameters and vehicle dynamics that are *used* (e.g., target velocity) and *defined* (e.g., updated velocity) in the function. Our goal in this analysis is to refine the interested controller function and investigate the value changes of those state variables and their mutual dependencies to build the input and output vector of the learning model. This helps us to construct a transparent model with explainable intermediate concepts [42, 52].

To pinpoint the memory locations of these variables, semantic annotation works [45, 69] use the mathematical templates of controller functions to decompose the control logic. These works need extra information about the RAV platform (e.g., memory layout) that varies for RAV applications. Instead, we use static dataflow analysis via LLVM passes [8] to identify those state variables in the interested code segment. Specifically, We design the dataflow analysis in three procedures as shown in Algorithm 1: (1) We extract

[4] In interpretable AI domain, explainability is often referred as giving explanations of a particular decision, and the root cause of the intrusion events for RAVs is studied in [25, 48]

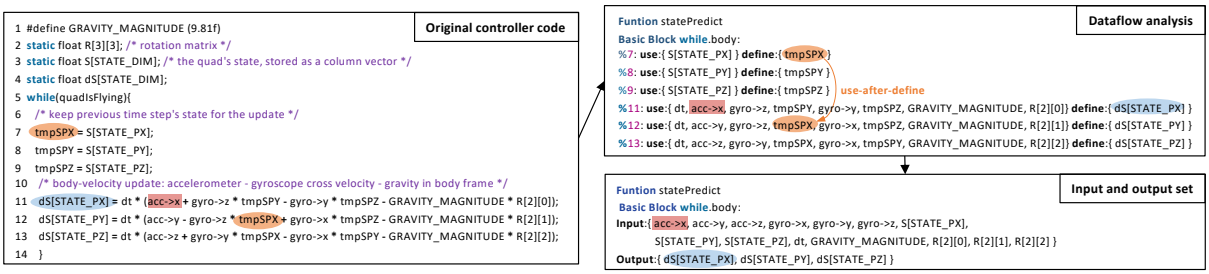


Figure 3: Dataflow analysis example for input and output nodes generation of neural network. Code segments are taken from Crazyflie 2.0 firmware and simplified for readability.

its basic blocks list from entry point to exit point for the identified controller segment (Line 3). (2) We build the *input* set, as the set of variables that are used in the basic block before being defined, and the *output* set, as the set of all the variables assignment within each basic block (Line 4-7). (3) We merge this information through instructions and solve the dataflow equations in a conservative way (Line 10-12), that is, we will include both the variable itself and possible alias of the same variable observed in intermediate parameters, array accesses, and indirect reference processes. Note that, alias objects are taken conservatively here by LLVM pointer analysis [7, 53] because they will be marked as repeated features and pruned in the preprocessing of the training dataset. For each controller segment, we finalize the *input* and *output* set to describe the value changes of state variables. In addition, we can also adjust the final variable sets based on a few other factors outside of the code segment such as macro definitions and global variables.

Algorithm 1 Dataflow analysis pass for basic blocks

1: **Initialize:** $Input[S] \leftarrow \phi; Output[S] \leftarrow \phi$; for all $S \in CFG$
2: **Repeat**
3: **for all** $S \in CFG$ **do** ▷ explore all basic blocks
4: $Input' \leftarrow \bigcup_{n \in pred[S]} Input[n]$ ▷ get *inputs* from predecessors
5: $Output' \leftarrow \bigcup_{n \in pred[S]} Output[n]$ ▷ get *outputs* from predecessors
6: $Input[S] \leftarrow use[S] \cup Input'$ ▷ take *use* variable as input
7: $Output[S] \leftarrow def[S] \cup (Output' - use[S])$ ▷ build output vector
8: **end for**
9: **Until**
10: **for all** $S \in CFG$ **do** ▷ stop until all dataflow information get collected
11: $Input' = Input[S]$ **and** $Output' = Output[S]$
12: **end for**

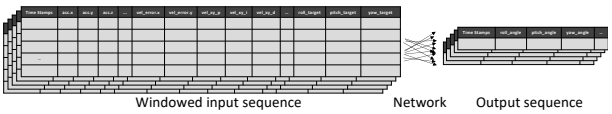
Case Study. The dataflow information is processed and calculated in the LLVM IR level. Figure 3 presents a simplified case study. For instance, we can observe that `S[STATE_PX]` is used to define `tmpSPX` in Line 7, and `tmpSPX` is used to futhur define another variable `dS[STATE_PY]` in Line 12. That means the value of `dS[STATE_PY]` indrectly depends on `S[STATE_PX]`, and `tmpSPX` is an intermediate temporary variable. In Line 11, the variable `acc->x` is used to define `ds[STATE_PX]` and there is no further updates for them. Therefore, `acc->x` is considered as *input* and `ds[STATE_PX]` is considered as *output* in this code segment. After summarizing all the variable changes and their mutual dependencies for each instruction, we obtain the `Input` set for the basic block, which is used to calculate those variables in `Output` set. As shown in Figure 5, the final `Input` set will be the input nodes for neural network models and `Output` set will be the output nodes.

4.3 Learning-based Approximate Model Construction

After identifying the critical controller segment (e.g., velocity control) and its input vector (e.g., target velocity, PID sub-module measurements) and output vector (e.g., updated velocity) in the above steps, we collect the benign experimental dataset to train the learning model and compress it for the practical deployment at runtime.

Data Collection and Preprocessing. To collect an adequate dataset for model training, we randomly generate the legitimate path-following missions under different environmental conditions (e.g., wind). Except for those physical dynamics already in the default RAVs system logger (e.g., attitude angles in ArduCoper), we employ Valgrind [13, 27] to trace the memory readings and writings of intermediate state variables in the identified controller code disassembly. Therefore, we collect the time-series dataset in real benign experiments. Note that we can generate a good amount of sample data in ArduCopter [11] since its default logging frequency is about 16.67HZ (i.e., every 60 milliseconds), which means, about 900 to 1000 valid samples (e.g., input and output vector pairs) will be generated in a typical 60 seconds mission (e.g., takeoff, follow the paths and travel through 5 waypoints, land) with ArduCopter simulators.

(a) Vector-to-vector training dataset



(b) Sequence-to-sequence training dataset with sliding window

Figure 4: Mini-Me converts the collected time-series dataset into windowed sequential dataset

For the collected dataset, we normalize all the training input and output data into floating-point values by using scaling functions based on their respective dimensions. For instance, if we observe that the computation results of the integration sub-module in the PID velocity controller usually range between -30 to 30, the velocity error (between current and target velocity states) could range

between -500 to 500. Therefore, this preprocessing step aims to mitigate the aggressive disturbance caused by those some input features with a large scalability sequence during the gradient descent calculation of the back-propagation process. Given the condition that the RAV's logging data are in time-series, we define a sliding window with fixed size to covert the training samples from vector-to-vector into sequence-to-sequence as shown in Figure 4. In other words, we use a sequence of inputs (e.g., target attitude) in a past period of time to predict the next outputs (e.g., updated attitude), which improves the accuracy and robustness of our learning model.

Neural Network Topology and Construction. To construct a semantically similar approximation of the targeted code segment, Mini-Me can use multiple kinds of neural network topology based on applications. In particular, we use the long short-term memory (LSTM) neural network [39], a variant of recurrent neural network (RNN), to learn the numerical mapping between the input and output sequences of the target controller segments in RAVs. The size and concrete elements of these sequences depend on the aforementioned dataflow analysis, which includes the vehicle dynamics (e.g., attitude, velocity of the RAV's body frame), sensor measurements and intermediate control state variables (e.g., PID sub-module parameters). Therefore, we construct the transparent inputs and outputs with explainable intermediate concepts [42, 52] for the neural network model.

As shown in Figure 5, Mini-Me's LSTM model is designed to be lightweight, consisting of the LSTM layer(s) and dense layer(s). The stacked LSTM layers aim to learn the temporal representation of non-linear relations between input and output sequences. In contrast to a typical classifier, our LSTM network does not comprise a *softmax* layer. Instead, we add fully connected dense layers at the end of the model to convey those linear relations between input and output sequences. We define the activation functions accordingly to achieve the empirically best performance on multivariate time-series forecasting.

The advantages of LSTM topology in Mini-Me over the general machine learning methodologies (e.g., SVM, decision tree, logistic regression with a ridge or lasso estimator) lie on the characteristics of training data samples. These traditional learning models are usually used for classification purposes with *limited* number of target labels. However, our training datasets are the *time-series* sequential samples for the forecasting purpose (e.g., predict updated vehicle dynamics), which contain numerous control states and are difficult to be labeled. In addition, LSTM is the most mentioned model used in time-series data training and forecasting because (1) it uses a sequence of samples (Figure 4b) as opposed to single input sample (Figure 4a) at each time. (2) it overcomes the vanishing gradient problem [20], which commonly exists on training processes (e.g., back-propagation) of gradient-based learning methods (e.g., CNN, simple RNN [33]).

Offline Training. Mini-Me trains the LSTM model in an iterative manner using classic feed-forward calculation then backpropagation. We initialize the network with random weights and hidden states with zeros. The training samples are shuffled and put into the model by batch. Each dense layer are activated by *relu* function and each LSTM layer are activated by *sigmoid* function. The whole network is trained end-to-end and the weights are updated by measuring the mean-square-error (MSE) between the predicted and the

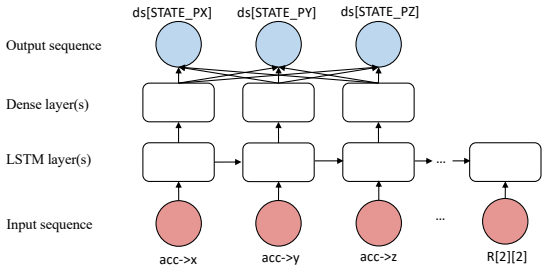


Figure 5: Mini-Me approximate computing model example based on LSTM neural network topology

target output sequence. We employ the Adam [50] optimizer and the total training epoch is set between 200 to 500.

In fact, given the multi-dimensional inputs and outputs, it is not a trivial challenge to determine the neural network topology and find its best hyperparameters since the training may not converge or even diverge (e.g., the training losses fail to move towards global or local minimums with a decreasing trend). Therefore, loss function MSE is not the only measure we use to evaluate the overall training quality. We employ the hyperparameter optimization framework to tune the hyperparameter settings (e.g., learning rate) and partial network structure choices (e.g., number of LSTM layers) in extensive trials, and our design optimization objective is to minimize the cross-validation loss when testing the NN model on new mission data sets, which guarantees the convergence and precision of our model. For instance, we suggest a new floating-point value from the log domain between 1e-1 to 1e-7 in every trial to search for the best learning rate. After all the tuning trials finish, we freeze the best available model and store it for further inference.

Neural Network Model Compression. In this step, Mini-Me significantly increases the execution speed (especially for inference) of the LSTM-based approximate model by utilizing neural network compression, which makes it easier to get deployed on a resource-constrained RAV system (e.g., low memory usage). As shown in Fig 6, Mini-Me performs the post-training quantization to quantize the weights and activations of each layer in the model. By converting the weights and inputs into integer types (e.g., 32-bit floating-point to 8-bit integer), Mini-Me consumes less memory bandwidth and achieves faster inference computation. Meanwhile, the trader-off with quantization is that Mini-Me degrades the model accuracy due to the reduced-precision arithmetic calculations in the network. In Mini-Me, the decrease of accuracy results in the enlargement of the error between the estimated state outputs (from the approximate model) and the perceived outputs (from the running control logic). Mini-Me mitigates this by applying confidence interval in runtime detection.

4.4 Runtime Monitoring

Binary Rewriting. To insert the Mini-Me detection function into the main control loop of the ARM binary in a RAV, We employ the *trampoline* binary rewriting technique [63] that is similar with [27]. The execution of the inserted monitoring function needs the function arguments and declarations. We reuse the identified input and output state variables as the arguments to construct the monitoring

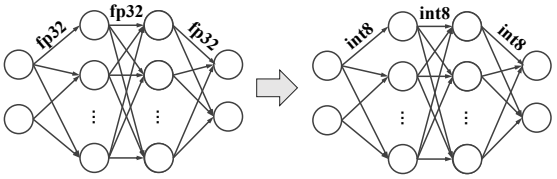


Figure 6: Mini-Me neural network model compression using static quantization

function. To ensure the reformed program declares the external function call and supports related neural network libraries, we precompile the source code of neural network inference as well as the supplementary functions, and then link all the object files to the ARM binary during its compiling and executing as a runtime library to get the final binary executable. we insert an external function call of the monitoring function and set the insert point after the target controller segment. Therefore, we successfully calculate the estimated control state outputs from the neural network model inference at runtime.

Attack Detection via Confidence Interval. To monitor the controller integrity in realtime, we calculate the Euclidean distance between the perceived outputs Y and estimated outputs $\hat{Y}$ as the estimation error E in equation 1 for all the benign data samples (e.g., independent testing data).

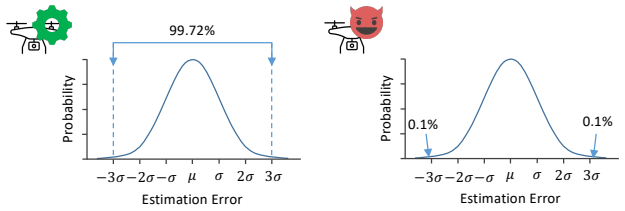
$$E = \sqrt{\left(Y - \hat{Y}\right)^2} = \sqrt{\sum_{k=1}^{n} (y_k - \hat{y}_k)^2} \tag{1}$$

$$F(e) = P(E \leqslant e) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{e} \exp^{-\frac{(t-\mu)^2}{2\sigma^2}} dt \tag{2}$$

All the benign estimation errors for the neural network approximate model form a closely normal distribution with a mean μ and standard deviation σ as shown in Figure 7. The cumulative distribution function $F(e)$ in equation 2 describes the probability that the estimation error E is less than or equal to a user-defined error threshold e, which determines the confidence of its estimation. If we define e as the safe interval between $\mu - 3\sigma$ to $\mu + 3\sigma$, the estimation error has the probability of 99.7% to fall in this interval in all benign experiments as shown in Figure 7a. In contrast, the probability that the estimation error falls out of this interval is less than 0.03%, which very likely happens only when attacks occur. Thus, it could be an indication that the monitored controller logic is compromised as shown in Figure 7b. In very few corner cases of benign samples, their estimation errors might also fall out of the safe interval and are marked as attacks, that is, false positive alarms.

In fact, the real-time estimation error of every new sample (i.e., control state updates) for online monitoring should obey the same statistical distribution generated from benign experiments if the integrity of monitored control logic holds. Therefore, the attacks are detected if we observed the real-time estimation error significantly violates the confidence interval of the benign error distribution.

It is important for Mini-Me to notify the system when a malicious execution is detected. Upon receiving the alert, we assume there are some secure operations that can be utilized to handle



(a) Benign Runs (b) Attack Runs

Figure 7: Runtime estimation error probability in benign and attack experiments

the responsive countermeasure if the control logic is compromised. Mini-Me does not recover the correct sensor data or original control logic. Instead, Mini-Me can boot the actuators into a failsafe mode which is defined by the user (e.g., safely land at a pre-determined location in a RAV use case) and prevent the RAV from a crash.

5 EVALUATION

In this section, we conduct and test Mini-Me in one real-world nano-quadcopter Crazyflie 2.0 [12] and widely-adopted open-source RAV virtual models, ArduPilot [11] and PX4 [9]. We evaluate the Mini-Me's real-time detection capability against five kinds of data-oriented attacks in terms of its effectiveness and performance. Our experiments answer the three main questions:

- **Q1**: How accurate is Mini-Me at learning linear and nonlinear controller functions?
- **Q2**: How effective is Mini-Me at detecting the data-oriented attacks?
- **Q3**: What is the runtime performance overhead of Mini-Me?

5.1 Implementation and Experimental Setup

We use llvm-mctoll [78] to lift the RAV's controller executable (ARM architecture) and use RetDec [4] as the decompiler to annotate the binary instructions with addresses. We implement two individual LLVM passes to identify the critical controller functions and perform the dataflow analysis to pinpoint the state variables in the LLVM IR. The neural network models are implemented in Python 3.7.4 based on Keras 2.4.3 with tensor flow 2.2.0 as the back-end. We implement the LSTM neural network models (e.g., stacked LSTM layers and fully connected layer) on the basis of Keras sequential model for both linear and nonlinear target controller functions. The training data used in LSTM model training are generated from real flight operations via autonomous scripts of mission generation and running.

Figure 8: RAVs in evaluation: Crazyflie 2.0, ArduCopter-based 3DR IRIS+, PX4-based 3DR Solo

For the experimental platform, we implement the prototype of Mini-Me on five different RAV systems including one real vehicle crazyflie 2.0 (quadcopter) and another four ArduCopter-based and PX4-based virtual vehicles since they are widely adapted and open-sourced, providing ground truth experiments for our study. All these RAVs are equipped with sensors, communication protocol, configuration and an ARM controller. Specifically, crazyflie 2.0 is designed as a double-MCU architecture to enrich its functionality. The main chip STM32F405 contains a 168MHz ARM Cortex-M4 processor, a 196KB static RAM and a 1MB flash ROM, which reads data from the inertial measurement unit (IMU) sensor, a combination of a 3-axis gyroscope, 3-axis accelerator, 3-axis magnetometer and a digital motion processor (DMP), to operate motors and perform flight control. For the virtual RAVs, we use the ArduCopter flight controller with its default APM SITL [5] simulator and PX4 controller with Gazebo [1] simulator. The simulation runs on 64-bit Ubuntu virtual machine of VMware and 4 processor cores @2.6GHz with 8 GB RAM.

5.2 Effectiveness of Attack Detection

Model Convergence and Error Distribution. After offline model construction, these data flow results constitute the training inputs and outputs for our neural network, which includes the rotation matrix, quadcopter state matrix, the position vector in both the global frame and its body frame, the attitude quaternion vector and attitude errors for prediction. We launch 30 different missions for each RAV and collect running data by mission, and each of them contains 900 to 1000 time instances on a 60ms logging frequency (averagely 52 seconds in total for each mission) excluding lifting and landing periods.

Due to the confined computing capacity and memory size in RAV, we train the neural network on a Linux PC and store its structural and weights information when the training process finishes. In order to achieve the best NN model in terms of convergence and accuracy, we apply the hyperparameter tuning framework optuna [3] to optimize the model training. We search for 5 different hyperparameters and configurable structural parameters including learning rate, the numbers of LSTM layers, the number of fully connected layers, batch size and the sliding window size. We suggest the floating-point value in log domain between 1e-1 to 1e-7 for the learning rate. We suggest integers between 1 to 4 for each number of LSTM and fully connected layers. The range of sliding window is set between 5 to 30 and we suggest numerical options of 32 to 512 for the batch size. We train the LSTM model with the mean square error as loss function and *adam* [50] as optimizer. Figure 9 shows the convergence of the LSTM-based approximation model in repeated experiments, the LSTM model will reach convergence at around 100 to 150 epochs as shown in the x-axis. Also, the validation losses in most epochs are greater than the training losses, which indicates there is no over-fitting on training samples. For the total dataset we collected, we split 80 percent of the data samples to train the NN model and keep the remaining 20 percent for validation purposes. The testing error distributions of LSTM models are shown in Figure 10, which indicates the reliant convergence of its training on vehicle data collected on 30 missions, that means, Mini-Me does not require a massive training dataset compared to

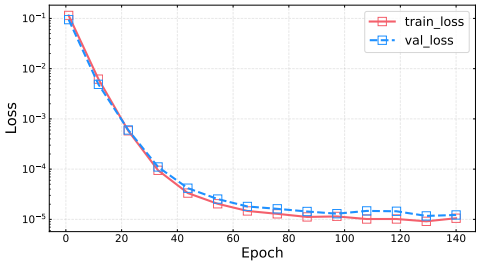


Figure 9: Convergence of LSTM-based approximate model

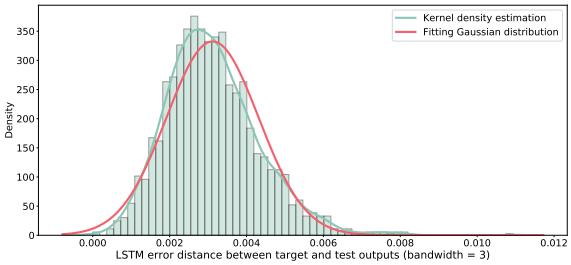
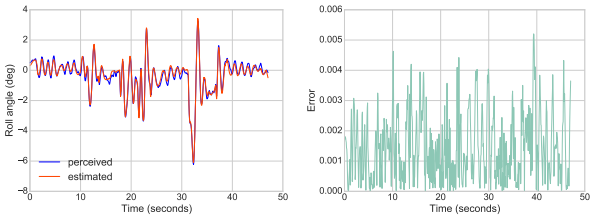


Figure 10: LSTM-based approximate model kernel density estimation and error distribution function generated from the 30 benign training missions

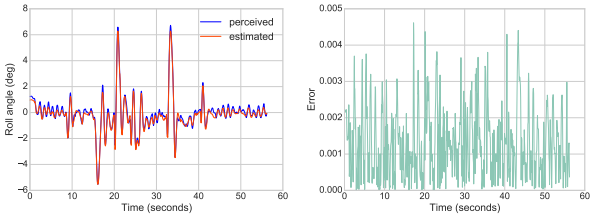
control invariant estimation work [27] due to its lightweight and concise model structural construction. In addition, we create the error distribution function based on the errors (i.e., distance between the perceived and estimated control state vector) for all testing samples. Note that the errors are measured after the normalization of testing samples, hence they are numerically smaller than the actual difference of vehicle dynamics. The distribution and probability density function provide us the average value μ and variance σ, which helps us determine the threshold (see Figure 13) as the error boundaries from the point of view of statistical probability.

Model Accuracy on Different Vehicles. We evaluate the generality of our proposed monitoring framework by showing the model accuracy for different vehicles and different control models. We establish the Mini-Me's approximate models for roll angle rate control functions extracted from ArduCopter control model in 3DR IRIS+ and PX4 control model in 3DR Solo. The results in Figure 11a and 11b show that a LSTM-model is capable to learn different control model behaviors in autonomous flight missions once the model is respectively trained for the certain vehicle dynamics and its control model. We also observe that the roll angles estimated by Mini-Me closely approximate the perceived changes of the roll angle, and the distances between them are measured as errors on the right side of Figure 11.

In addition, we extract another quaternion-based attitude controller in Crazyflie 2.0 for approximation (as opposed to angle-based RAV attitude stabilizer). As shown in Figure 12, the x-axis corresponds to operation time (in original timestamp) once the RAV starts flying along the testing trajectory, and the y-axis represents the measurement for the four different elements in the attitude quaternion that are used in rotation calculations and also reflects



(a) ArduCopter-based 3DR IRIS+ control model



(b) PX4-based 3DR Solo control model

Figure 11: Different approximate models get trained to estimate the control state variable of different vehicles

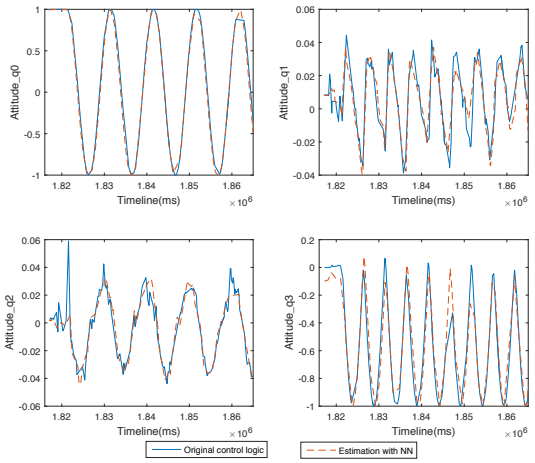
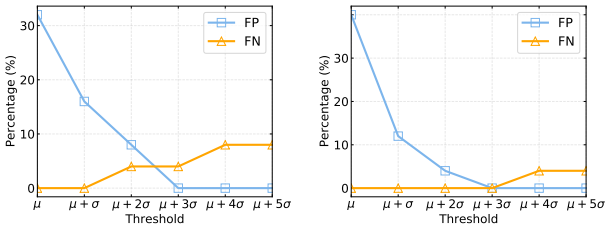


Figure 12: LSTM network approximation results for four attitude parameters in practical flight experiments

the flight attitude amidst missions. We observe that the NN-based approximation model learns the original control logic and accurately predicts the same flight movement patterns in our test cases. Therefore, we observe that our monitoring framework is agnostic to the RAV's controller software and accessible to build and deploy for different vehicles.

Attack Detection and FP/FN Rates. Figure 13 presents the results of false positive (FP) rates and false negative (FN) rates for attack detection under different thresholds. After deploying our monitoring framework on the RAV, we execute 25 benign missions in various environmental conditions to measure FP by counting how many false alarms are created. We launch the same 25 missions again and start attacks during the flight to measure FN how many attacks are ignored. Since the threshold is relevant to the probability calculated by the error distribution, we test six different thresholds based on the average error μ and variance σ. From figure 13, we find that a larger threshold generally brings about lower FP rates and relatively higher FN rates. Specifically, a larger threshold has lower FP rates when we test in unfavorable environments (e.g., various wind speeds) because the error distribution has a small deviation compared to prior training and testing missions. However, a larger threshold also leads to a higher FN rate especially when attackers compromise the RAV controller states in a gradual manner and long period of time because the malicious states do not have radical changes. More discussion about stealthy attacks is in Section 6. Therefore, we choose the threshold close to $\mu + 3\sigma$ to achieve the best combination of FP and FN rates. Overall, we observe 0% FP and 4% FN on the ArduCopter-based IRIS+ model with only one attack evading the detection. For Crazyflie 2.0 model, we observe 0% FP and 0% FN with no attack evading the detection by Mini-Me.



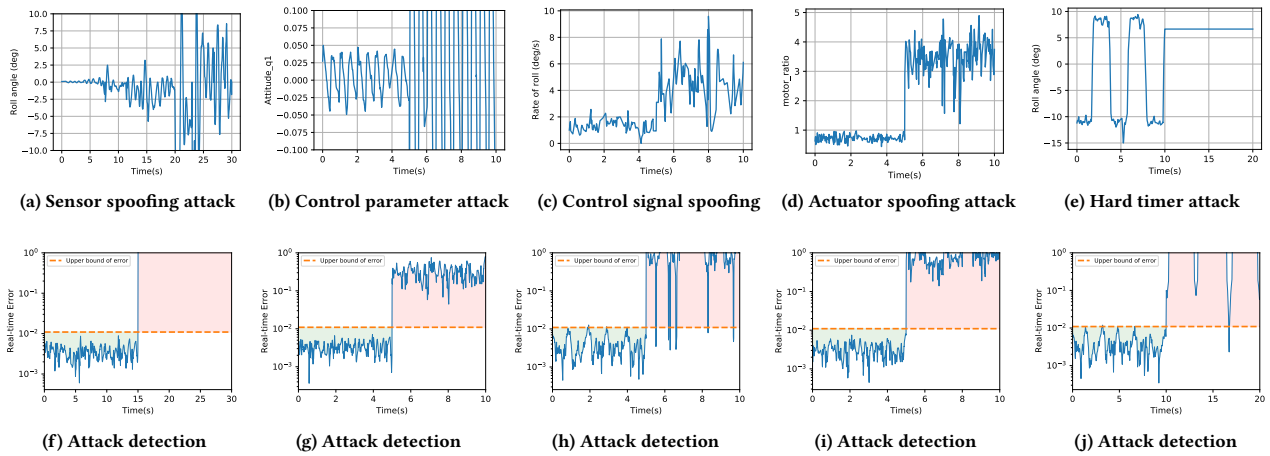
(a) ArduCopter-based 3DR IRIS+ (b) Crazyflie 2.0

Figure 13: False positive (FP) rates and false negative (FN) rates for different thresholds

5.3 Case Studies

Based on the current known attack cases relevant to our application, we create five different attack examples to show how Mini-Me monitors the estimation errors in real-time and detects possible exploitation.

Sensor Spoofing Attack. For the RAV and its RTOS, sensor spoofing attacks may occur in physical components (e.g., GPS signal tampering, vulnerability in sensor readings). When we discuss the condition in which the attacker compromises the sensor readings or takes over the sensor signals to mislead the RAV to unsafe operations, we assume that the attacker's goal is to corrupt the vehicle dynamic measurements (e.g., roll angle) from the external physical channels, which are essential for vehicle filtering and stabilizer. Our solution is able to prevent this kind of attack by comparing vehicle dynamic outputs from the original control logic and approximate model. Since the manipulation of partial or entire sensor readings will cause the malicious misbehavior, our pre-trained neural network reads the last updating sensor values augmented with other control state variables to predict the expected behaviors of the RAV, reflected as the parameter updates in the RAV's attitude and velocity vectors. This significant mismatch of output vectors will result in abnormal real-time error changes and be detected by Mini-Me.



(a) Sensor spoofing attack (b) Control parameter attack (c) Control signal spoofing (d) Actuator spoofing attack (e) Hard timer attack

(f) Attack detection (g) Attack detection (h) Attack detection (i) Attack detection (j) Attack detection

Figure 14: Mini-Me estimation error and real-time detection in practical attack cases

In our test case, Crazyflie 2.0 is set to fly along a testing trajectory. We launch the sensor spoofing attack [47] at time instance 15.0s after its taking off to disrupt the inertial sensor readings and compromise roll angle measurements, which causes failure to update state parameters in velocity and position controllers as shown in Figure 14a. In more detail, the controller task is still running, but the sensor readings and state estimation used to measure RAV's attitudes are compromised. Our results show a sharp increment in real-time monitoring error, which breaks the upper boundary generated from the testing estimation error distribution as marked out in Figure 14f. Thus, with the protection of Mini-Me, the NN-based approximation model recognizes sudden changes of the attitude measurements and detects the sensor spoofing attack.

Control Parameter Attack. For real-time RAVs, any disturbance or intrusion to control parameters can cause unsafe operations and even harm to the hardware components. For instance, the RAV stabilizer logic calculates the RAV velocity based on attitude errors from updated sensor measurements incorporating the current attitude information. The attacker might leverage control semantic bugs [49] and modify the security-critical parameters like *rollRate*, an important parameter for attitude resolution, to compromise the control logic and act disruptively amidst a flight mission.

In our experiments, a RAV is tasked with an automatic path-following mission, moving through a series of sequential waypoints. During its mission, we modify the control parameters in `attitude_pid_controller` to push one dimensional of attitude measurements to a very large scale from time instance 5.0s as shown in Figure 14b. LSTM approximation model successfully detects this sudden tampering on control logic because the real-time monitoring errors become greater than the pre-determined upper boundary as presented in Figure 14g. The reason why estimated errors do not fluctuate very sharply is that the scalar in the preprocessing layer leverages this impact of shape increment in the output vector. Our solution Mini-Me, equipped with LSTM neural network topology, proves its effectiveness in detecting control parameter attacks by monitoring the estimation error.

Control Signal Spoofing. In control signal spoofing attacks, an adversary can penetrate the control signals using various wireless communication techniques. The RC disturbance attack [47] could corrupt the RAV's RC signal handling and lead to unsafe flight operations. The GPS signal spoofing [43, 72] can send malicious signals to cause operation disturbance.

In our experiments, we launch a control signal spoofing attack in [27]. The motor pulse width modulation (PWM) signals are used in 3DS IRIS+ to adjust the motor's rotation speed and attitude updates from the state estimation process. We launch the attack at time instance 5.0s in Figure 14c, which results in a significant disturbance to the attitude-related parameters in state estimation including the `RollRate` (i.e., rate of roll angle). For an autonomous flight mission, the `RollRate` performs an important role in RAV velocity control algorithms such as PID and Kalman filter. As shown in Figure 14h, our solution recognizes these malicious changes of the real-time error between the predicted outputs from Mini-Me and perceived outputs from current running control logic.

Actuator Spoofing Attacks. The autonomous RAVs run control algorithms with the sensor readings to send commands to actuators. As a subset of control signal spoofing attacks, the actuator spoofing attacks specifically compromise actuator signal variable calculations using control-semantic bugs. Through feeding manipulated actuator signals, false data will be injected and disrupt the actuator (e.g. motors) measurements in state estimators.

In this case, we launch a typical actuator spoofing attack via input validation bugs [49] and misconfiguring the actuator parameters within 3DS IRIS+. Specifically, we manipulate the coefficient parameters of actuator signal PWM at time instance 5.0s amidst an autonomous flight mission. We define a parameter reset function in python scripts to raise the motor ratio values to five times more than their normal readings like Figure 14d, resulting in the compromised movement diagnosis. When the actuator signal gets compromised, the original control logic and Mini-Me will receive the same input data but generate different actuator outputs. As shown in Figure 14i, this kind of change will result in an unmatched input-output mapping based on the NN model forecasting; in other words, the

Table 1: Mini-Me detection evaluation on multi-category attack cases in comparison with existing cyber-physical attack detection methods. ✔: DETECTED ✗: NOT DETECTED

Attack Cases	Attack Surface		Detection			
	Attack Point	Description	Mini-Me	CI [27]	SAVIOR [59]	MAYDAY [48]
Sensor Spoofing Attack	Sensor reading	Malicious sensor intrusion	✔	✔	✔	✗
Control Parameter Attack	Control parameter	Tamper the control logic	✔	✗*	✗	✔
Control Signal Spoofing	State estimator	Mislead the actuator	✔	✔	✔	✔
Actuator Spoofing Attack	Actuator parameter	Mislead the actuator	✔	✔	✔	✔
Hard Timer Attack	SysTick hardware timer	Slow down real-time response	✔	✔	✔	✗

* The control parameter attack is only partially detected by CI. However, those attacks triggered by control-semantic bugs (e.g., missing range check) can not be detected

Table 2: Mini-Me training and evaluation results on different consumer-grade drone platforms. (CSO: Code Size Overhead, RPO: Runtime Performance Overhead)

Vendor	Model	Firmware	Performance	
			CSO (%)	RPO (%)
Bitcraze	Crazeflie 2.0	cf2_2019.02	0.18	3.1
3D Robotics	3DR IRIS+	ArduCopter 3.4	0.22	2.1
ArduPilot	APM SITL	ArduCopter 3.4	0.44	2.9
ArduPilot	APM SITL	ArduCopter 3.6	0.48	2.7
3D Robotics	3DR Solo	PX4 Pro 1.7	0.35	2.0

error between original control logic and approximate model has already exceeded the security range given by Mini-Me. We observe that real-time estimation error exceeds the upper bound, thus our solution Mini-Me can detect this actuator spoofing attack.

Hard Timer Attack. The task deadlines and queues monitoring function in RAVs are usually relying on the timer-related values in the global macro definition. As a type of system clock attack, the hard timer attack will write maximum values into timers to compromise the deadline-based task management and significantly slow down the real-time response. We launch the hard timer attack at time instance 10.0s to write a large value into *TIMER_PERIOD*. This attack will cause the failure of queues and task management in Crazyflie 2.0 and result in the entire control program crash. The roll angle is used for RAV's attitude adjustment by state estimator and other control algorithms. When it stops updating as shown in Figure 14e, our NN-based approximation model will calculate estimation based on the last signals. In addition, the original control algorithm has extremely low responsiveness towards the sensor data readings. Therefore, Mini-Me successfully recognizes this kind of attack by monitoring that real-time estimation error in Figure 14j has exceeded the upper bound of the benign error range.

5.4 Performance Overhead

Space overhead. We measure the code size overhead on all five RAV models. During a boot-up routine, the RAV software application initializes the portions in volatile memory and load RTOS and program codes from the flash memory, which also includes the read-only data along with other system-level data such as event logs. As shown in Table 2, Mini-Me has an average space overhead of 0.34%, which is negligible for practical deployment on resource-constraint embedded systems like RAVs. Note that we do not include the offline training libraries and modules in the space overhead since they are not used in the forecasting process once the trained model gets deployed.

Runtime overhead. The measurements of runtime overhead on all the five RAV models are presented in Table 2. The deployment of Mini-Me does not violate any deadline constraints or scan cycles in our experiments. The overhead of all five RAV models is 2.6% on average since Mini-Me only calculates the inference output (i.e., real-time monitoring error) based on mathematical operations of the trained model, which run much faster in reduced precision. The training and hyperparameter tuning of the NN model is very time-consuming. In each tuning trial, we suggest a new set of hyperparameters and a network topology to train and do cross-validation on test sets. These model training and optimization processes usually take hundreds of trials to determine the best model but they are all offline for Mini-Me.

6 DISCUSSION

Comparison with Prior Works. Based on the study cases above, we demonstrate the attack detection and protection of Mini-Me. Compared to recent control invariant-based works (CI [27], SAVIOR [59]) in Table 1, they mainly focus on the external sensor spoofing attacks and barely explore the internal control logic of RAV software. Instead of building a similar estimation model based on high-level vehicle dynamics and sensors, Mini-Me is a fine-grained solution that explores the semantic information of controllers using dataflow analysis, and creates a concise neural network-based "replica" with explainable intermediate concepts. Compared to bug tracking and parameter pinpoint tools like RVFuzeer [49] and Mayday [48] in Table 1, they are mainly focusing on the semantic bug investigation and traceback of control program execution. Mini-Me organically combines the dataflow information of internal control program execution with the external sensor measurements to build a more comprehensive and explainable estimation model.

Compared to other NN-based or learning-based system modeling solutions ([24, 40, 44, 71, 75]), Mini-Me can not only detect the cyber-oriented attacks but also defense against malicious tampering of internal control logic. The prior works mainly focus on extracting the operational stability-related vehicle dynamics to construct a systematical model with learning techniques. They did not sufficiently explore the internal semantic information in control logic. Thus, the sensor, actuator, physical dynamics in such a "black box", especially for highly non-linear cyber-physical systems, will result in non-transparent training and lack explainability [42] even with satisfying accuracy. In contrast, Mini-Me uses function-level time

profiling and dataflow analysis to study the variable changes and mutual dependencies of control state variables and intermediate variables, which provides us with concrete input and output sets for training a neural network-based approximate computing model. Furthermore, Mini-Me tunes and trains the LSTM-based neural network model and precisely estimates time-series data to defense all the five kinds of attacks listed in Table 1.

Platform Applicability. Regarding Mini-Me's applicability for different RAV models, Mini-Me's analysis (including its automated LLVM passes, data flow analyses, neural network training and error monitoring) is performed automatically and is agnostic to the RAV's controller software, which can be easily expanded to other RAV platforms as described in Table 2 with very low code size overhead and runtime performance overhead. Most drones nowadays (including Crazyflie) run their controllers on ARM processors, which our evaluations are based on. In terms of the minimum system requirements, Mini-Me does not require additional hardware and runs on the drone's processor with no impact on the main flight operation as empirically shown by our experiments.

Limitations and Stealthy Attacks. Mini-Me's main objective is to detect attacks on the controller that result in fairly distinguishable behavioral divergence from its legitimate execution. However, there have been recent studies on stealthy (or mimicry) attacks [29, 59] that leverage gradual and minimal data value manipulations to evade anomaly detection solutions [27]. For instance, the attacker might be aware of the defense mechanism and launch the stealthy attacks by crafting malicious parameter values, which will not make the monitoring model observe large enough deviation and therefore go undetected. The existing robustness design of RAV's control model and algorithms is not sufficient to mitigate these threats. Therefore, detecting such attacks is a real challenge in practical settings without affecting the false positive rates of the detector negatively. Further research is required to make sure such attacks can be detected accurately.

For Mini-Me's accurate performance and attack detection, it is important to make sure that the DNN model will imitate the original controller's behavior in all feasible system states including the execution paths of the controller software. In the case of complex and large-scale controller algorithms and software, complete code and system state space coverage could face scalability issues. As the result, reverse engineering analysis of the controller and construction of DNN models may require more static and dynamic techniques such as program-level symbolic executions [57] that we did not explore in this work.

7 RELATED WORKS

Drone Attacks. The prior attacks exploiting the RAV's attack surfaces focus on the external sensor readings and remote configuration interfaces. Sensor spoofing [31, 68, 72] has been identified as one of the most common forms of cyber-attacks towards drones these days. Optical sensor flow attack [31] demonstrate that attackers can form implicit control channels and take direct control over a drone by tricking the optical flow sensing. GPS spoofing attacks [72] achieve convert satellite-lock takeover by generating the malicious signals to spoof the GPS receivers. Authors in [68] present an attack on drones equipped with vulnerable gyroscope sensors by using intentional sound noise, which easily causes drone crashes. As a type of cyber-physical system (CPS), drones are also facing the challenges of various CPS attacks such as false data injection attacks [56, 70, 76, 77] and control logic manipulation [38, 54]. Those forged sensor data and malicious control programs can be sent and mislead the state estimators in drones for actuators operation corruption.

Defenses. To defend against these threats, possible solutions focused on verifying the control logic with safety checks before its execution in system actuators. TSV [57] goes even further to combine symbolic execution with model checking and improves the proof-based approaches by lumping together the safety constraints from transformed intermediate language expression. In response, anomaly-based solutions [21, 61, 80] have been proposed to detect malicious and unexpected behaviors by strengthening the robustness and reliability of state estimators. Other recent works [26, 36, 83] focus on distinguishing anomalous behavioral patterns as opposed to the signature-based paradigm or detecting attacks with received sensor measurements. Regarding Mini-Me vs. CFG checking solutions (which we assume are the CFI techniques [15, 30, 35, 76]), controller attacks can exploit a variety of vulnerabilities such as control semantic bugs or data-oriented attacks that affect the controller's operations without violating the control flow. While CFI techniques are able to detect the control flow attacks, other attack types can evade CFI as shown by the past work. Mini-Me's defense is attack type-agnostic and can detect controller misbehaviors (it's I/O profiles from/to sensors/actuators) no matter how the controller software gets attacked. The learning-based defenses [24, 40, 44, 71, 75] use neural network-based models to monitor malicious intrusions at runtime. This kind of learning-based approach usually requires sensor readings, actuator behaviors and accurate definitions of system dynamics. However, the generated systematical model is opaque during training and lacks explainability. The adaptive neural networks [23, 64, 82] are used as online approximators to estimate the nonlinear fault function in the fault-tolerant control scheme. But their solutions cannot protect against real-time control logic attacks and also have limited guarantees in highly nonlinear systems. For our solution, we do not need to extract the physical dynamics or identify detailed specifications. Mini-Me automatically optimizes the neural network topology and generates safe predictions via the trained NN model for runtime attack monitoring.

8 CONCLUSION

With many challenges arising about the vulnerabilities of embedded controllers in cyber-physical systems, we propose Mini-Me, an automated, precise and robust framework for checking the validity of current control logic. We utilize the combination of reverse engineering technique and dataflow analysis to identify the variable changes and their dependencies in the critical controller functions of RAVs. We develop a lightweight neural network model to learn an extracted control function, which is deployed for the online detection of a wide array of data-oriented attacks. We show the novelty and effectiveness of Mini-Me in the experiments on a real physical RAV system Crazyflie 2.0 and simulated RAVs.

ACKNOWLEDGMENTS

This material is based upon work supported by the Department of Energy under Award Number DE-OE0000780, Cyber Resilient Energy Delivery Consortium (CREDC).

REFERENCES

[1] 2014. *Gazebo: Open Source Robotics Foundation.* http://gazebosim.org/.
[2] 2020. *3rd Eye Scene.* https://github.com/csiro-robotics/3rdEyeScene.
[3] 2020. *Optuna: A hyperparameter optimization framework.* https://github.com/optuna/optuna.
[4] 2020. *RetDec: a retargetable machine-code decompiler based on LLVM.* https://retdec.com/.
[5] 2020. *SITL Simulator (Software in the Loop).* https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html.
[6] 2021. *Black Magic Probe.* https://github.com/blacksphere/blackmagic/wiki.
[7] 2021. *LLVM Alias Analysis Infrastructure.* https://llvm.org/docs/AliasAnalysis.html.
[8] 2021. *LLVM Pass Framework.* https://llvm.org/docs/WritingAnLLVMPass.html.
[9] 2021. *PX4 Pro Open Source Autopilot - Open Source for Drones.* http://px4.io.
[10] Retrieved July 1, 2020. *DHL parcelcopter launches initial operations for research purposes.* https://www.dhl.com/en/press/releases/releases_2014/group/dhl_parcelcopter_launches_initial_operations_for_research_purposes.html.
[11] Retrieved July, 2020. *ArduPilot: versatile, Trusted, Open Autopilot software for drones and other autonomous systems.* https://ardupilot.org/about.
[12] Retrieved September 1, 2020. *The Crazyflie 2.0, a lightweight, open source flying development platform.* https://www.bitcraze.io/products/old-products/crazyflie-2-0/.
[13] Retrieved September 1, 2020. *Valgrind: the instrumentation framework for building dynamic analysis tools.* https://valgrind.org/.
[14] Alireza Abbaspour, Kang K Yen, Shirin Noei, and Arman Sargolzaei. 2016. Detection of fault data injection attack on uav using adaptive neural network. *Procedia computer science* 95 (2016), 193–200.
[15] Tigist Abera, Raad Bahmani, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, and Matthias Schunter. 2019. DIAT: Data Integrity Attestation for Resilient Collaboration of Autonomous Systems.. In *NDSS*.
[16] Sridhar Adepu, Ferdinand Brasser, Luis Garcia, Michael Rodler, Lucas Davi, Ahmad-Reza Sadeghi, and Saman Zonouz. 2020. Control behavior integrity for distributed cyber-physical systems. In *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 30–40.
[17] Naif Saleh Almakhdhub, Abraham A Clements, Saurabh Bagchi, and Mathias Payer. 2020. μRAI: Securing embedded systems with return address integrity. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26.*
[18] Amazon. Retrieved July 1, 2020. *First Prime Air Delievery.* https://www.amazon.com/Amazon-Prime-Air/b?node=8037720011.
[19] Domagoj Babic and Alan J Hu. 2008. Calysto: scalable and precise extended static checking. In *Proceedings of the 30th international conference on Software engineering*. 211–220.
[20] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks* 5, 2 (1994), 157–166.
[21] Rakesh B Bobba, Katherine M Rogers, Qiyan Wang, Himanshu Khurana, Klara Nahrstedt, and Thomas J Overbye. 2010. Detecting false data injection attacks on dc state estimation. In *Preprints of the First Workshop on Secure Control Systems, CPSWEEK*, Vol. 2010.
[22] Nicholas Carlini, Antonio Barresi, Mathias Payer, David Wagner, and Thomas R Gross. 2015. Control-Flow Bending: On the Effectiveness of Control-Flow Integrity.. In *USENIX Security Symposium*. 161–176.
[23] Mou Chen, Peng Shi, and Cheng-Chew Lim. 2015. Adaptive neural fault-tolerant control of a 3-DOF model helicopter system. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 46, 2 (2015), 260–270.
[24] 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. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 648–660.
[25] Long Cheng, Ke Tian, and Danfeng Yao. 2017. Orpheus: Enforcing cyber-physical execution semantics to defend against data-oriented attacks. In *Proceedings of the 33rd Annual Computer Security Applications Conference*. 315–326.
[26] Steven Cheung, Bruno Dutertre, Martin Fong, Ulf Lindqvist, Keith Skinner, and Alfonso Valdes. 2007. Using model-based intrusion detection for SCADA networks. In *Proceedings of the SCADA security scientific symposium*, Vol. 46. Citeseer, 1–12.
[27] Hongjun Choi, Wen-Chuan Lee, Yousra Aafer, Fan Fei, Zhan Tu, Xiangyu Zhang, Dongyan Xu, and Xinyan Xinyan. 2018. Detecting attacks against robotic vehicles: A control invariant approach. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 801–816.
[28] Abraham A Clements, Naif Saleh Almakhdhub, Khaled S Saab, Prashast Srivastava, Jinkyu Koo, Saurabh Bagchi, and Mathias Payer. 2017. Protecting Bare-metal Embedded Systems With Privilege Overlays. In *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 289–303.
[29] Pritam Dash, Mehdi Karimibiuki, and Karthik Pattabiraman. 2019. Out of control: stealthy attacks against robotic vehicles protected by control-based techniques. In *Proceedings of the 35th Annual Computer Security Applications Conference*. 660–672.
[30] Lucas Davi, Matthias Hanreich, Debayan Paul, Ahmad-Reza Sadeghi, Patrick Koeberl, Dean Sullivan, Orlando Arias, and Yier Jin. 2015. HAFIX: Hardware-assisted flow integrity extension. In *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 74.
[31] Drew Davidson, Hao Wu, Robert Jellinek, Vikas Singh, and Thomas Ristenpart. 2016. Controlling UAVs with Sensor Input Spoofing Attacks.. In *WOOT*.
[32] Lisa Nguyen Quang Do, Karim Ali, Benjamin Livshits, Eric Bodden, Justin Smith, and Emerson Murphy-Hill. 2017. Just-in-time static analysis. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 307–317.
[33] Jeffrey L Elman. 1990. Finding structure in time. *Cognitive science* 14, 2 (1990), 179–211.
[34] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. 2012. Neural acceleration for general-purpose approximate programs. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 449–460.
[35] Sriharsha Etigowni, Shamina Hossain-McKenzie, Maryam Kazerooni, Katherine Davis, and Saman Zonouz. 2018. Crystal (ball): I Look at Physics and Predict Control Flow! Just-Ahead-Of-Time Controller Recovery. In *Proceedings of the 34th Annual Computer Security Applications Conference*. ACM, 553–565.
[36] Sriharsha Etigowni, Dave Jing Tian, Grant Hernandez, Saman Zonouz, and Kevin Butler. 2016. CPAC: securing critical infrastructure with cyber-physical access control. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*. ACM, 139–152.
[37] Fan Fei, Zhan Tu, Ruikun Yu, Taegyu Kim, Xiangyu Zhang, Dongyan Xu, and Xinyan Deng. 2018. Cross-layer retrofitting of UAVs against cyber-physical attacks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 550–557.
[38] Luis Garcia, Ferdinand Brasser, Mehmet H Cintuglu, Ahmad-Reza Sadeghi, Osama Mohammed, and Saman A Zonouz. 2017. Hey, my malware knows physics! attacking plcs with physical model aware rootkit. In *Proceedings of the Network & Distributed System Security Symposium, San Diego, CA, USA*. 26–28.
[39] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. 1999. Learning to forget: Continual prediction with LSTM. (1999).
[40] Jonathan Goh, Sridhar Adepu, Marcus Tan, and Zi Shan Lee. 2017. Anomaly detection in cyber physical systems using recurrent neural networks. In *2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE)*. IEEE, 140–145.
[41] Jared Green. Retrieved July 15, 2020. *Drones Will Elevate Urban Design.* https://www.smartcitiesdive.com/ex/sustainablecitiescollective/drones-will-elevate-urban-design/1053491/.
[42] Wenbo Guo, Dongliang Mu, Jun Xu, Purui Su, Gang Wang, and Xinyu Xing. 2018. Lemna: Explaining deep learning based security applications. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 364–379.
[43] Todd E Humphreys, Brent M Ledvina, Mark L Psiaki, Brady W O'Hanlon, and Paul M Kintner. 2008. Assessing the spoofing threat: Development of a portable GPS civilian spoofer. In *Radionavigation laboratory conference proceedings*.
[44] Khurum Nazir Junejo and Jonathan Goh. 2016. Behaviour-based attack detection and classification in cyber physical systems using machine learning. In *Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security*. ACM, 34–43.
[45] Anastasis Keliris and Michail Maniatakos. 2018. Icsref: A framework for automated reverse engineering of industrial control systems binaries. *arXiv preprint arXiv:1812.03478* (2018).
[46] Andrew J Kerns, Daniel P Shepard, Jahshan A Bhatti, and Todd E Humphreys. 2014. Unmanned aircraft capture and control via GPS spoofing. *Journal of Field Robotics* 31, 4 (2014), 617–636.
[47] Chung Hwan Kim, Taegyu Kim, Hongjun Choi, Zhongshu Gu, Byoungyoung Lee, Xiangyu Zhang, and Dongyan Xu. [n.d.]. Securing Real-Time Microcontroller Systems through Customized Memory View Switching. ([n. d.]).
[48] Taegyu Kim, Chung Hwan Kim, Altay Ozen, Fan Fei, Zhan Tu, Xiangyu Zhang, Xinyan Deng, Dave Jing Tian, and Dongyan Xu. 2020. From Control Model to Program: Investigating Robotic Aerial Vehicle Accidents with {MAYDAY}. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*. 913–930.
[49] Taegyu Kim, Chung Hwan Kim, Junghwan Rhee, Fan Fei, Zhan Tu, Gregory Walkup, Xiangyu Zhang, Xinyan Deng, and Dongyan Xu. 2019. RVFUZZER: finding input validation bugs in robotic vehicles through control-guided testing. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 425–442.

[50] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[51] Patrick Koeberl, Steffen Schulz, Ahmad-Reza Sadeghi, and Vijay Varadharajan. 2014. TrustLite: A security architecture for tiny embedded devices. In *Proceedings of the Ninth European Conference on Computer Systems*. 1–14.

[52] Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. 2020. Concept bottleneck models. In *International Conference on Machine Learning*. PMLR, 5338–5348.

[53] William Landi and Barbara G Ryder. 2004. A safe approximate algorithm for interprocedural pointer aliasing. *ACM SIGPLAN Notices* 39, 4 (2004), 473–489.

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

[55] Yanlin Li, Jonathan M McCune, and Adrian Perrig. 2011. VIPER: Verifying the integrity of peripherals' firmware. In *Proceedings of the 18th ACM conference on Computer and communications security*. 3–16.

[56] Stephen McLaughlin and Saman Zonouz. 2014. Controller-aware false data injection against programmable logic controllers. In *Smart Grid Communications (SmartGridComm), 2014 IEEE International Conference on*. IEEE, 848–853.

[57] Stephen E McLaughlin, Saman A Zonouz, Devin J Pohly, and Patrick D McDaniel. 2014. A Trusted Safety Verifier for Process Controller Code.. In *NDSS*, Vol. 14.

[58] Job Noorman, Pieter Agten, Wilfried Daniels, Raoul Strackx, Anthony Van Herrewege, Christophe Huygens, Bart Preneel, Ingrid Verbauwhede, and Frank Piessens. 2013. Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base. In *22nd {USENIX} Security Symposium ({USENIX} Security 13)*. 479–498.

[59] Raul Quinonez, Jairo Giraldo, Luis Salazar, Erick Bauman, Alvaro Cardenas, and Zhiqiang Lin. 2020. {SAVIOR}: Securing Autonomous Vehicles with Robust Physical Invariants. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*. 895–912.

[60] Ihab Samy, Ian Postlethwaite, and Dawei Gu. 2008. Neural network based sensor validation scheme demonstrated on an unmanned air vehicle (UAV) model. In *2008 47th IEEE Conference on Decision and Control*. IEEE, 1237–1242.

[61] Henrik Sandberg, André Teixeira, and Karl H Johansson. 2010. On security indices for state estimators in power networks. In *First Workshop on Secure Control Systems (SCS), Stockholm, 2010*.

[62] Hovav Shacham. 2007. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 552–561.

[63] Hovav Shacham, Matthew Page, Ben Pfaff, Eu-Jin Goh, Nagendra Modadugu, and Dan Boneh. 2004. On the effectiveness of address-space randomization. In *Proceedings of the 11th ACM conference on Computer and communications security*. 298–307.

[64] Qikun Shen, Bin Jiang, Peng Shi, and Cheng-Chew Lim. 2014. Novel neural networks-based fault tolerant control scheme with fault alarm. *IEEE transactions on cybernetics* 44, 11 (2014), 2190–2201.

[65] Qingkai Shi, Xiao Xiao, Rongxin Wu, Jinguo Zhou, Gang Fan, and Charles Zhang. 2018. Pinpoint: Fast and precise sparse value flow analysis for million lines of code. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 693–706.

[66] Yan Shoshitaishvili, Ruoyu Wang, Christopher Salls, Nick Stephens, Mario Polino, Andrew Dutcher, John Grosen, Siji Feng, Christophe Hauser, Christopher Kruegel, et al. 2016. Sok:(state of) the art of war: Offensive techniques in binary analysis. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 138–157.

[67] Yasser Shoukry, Paul Martin, Paulo Tabuada, and Mani Srivastava. 2013. Non-invasive spoofing attacks for anti-lock braking systems. In *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 55–72.

[68] Yunmok Son, Hocheol Shin, Dongkwan Kim, Youngseok Park, Juhwan Noh, Kibum Choi, Jungwoo Choi, and Yongdae Kim. 2015. Rocking drones with intentional sound noise on gyroscopic sensors. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*. 881–896.

[69] Pengfei Sun, Luis Garcia, and Saman Zonouz. 2019. Tell Me More Than Just Assembly! Reversing Cyber-Physical Execution Semantics of Embedded IoT Controller Software Binaries. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 349–361.

[70] Laszlo Szekeres, Mathias Payer, Tao Wei, and Dawn Song. 2013. Sok: Eternal war in memory. In *2013 IEEE Symposium on Security and Privacy*. IEEE, 48–62.

[71] HA Talebi and RV Patel. 2006. An intelligent fault detection and recovery scheme for reaction wheel actuator of satellite attitude control systems. In *2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control*. IEEE, 3282–3287.

[72] Nils Ole Tippenhauer, Christina Pöpper, Kasper Bonne Rasmussen, and Srdjan Capkun. 2011. On the requirements for successful GPS spoofing attacks. In *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 75–86.

[73] Minh Tran, Mark Etheridge, Tyler Bletsch, Xuxian Jiang, Vincent Freeh, and Peng Ning. 2011. On the expressiveness of return-into-libc attacks. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 121–141.

[74] Timothy Trippel, Ofir Weisse, Wenyuan Xu, Peter Honeyman, and Kevin Fu. 2017. WALNUT: Waging doubt on the integrity of mems accelerometers with acoustic injection attacks. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 3–18.

[75] Qing Wu and Mehrdad Saif. 2007. Repetitive learning observer based actuator fault detection, isolation, and estimation with application to a satellite attitude control system. In *2007 American Control Conference*. IEEE, 414–419.

[76] Yubin Xia, Yutao Liu, Haibo Chen, and Binyu Zang. 2012. CFIMon: Detecting violation of control flow integrity using performance counters. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*. IEEE, 1–12.

[77] Le Xie, Yilin Mo, and Bruno Sinopoli. 2010. False data injection attacks in electricity markets. In *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*. IEEE, 226–231.

[78] S. Bharadwaj Yadavalli and Aaron Smith. 2019. Raising Binaries to LLVM IR with MCTOLL (WIP Paper). In *Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems* (Phoenix, AZ, USA) *(LCTES 2019)*. Association for Computing Machinery, New York, NY, USA, 213–218. https://doi.org/10.1145/3316482.3326354

[79] Chen Yan, Wenyuan Xu, and Jianhao Liu. 2016. Can you trust autonomous vehicles: Contactless attacks against sensors of self-driving vehicle. *DEF CON* 24, 8 (2016), 109.

[80] Man-Ki Yoon, Bo Liu, Naira Hovakimyan, and Lui Sha. 2017. Virtualdrone: virtual sensing, actuation, and communication for attack-resilient unmanned aerial systems. In *Proceedings of the 8th international conference on cyber-physical systems*. 143–154.

[81] Jie Zhou, Yufei Du, Zhuojia Shen, Lele Ma, John Criswell, and Robert J Walls. 2020. Silhouette: Efficient protected shadow stacks for embedded systems. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*. 1219–1236.

[82] Qi Zhou, Peng Shi, Honghai Liu, and Shengyuan Xu. 2012. Neural-network-based decentralized adaptive output-feedback control for large-scale stochastic nonlinear systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 42, 6 (2012), 1608–1619.

[83] Saman Zonouz, Katherine M Rogers, Robin Berthier, Rakesh B Bobba, William H Sanders, and Thomas J Overbye. 2012. SCPSE: Security-oriented cyber-physical state estimation for power grid critical infrastructures. *IEEE Transactions on Smart Grid* 3, 4 (2012), 1790–1799.

[84] Saman Zonouz, Julian Rrushi, and Stephen McLaughlin. 2014. Detecting industrial control malware using automated PLC code analytics. *IEEE Security & Privacy* 12, 6 (2014), 40–47.