RvPlayer: Robotic Vehicle Forensics by Replay with What-if Reasoning

Hongjun Choi
Purdue University
choi293@purdue.edu

Zhiyuan Cheng
Purdue University
cheng443@purdue.edu

Xiangyu Zhang
Purdue University
xyzhang@cs.purdue.edu

***Abstract*—Robotic vehicle (RV) attack forensics identifies root cause of an accident. Reproduction of accident and reasoning about its causation are critical steps in the process. Ideally, such investigation would be performed in real-world field tests by faithfully regenerating the environmental conditions and varying the different factors to understand causality. However, such analysis is extremely expensive and in many cases infeasible due to the difficulties of enforcing physical conditions. Existing RV forensics techniques focus on faithful accident reproduction in simulation and hence lack the support of causality reasoning. They also entail substantial overhead. We propose RvPlayer, a system for RV forensics. It supports replay with what-if reasoning inside simulator (e.g., checking if an accident can be avoided by changing some control parameter, code, or vehicle states). It is a low-cost replacement of the expensive field test based forensics. It features an efficient demand-driven adaptive logging method capturing non-deterministic physical conditions, and a novel replay technique supporting various replay policies that selectively enable/disable information during replay for root cause analysis. Our evaluation on 6 RVs (4 real and 2 virtual), 5 real-world auto-driving traces, and 1194 attack instances of various kinds reported in the literature shows that it can precisely pinpoint the root causes of these attacks without false positives. It has only 6.57% of the overhead of a simple logging design.**

I. Introduction

Robotic vehicles (RV) are becoming increasingly popular due to the advances of AI and the fast-growing applications of these vehicles [1], [2]. They hence become an important target of attackers. RVs are combinations of cyber and physical components, which introduce a broader attack surface compared to traditional computation systems. Various attacks have been developed for RV systems, including sensor spoofing [3]–[6], side-channels [7], [8], and APTs [9]–[11], attacking both the cyber and the physical domains. As RV attacks often have physical consequences, endangering properties and even human lives, RV security assurance is an important challenge. Many existing defense techniques fall into the category of anomaly detection [12]–[15] that utilizes models and invariants to detect behavioral anomalies at runtime or offline. Attack resilient approaches [16]–[18] recover systems from attacks to continue normal operations.

In this paper, we focus on *RV forensics*, which is to identify root cause in post-accident investigation (e.g., determine which parameter update caused an accident or which sensor was spoofed). This is an orthogonal challenge to the aforementioned ones. Note that forensics is particularly important when attacks are stealthy and/or have a long duration, making runtime detection difficult. The control of an RV is through a loop that involves control software, vehicle, and the environment. As such, external inputs to cyber components (e.g., sensor readings) are the aggregation of *both the environmental conditions and the effects of motor thrusts decided by the previous control loop iteration*. Many existing RV or Cyber Physical System (CPS) forensics techniques, however, use an open loop abstraction and focus only on cyber components [19]–[22]. They replay the observed external inputs and do not distinguish effects from the environment and from the vehicle. Therefore, although they can faithfully replay an accident, they do not support *what-if reasoning*, a key step in causality analysis [23], [24]), which changes individual factors of a system (e.g., reverts a malicious parameter update), replays physical world disturbances, and observes consequences (e.g., if the accident is successfully avoided). A factor is considered a root cause if enabling/disabling it induces/avoids the accident. Recently, a state-of-the-art RV problem diagnosis tool MAYDAY [25] utilizes additional runtime program logs and threshold based physical anomaly detection. It focuses on accidents caused by control software bugs. It can narrow down root cause to a program region. It does not focus on accidents caused by external perturbations such as sensor spoofing.

Ideally, RV forensics shall be performed through real-world field tests in which the same environmental conditions are faithfully regenerated and the accident is replayed with various what-if changes to understand causation. However, this is extremely expensive and in many cases infeasible due to the difficulty of recreating same environmental disturbances. Therefore, we propose a novel low-cost replacement that enables forensics in simulation. It selectively and efficiently records critical runtime information with an adaptive frequency. Specifically, it constructs a dynamics model of the subject RV and runs it as a shadow system to the real one during operation. When the model can correctly predict the real system behaviors, the RV is considered not having substantial environmental disturbances and a low logging frequency is used. Otherwise, high frequencies proportional to the level of anomaly are used. Different from traditional CPS replay techniques that treat external inputs from the physical world as a whole, our technique decouples it to environmental disturbances and effects by motor thrusts such that the former can be saved and replayed independently from the latter, achieving reproduction of physical environments. During replay, it leverages the aforementioned model (in simulation) to regenerate the states that were not recorded during operation.

Network and Distributed Systems Security (NDSS) Symposium 2022
24-28 April 2022, San Diego, CA, USA
ISBN 1-891562-74-6
https://dx.doi.org/10.14722/ndss.2022.24244
www.ndss-symposium.org

It also re-applies the recorded disturbances to recreate the environmental conditions. More importantly, it supports a wide range of policies that selectively enable/disable certain information during replay to allow what-if reasoning for root cause identification. More details can be found in Section III.

Our contributions are summarized as follows.

- We develop an efficient demand-driven logging method that features an adaptive logging frequency and decoupling/recording environmental disturbances.
- We develop a novel replay engine that supports various replay policies needed in the what-if reasonings for different kinds of attack. These policies allow disabling/enabling individual events (e.g., parameter updates) and sensors, and changing controller code during replay. The replayed runs (inside the simulator) closely resemble the real test executions (in the physical world) with the same changes applied.
- We develop a prototype RvPlayer and evaluate it on 6 RVs, including drones and rovers, with 4 real vehicles and 2 virtual ones, and on 5 real-world auto-driving operation traces. We also test it on normal operations with real-world considerations (e.g., GPS urban canyon, wind, etc) and popular attacks, including gradual spoofing [4], [6], split-second spoofing [26], parameter tampering [12], code vulnerabilities that can be exploited through physical conditions [27], and sensor fusion spoofing for auto-driving cars [4], with 1194 attack instances. The results show that the logging component is highly efficient, reducing the logging overhead of a naive design by 93.43%. In contrast, due to the different design goal, MAYDAY's space overhead is 40.5 times of RvPlayer's. Our replay policies can precisely pinpoint the causes for all the attacks we study without false positives and identify the attack start-times. In contrast, forensics methods adapted from anomaly detection methods can only identify root causes for very limited attacks, i.e., 32 of the 53 split-second sensor spoofing and sensor fusion spoofing, and miss others. MAYDAY does not handle 6 out of the 12 parameter tampering attacks we studied.

Threat Model. Our threat model is consistent with that in the literature [12], [25], [28]. We make the following assumptions. *(i)* The attacker cannot directly access the internals (i.e., control program) of the target vehicle. Instead, he performs attack through *external* means (e.g., parameter modifications by external commands and sensor spoofing), with the goal of disrupting the planned operations, damaging the vehicle, and covering his tracks. *(ii)* The life-cycle of an attack can consist of multiple steps: intrusion, payload triggering, and symptom manifestation. Intrusion is an act of entering a system and placing malicious components (i.e., payloads). Next, payloads can be triggered immediately or long after intrusion when the system encounters a specific physical condition. The attacker knows what conditions can trigger the planned payload but does not necessarily fabricate the conditions right away. Symptoms of attack (e.g., crashes) can appear immediately or after a certain period of time since payload triggering. *(iii)* The attacker is knowledgeable about the built-in logging functionalities. The attacker gets to know how to attack a specific vehicle by acquiring a copy from the market or even reverse engineering the vehicle's dynamics and control parameters by observing its flights. He cannot directly corrupt the logging component or the generated logs, but he can indirectly compromise them through spoofing. Secure execution environment is provided for the logging component. We assume that the vehicle and recorded logs are accessible during forensics. This assumption is the same as that in the log-based forensics techniques [25], [29].

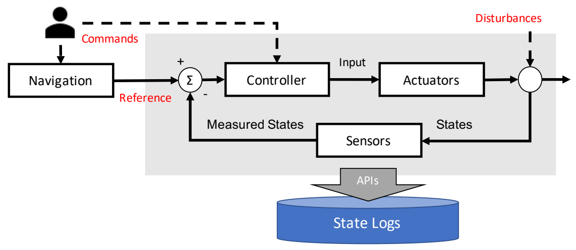


Fig. 1: Typical feedback control loop architecture with state logs

II. Background and Motivation

A. Background: Feedback Control Loop and State Logging

A control system typically uses a feedback-driven and loop-based architecture as shown inside the gray box in Figure 1. Given a reference state (e.g., a waypoint to reach), the system continuously observes the differences between the reference and the current states (measured by sensors), computes the needed control signals, based on some control algorithm with specific parameters, and drives the actuators to reach the target. Observe that the control system mainly takes two kinds of external inputs and operates accordingly. The first kind is called *cyber events* that are high-level instructions from the navigation component or the user, such as user commands to modify a control parameter or setting a reference state. These events are discrete and can be intercepted, stored, and replayed during forensic analysis. The second is physical disturbances (e.g., wind, air resistance, and road friction) from the environment. They are continuous and largely non-deterministic. Different from cyber events, logging and replaying physical signals is challenging, entailing substantial overhead.

Typically, a control program has a few threads, including the monitor thread, main thread, timer thread, IO thread, etc. The control loop is in the main thread and has a higher priority than the others, whereas the IO thread in charge of writing log data to SD card has a low priority. A real-time OS manages the scheduling of the different threads. Additionally in the control loop, there are many tasks in charge of different functions, including navigation, attitude control, fall-safe check, sensor data update, remote communication, data logging, etc. These tasks have their own predefined frequencies and get executed once in a few iterations. In order to maintain a fixed control loop frequency, each iteration has a hard time constraint. In iterations with many tasks scheduled, time-consuming tasks may be skipped (preempted) together with their logging requests, to meet the time constraint.

Modern RV systems often have built-in support to record system states as time-series data. The recorded states may include sensor measurements, control references, actuation signals, and control parameters. For example, in the on-board dataflash log for Ardupilot [30], there are `ACC, GYR,` IMU, IMT, MAG, BARO, GPS, and GPA message types for recording sensor measurements, RCIN and RCOU for actuator signals. AHR2, ATT, and POS for physical status, NKF, PID, MAVC, etc. for control statistics and events.

B. Motivating Example

Advanced stealthy attacks [4], [6], [7], [12], [31] can hide malicious activities from detection techniques. The consequences of these attacks may only become observable long after intrusion. In the following, we present a realistic stealthy attack, *parameter tampering attack* [32], as our motivating example. The attacker issues a malicious command to set a control parameter to an invalid value, and the tampered parameter impacts control operations under a certain triggering condition (e.g., sharp turn with wind). Thus, the malicious payload usually lurks for a long period of time, and attack ramifications occur afterwards. It is hence challenging for existing techniques to detect the root cause or conduct forensic analysis.

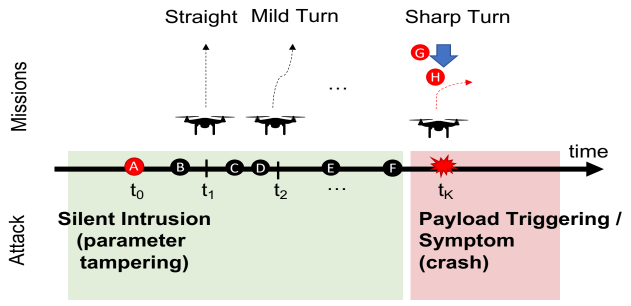

Straight
Mild Turn
Sharp Turn
Missions
G
H
time
A B C D E F
t0 t1 t2 tK
Attack
Silent Intrusion (parameter tampering)
Payload Triggering / Symptom (crash)

Fig. 2: An example of advanced attack. A-G: cyber events and important physical disturbances: a parameter is tampered with at A; the drone makes a scheduled sharp turn at H with strong wind gust denoted by G. The arrows at H and G denote the directions.

Attack Scenario. A package delivery drone mostly flies straight and occasionally makes turns in its daily missions. Figure 2 illustrates an example of multiple missions over time. At time t_0, the attacker issues a malicious command A to enlarge the proportional gain of horizontal velocity's PID controller from 0.6 to 5.7, by tampering with parameter VEL_XY_P in the Ardupilot system. During the $k-1$ deliveries starting at t_1, t_2, ..., t_{k-1} (over several days), respectively, the drone performs simple flights including end-to-end straight routes and mild turns. Various normal commands (B-F) are also issued in the duration. At time t_k (i.e., the kth delivery after the parameter tampering), it flies straight and then performs a scheduled sharp turn (e.g., north to the east) at G, with Beaufort level 6 wind gust [33] (that may move tree branches) blowing from north to south. The environmental condition triggers the injected malicious behavior. In other words, the drone would turn successfully even with the wind, were not for the compromised parameter. Specifically, VEL_XY_P is responsible for the proportional gain of horizontal velocity's PID controller. The update does not have any immediate effect as the drone does not have any aggressive horizontal movement at that moment. Later, upon the triggering condition, it causes the drone to crash. The high control gain leads to sudden acceleration change and divergence of velocity control, and thus the accumulated impact causes a crash. Note that the attack consequences happen long after the initial parameter tampering. There are many legitimate parameter changes in the duration.

Ideally, a forensics technique would faithfully record all the cyber-events and environmental disturbances during operation and replay them *with mutations* in post-mortem analysis (e.g., through multiple field tests). However, existing techniques fall short in various aspects.

Limitations of Existing Approaches. A wide spectrum of runtime anomaly detection techniques [12]–[15], [34] developed by the security community can detect attack ramifications/symptoms. However, a root cause that occurred long before attack consequences is difficult to locate. Efficient state logging and replay based forensic techniques are essential.

(1) *Existing Logging Techniques Miss Critical Evidence Due to Resource Constraints*. Existing RV/CPS data loggers [35], [36] record operational data in non-volatile memory (usually onboard). A prominent challenge is to handle storage overhead. Control loops usually run at a high frequency (e.g., 400Hz). At each iteration, messages of the types mentioned at the end of Section II-A are generated. If all message types are considered, the memory consumption is 13.82 GB/day for a PX4 drone, and 5.30GB/day for Ardupilot. As such, a typical 32GB storage can only record 6 days' flight for Ardupilot, without considering space consumption caused by other data such as video and audio. The systems would automatically delete log entries when the space is not enough. The real-time OS and the control algorithm may preempt/discard logging operations when they are under time pressure. As such, state logs are rarely complete. In a real drone flight such as those in the motivating example, due to the low CPU speed, 85.6% of log events are missing if FAST_ATT and PID logging options are turned on (Details are available in Section S-A of our online document [37].).

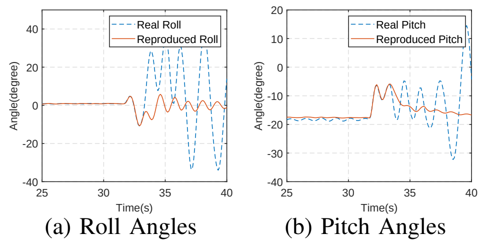


Fig. 3: Comparisons between the real traces of motivating example and the replayed traces without environmental disturbances

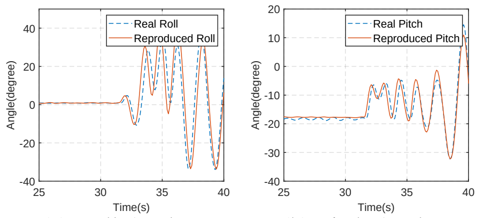
(2) *Existing Techniques Lack Sufficient Replay Support*. There are techniques that log cyber-events and generate operation trace from such events in a simulation environment for forensic analysis [38]–[40]. While these techniques are sufficient in diagnosing deterministic bugs/attacks, they can hardly analyze those that are conditioned on *nondeterministic* environment disturbances. Figure 3 shows the operation traces (i.e., roll and pitch) of the vehicle in our attack example when environmental disturbances are precluded, and its comparison with the real operation traces. Observe that the crash vanishes and there are substantial errors in the two traces, illustrating the importance of considering environmental disturbances.

There are also classic program/system record and replay techniques [39], [41]–[45] that can faithfully reproduce cyber-space execution, for forensics and debugging. These techniques aim at faithful reproduction, and lack the support of what-if reasoning, which entails replaying an execution in the presence of changes (e.g., configuration and program changes). Replay in the cyber space with changes is in general highly challenging *as these changes may lead to program execution path differences such that log entries may mis-align with the replayed execution, causing replay exceptions [46]*. In addition, although these techniques can record sensor readings, which are the joint effects of motor thrusts and environmental disturbances, they can hardly isolate the latter for environment reproduction. Furthermore, replay with what-if changes in real-world is extremely expensive and recreating the same environmental disturbances is difficult since these are non-deterministic and hard to measure individually. These features make what-if analysis for RVs more challenging.

Recent CPS Forensic Technique MAYDAY [25]. MAYDAY aims to identify a potentially problematic control code for a runtime accident. A control system consists of a set of controller components (or controllers in short) controlling different physical aspects of a vehicle. These controllers have inter-dependences. MAYDAY leverages a pre-constructed dependency graph between controllers. When an accident happens, denoted as states substantially deviated from their references at some controller (e.g., position discrepancy), MAYDAY leverages the graph to back-trace to the controller that has the initial state corruption. The human inspector then looks into the code region of the controller to find the root cause. To do so, MAYDAY instruments control program and records all control states and reference points in each controller, which enables source code level debugging. While it is highly effective in diagnosing controller bugs and mission command bugs, in accidents involved external perturbations (that we target), due to the highly iterative nature of controller computation, small state corruption may be quickly propagated to other controllers. It hence becomes difficult to identify the initial root cause controller using a method like MAYDAY.

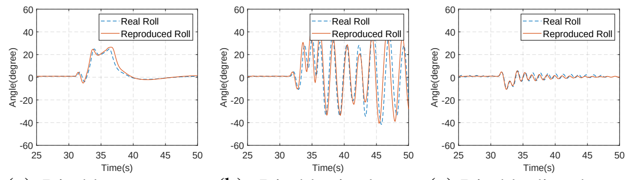
For our motivating example, since the crash is triggered by environmental conditions instead of cyber-space state corruption in some controller, the first controller reacts to the environmental disturbance may not be the controller with the compromised parameter. Actually in 0.1 seconds after the sharp turn with the wind gust, all controllers are considered abnormal by MAYDAY. Although a parameter of the velocity controller is compromised, MAYDAY reports the position controller as the root. Even if it identified the velocity controller correctly, it remains a challenge to determine what is wrong in the controller as the state corruption could be induced by a bug in the code or a parameter update, not to mention the 5 other parameter updates for the same controller over the attack duration. In such a case, what-if reasoning is needed to determine a particular parameter update is the cause of accident. Despite its effectiveness in locating problematic controller(s), MAYDAY does not perform what-if analysis as it does not replay missions. Our technique is hence complementary to MAYDAY.

Our Approach. Our technique records cyber events, states and environmental disturbances. To reduce space consumption, it is demand-driven and only records when disturbances trigger state changes of the vehicle that cannot be predicted by the dynamics model. In addition, it features the ability of reverse engineering the *aggregated* effects of all environmental disturbances and decoupling them from the effects by motor thrusts. The former is hence recorded (and later reapplied during replay). Note that many environmental conditions cannot be measured by RVs. For instance, wind gusts or precipitation is a typical environmental condition affecting drone's behavior. However, most autopilot systems and commercial vehicles cannot provide the individual measurements of external environmental effects (e.g., aerodynamic effects and magnetic forces) unless a system is equipped with designated sensors. In our motivation attack, our logging component consumes space at a rate of 0.17GB/day (compared to 4.04GB/day with builtin logging).



(a) Roll Angles (b) Pitch Angles

Fig. 4: Comparisons between the real traces and the replayed traces with captured disturbances



(a) Disable root cause parameter (b) Disable irrelevant parameter (c) Disable disturbance

Fig. 5: What-if reasoning by changing settings

In the forensics stage, our technique supports faithful replay and replay with what-if reasoning. Faithful replay is achieved by rerunning the mission in a simulation environment and reapplying the recorded environmental disturbances. Since we leverage simulation during forensics, we can replay the accident as many times as we want. The replay of recorded disturbances does not require any environment simulation plugins (e.g., wind /wind-gust plugin). We also support what-if reasoning by automatically making changes during replay, such as disabling parameter updates, injecting new updates, and changing controller code. This is enabled by our capabilities of isolating environmental disturbances and reapplying them during replay. In our example, the vehicle crashes when making a sharp turn under wind gust. Figure 4 shows that our technique can faithfully reproduce the crash. In addition, Figure 5 shows that when we mutate the conditions, including figure (a) disabling the root cause parameter update; (b) disabling an irrelevant parameter update; and (c) disabling the environmental disturbances, how the vehicle behaviors change with these mutations during replay, and how the replayed traces differ from the real traces with the same mutations. In other words, on one hand we change the settings and replay the cyber events and the recorded external disturbances with the changed settings to get the first set of traces. On the other hand, we also *re-execute* the mission with the changed settings and regenerate

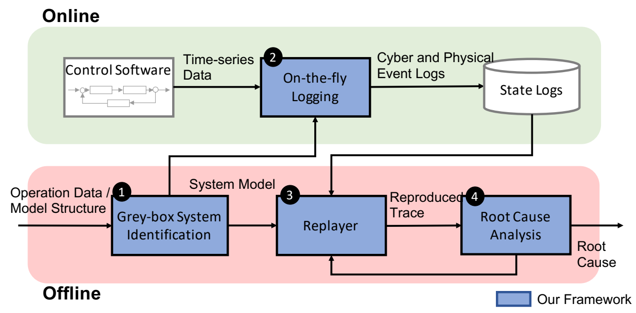


Fig. 6: RvPlayer Overview

the same set of environmental disturbances (e.g., wind gust)[1] to get the second set of operation traces. Then we compare the two sets. Observe that the replayed traces (the red solid curves in Figure 5) closely resemble the re-execution traces (the blue dashed curves), illustrating the correctness of our what-if reasoning. In addition, the results clearly demonstrate that the `VEL_XY_P` parameter update is the root cause and it needs the triggering condition of the wind. Missing either precludes the crash. That is, the drone does not crash in Figure 5 (a) or (c) as the curves are stablized towards the end, while it does in (b) as the fluctuation aggravates.

In the presence of sensor spoofing, logs are corrupted. A faithful replay technique will likely fail. RvPlayer leverages information redundancy and dependences in sensor fusion such that it systematically disables sensors according to their dependence order, to identify spoofed sensors and the spoofing attack starting time (Section IV-B).

III. Design

Figure 6 presents the work-flow of our technique. First, RvPlayer derives the subject vehicle's dynamics model via grey-box system identification (❶) based on the vehicle's operation profile and a model structure pre-selected for a vehicle category. The model describes the expected vehicle behaviors, given the current states and external signals, including control signals and environmental disturbances. The model is used to facilitate demand-driven logging, namely, we adaptively log when the vehicle's states deviate from the model's prediction. This is because during replay we can use the model to regenerate the states that can be precisely predicted. Second, our logger (❷) instruments the control software and records states and environmental disturbances. It uses a high logging frequency to capture disturbances accurately when the vehicle's behaviors deviate from the model prediction and a low frequency otherwise. Note that RvPlayer logs minimal state information even when behaviors are predictable. Such information is used for regular correction of accumulated errors during replay, which are inevitable. Upon an accident of interest, the replayer (❸) replays the same mission with the recorded disturbances and states. Finally, our root cause analysis (❹) identifies the root cause by performing systematic what-if analysis. Details will be discussed in the following.

A. RV Modeling by System Identification

We leverage a control system engineering methodology *system identification* (SI) [48], to build a dynamics model for the target vehicle. The nominal physical behaviors of a vehicle can be described by a set of mathematical equations constituting the model. Model equations are readily available for various dynamic systems, from simple dynamics such as robotic arms [49] and water tanks [50] to complex ones such as multi-copter [51] and ground vehicles [52]. Coefficients/parameters of these equations are vehicle specific. SI techniques allow us to estimate them from measured input-output data of the vehicle's normal operations. Most system dynamics can be described by the following general equations.

$$\begin{aligned} \dot{s} &= \mathcal{F}(s) + \mathcal{G}(s)u + d, \\ y &= \mathcal{H}(s) + v \end{aligned} \tag{1}$$

Here, s is a system state vector, y a system output vector, u control inputs, d a disturbance vector, and v a measurement noise vector. $\mathcal{F}$, $\mathcal{G}$, and $\mathcal{H}$ are nonlinear nominal system function, input function, and output function, respectively. As such, the two equations determine the first derivative of states denoted as $\dot{s}$ and the output y based on the current state and the control inputs, considering the disturbances and noise. Note that the next state values are computed from the current states and the derivatives. We use a nonlinear grey-box SI [53], which leverages pre-existing formulas (called dynamics model). Grey-box SI is more accurate than black-box SI as it leverages the known dynamics model. The constructed models for a quadrotor and a ground vehicle (i.e., concretizations of Equation (1)) are available in Section S-B of our online document. Dynamics models do not model environments. As such, nontrivial environmental disturbances must cause deviations and be recorded. Model accuracy hence has effects on our log reduction efficacy but its influence is marginal compared to environmental disturbances (see Appendix E).

B. On-the-fly Logging

RvPlayer logs the following information to support replay and what-if reasoning: timestamp, control references, system states, sensor measurements, control parameters, and environmental disturbances by instrumenting the event dispatcher and the main control loop. In particular, references are inputs from the navigation component, reflecting the targets; system states include positions and attitudes; and control parameters are configurable coefficients for control algorithms. It separates the information into three sets: (i) *the environmental disturbances that are recorded in an adaptive fashion* (i.e., using a higher sampling frequency when the vehicle has nontrivial deviation from dynamics model's prediction), (ii) *state information including positions, attitudes, and sensor measurements that are recorded with a much lower but regular frequency (2HZ)*, and (iii) *discrete events that are faithfully recorded without any reduction such as parameter updates*. The second is used to correct accumulated errors that are inevitable during replay and to diagnose spoofing attacks. Note that environmental disturbances are difficult to measure with precision. A typical commodity quadrotor has only inertial, magnetic, and GPS sensors, which are not sufficient to provide comprehensive measurements of environmental conditions. To address the challenge, our adaptive logger calculates the aggregated effects of environmental disturbances and only records such effects, avoiding the need of measuring a wide spectrum of environmental conditions.

[1]Since it is infeasible to deterministically regenerate the same disturbances in the physical world, we use simulation with environmental plugins [47].

1) Computing Aggregated Effects of Environmental Disturbances: Our technique decouples the effects by external disturbances from those by motor thrusts so that the former can be recorded and replayed for environment reconstruction. The idea is that environmental disturbances affect the vehicle through forces, which are reflected in accelerations, including both linear and angular accelerations. However, acceleration values depend on not only external forces, i.e., environmental conditions, but also internal forces (e.g., thrust) that are generated by motors under the control of the vehicle system. We hence develop a method to decouple the two.

The idea is inspired by *Disturbance Observer* (DOB)-based control [54], [55] which is a widely used robust control methodology. It estimates external forces by utilizing the dynamics model and measured states of a target system, and then feeds back the estimation (of external forces) into a control loop to improve the robustness of system control. While we do not aim at robust control, we leverage the estimation method in DOB to approximate environmental disturbances.

As mentioned in Section III-A, a dynamics model denotes a vehicle's normal behavior *without disturbances*. We have the following two dynamics equations.

$$\begin{aligned} \dot{s} &= \mathcal{F}(s) + \mathcal{G}(s)u + d, \\ \dot{s_m} &= \mathcal{F}_m(s) + \mathcal{G}_m(s)u \end{aligned} \tag{2}$$

The first one is the same as that in Equation (1). Recall s denotes the real vehicle states (and $\dot{s}$ its derivative), u control input, d disturbances, and $\mathcal{F}$ and $\mathcal{G}$ system nominal function and input function, respectively. The second equation is the dynamics denoted by the model with s_m the model prediction of vehicle states. $\mathcal{F}_m$ and $\mathcal{G}_m$ are derived by system identification. Observe that the derivative of prediction (used to determine next states) is computed from the (current) real vehicle states s without disturbances. Therefore, the estimated external disturbances denoted as $\hat{d}$ can be derived as follows.

$$\hat{d} = \dot{s} - (\mathcal{F}(s) + \mathcal{G}(s)u) \approx \dot{s} - \dot{s}_m \tag{3}$$

We assume $\mathcal{F}(s) \approx \mathcal{F}_m(s)$ and $\mathcal{G}(s) \approx \mathcal{G}_m(s)$. Intuitively, the aggregated external disturbances can be approximated by the derivative differences between model and measured states.

2) Adaptive Logging: If vehicle states have non-trivial deviation from model prediction i.e., including significant disturbances or anomalies), the sampling frequency is increased, with the maximum being per control loop iteration. Specifically, in a control loop iteration at time t, let $s^i(t)$ be the ith value in a 12 dimension state vector that includes positions (x, y, and z), velocities ($\dot{x}$, $\dot{y}$, $\dot{z}$), attitudes (roll ϕ, pitch θ, yaw ψ) and angular velocities, and $\hat{s}^i(t)$ its prediction. The prediction deviation $\mathcal{E}^i(t)$ is the absolute difference of the two. Through a number of profile runs, we collect the distribution of $\mathcal{E}^i$. $\mathcal{E}^i_{max}$, the maximum deviation *in normal situations*, is defined as the third quartile (Q3) plus 1.5 times interquartile range (IQR) of the deviation distribution, which corresponds to the maximum value of box-plot in descriptive statistics [56].

The logging frequency is hence adapted as follows.

$$f(t+1) = \alpha(t) \cdot f_m \tag{4}$$

Here, $f(t+1)$ is the logging frequency at the next timestamp, and f_m the main loop frequency that is the maximum possible logging frequency. Coefficient $\alpha(t)$ is computed as follows.

$$\alpha(t) = \begin{cases} \max\limits_{i\in\{1,\ldots,n\}} \left(\dfrac{\mathcal{E}^i(t)}{\mathcal{E}^i_{max}}\right), & \text{if } \forall \mathcal{E}^i(t) \leq \mathcal{E}^i_{max} \\ 1, & \text{otherwise} \end{cases} \tag{5}$$

When the prediction deviation $\mathcal{E}^i(t)$ is smaller than $\mathcal{E}^i_{max}$ for all i's, the future logging frequency is set to the maximum ratio of the current deviation to the corresponding maximum normal deviation among the individual states (and hence smaller than 1). Intuitively, it means that the current state value is likely normal (and can be accurately predicted by the model). Hence, it may not need to be recorded. When any $\mathcal{E}^i(t)$ is larger than $\mathcal{E}_{max}$, the system logs at the highest frequency. In Appendix D, our experiment shows that results are not sensitive to $\mathcal{E}_{max}$.

C. Replayer

Our replayer takes the recorded information and replays the mission in simulation. The replay can be customized by different *policies* for what-if reasoning, by selectively enabling/disabling various recorded information during replay. Here, we introduce the *vanilla replay* that aims to *faithfully replay the discrete events (e.g., parameter updates) and environmental disturbances.* Customized replay policies will be discussed in the next section. During faithful replay, recorded events and disturbances are retrieved from the log files based on their timestamps and applied in order. To correct accumulated intrinsic errors, recorded states such as positions are regularly synchronized with the simulated vehicle.

Algorithm 1 Replay

```
    Input: $t$ iteration, $D$ online log
           $s(t)$ states, $a(t)$accelerations
           $K$ synchronization interval counter
    Output: $s(t+1)$ next states
 1: procedure REPLAY
 2:     $E, t_e = getNextEvent(D)$                       ▷ discrete event
 3:     if $t = t_e$ then
 4:         $executeEvent(E)$
 5:     end if
 6:     $d = getMostRecentDisturbance(D, t)$
 7:     $a(t) \leftarrow a(t) + d$                          ▷ apply disturbances
 8:     $s(t+1) \leftarrow simulation(s(t), a(t))$          ▷ update next states
 9:     if $(t \bmod K) = 0$ then                          ▷ Error correction
10:         $s(t) \leftarrow getPosition(t, D)$
11:     end if
12: end procedure
```

Algorithm 1 presents the replay procedure for linear states. Attitude states replay is similar and elided. The replay function is invoked at each iteration of the main control loop. In lines 3-5, it checks the timestamp of next discrete event. If it matches the current time, the event is executed. In lines 6-7, it uses the current time to retrieve the most recent recorded disturbance and applies it to the virtual vehicle. Note that since real disturbances are continuous while the recorded ones are discrete, we use the most recent recorded values to achieve a kind of interpolation/smoothing. In lines 9-11, a counter is used to facilitate regular position error correction.

IV. Root Cause Analysis Based on Replay with What-if Reasoning

A wide range of attacks have been developed against robotic RVs. Most of them can be classified into the following categories: (i) cyber-only attacks that exploit traditional zero-day vulnerabilities to crash or gain control of the system [2], [57], [58]; (ii) cyber-space state corruption (e.g., parameter tampering) or latent defects that can cause physical accidents when certain environmental conditions are satisfied [12], [27], [32]; (iii) spoofing attacks that compromise sensor readings such as GPS and Microelectromechanical systems (MEMS) sensors spoofing [4]–[6]; and (iv) physical attacks that enforce extreme environmental conditions (e.g., strong wind and obstacles). There have been a large body of existing forensics works for category (i) attacks. Hence, we focus on how to use different replay policies to diagnose the other three kinds. We assume these attacks are not adapted to evade our technique, that is, we discuss their original forms. Discussion of adaptive attack can be found in Section VI.

Definition of Accident and Reproduction of Accident. We denote an *accident* as corrupted states at a specific time (e.g. crash locations or large position/attitude offsets). We assume the *genuine* (i.e., not spoofed) accident states are available. The assumption is reasonable in the context of forensics. Because even in the presence of spoofing, when an accident occurs, causing termination of operation (e.g., crash or landing/stopping at a substantially deviated location), the true location of accident can be precisely measured, without being affected by the compromised states of victim RV. We say *an accident is reproduced during forensics if the replayed run reproduces the genuine accident states.* Accidents in which the vehicle is no longer accessible are beyond our scope (and most forensics techniques). □

Accidents caused by *spoofing* and *non-spoofing* attacks have different natures, entailing different replay policies. In the presence of spoofing, the RV internal states (and hence the logs) are substantially corrupted. Direct reasoning based on such corrupted logs likely yields meaningless results. The above assumption (of genuine states upon accident) is hence critical to spoofing attack forensics as it provides a clean reference point to identify clean and usable information from the corrupted log, if any.

Replay with What-if Reasoning. What-if reasoning is a common reasoning method that analyzes causality by changing causes and observing outcomes [23], [24]. Intuitively, for two distinct events A and B, we can reason as follows. If B happens after A and B does not happen in the absence of A, then A is a cause of B. More formally, A causes B because "if A then B" and its counterfactual "if not A then not B" are true. Our root cause analysis technique utilizes multiple replays with what-if analysis to identify what causes the accident. During replay, what-if reasoning enables us to analyze the causality of the attack by selectively enabling/disabling the recorded cyber events, sources of sensory data, program codes, and external physical disturbances.

Diagnosis Procedure. Given the assumptions and definitions, the overall procedure is as follows.

(1) *Non-spoofing Attacks.* If an accident can be faithfully reproduced by the vanilla replay, the recorded data is clean and the accident is not caused by spoofing, but rather by other attacks such as parameter tampering and defective code. To diagnose non-spoofing attacks, RvPlayer abstracts parameter updates, external disturbances and code changes as configuration variables for replay. A genetic algorithm is used to find the minimal configuration changes that can preclude the accident. The root cause is hence derived from the minimal set.

(2) *Spoofing Attacks.* If the accident cannot be faithfully reproduced by the vanilla replay, some sensor(s) must have been spoofed. The genetic algorithm is no longer applicable for spoofing attacks because spoofing signals cannot be considered as instantaneous events as they usually last for a time duration in order to have the desired effects. Modern RV systems often have multiple sensors that measure various kinds of information. There is often overlap between sensors, meaning two sensors measuring a same physical property in different manners (e.g., both GPS and barometer provide altitude information). In addition, sensors may measure inter-dependent properties (e.g., linear accelerations and linear positions). RvPlayer leverages such overlap and dependencies to infer spoofed sensors and also spoofing start time. It first replays based on sensor readings that can be directly compared with the genuine accident information. After validating these sensor readings, it further uses them as reference in follow-up replay runs to validate other sensors that overlap or have dependence with the validated ones. The process repeats until the spoofed sensors are found. Sometimes determining if a case is due to spoofing may appear subtle. Such an example is discussed at Appendix A.

A. Diagnosing Non-spoofing Attacks with What-if Reasoning

In this paper, we focus on two kinds of non-spoofing attacks: parameter tampering [12], [25] and defective safety-check conditions [27]. Safety checks determine if an RV is in some critical condition, e.g., crashing and out-of-control, such that counter-measures should be activated such as landing, turning off motors, and shooting a parachute. They are an integral part of modern RV control software. These checks are usually comparisons with some constant thresholds. Defective safety checks are due to problematic thresholds. Under certain environmental conditions (e.g., crafted by the attacker), the RV may fail to detect a real crash (i.e., false negative) or raise a false alarm (i.e., false positive).

The what-if reasoning in diagnosing these non-spoofing attack is through a large number replays driven by a genetic algorithm. Intuitively, we use the genetic algorithm to systematically disable discrete events during replay and change thresholds in safety checks to see if such changes can avoid the accident. The smallest set(s) of changes identified by the genetic algorithm denotes the root cause.

We define a replay run as a function as follows.

$$\mathcal{R}(x_1, x_2, ..., x_m,\ y_1, y_2, ...y_n) \text{ in range } \{0, 1\},$$

with x_j a boolean configuration variable denoting if a recorded event e_i is enabled during replay (e.g., a parameter update event), and y_i a real configuration variable in a range $[lb_i, ub_i]$ denoting the threshold used in a safety check. The replay function yields 1 when it reproduces the accident, 0 otherwise.

As such, we have $\mathcal{R}(x_1{=}1,\ x_2{=}1,\ ...,\ x_m{=}1,\ y_1{=}\theta_1,\ ...,\ y_n{=}\theta_n){=}1$, with θ_1, ..., θ_n the current threshold values of the n safety checks, because the configuration $\langle x_1{=}1,\ x_2{=}1,\ ...,\ x_m{=}1,\ y_1{=}\theta_1,\ ...,\ y_n{=}\theta_n\rangle$ denotes the vanilla replay.

Given a configuration from the entire space, denoted as $c \in C$, a mutant $c' \in C$ can be acquired by changing the values of a set of configuration variables in their ranges. For example, a mutant $\langle x_1{=}0,\ x_2{=}1,\ ...,\ x_m{=}1,\ y_1{=}\theta_1,\ ...,\ y_n{=}\theta_n \rangle$ means that we disable the first discrete event during replay. With such abstraction, the diagnosis problem is reduced to finding the minimum mutation to the initial configuration that can suppress the accident and in the mean time does not affect normal operations (i.e., the operation trace before the accident can be reproduced). Note that the latter is needed otherwise a simple mutation that shuts down the RV will suffice.

RvPlayer uses a genetic algorithm to find the minimum mutation. Starting from an initial population, i.e., a set of configurations c_1, c_2, ..., and c_p. An *offspring* configuration can be derived as a mutant of some existing configuration, or by a *cross-over* operation on a pair of existing configurations, i.e., exchanging the values of a subset of corresponding configuration variables [59]. These offsprings form the new generation. Each configuration denotes a replay run. RvPlayer selects the *healthy* offsprings by a *fitness* function computed from dynamic information collected during replay. The process repeats and the fitness of new generations continues to improve.

Our *fitness* function requires that (1) the replay function yields 0 with normal mission completion. (i.e., accident suppressed); (2) the set of disabled discrete events is small; and (3) the set of changed safety check thresholds is small and their value changes are small.

Given an initial configuration $c_0 \in C$ such that $R(c_0) = 1$, finding the root cause (i.e., minimum mutation) can be formulated as a multi-objective optimization (MOP) problem [60].

$$\begin{aligned} \textbf{minimize } & \mathcal{F}(c) = \{f_1(c), f_2(c), ..., f_m(c)\} \\ \textbf{subject to } & \mathcal{G}(c) = \{g_1(c), g_2(c), ..., g_n(c))\}, \quad c \in C. \end{aligned} \tag{6}$$

where C is the configuration space, $\mathcal{F}: C \to R^m$ consists of m objectives denoted as $f_1, ..., f_m$ and R^m is the objective space. $\mathcal{G}$ is defined by n constraints denoted as $g_1, ..., g_n$. For our fitness functions, $g_1(c) = \{\mathcal{R}(c) = 0\}$ denotes the condition (1), $f_1(c)$ is the condition (2), and f_2 and f_3 denote the condition (3), where each function f_i is defined as $\Delta_i(c_0, c)$, where $\Delta_i(\cdot)$ denotes a distance function using the L_p-norm distance of an offspring from the initial configuration. We use L_1 for f_1 and f_2, and L_2 for f_3. To select healthy offsprings for the genetic algorithm, we leverage the non-dominated sorting algorithm NSGA-II [61] based on our multi-objective fitness function. An example of how RvPlayer identifies the root cause for a defective safety-check attack can be found in Section S-C of our online document. Genetic algorithm is widely used in searching for discrete event root causes that may be correlated, that is, multiple working together to cause failure [62]–[64]. A popular alternative is gradient descent, which requires the domain to be continuous, which is not our case as we reason about the presence/absence of an event.

B. Diagnosing Spoofing Attacks with What-if Reasoning

When the vanilla replay cannot reproduce an accident, it is likely due to spoofing. The goal is hence to identify the sensor(s) that are spoofed and the time that the spoofing started. Many spoofing attacks such as gradual spoofing [4], [6] only introduce small errors. Locating the starting time entails distinguishing the spoofing errors and the intrinsic errors, which is challenging.

RvPlayer leverages a *state computation graph* (SCG) constructed for each RV category that describes how sensor readings are used and fused to derive system states. The states are fed to the control software to compute control signals in each iteration of the main control loop. SCG also denotes sensor dependences and redundancy. Starting from a set of states/sensors that are already verified (to have genuine values), RvPlayer identifies a set of *verifiable sensors* whose genuineness can be determined by using only the verified ones in replay. Initially, the set only contains the genuine accident states. After RvPlayer identifies a spoofed sensor, it further analyzes the second order derivative of the spoofed signal during replay to determine the start time of spoofing.

State Computation Graph (SCG). In an SCG, nodes denote states (in oval) or sensor readings (in yellow and blue boxes) and edges denote dataflow, that is, a source state/sensor is used in computing the value for a target state. Operator $\otimes$ denotes a fusion operation, meaning that the output signal is a weighted sum of the multiple input signals. States are classified to *plain states* such as linear and angular positions, *first-order states* such as velocities, and *second-order states* such as accelerations.

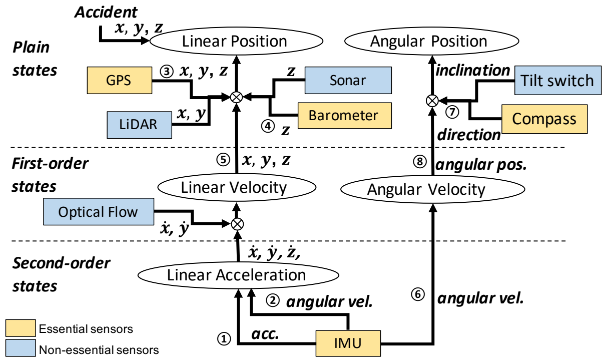


Fig. 7: State computation graph for drones

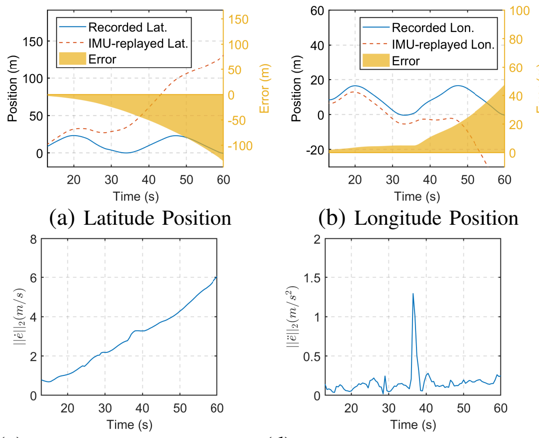
Figure 7 shows an example SCG for drones. Observe that the acceleration and angular velocity signals of the IMU sensor(s) at the bottom are used to compute linear accelerations (edges ① and ②). Angular velocities are needed in *frame transfer* from the IMU's *sensor frame* to the *world frame* [65]. Intuitively, IMU readings are represented in a coordinate system regarding the IMU sensor (i.e., how the sensor sees the RV's movements) whereas readings in the world frame describes how the world sees the RV's movements. Linear acceleration can be further used to derive linear velocity. The computed velocity signal is fused with GPS readings and barometer readings to derive the final linear position states (edges ③-⑤). Additional sensors (in blue boxes) can provide more redundancies of sensors readings. Similarly, the computed angular velocities are fused with compass readings and tilt readings to derive angular position. □

Identifying Spoofed Sensor Reading(s) by Gradual Validation Based on SCG. Starting from an initial state/sensor-reading verified by directly comparing with the genuine accident states, RvPlayer gradually verifies more and more states/sensor-readings, until the spoofed ones are identified. A typical process is to first verify the low order sensors (e.g., GPS) and then the higher order ones. There are two reasons. First, the genuine accident states are usually low order (e.g., linear positions). Second, using high order sensor readings (e.g., IMU) in replay may lead to non-trivial drifting errors over time [66] due to the intrinsic uncertainty of these readings. As such, verifying low order sensor readings (e.g., GPS) via the replayed results by higher order (verified) sensor readings (e.g., IMU) may be error-prone. Specifically, given a set of states/sensor-readings that have been verified, a sensor-reading (beyond the set) becomes *verifiable* if a dataflow path from the sensor meets with another dataflow path from verified sensor(s)/state(s) at a fusion operator. For example, we use $\{x@GPS,\ y@GPS\}$ to denote that the initial verified set contains the x and y position readings of GPS, as they can be easily verified by comparing to the genuine crash sites. Given this set, IMU becomes verifiable as the dataflow path consisting of ①, ②, and ⑤ meets with that from GPS (i.e., ③) at a fusion operator. In other words, the two data paths denote redundancy and hence can be used for verification. After IMU is verified, barometer and compass become verifiable.

To determine if a verifiable reading is spoofed, the replay policy is the following. RvPlayer disables the data path from sensors that have been verified at the fusion operator and replays using only the readings from a verifiable reading. The replayed states are then cross-checked with states when replaying exclusively with verified sensors to confirm the validity of the verifiable reading. In the drone case, to verify IMU, RvPlayer replays exclusively using IMU (without using GPS) and then compares the replayed run with that using only GPS. The process is automated given the SCG, whose construction is one-time effort. In Section F, we present a case study how RvPlayer verifies sensors gradually and identifies the source of spoofing.

There are cases that none of the sensors is verifiable, meaning that there is not enough reference to determine which of these sensor(s) is spoofed. RvPlayer will conservatively claim all these sensors are spoofed. This may occur when multiple sensors are spoofed. For example, if both GPS and IMU are spoofed, there is no way RvPlayer can determine if barometer and compass are spoofed. Note that this is not a technical limitation of RvPlayer, but rather due to the intractability of the problem.

Further Pinpointing Spoofing Start-time by Second Derivative Analysis. After identifying spoofed sensor readings, RvPlayer further identifies the spoofing start-time. This is critical to understanding attack causation. However, a direct comparison of replayed results of sensors (such as GPS and IMU) to identify spoofing start-time is practically difficult due to the intrinsic drifting errors. We observe that the second derivative analysis of replay errors can disclose spoofing start-time. In the following, we formally explain the method using a case in which GPS is spoofed and IMU is genuine. Note that here we are not using IMU to determine if GPS is spoofed. Instead, we assume that we have determined that GPS is spoofed (using the validation process mentioned ealier) and here we are using the (genuine) IMU to determine the GPS spoofing start-time. Specifically, when RvPlayer replays using only IMU, the position $\hat{p}(t)$ is integrated from the linear acceleration reading $\hat{a}(t)$ of IMU that is composed of the true value $a(t)$ and a small intrinsic error $b(t)$ as shown in Equation (7). In Equation (8), $p(t)$ denotes the true position integrated from true acceleration. The second term denotes the drifting error.



(a) Latitude Position (b) Longitude Position

(c) First Derivative of Error (d) Second Derivative of Error

Fig. 8: Second derivative analysis of GPS spoofing start-time.

$$\hat{p}(t) = \iint_0^t \hat{a}(t)\ dt = \iint_0^t a(t)\ dt + \iint_0^t b(t)\ dt \tag{7}$$

$$\hat{p}(t) = p(t) + \iint_0^t b(t)\ dt \tag{8}$$

$$p_m(t) = p(t) + d(t) \tag{9}$$

Since GPS is spoofed, the measured position $p_m(t)$ is composed of the true position and the injected error $d(t)$ as shown in Equation (9). Hence, the replay error is:

$$e(t) = p_m(t) - \hat{p}(t) = d(t) - \iint_0^t b(t)\ dt \tag{10}$$

Let the start-time of spoofing be t_0 and the injected offset increase gradually at a rate c after t_0 (a typical setup of gradual spoofing [6]). Then $e(t)$ can be rewritten as Equation (11).

$$e(t) = \begin{cases} -\iint_0^t b(t)\ dt & ,t < t_0 \\ c \cdot (t - t_0) - \iint_0^t b(t)\ dt & ,t \geq t_0 \end{cases} \tag{11}$$

The first and second derivatives of replay error, $\dot{e}(t)$ and $\ddot{e}(t)$, are then shown in Equation (12) and Equation (13). The former has a linear order of growth and the latter is the IMU intrinsic error which tends to be small before and after t_0.

$$\dot{e}(t) = \begin{cases} -\int_0^t b(t)\ dt & t < t_0 \\ c - \int_0^t b(t)\ dt & t > t_0 \end{cases} \tag{12}$$

$$\ddot{e}(t) = \begin{cases} -b(t) & t < t_0 \\ -b(t) & t > t_0 \end{cases} \tag{13}$$

Note that they are not continuous at the attack time t_0 because $c > 0$. In practice, the recorded positions are sampled discretely at a frequency f_s. Let t_a be the last sample before t_0 and t_b the first sample after t_0 ($\Delta t = t_b - t_a = 1/f_s$), then $\ddot{e}(t_0)$ can be approximated by Equation (14). Combined with Equation (13), $\ddot{e}(t)$ can be written as Equation (15). Observe that the value at t_0 is much larger than the rest, which serves as a strong indicator of start-time of spoofing.

$$\ddot{e}(t_0) = \frac{\dot{e}(t_b) - \dot{e}(t_a)}{t_b - t_a} = \frac{c - \int_{t_a}^{t_b} b(t)\, dt}{\Delta t} \doteq cf_s - b(t_0) \quad (14)$$

$$\ddot{e}(t) \doteq \begin{cases} -b(t) & , t \neq t_0 \\ cf_s - b(t_0) & , t = t_0 \end{cases} \quad (15)$$

Intuitively, the second derivative analysis undoes the accumulation of drifting error, and inverses it to the small error term $b(t)$ such that the spoofing effect clearly stands out. The method also works when IMU is spoofed and GPS is genuine. The formal analysis is elided.

Figure 8 shows the second derivative analysis for a gradual GPS longitude spoofing started at the 36.3s. As shown in (a) and (b), the recorded latitude and longitude positions (from the GPS and the replayed states by IMU) have increasing errors, due to drifting and spoofing. By directly observing the positions and errors, it is very difficult to identify the spoofing and the start-time. Figures (c) and (d) show the the first and second derivatives of the longitude error. The first derivative is an ascending curve, whereas the second derivative has a surge that clearly suggests the spoofing start-time.

V. Evaluation

A. Evaluation Setup

Implementation. RvPlayer includes: (1) a non-linear modeling method with System Identification in MATLAB; (2) an online adaptive logging module for each target control program; (3) a replayer; (4) a root-cause analyzer that systematically performs what-if reasoning to identify root cause. The logger records runtime values through built-in logging APIs in the individual systems. Modern vehicles commonly utilize sensor fusion, which combines information from multiple sensors to generate a single output by voting or weighted averaging, e.g., using *Extended Kalman Filter* (EKF) [67]. RvPlayer records raw sensor measurements instead of states after fusion since corrupted sensor information could be filtered out by fusion. RVs also often use filters (e.g., Butterworth low-pass filter [68]) to suppress noises. As such, spoofing signals need to have certain strength to get through filtering and fusion.

Subject Systems. We evaluate our technique on 3 control programs with 5 different types of real/simulated vehicles. Additionally, we use real-world traces for 2 autonomous cars from [4]. The vehicles perform diverse physical maneuvers, and the control programs have various software stacks, supporting that our technique provides a generic solution.

TABLE I: Subject Systems

Controller	Version	SLOC	Vehicle Type	Log Freq	Vehicle	# State	Sensors
Ardupilot	4.0.1	267,593	Quadrotor	400	SimQuad	12	G I B C R
OpenSolo	4.0.0	413,012	Quadrotor	400	3DR Solo	12	G I B C
Ardupilot	4.0.1	267,593	Quadrotor	400	IRIS+	12	G I B C
APMRover2	4.0.0	257,444	4-Wheel Rover	50	SimRover	6	G I C
APMRover2	3.2	203,266	4-Wheel Rover	50	Erle-Rover	6	G I C

G: GPS, I: IMU, B: Barometer, C: Compass, R: Rangefinder

Table I shows the drones and rovers used in our evaluation. The simulated vehicles (i.e., SimQuad and SimRover) run on

TABLE II: Real-world AD Datasets

Source	Road	Real Traces (#)	Attack Instances (#)	Freq (Hz)	Duration (s)	Sensor Data
Baidu Apollo	Urban	1	256	100	257	GPS LiDAR IMU
KAIST Complex	Urban / Highway	4	890	100	553-1937	GPS LiDAR IMU

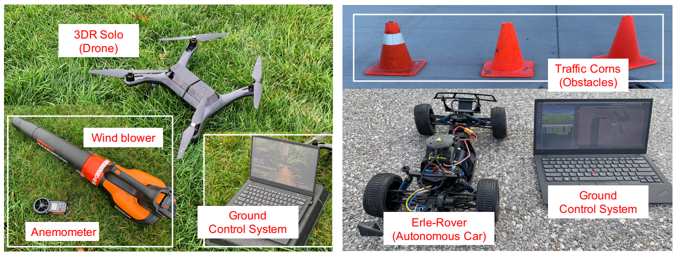


Fig. 9: Real vehicles and experimental tools. Aerial vehicle setup (left): drone, GCS, wind generator, and anemometer. Ground vehicle setup (right): traffic cones, GCS, and autonomous vehicle.

Ubuntu 18.04LTS with Intel i7-8700 and 32GB RAM. The real aerial vehicle, 3DR Solo (left in Figure 9), contains an open-source autopilot system, Pixhawk 2.1, running on an ARM Cortex-M4 STM32F427 CPU with 256KB RAM, a 32GB SD card and various onboard sensors. IRIS+ contains a Pixhawk 1 with a 8GB SD card and various sensors. The ground vehicle, Erle-rover (right in Figure 9), runs on Erle-brain, which is a Linux-based autopilot system running on an ARM Coretex-A7 with 1GB RAM, 16GB flash memory, and sensors.

For auto-driving cars, we follow a similar setup in [4] to use real-world operation trace datasets, due to the lack of real auto-driving cars. Table II shows the two datasets used in our evaluation. We use one trace from the Apollo [69] dataset that includes sensor traces of urban driving and 4 traces from the KAIST Complex Urban [70] dataset that provides complex driving environments.

Missions and Disturbances. We evaluate our technique with various autonomous missions (e.g., flight plans) and different types/levels of environmental disturbances. The autonomous missions consist of straight-line mission, square mission, zig-zag mission and star-shape mission, covering diverse maneuvers. The straight-line mission is to move towards north for 30 meters, then stop/land. The square mission has a square-shape trajectory with 30 meters edge length and 4 waypoints. In the zig-zag mission, the vehicles moves in a zig-zag fashion (so that they turn both left and right) with 11 waypoints. The star-shape mission has a hexagonal-star trajectory with 13 waypoints with sharp turns. During these missions, the real vehicles have random natural winds as environmental disturbances. For the simulated vehicles, no environmental disturbances are applied to the first two missions (i.e. straight and square missions) and random winds from various directions with 0-22 mph speed are applied to the zig-zag and star-shape missions. We also evaluate RvPlayer's performance under different levels of environmental disturbances. We have three settings: small, medium and strong. In testing 3DR Solo, we used wind blower and anemometer (left bottom in Figure 9). The wind blower can generate multiple levels of wind at different distances from the vehicle. We use the wind blower at 2m, 1m, and 0.5m away from a hovering drone, creating 10mph (Beaufort Level 3), 20mph (Level 5) and 50mph (Level 9) wind disturbances, respectively. For the Erle-Rover, the three levels of disturbances are generated using traffic cones (top right in Figure 9) and the wind blower. Small and medium effects are generated using the wind blower 45-degree and directly ahead of the vehicle, respectively. Strong effect comes from hitting the center of a traffic cone. For the simulated vehicles, the different levels of environmental effects are generated by environmental plugins [47] in simulation. We set `SIM_WIND_SPD=4`, `SIM_WIND_SPD=9` and `SIM_WIND_SPD=20` to generate the three levels of external disturbances.

RV Attacks. We implemented 5 kinds of popular RV attacks (with 1194 attack instances with 48 on drones and rovers, and 1146 on auto-driving cars) from the literature: parameter tampering [12], [32], short-duration sensor spoofing [3], [26], gradual sensor spoofing [6], multi-sensor fusion (MSF) attack [4] and defective safety check attack [27]. They are on the square route mission (M1) and the zig-zag route (M2). Details about RV Attacks, the implementation of RvPlayer, datasets used, and demos are available at https://sites.google.com/view/rvplayer.

B. *Efficiency and Effectiveness of Logging and Replay*

We performed several experiments to evaluate the logging and replay functionalities of RvPlayer, focusing on efficiency and effectiveness. For efficiency, we measure space and runtime overheads, and show the log reduction by our adaptive logging method. For effectiveness, we show that RvPlayer reproduces original traces of simulated and real flights under different scenarios (i.e., different missions and disturbances) with small errors. We further show that RvPlayer is effective to find root cause of accidents of real-world flights in an urban area where diverse errors are introduced (e.g., GPS urban canyon, wind, etc.). We also show that the reproduction results are stable, by studying the tradeoffs between replay errors and log reduction rates.

Performance Overhead. RvPlayer enlarges the firmware size by 0.10% to 0.28% on average. Without adaptive logging, Ardupilot-based drones have the space consumption rate of 4.04GB/day and APMRover-based rovers have the rate of 0.35GB/day. After adaptive logging, RvPlayer reduces more than 90% of the space consumption for normal operations. In contrast, MAYDAY's space consumption is 40.5 times of RvPlayer's. The runtime overhead of RvPlayer is lower than 8%. In contrast, MAYDAY's overhead is 19.6%. Additionally, we measure the CPU utilization rate for the real vehicles. The rate increases from 55.19% to 57.07% and 20.03% to 20.46% for 3DR Solo and Erle-rover respectively. Such marginal overhead does not affect normal operations. Details can be found in Appendix B.

Replay Accuracy. We show how accurately RvPlayer reproduces the original executions of simulated and real flights in various conditions. We log states at the highest frequency possible (i.e., every control loop iteration) and use them as the ground truth. We then compare the replayed executions with the ground truth. We also evaluate a baseline in which we replay without using the recovered disturbances. For all the RVs and missions, the mean position error of RvPlayer is less than 0.73 meter and the mean attitude error of reproduction is less than 6.4 degrees. In comparison, the baseline's mean position and attitude errors are 364.8% and 290.5% larger, and 361.3% and 453.6% larger in high variance durations. Details and a case study can be found in Appendix C.

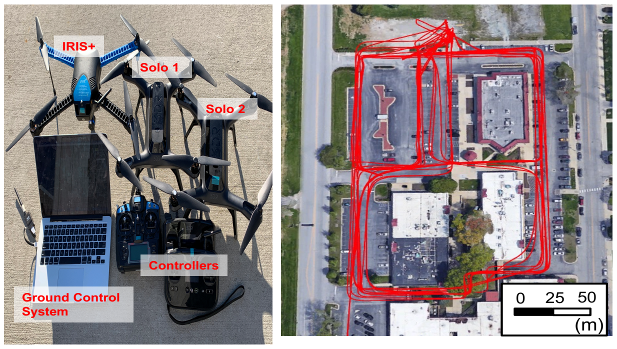


Fig. 10: Real drones and real flight trajectories in an urban area

Problem Diagnosis in Real-world Missions. In reality, there are lots of (small) errors that may cause accidents (e.g, substantial deviations). Hence in this experiment, we run physical experiments with real drones in an urban area where diverse errors (e.g. GPS signal loss, wind, etc) can be encountered. We use missions that are (relatively) long lasting with some reaching the maximum battery capacity and have multiple waypoints (being repeated many times). Besides natural errors (e.g., wind and intrinsic GPS errors), we artificially generate GPS urban canyon effects via an injected code snippet in the control loop, which intentionally degrades GPS signals[2]. We activate these effects randomly with various duration in the missions. We consider deviation larger than one meter as an accident and then study if RvPlayer can determine the real root causes.

We use three drones of two models: 3DR Solo1, Solo2 (without GPS glitch protection), and IRIS+ as shown in Figure 10 (left). The missions are shown in Figure 10 (right). The flight distances are all longer than 3km. Due to safety concerns, when an accident occurs (i.e., a large deviation), we let the vehicles land via our fail-safe function[3]. We perform the experiments over several days, trying to maximize the range of weather conditions we cover. We have covered wind speed from 0-12mph, temperature from 27-61°F and humidity from 35-78%.

Table III summarizes the results. The first column shows the error conditions, the second column the subject vehicles, the third and fourth columns flight distance and time, the fifth column the maximum position deviation from the target trajectory, the sixth column the maximum position error during replay, the seventh column reporting if an accident occurs, the eighth column the identified root cause, the ninth column the error of identified root cause start-time, and the tenth column the SCG path that RvPlayer uses in replay. The winds are not strong enough to induce accidents, whereas the artificial

[2]Natural urban canyon effects are unlikely in the geographical locations of the authors.

[3]All the real drones have been registered with the FAA and controlled by a certified operator with LAANC (Low Altitude Authorization and Notification Capability) permission.

TABLE III: RvPlayer with real-world flights

Conditions	Vehicle	FD (km)	FT (min)	D (m)	RErr (m)	A (Y/N)	R	TErr (sec)	SCG Path
Wind (0-4mph)	Solo1	3.26	10.8	0.09	0.2	No	-	-	-
	Solo2	3.23	10.5	0.08	0.2	No	-	-	-
	IRIS+	3.14	13.1	0.11	0.32	No	-	-	-
Wind (4-8mph)	Solo1	3.58	11.8	0.10	0.21	No	-	-	-
	Solo2	3.60	11.9	0.10	0.2	No	-	-	-
	IRIS+	3.14	13.2	0.13	0.31	No	-	-	-
Wind (8-12mph)	Solo1	3.61	12.0	0.12	0.37	No	-	-	-
	Solo2	3.60	12.0	0.11	0.39	No	-	-	-
	IRIS+	3.66	15.2	0.23	0.43	No	-	-	-
GPS Glitch (1-3s)	Solo1	3.25	10.8	0.38	0.49	No	-	-	-
	Solo2	3.36	11.2	2.93	0.41	**Yes**	GPS	0.0	G
	IRIS+	3.21	13.4	0.44	0.35	No	-	-	-
GPS Glitch (3-5s)	Solo1	3.38	10.8	0.51	0.38	No	-	-	-
	Solo2	1.32	4.4	4.52	0.42	**Yes**	GPS	0.0	G
	IRIS+	3.22	13.4	0.56	0.45	No	GPS	0.0	G
GPS Glitch (5s>)	Solo1	1.52	5.0	2.21	0.39	**Yes**	GPS	0.0	G
	Solo2	1.48	4.9	8.26	0.42	**Yes**	GPS	0.0	G
	IRIS+	2.18	9.08	3.34	0.39	**Yes**	GPS	0.0	G

FL: flight distance, FT: flight time, D: deviation, A: if accident is encountered, RErr: replay error, R: root cause identified, TErr: identified start-time error, N/A: not available

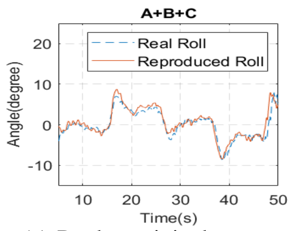
GPS urban canyon effects indeed cause a few failures. For all these accidents, RvPlayer accurately identifies the GPS signal loss as the root cause without any start-time error. Besides these normal flights, extreme wind conditions and the investigation results are available in Table V. In addition, we extensively evaluate urban canyon effects in different settings (e.g., effect duration and GPS protection enabled/disabled) with real drones. The evaluation results and demo videos are available in Section S-G of our online document.

Fidelity of What-if Analysis. In this experiment, we show that the what-if analysis on a simulator can produce high fidelity results as in physical field tests. In particular, we demonstrate that the replayed runs (on a simulator) with certain events disabled have small errors compared to the real runs (in the physical world) with the same set of events disabled.

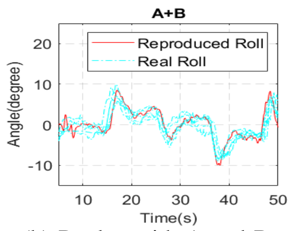
We use a mission with a square trajectory. During the flight, three paraemters are updated (A) `WP_YAW_BEHAVIOR`, (B) `WPNAV_SPEED` and (C) `FS_THR_VALUE`. We first collected a replay trace of the flight on a day with normal environmental conditions (with SE wind of level 2, light breeze). On the same day, we disabled different subsets of the update events, re-flied the same mission, and collected detailed state traces. For each configuration, we flew five times to mitigate environmental uncertainty. Later in the simulation environment, we used RvPlayer to replay the first trace multiple times, each having the corresponding subset of update events disabled. We then compared the replayed runs and the corresponding real runs. Figure 11 shows that the replay errors are small. It demonstrates that RvPlayer indeed can serve as an alternative to expensive field tests. Also observe that disabling different subsets leads to different state traces such that a faithful replay technique without what-if reasoning capabilities would not work.

C. Attack Root Cause Analysis

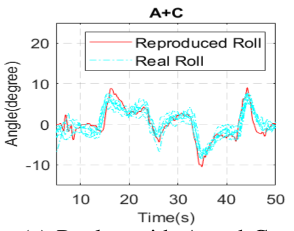
Parameter Tampering Attacks Results. Table IV summarizes the results of parameter tampering attack analysis. The second column shows the vehicles, the third column the attack consequences, the fourth column the number of parameters updated, the fifth column the root cause parameter updates, the sixth column the number of replays needed (to identify



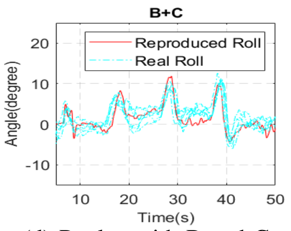
(a) Replay original trace



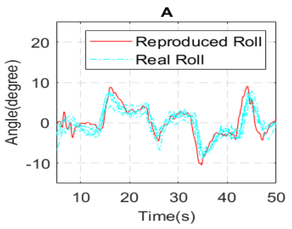
(b) Replay with A and B



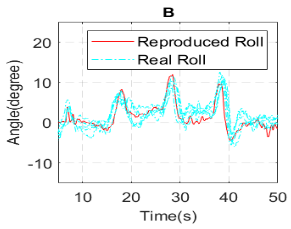
(c) Replay with A and C



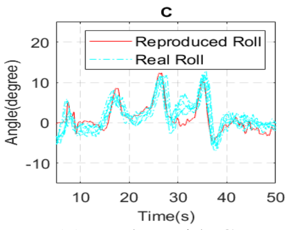
(d) Replay with B and C



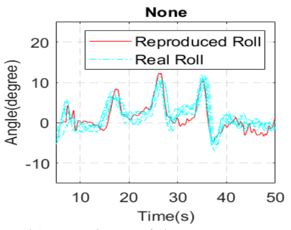
(e) Replay with A



(f) Replay with B



(g) Replay with C



(h) Replay without events

Fig. 11: Comparison of physically reruns and replayed runs under different combinations of cyber events. Figure (a) shows RvPlayer can faithfully replay the first flight (with none of the events disabled) and the error is negligible. Figures (b-h) show that even when different events are disabled, e.g., (c) showing B disabled, RvPlayer can closely reproduce the corresponding real runs.

the root causes), the seventh column the total analysis time, and the last column reporting if RvPlayer identifies the root causes. Some of the attacks require compromising more than one parameters. Observe that RvPlayer can precisely identify the root causes for all the attack cases, including those having multiple root causes, with reasonable analysis time. The large number of replays needed indicate that such what-if forensics analysis cannot be afforded in field tests. Our technique can also correctly distinguish the attacks by strong external disturbances from parameter tamperings. Advanced attackers may exploit multiple root causes to make the search process of RvPlayer more difficult. We discuss this advanced attack in Section S-F of our online document.

Short-duration and Gradual Sensor Spoofing Attacks Analysis. We mix the 23 spoofing attacks with 3 physical attacks.

TABLE IV: Parameter Tampering Attack Analysis Result

Mission &	Attack	V	Attack Consequence	PC (#)	Root Cause	RC (#)	RT (min)	ID?
M1	A1	SQ	Crash	10	`ATC_RAT_RLL_P`	133	53	YES
	A2	SL	Crash	10	`ATC_RAT_RLL_P`	112	34	YES
	A3	SQ	Oscillation	10	`PSC_VELXY_P`	148	74	YES
	A4	SQ	Crash	10	`ATC_RAT_PIT_I` `ATC_RAT_PIT_FF`	154	64	YES
	A5	SR	Oscillation	10	`ATC_STR_RAT_P`	129	97	YES
	A6	ER	Oscillation	10	`ATC_STR_RAT_P`	119	44	YES
	A7	SQ	Drift away	9	External Dist.	87	62	YES
	A8	SQ	Oscillation	9	External Dis.	74	51	YES
	A9	SR	Crash	9	External Dis.	89	68	YES
M2	A10	SQ	Crash	10	`ATC_RAT_RLL_P`	159	101	YES
	A11	SL	Crash	10	`ATC_RAT_RLL_P`	140	54	YES
	A12	SQ	Oscillation	10	`PSC_VELXY_P`	125	85	YES
	A13	SQ	Crash	10	`ATC_RAT_PIT_I` `ATC_RAT_PIT_FF`	178	116	YES
	A14	SR	Oscillation	10	`ATC_STR_RAT_P`	143	141	YES
	A15	ER	Oscillation	10	`ATC_STR_RAT_P`	124	33	YES
	A16	SQ	Drift away	9	External Dist.	102	99	YES
	A17	SQ	Oscillation	9	External Dis.	96	94	YES
	A18	SR	Crash	9	External Dis.	83	64	YES

V: Vehicle, SQ: SimQuad, SR: SimRover, SL: 3DR Solo, ER: Erle-Rover
PC: Parameter Count, RC: Replay Count, RT: Total Replay Time, ID: Identified

TABLE V: Sensor Spoofing Attack Analysis Result

Attack	Attack Impact	V	AST (s)	ADT (s)	RC (#)	RT (s)	IDR	STE	SCG Path
GPS Gradual	Crash	SQ SR SL ER	19.23 20.78 16.12 11.14	44 43 53 10	2	128 153 138 43	GPS	+0.77 +0.72 +0.88 +0.86	G
Barometer Gradual	Crash	SQ SL	25.43 12.53	6 7	3	93 58	Baro	+0.57 +0.97	GB
Barometer Short-time	Crash	SQ SL	16.96 9.89	1 1	3	61 128	Baro	+0.54 +0.61	GB
Rangefinder Gradual	Crash	SQ	21.24	6	4	108	Range-finder	+0.76	GBR
Rangefinder Short-time	Crash	SQ	22.12	1	4	106	Range-finder	+0.88	GBR
Rangefinder & Barometer	Crash	SQ	23.78	1	5	243	Baro Range	+0.67 +0.67	GBR
Gyroscope Short-time	Crash	SQ SR SL	18.92 42.82 12.03	0.5 0.5 0.5	5	212 242 106	IMU	+1.58 +0.68 +0.97	GBRI
Gyroscope Gradual	Crash	SQ SR	32.52 36.11	8 8	5	204 310	IMU	+1.98 +0.89	GBRI
Accel. Short-time	Crash	SL	10.47	1	5	68	IMU	+0.99	GBRI
Accel. Gradual	Crash	SQ SR SL	36.88 31.46 4.99	7 7 6	5	228 287 57	IMU	+1.12 +1.04 +1.01	GBRI
Gyro.&Acc. Gradual	Crash	SQ SR	36.88 36.32	7 12	5	228 242	IMU	+1.12 +1.18	GBRI
Compass Gradual	Crash	SQ SR	8.7 18.92	32 67	6	535 669	Compass	+2.3 +1.08	GBRIC
Strong Dist.	Crash	SQ SR	30.52 30.72	16 15	2	93 92	Dist.	- -	-
Strong Dist.	Drift Away	SQ	13.39	18	2	118	Dist.	-	-
Strong Dist.	Strong Oscillation	SQ	16.53	21	2	82	Dist.	-	-

V: Vehicle, SQ: SimQuad, SR: SimRover, SL: 3DR Solo, ER: Erle-Rover,
AST: Attack Start Time, ADT: Attack Duration Time, RC: Replay Count,
RT: Total Replay Time, IDR: Identified Root Cause, STE: Identified Start-Time Error

Table V summarizes the results. The first column shows the spoofed sensor and the spoofing method, the second column the attack consequence, the third targeted vehicle(s), the fourth column the start-time of attack, the fifth column the attack duration, the sixth column the number of replays, the seventh column total replay time, the eighth column identified root-cause, the ninth column the error of identified start-time, the tenth column the SCG path (Section IV-B) that RvPlayer uses in replays. Observe that the root causes are correctly detected for all these attacks and the start times are identified with mostly around 1 second errors. Observe that although the SCG paths are different, they are the prefixes of the same validation path. The last three attacks show that when the accidents are not caused by spoofing, RvPlayer correctly and exhaustively validates all sensors along the SCG path and does not report any spoofing, that is, it produces no false positives.

MSF Attacks Analysis. Table VI summaries the results for the MSF attacks. In the table, we show the datasets and the trace ids (columns 1 and 2). The traces are obtained from different types of roads (column 3) with different lengths (column 4), and the required deviations for a successful attack (column 5) are correspondingly different based on the width of roads. Based on the optimal attack parameters in the original paper (column 6), a number of attack are performed with different start-times (column 7). Column 8 reports the number of successful attacks (i.e., achieving the required deviation) with the average attack duration in column 9. Column 10 shows that the average difference between the spoofed GPS position and the accident position. According to the SCG for AD cars, RvPlayer leverages data redundancy in linear positions from LiDAR and GPS. It first compares $\{x@GPS,\ y@GPS\}$ and $\{x@LiDAR,\ y@LiDAR\}$ to the accident position. This allows to decide which sensor is spoofed. Figure 12 shows the average position differences of GPS and LiDAR from the genuine accident positions (normalized to zeros) upon accidents. Observe that the LiDAR positions are very close to the ground truth. In contrast, the GPS position differences are around 5 to 11 meters, exceeding normal GPS errors in modern AD systems. The last column shows the average error of identified start-time of spoofing. RvPlayer identifies a sudden increase of the second derivative of replay errors using GPS and LiDAR. Observe that the means error is less than 0.9 seconds for all the attacks.

Defective Safety-check Attack Results. Table VII summarizes the results for attacks exploiting defective safety-checks. The first column shows the safety-check functions from various control programs, the second column the defect type, the third column the triggering conditions, the fourth column the attack consequences, the fifth column the number of predicates involved, the sixth the number of replays, the seventh total replay time, the eighth the number of identified root causes. Observe that RvPlayer successfully identifies all the root causes. Case studies can be found in Appendix F.

D. Comparison with Other RV Forensics Techniques

We compare RvPlayer with two other techniques: Software Sensor (SS) [17], and MAYDAY [25]. SS builds a model for a physical sensor (called *software sensor*) that can convert other sensors' readings to approximate the sensor's reading. While SS was proposed for robust control, software sensors' reading can be compared with their physical counter-parties to identify attacks and start-times. That is, any substantial deviation of a sensor's readings from its software counterparts indicates the sensor is spoofed. We use the 38 RV attacks from the previous experiment (Table IV, V, VII), randomly selected 30 MSF attacks for AD, and 10 accidents by natural disturbances. Table VIII shows the results. Each row represents a kind of attack and the number of instances. The columns

Source	Trace	Road Type	Data Len. (s)	Required Dev. (m)	Attack Par. (d/f)	Attacks (#)	Accidents (#)	Attack Dur. (s)	Pos. Diff. (m)	Time Err. (s)
Apollo	ba-local	Urban	257	2.405	0.6 / 1.5	256	126	1.63	5.13	0.7608
KAIST Complex Urban	ka-local31	Urban	1014	2.405	0.5 / 1.2	217	208	1.92	4.37	0.0034
	ka-local07	Urban	553	2.405	0.3 / 1.1	231	228	2.07	6.61	0.0017
	ka-highway17	Highway	1186	2.855	0.3 / 1.2	162	162	10.64	6.88	0.0063
	ka-highway06	Highway	1937	2.855	1.1 / 1.5	280	277	9.30	11.12	0.9027

TABLE VI: MSF Attack (through GPS Spoofing) [4] for Autonomous Driving

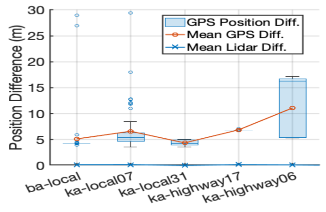


Fig. 12: Position difference

TABLE VII: Root Cause Analysis for Defective Safety-check attack

Software / Check Function	Defect (FN)	Triggering Physical Condition	Accidents ($\mathcal{R}(\cdot)$=1)	Mutated Conditions (#)	Replays (#)	Time (m)	Root Cause	Replay Constraints ($\mathcal{R}(\cdot)$=0)
ArduCopter / Crash checker	Missing crash	Wall	Out of Control	7 Predicates	144	73	2 Predicates	Checking successful detection and disarm
APMRover2 / Crash checker	Missing crash	Person (Dummy)	Injury	8 Predicates	152	113	2 Predicates	Checking successful detection and stop
PX4 / Ground contact	Detection fail	Wind gust	Fly away	4 Predicates	48	24	1 Predicate	Checking an alert message

TABLE VIII: Comparison with Other RV Forensics Techniques

Attack	#	SS				MAYDAY				RvPlayer			
		TP	FP	FN	STE	TP[†]	FP	FN	STE	TP	FP	FN	STE
P	12	✗	✗	✗	✗	6	6	0	0.57	12	0	0	0
S	23	10	0	13	1.13	✗	✗	✗	✗	23	0	0	1.02
C	3	✗	✗	✗	✗	✗	✗	✗	✗	3	0	0	0
A	30	22	0	8	4.37	✗	✗	✗	✗	30	0	0	0.73
D	10	✗	✗	✗	✗	✗	✗	✗	✗	0	0	0	✗
Total	78	32	0	21	-	6	6	0	-	78	0	0	-

- P: parameter attack, S: sensor spoofing, C: safety check attack, A: AD MSF attack, D: disturbance
- ✗: Not available
[†] MAYDAY does not pinpoint real root cause parameter(s), rather it reports code region associated with the parameters (we count this as TP).

contain the results for different techniques, including the number of false positives/negatives and the average start-time error. For SS and MAYDAY, we use the moment they detect anomaly as the attack start time. Symbol ✗ means that the technique does not support analyzing an attack. Observe that SS and MAYDAY each can analyze only one or two kinds of attack, whereas RvPlayer supports all. Even for the attack categories SS and MAYDAY focus, RvPlayer outperforms by having no false positives or negatives, and precise attack start time. As discussed in Section II, MAYDAY relies on thresholds to find root causes and in many attacks, multiple code components easily exceed the thresholds. SS compares physical sensor readings with approximated sensor readings to report root causes. In many gradual spoofing attacks, the difference between the two readings is marginal, leading to false negatives. Five case studies of analyzing four different kinds of attacks can be found in Appendix F.

VI. Discussion

Model Accuracy. Our model is not perfect as it is constructed by regression analysis (i.e., SI) from traces. A "less perfect" model means that we may log a bit more information than necessary because RvPlayer simply records the states that cannot be precisely predicted. We evaluate the impact of model accuracy on the performance of RvPlayer in Appendix E.

TABLE IX: Sensors and Data Redundancy

Data Type	GPS	Gyro	Accel	Mag	Baro	Sonar	Cam*	LiDAR	RADAR
Positional	✓		✓		✓	✓	✓	✓	✓
Rotational		✓		✓		✓	✓	✓	✓
Distance[†]					✓	✓	✓	✓	✓

✓: Included. [†] Relative position to objects or the ground. *With post-processing via AI models

Multiple Sensor Attacks and Software Hijacking. We rely on SCG to identify spoofed sensor reading(s). When multiple sensors are attacked, RvPlayer may still be effective if there is at least a clean path in the SCG, In practice, it may be difficult to attack all paths in SCG simultaneously. In addition, modern RVs heavily leverage sensor fusion, in which multiple sensors of the same or different kinds provide overlapping/redundant information. Table IX shows data redundancy in the current practice of sensor fusion. RvPlayer belongs to forensic analysis, which is opportunistic. If all evidences are indeed destroyed, it is unlikely that any forensic technique, including RvPlayer, can work. If the control software is hijacked, the log cannot be trusted. RvPlayer will be ineffective.

Adaptive Attacks. If the attacker is aware of RvPlayer, he could devise adaptive attacks that aim to evade our technique. The first possible adaptive attack aims at the adaptive sampling strategy. Specifically, the attack can be made stealthy such that the errors are not large enough to trigger high-frequency logging. RvPlayer is resilient to the attack to some extent. In particular, it regularly records vehicle states (with a low frequency) such that the replayed states are regularly synchronized with these recorded states. According to our definition of accident and reproduction of accident in Section IV, we assume the genuine states are known upon accidents. Therefore, during forensics, RvPlayer can reproduce the (spoofed) vehicle states through regular state synchronization. They are different from the genuine states, allowing to reproduce the attack. However since the errors are very small, RvPlayer may not precisely pinpoint the attack start time without detailed logs. To mitigate this limitation, we could leverage on-the-fly time-series compression algorithms [71], [72] to compress logs, or offload the collected traces to an edge router. Our goal is attack forensics. If the objective is to secure RV systems after all, existing on-the-fly defense techniques [12], [28], [73] can be leveraged to prevent such attacks in the first place.

Another adaptive attack is to induce drastic errors such that RvPlayer logs at the max rate and fills up the storage space, flushing out important events associated with the root cause. However, to fill up the 32GB storage space (excluding other data like videos), the attack has to last for 16 hours. The vehicle hence likely crashes (with such large errors) long before the space runs out. In addition, a simple online checker can be employed to detect such long-lasting mis-prediction.

VII. Related Work

Autonomous CPS Attack Countermeasures. In recent years, significant efforts have been invested on CPS attack countermeasures. Runtime attack detection approaches detect physical anomalies at runtime via control invariants [12], [13], [28], [74], redundancy-based technique [75], [76], and rule-based detection [77]–[80]. Runtime attack recovery techniques mitigate attack impact via software sensors [17], fault identification and isolation [73], [81] and checkpoint-based recovery [16]. Meanwhile, there have been other efforts on control program vulnerability identification. SI-based testing technique [82] provides a test oracle for bug localization. CPI [27] presents a technique to identify cyber-physical inconsistency vulnerabilities in safety checks of control software. These techniques provide a preventive or protective countermeasure. However, state-the-art CPS attacks (e.g., stealthy and APT-style attack [4], [6], [12], [26]) may evade them. In contrast, our technique aims to provide a postmortem analysis to find the root causes of sophisticated attacks. They are hence complementary.

Attack Provenance Analysis. Existing log-based causality analysis [42], [83]–[88] utilize event logs (e.g., syscalls). They correlate events through dependences to trace back attack root causes or trace forward to identify attack damages. Tainting-based analysis [89]–[93] utilizes provenance propagation. It starts from given taint sources and monitors information flow to capture program dependencies during execution. Both approaches induce significant runtime and space overhead. Also, they are originally designed for cyber-attack investigation and not suitable to handle physical attack vectors in resource-constrained CPS. Record and replay based provenance analysis [39], [41]–[45], [84], [94] are also a widely used approach to attack investigation. They record program inputs and replay original executions to analyze the root cause of attacks. These approaches are effective in deterministic software-oriented (i.e., cyber) execution environments. Our approach is inspired by traditional replay-based provenance analysis. It however focuses on reproducing physical system behaviors. Our novelty lies in that we isolate and record non-deterministic environmental conditions and exploit them in what-if analysis.

CPS Forensic Analysis. CPS forensics is less investigated. Some research efforts have been placed on various types of CPS such as industrial control system [19], [95], smart grid [20], smart home [21], and autonomous CPS [22]). However, these existing methods are mostly dedicated to cyber components and networks. For autonomous CPS, MAYDAY [25] discussed in Section II presented a post-investigation technique by connecting program analysis and physical anomaly detection. It supports source code level debugging and is highly effective in locating defective controller and mission command bugs. It does not handle convoluted anomaly and requires heavy-weight program instrumentation and trace collection. Our technique supports what-if reasoning to investigate various kinds of attacks such as parameter-tampering attacks, sensor spoofing attacks and defective codes attacks with low overhead. This makes it complementary to MAYDAY.

VIII. Conclusion

We present RvPlayer, a novel forensic analysis technique for RVs. RvPlayer captures environmental disturbances at runtime and replays them in simulation to reconstruct actual attack traces offline. Our adaptive logging technique is efficient in meeting resource-constrained CPS requirements by adjusting a logging frequency based on a target-specific predictive model and recording only significant runtime disturbances. Our what-if analysis technique supports various replay policies for different attacks to identify root causes, by selectively enabling/disabling recorded information. Our evaluation and case studies with 5 types of state-of-the-art RV attacks (1194 instances) on various simulated and real vehicles demonstrate our technique's effectiveness and efficiency. It also shows that RvPlayer outperforms the state-of-the-art.

Acknowledgements

We thank the anonymous reviewers for their constructive comments. This research was supported, in part by NSF 1901242 and 1910300, ONR N000141712045, N000141410468 and N000141712947, IARPA TrojAI W911NF-19-S-0012, and DARPA VSPELLS (under subcontract 1119482 with BAE Systems). Any opinions, findings, and conclusions in this paper are those of the authors only and do not necessarily reflect the views of our sponsors.

References

[1] C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. Jesus, R. Berriel, T. M. Paixão, F. Mutz *et al.*, "Self-driving cars: A survey," *Expert Systems with Applications*.

[2] B. Nassi, R. Bitton, R. Masuoka, A. Shabtai, and Y. Elovici, "Sok: Security and privacy in the age of commercial drones," in *SP*, 2021.

[3] Y. Tu, Z. Lin, I. Lee, and X. Hei, "Injected and delivered: Fabricating implicit control over actuation systems by spoofing inertial sensors," in *USENIX Security*, 2018.

[4] J. Shen, J. Y. Won, Z. Chen, and Q. A. Chen, "Drift with devil: Security of multi-sensor fusion based localization in high-level autonomous driving under GPS spoofing," in *USENIX Security*, 2020, pp. 931–948.

[5] T. Trippel, O. Weisse, W. Xu, P. Honeyman, and K. Fu, "Walnut: Waging doubt on the integrity of mems accelerometers with acoustic injection attacks," in *EuroS&P*, 2017.

[6] K. C. Zeng, S. Liu, Y. Shu, D. Wang, H. Li, Y. Dou, G. Wang, and Y. Yang, "All your GPS are belong to us: Towards stealthy manipulation of road navigation systems," in *USENIX Security 18*, 2018.

[7] M. Luo, A. C. Myers, and G. E. Suh, "Stealthy tracking of autonomous vehicles with cache side channels," in *USENIX Security*, 2020.

[8] M. A. Al Faruque, S. R. Chhetri, A. Canedo, and J. Wan, "Acoustic side-channel attacks on additive manufacturing systems," in *ICCPS*, 2016.

[9] K. Krombholz, H. Hobel, M. Huber, and E. Weippl, "Advanced social engineering attacks," *Journal of Information Security and applications*, vol. 22, pp. 113–122, 2015.

[10] C. Tankard, "Advanced persistent threats and how to monitor and deter them," *Network security*, vol. 2011, no. 8, pp. 16–19, 2011.

[11] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Security & Privacy*, vol. 9, no. 3, pp. 49–51, 2011.

[12] H. Choi, W.-C. Lee, Y. Aafer, F. Fei, Z. Tu, X. Zhang, D. Xu, and X. Deng, "Detecting attacks against robotic vehicles: A control invariant approach," in *CCS*, 2018, pp. 801–816.

[13] Y. Chen, C. M. Poskitt, and J. Sun, "Learning from mutants: Using code mutation to learn and monitor invariants of a cyber-physical system," in *SP*. IEEE, 2018, pp. 648–660.

[14] A. Abbaspour, K. K. Yen, S. Noei, and A. Sargolzaei, "Detection of fault data injection attack on uav using adaptive neural network," *Procedia computer science*, vol. 95, pp. 193–200, 2016.

[15] K. N. Junejo and J. Goh, "Behaviour-based attack detection and classification in cyber physical systems using machine learning," in *CPS-Sec*, 2016.

[16] F. Kong, M. Xu, J. Weimer, O. Sokolsky, and I. Lee, "Cyber-physical system checkpointing and recovery," in *ICCPS*. IEEE, 2018, pp. 22–31.

[17] H. Choi, S. Kate, Y. Aafer, X. Zhang, and D. Xu, "Software-based realtime recovery from sensor attacks on robotic vehicles," in *RAID*, 2020, pp. 349–364.

[18] M.-K. Yoon, B. Liu, N. Hovakimyan, and L. Sha, "Virtualdrone: virtual sensing, actuation, and communication for attack-resilient unmanned aerial systems," in *ICCPS*, 2017, pp. 143–154.

[19] R. Chan and K.-P. Chow, "Forensic analysis of a siemens programmable logic controller," in *International Conference on Critical Infrastructure Protection*. Springer, 2016, pp. 117–130.

[20] E. Sohl, C. Fielding, T. Hanlon, J. Rrushi, H. Farhangi, C. Howey, K. Carmichael, and J. Dabell, "A field study of digital forensics of intrusions in the electrical power grid," in *CPS-SPC*, 2015.

[21] Q. Do, B. Martini, and K.-K. R. Choo, "Cyber-physical systems information gathering: A smart home case study," *Computer Networks*, vol. 138, pp. 1–12, 2018.

[22] M. Cebe, E. Erdin, K. Akkaya, H. Aksu, and S. Uluagac, "Block4forensic: An integrated lightweight blockchain framework for forensics applications of connected vehicles," *IEEE Communications Magazine*, vol. 56, no. 10, pp. 50–57, 2018.

[23] D. Lewis, "Causation," *The journal of philosophy*, vol. 70, no. 17, pp. 556–567, 1974.

[24] J. Pearl, *Causality*. Cambridge university press, 2009.

[25] T. Kim, C. H. Kim, A. Ozen, F. Fei, Z. Tu, X. Zhang, X. Deng, D. J. Tian, and D. Xu, "From control model to program: Investigating robotic aerial vehicle accidents with MAYDAY," in *USENIX Security*, 2020.

[26] B. Nassi, Y. Mirsky, D. Nassi, R. Ben-Netanel, O. Drokin, and Y. Elovici, "Phantom of the adas: Securing advanced driver-assistance systems from split-second phantom attacks," in *CCS*, 2020.

[27] H. Choi, S. Kate, Y. Aafer, X. Zhang, and D. Xu, "Cyber-physical inconsistency vulnerability identification for safety checks in robotic vehicles," in *CCS*, 2020, pp. 263–278.

[28] R. Quinonez, J. Giraldo, L. Salazar, E. Bauman, A. Cardenas, and Z. Lin, "SAVIOR: Securing autonomous vehicles with robust physical invariants," in *USENIX Security*, 2020, pp. 895–912.

[29] M. Zhang, C.-Y. Chen, B.-C. Kao, Y. Qamsane, Y. Shao, Y. Lin, E. Shi, S. Mohan, K. Barton, J. Moyne *et al.*, "Towards automated safety vetting of plc code in real-world plants," in *SP*, 2019.

[30] "ArduPilot :: Home," 2010, http://ardupilot.org/.

[31] P. Dash, M. Karimibiuki, and K. Pattabiraman, "Out of control: stealthy attacks against robotic vehicles protected by control-based techniques," in *ACSAC*, 2019, pp. 660–672.

[32] T. Kim, C. H. Kim, J. Rhee, F. Fei, Z. Tu, G. Walkup, X. Zhang, X. Deng, and D. Xu, "Rvfuzzer: finding input validation bugs in robotic vehicles through control-guided testing," in *USENIX*, 2019.

[33] "Beaufort Wind Scale," 2017, https://www.weather.gov/mfl/beaufort.

[34] P. Guo, H. Kim, N. Virani, J. Xu, M. Zhu, and P. Liu, "Roboads: Anomaly detection against sensor and actuator misbehaviors in mobile robots," in *2018 48th DSN*. IEEE, 2018, pp. 574–585.

[35] "Onboard Message Log Messages — Copter documentation," 2010, https://ardupilot.org/copter/docs/logmessages.html.

[36] "ULog File Format · PX4 Developer Guide," 2010, https://dev.px4.io/master/en/log/ulog_file_format.html.

[37] "RVPlayer Online Document," https://sites.google.com/view/rvplayer.

[38] M. Sheldon and G. V. B. Weissman, "Retrace: Collecting execution trace with virtual machine deterministic replay," in *MoBS 2007*, 2007.

[39] G. Walkup, S. Etigowni, D. Xu, V. Urias, and H. W. Lin, "Forensic investigation of industrial control systems using deterministic replay," in *CNS*. IEEE, 2020, pp. 1–9.

[40] O. Landsiedel, E. M. Schiller, and S. Tomaselli, "Libreplay: Deterministic replay for bug hunting in sensor networks," in *European Conference on Wireless Sensor Networks*. Springer, 2015, pp. 258–265.

[41] R. Chandra, T. Kim, M. Shah, N. Narula, and N. Zeldovich, "Intrusion recovery for database-backed web applications," in *SOSP*, 2011.

[42] A. Goel, K. Po, K. Farhadi, Z. Li, and E. De Lara, "The taser intrusion recovery system," in *SOSP*, 2005, pp. 163–176.

[43] Z. Guo, X. Wang, J. Tang, X. Liu, Z. Xu, M. Wu, M. F. Kaashoek, and Z. Zhang, "R2: An application-level kernel for record and replay." in *OSDI*, vol. 8, 2008, pp. 193–208.

[44] S. M. Srinivasan, S. Kandula, C. R. Andrews, Y. Zhou *et al.*, "Flashback: A lightweight extension for rollback and deterministic replay for software debugging," in *USENIX ATC*, 2004, pp. 29–44.

[45] S. Narayanasamy, G. Pokam, and B. Calder, "Bugnet: Recording application-level execution for deterministic replay debugging," *IEEE Micro*, vol. 26, no. 1, pp. 100–109, 2006.

[46] Y. Kwon, D. Kim, W. N. Sumner, K. Kim, B. Saltaformaggio, X. Zhang, and D. Xu, "Ldx: Causality inference by lightweight dual execution," in *ASPLOS*, 2016, pp. 503–515.

[47] "Environmental disturbance simulation · Ardupilot," 2020, https://ardupilot.org/dev/docs/using-sitl-for-ardupilot-testing.html.

[48] L. Ljung, "System identification: Theory for the user," 1987.

[49] A. K. Bejczy and R. Paul, "Simplified robot arm dynamics for control," in *CDC*. IEEE, 1981, pp. 261–262.

[50] N. Petit and P. Rouchon, "Dynamics and solutions to some control problems for water-tank systems," *IEEE Transactions on Automatic Control*, vol. 47, no. 4, pp. 594–609, 2002.

[51] J. Li and Y. Li, "Dynamic analysis and pid control for a quadrotor," in *ICMA*. IEEE, 2011, pp. 573–578.

[52] J. Wang and R. G. Longoria, "Coordinated and reconfigurable vehicle dynamics control," *IEEE Transactions on Control Systems Technology*, vol. 17, no. 3, pp. 723–732, 2009.

[53] "Nonlinear Model Identification - MATLAB and Simulink," 2021, https://www.mathworks.com/help/ident/nonlinear-model-identification.html.

[54] W.-H. Chen, J. Yang, L. Guo, and S. Li, "Disturbance-observer-based control and related methods—an overview," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 2, pp. 1083–1095, 2015.

[55] S. Li, J. Yang, W.-H. Chen, and X. Chen, *Disturbance observer-based control: methods and applications*. CRC press, 2014.

[56] J. W. Tukey, *Exploratory data analysis*. Reading, MA, 1977, vol. 2.

[57] J. Garcia, Y. Feng, J. Shen, S. Almanee, Y. Xia, and Q. A. Chen, "A comprehensive study of autonomous vehicle bugs," in *ICSE*, 2020.

[58] C. L. Krishna and R. R. Murphy, "A review on cybersecurity vulnerabilities for unmanned aerial vehicles," in *SSRR*, 2017.

[59] K. Deb and R. B. Agrawal, "Simulated binary crossover for continuous search space," *Complex systems*, vol. 9, no. 2, pp. 115–148, 1995.

[60] K. Deb, *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons, 2001, vol. 16.

[61] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[62] M. Harman and P. McMinn, "A theoretical and empirical study of search-based testing: Local, global, and hybrid search," *IEEE Transactions on Software Engineering*, vol. 36, no. 2, pp. 226–247, 2009.

[63] F. Gross, G. Fraser, and A. Zeller, "Search-based system testing: high coverage, no false alarms," in *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, 2012, pp. 67–77.

[64] K. Praditwong, M. Harman, and X. Yao, "Software module clustering as a multi-objective search problem," *IEEE Transactions on Software Engineering*, vol. 37, no. 2, pp. 264–282, 2010.

[65] T. Luukkonen, "Modelling and control of quadcopter," *Independent research project in applied mathematics, Espoo*, vol. 22, 2011.

[66] N. El-Sheimy and A. Youssef, "Inertial sensors technologies for navigation applications: State of the art and future trends," *Satellite Navigation*, vol. 1, no. 1, pp. 1–21, 2020.

[67] R. E. Kalman *et al.*, "Contributions to the theory of optimal control," *Bol. soc. mat. mexicana*, vol. 5, no. 2, 1960.

[68] S. Butterworth *et al.*, "On the theory of filter amplifiers," *Wireless Engineer*, vol. 7, no. 6, pp. 536–541, 1930.

[69] "Baidu Apollo," 2021, https://github.com/ApolloAuto/apollo.

[70] J. Jeong, Y. Cho, Y.-S. Shin, H. Roh, and A. Kim, "Complex urban dataset with multi-level sensors from highly diverse urban environments," *Int J Rob Res*, vol. 38, no. 6, pp. 642–657, 2019.

[71] V. Sundaram, P. Eugster, and X. Zhang, "Prius: Generic hybrid trace compression for wireless sensor networks," in *ACM Sensys*, 2012.

[72] D. Blalock, S. Madden, and J. Guttag, "Sprintz: Time series compression for the internet of things," *ACM IMWUT*, 2018.

[73] M. Saied, B. Lussier, I. Fantoni, H. Shraim, and C. Francis, "Fault diagnosis and fault-tolerant control of an octorotor uav using motors speeds measurements," *IFAC-PapersOnLine*, vol. 50, no. 1, 2017.

[74] C. Feng, V. R. Palleti, A. Mathur, and D. Chana, "A systematic framework to generate invariants for anomaly detection in industrial control systems." in *NDSS*, 2019.

[75] M.-K. Yoon, B. Liu, N. Hovakimyan, and L. Sha, "Virtualdrone: virtual sensing, actuation, and communication for attack-resilient unmanned aerial systems," in *ICCPS*. ACM, 2017, pp. 143–154.

[76] F. Fei, Z. Tu, R. Yu, T. Kim, X. Zhang, D. Xu, and X. Deng, "Cross-layer retrofitting of uavs against cyber-physical attacks," in *ICRA*, 2018.

[77] R. Mitchell and R. Chen, "Behavior rule specification based intrusion detection for safety critical medical cyber physical systems," *IEEE Trans Dependable Secure Comput*, vol. 12, no. 1, pp. 16–30, 2015.

[78] ——, "Adaptive intrusion detection of malicious unmanned air vehicles using behavior rule specifications," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 44, no. 5, pp. 593–604, 2014.

[79] C. Zimmer, B. Bhat, F. Mueller, and S. Mohan, "Time-based intrusion detection in cyber-physical systems," in *ICCPS*, 2010, pp. 109–118.

[80] S. Bak, K. Manamcheri, S. Mitra, and M. Caccamo, "Sandboxing controllers for cyber-physical systems," in *ICCPS*, 2011, pp. 3–12.

[81] V. Stepanyan, K. S. Krishnakumar, and A. Bencomo, "Identification and reconfigurable control of impaired multi-rotor drones," in *AIAA Guidance, Navigation, and Control Conference*, 2016, p. 1384.

[82] Z. He, Y. Chen, E. Huang, Q. Wang, Y. Pei, and H. Yuan, "A system identification based oracle for control-cps software fault localization," in *ICSE*. IEEE, 2019, pp. 116–127.

[83] S. Krishnan, K. Z. Snow, and F. Monrose, "Trail of bytes: efficient support for forensic analysis," in *CCS*, 2010, pp. 50–60.

[84] T. Kim, X. Wang, N. Zeldovich, M. F. Kaashoek *et al.*, "Intrusion recovery using selective re-execution." in *OSDI*, 2010, pp. 89–104.

[85] S. T. King and P. M. Chen, "Backtracking intrusions," in *SOSP*, 2003.

[86] K. H. Lee, X. Zhang, and D. Xu, "High accuracy attack provenance via binary-based execution partition." in *NDSS*, 2013.

[87] Y. Kwon, F. Wang, W. Wang, K. H. Lee, W.-C. Lee, S. Ma, X. Zhang, D. Xu, S. Jha, G. F. Ciocarlie *et al.*, "Mci: Modeling-based causality inference in audit logging for attack investigation." in *NDSS*, 2018.

[88] S. Ma, J. Zhai, F. Wang, K. H. Lee, X. Zhang, and D. Xu, "Mpi: Multiple perspective attack investigation with semantic aware execution partitioning," in *USENIX Security*, 2017, pp. 1111–1128.

[89] S. Ma, X. Zhang, and D. Xu, "Protracer: Towards practical provenance tracing by alternating between logging and tainting." in *NDSS*, 2016.

[90] E. Bosman, A. Slowinska, and H. Bos, "Minemu: The world's fastest taint tracker," in *RAID*. Springer, 2011, pp. 1–20.

[91] J. Clause, W. Li, and A. Orso, "Dytan: a generic dynamic taint analysis framework," in *ISSTA*, 2007, pp. 196–206.

[92] M. G. Kang, S. McCamant, P. Poosankam, and D. Song, "Dta++: dynamic taint analysis with targeted control-flow propagation." in *NDSS*, 2011.

[93] F. Qin, C. Wang, Z. Li, H.-s. Kim, Y. Zhou, and Y. Wu, "Lift: A low-overhead practical information flow tracking system for detecting security attacks," in *MICRO'06*. IEEE, 2006, pp. 135–148.

[94] D. Geels, G. Altekar, P. Maniatis, T. Roscoe, and I. Stoica, "Friday: Global comprehension for distributed replay." in *NSDI*, 2007.

[95] I. Ahmed, S. Obermeier, M. Naedele, and G. G. Richard III, "Scada systems: Challenges for forensic investigators," *Computer*, vol. 45, no. 12, pp. 44–51, 2012.

[96] U. Zölzer, *Digital audio signal processing*. Wiley, 2008, vol. 9.

Appendix

A. *A Subtle Spoofing Case*

The attacker jams the camera sensor to disable the obstacle detection system, making it impossible to recognize an object in front, then the drone will collide with the obstacle. According to the definition of an accident in Section IV, we assume genuine accident states are collected, which may not be the states observed by the vehicle. In this case, the genuine states include the presence of an obstacle, while the observed states do not include the obstacle. According to the definition of reproduction of an accident, an accident is considered reproduced when all genuine states are reproduced. In this case, they are not as the obstacle is invisible. Hence, the accident is considered not reproduced and a spoofing case. Intuitively, the vehicle's failure to see the obstacle upon crash indicates spoofing. Note that our SCG-based method can handle such cases as other sensors like LiDAR allow us to localize the compromised sensor.

B. *Performance Overhead*

Please see Section S-D of our online document.

C. *Replay Accuracy*

Please see Section S-E of our online document.

D. *Effect of Parameter $\mathcal{E}_{max}$*

Please see Section S-H in our online document.

E. *Impact of Model Accuracy*

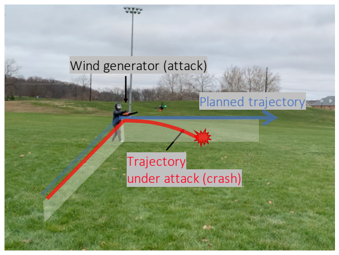
Please see Section S-I of our online document.

F. *Case Studies*

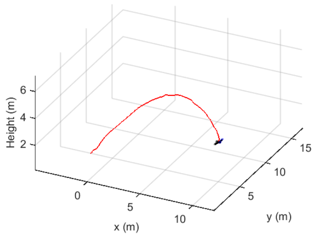
In this section, we demonstrate various case studies to show effectiveness of RvPlayer in attack investigation.

Parameter Tampering Attack. In the first case, we launch an APT-like parameter tampering attack [12], [32], which requires a multi-staged and long-term procedure. The drone first performs five normal missions (e.g., straight flies) and 8 commands are issued to modify various parameters (available in our online document) during the flights. Among these parameters, `ATC_RAT_RLL_P` affects control behaviors significantly when the vehicle performs an extreme maneuver under harsh environmental conditions. After the five normal missions, the drone performs another mission which includes a 90-degree turn. During this flight, we generate a strong wind (i.e., 50 mph/h) via our wind generator to trigger malicious behaviors causing a crash. Note that the same speed of wind gust does not cause a crash during the same turn under the original parameter value. Figure 13a shows the snapshot from the actual accident under the attack.

Attack Investigation. To diagnose the attack, RvPlayer first reproduces the accident. It checks that the replayed run faithfully reproduces the genuine accident states in simulation to verify the attack is caused by a non-spoofing attack. Figure 13b

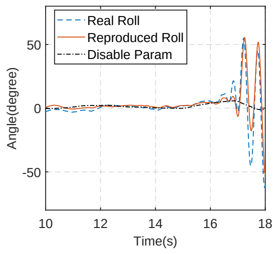


(a) Actual attack

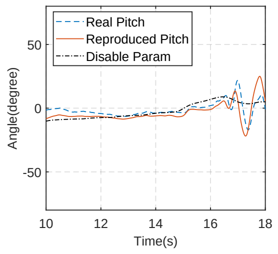


(b) Reproduced attack

Fig. 13: Snapshots from videos of the actual and reproduced parameter attacks (Videos: real attack [https://youtu.be/jD9D9lCdEjQ] and 3D simulation [https://youtu.be/T6b4-KD7LK0]



(a) Roll Angles



(b) Pitch Angles

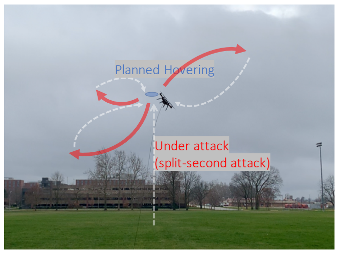
Fig. 14: Comparison of state changes during the real and reproduced parameter attacks

shows the replay trajectory. Figure 14 shows the comparison of attitude traces of the actual (in blue) and reproduced runs with all the updates enabled (in red) and disabled (in black). It is hence clear that the accident is caused by some parameter update(s). RVPLAYER then performs what-if reasoning to find the root cause update. Specifically, we collect the parameter update events $\langle e_1, e_2, ..., e_8\rangle$ from the runtime log and abstract them to boolean configuration variables $\langle x_1, x_2, ..., x_8\rangle$ for replay. Given the initial configuration $c_0 = \langle x_1{=}1,\ x_2{=}1,\ ...,\ x_8{=}1\rangle$ such that $\mathcal{R}(c_0) = 1$, RVPLAYER runs the genetic algorithm with our fitness function (defined in Section IV-A). After 4 evolutionary iterations (population size N=20), RVPLAYER reports $\langle x_1{=}1,\ ...,\ x_7{=}1, x_8{=}0\rangle$ (i.e., `ATC_RAT_RLL_P`) as the root cause.

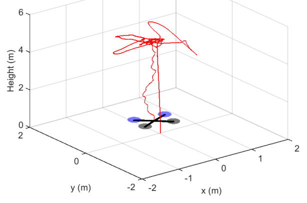
Short-duration IMU Sensor Attack. This is a *split-second attack* [26], [96]. It injects malicious signals between two logging events of a target sensor to trigger abnormal behaviors. As such, the malicious signal is not recorded. In the attack, we directly modified internal sensor measurements without losing generality, due to the lack of special attack devices (e.g., a distant vibration generator). Our attack code modifies the gyro's x-axis roll rate from the original value to 0.3. The attack frequency is 25Hz with 0.5 second duration. It is calibrated in a way that it avoids the regular state logging at the same frequency of 25Hz. The attack is launched at 12.03 seconds after taking off. Figure 15a illustrates the attack consequences.

Attack Investigation. Since we are uncertain if this is a spoofing attack, we first use RVPLAYER to perform a vanilla replay. Figure 15 shows snapshots from the videos of the actual and the reproduced attacks. In Figure 16a, we denote the recorded roll angle with the blue solid curve and the reproduced one with the red dashed curve. Observe that the reproduced roll angle in the vanilla replay is inconsistent with the recorded angle (e.g., the opposite peaks). We hence decide this is a sensor spoofing attack. Then, RVPLAYER starts the procedure of sensor spoofing analysis as the inconsistency could be induced by attack on any sensor. In the spoofing analysis, RVPLAYER first validates $\{x@GPS,\ y@GPS,\ z@GPS\}$ from GPS by comparing the accident position with the recorded GPS position. They match well, and hence GPS is verified. Given the SCG for drones with the verified set $\{z@GPS\}$, the barometer becomes verifiable, and RVPLAYER compares the replayed barometric altitude with GPS altitude. As shown in Figure 16b, they also match well, and the barometer is genuine. Next, RVPLAYER validates the IMU through replaying with the IMU data. IMU is verifiable regarding GPS and barometer, i.e., the set $\{x@IMU,\ y@IMU,\ z@IMU\}$ is verifiable. Figure 16c shows the already verified longitude position (i.e., $y@GPS$) denoted by the blue solid curve, the IMU-replayed position (i.e., $y@IMU$) denoted by the red dashed curve and their difference (i.e., replay error) denoted by the yellow area. Observe that the error starts growing at 14s. Through the second derivative analysis in Figure 16d, we observe a peak at 13s, which is identified as the start-time of IMU spoofing. Recall that the attack was launched at the 12th second. The small time error is often due to sampling interval.

Additional case studies (a gradual GPS spoofing attack and two defective code attacks) can be found online [37].

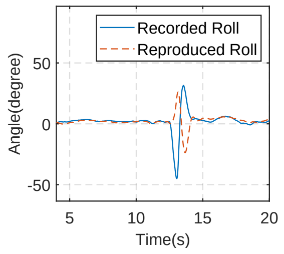


(a) Actual attack

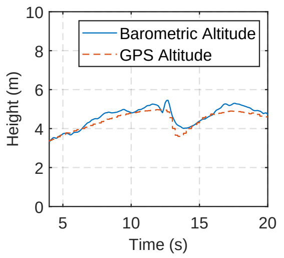


(b) Reproduced attack

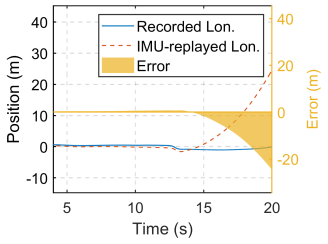
Fig. 15: Snapshots from videos of the actual and reproduced split-second attacks. (Videos: real attack [https://youtu.be/4Bj-Fnd5tVg] and 3D simulation [https://youtu.be/LKAfSEIx0-o])



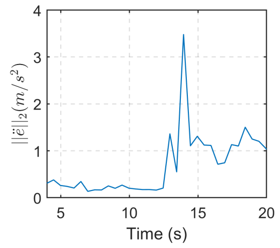
(a) Vanilla Replay



(b) Validate Barometer



(c) Validate IMU



(d) Second Derivative of Error

Fig. 16: Spoofing analysis of gyroscope attack.