Runtime Interchange of Enforcers for Adaptive Attacks: A Security Analysis Framework for Drones

Alex Baird
Dept. E.C.S.E.
The University of Auckland
Auckland, New Zealand
alex.baird@auckland.ac.nz

Hammond Pearce
Dept. E.C.E.
New York University
New York, USA
hammond.pearce@nyu.edu

Srinivas Pinisetty
School of Electrical Sciences
IIT Bhubaneswar
Bhubaneswar, India
spinisetty@iitbbs.ac.in

Partha Roop
Dept. E.C.S.E.
The University of Auckland
Auckland, New Zealand
p.roop@auckland.ac.nz

***Abstract*—Unmanned aerial drones are Cyber-Physical Systems (CPSs) with increasing availability, popularity, and capability. Although other aeronautical and safety-critical industries apply stringent regulations and design approaches, smaller drones tend to have much weaker and informal design requirements. Due to the strong open-source movement in this space, there are numerous opportunities for malicious actors to find weaknesses to attack drone systems, and in parallel develop their own rogue drones. These factors present a risk of damage to people and property in addition to compromise of integrity and availability. However, a formal framework for ethical hacking that combines attacker modelling and launching of attacks is lacking in the literature. To this end, we leverage runtime enforcement, combined with the idea of suspension from synchronous programming to develop the first such formal framework.**

The proposed framework enables the modelling of complex attack vectors on drones. To facilitate this, we propose a bespoke policy-based runtime enforcement framework called *enforcer interchange (EI)*. It is capable of both individual intent/target-specific attacks as well as more sophisticated combinations of attacks, which it manages by enabling and disabling *attack enforcers* at runtime in a context-aware manner. To demonstrate our framework, we utilise a quadcopter drone simulator and record the changes in the drone's behaviour as it executes a range of missions under different attacks. Our approach provides a framework for testing drones' resilience and defenses against malicious attacks, as well as exploring the capabilities of rogue drones.

I. Introduction

Unmanned aerial drones are Cyber-Physical Systems (CPSs) [1] as they intertwine software controllers, hardware actuators, and communication mechanisms for controlling the different tasks related to flight. There is a rapidly expanding global market for drones ranging from QR code displays in Shanghai [2] to passenger transport and delivery drones [3]. Another example sees Alphabet's Wing project already delivering to customers in the United States and Australia [4].

There are a range of communication architectures used in controlling drones, from hand held radio to satellite-based options, which introduces potential security vulnerabilities. For instance, a security analysis of the DJI Phantom 3 by Trujano et al. in [5] shows that drones, like many other technologies, have weaknesses. Likewise, Yaacoub et al. in [6] provide a deep review of the security of drones. One common factor is unencrypted communication between base station and drone [7], [8]. This work discusses the wide range of cybersecurity flaws typically found in current drone systems, highlights the inconsistent and lagging regulation for drone owners, and covers existing counter measures to these vulnerabilities. Thus, a malicious actor can cause considerable damage to life and property using a drone attack mechanism.

In established safety critical industries such as aeronautics, model-driven approaches are often required to ensure safety and security. However, this is not extended to small unmanned drones. With no expected design standard, there is an opportunity to develop a test bed of security attacks that drone owners and designers alike could use to measure the performance (i.e. safety) of the drone when under malicious attack.

In this work, we present formal models of attacks on a drone system, and a formal methodology to combine individual attacks for a range of scenarios. Our work falls into *runtime assurance* [9], where system behaviours are augmented by supervisory systems. In contrast to existing work, our approach enforces security breaches by dynamically selecting from a bank of runtime enforcers. Edit automata-like [10] runtime enforcers encompass the capabilities of typical drone attacks, providing a test-bed for developing more robust controllers.

This leads us to the main contributions of the paper. (1) We present a range of runtime enforcers which model attacks on communication in a quad copter drone system. (2) We present a formal approach, called Enforcer Interchange (EI), for combining runtime enforcers. EI extends existing techniques based on runtime enforcement of reactive systems [11], [12], which deal only with single enforcers. (3) We demonstrate individual attacks and a sophisticated combination of attacks for delivery drone example. This runs in a simulator for a popular open source drone control software, ArduPilot.

The rest of the paper is organised as follows: Section II introduces attack modelling and our drone case study. Section III provides preliminaries and notation which we apply. In Section IV we present runtime enforcers as a method for modelling attacks. In Section V we present Enforcer Interchange (EI) for complex attack modelling, with an example for our case study. In Section VI we present the impact of our attacks on the drone delivery system. Finally, Section VII presents an overview of the related work in this area, and Section VIII concludes the paper.

II. Attack Modelling for Drones

A. Attacks on Drones

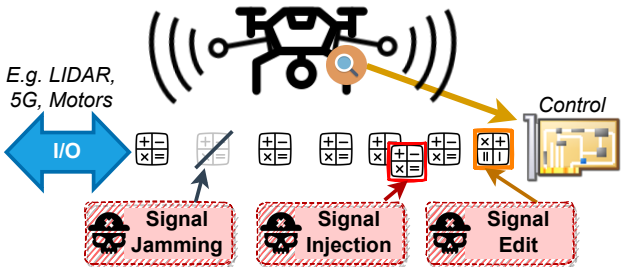


Figure 1. Drone attack model for jamming, injection, and editing capabilities

Given the wide attack space in existing drones, it is unsurprising various cyber-attacks have been demonstrated. For example, WiFi based jamming and deauthentication attacks in [13] allow denial of service, man-in-the-middle, and packet spoofing to give the attacker complete drone control. Other RF jamming approaches attempt to trigger a safe landing/return home [14] and to prevent sensing and communication [15]. Another method of attack spoofs GPS systems, allowing the attacker to control the drone's location [16], [17]. Hardware trojans in communication systems [18] are another attack vector which can inject false sensor data, alter firmware, and breach confidentiality [19].

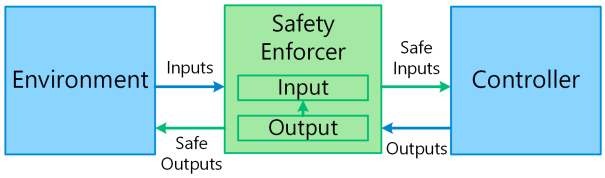
In this work we create a simplified attack model (Figure 1) which groups these and other attacks into the following categories, and ensure that our formal model can capture the semantics of each kind of attack:

- **Jamming**: a denial of service attack that prevents two way communication between base station and drone.
- **Injection**: creation of control and/or status signals.
- **Alteration**: editing control or status signals.

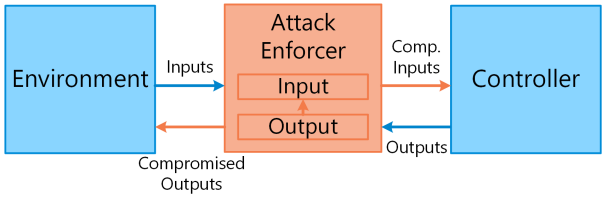
Runtime enforcement [20] offers an interesting avenue for creating an environment for ethical hacking over drones. An enforcer is used to ensure that a given set of desired policies hold over a given CPS. Coincidentally, the framework of edit automata [10] proposes three types of enforcement actions, namely suppression, where an input is *suppressed*, insertion, where a new input is *inserted* and editing where an input is *edited* by combing suppression with insertion. These roughly correspond to jamming, insertion, and alteration respectively. Thus, this provides us with a nice formal basis for developing the new ethical hacking framework. However, edit automata are not sufficient for this task as they were developed for non-real-time systems. Moreover, there is a need for developing a framework where complex attacks can be modelled and launched such that certain attack policies can be turned on and off as needed. Such requirements are not achievable using existing enforcement frameworks.

B. Attack Modelling with Runtime Enforcers

Runtime enforcement (RE) [21]–[23] is a formal mechanism which synthesises an *enforcer* to observe a black-box system



(a) Bidirectional runtime enforcement [12]



(b) Attack modelling with bidirectional runtime enforcement

Figure 2. Comparison of existing and attack runtime enforcement

and ensure a set of policies are satisfied. Before a policy is violated, the enforcer may block execution [21], insert or suppress events [23], or delay events [22]. For bidirectional reactive systems, input and output events are edited [12] to prevent violation.

As illustrated in Figure 2a, bidirectional safety enforcers perform, when necessary, some action on environment inputs and controller outputs to ensure the policy is satisfied at all times. In our work, we instead *flip* the application from ensuring safety to instead perform cyber-security attacks.

Illustrated in Figure 2b, our "attack enforcers" are placed in the conceptual "machine-in-the-middle" (MITM) configuration, where environment inputs can be altered to trick the controller, and outputs can be altered to trick the environment. We follow this as we design jamming, injection, and alteration attacks for drones in a delivery context.

C. Motivating Example: Delivery Drone

We consider a delivery drone, with the pre-programmed procedures for the following missions:

- **Flight test**: the drone takes off from it's home location, climbs to a set altitude where it hovers until the base station requests it to land.
- **Relocation**: the drone climbs to altitude, flies to set coordinates, and then descends to it's new home base.
- **Delivery**: the drone climbs to altitude, flies to a set location, descends to drop off a package, and then returns to the home location.

We implement these missions in a software-in-the-loop (SITL) simulator, shown in Figure 3, for ArduPilot, an open source drone control software platform. This software can run on a range of open and closed source drone hardware, with the simulator provided to test off the device, prior to live tests. To communicate with ArduPilot we use DroneKit, an open source abstraction layer in Python. We insert attack enforcers in this communication between the drone and base station as illustrated in Figure 4. From the attacker's perspective,

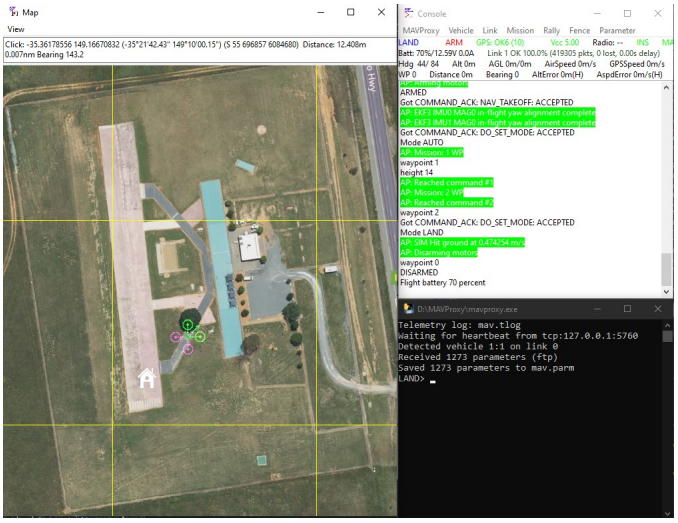


Figure 3. ArduPilot software-in-the-loop simulator for quadcopter drone

we assume a "grey box" system as the inputs and outputs are known and the missions are also known, but the internal workings of both drone and controller remain unknown.

Figure 4. Drone simulation system

III. Preliminaries and Notations

We consider Cyber-Physical Systems to have finite ordered sets of valued input channels $I = \{i_1, i_2, \ldots i_n\}$ and valued output channels $O = \{o_1, o_2, \ldots o_n\}$. For a variable (resp. channel) v, $\mathcal{D}_v$ denotes its domain, and for a finite ordered set of variables $V = \{v_1, \ldots, v_n\}$, $\mathcal{D}_V$ is the product domain $\mathcal{D}_{v_1} \times \cdots \times \mathcal{D}_{v_n}$.

Consider $n \in \mathbb{N}$, $\mathbb{B}_n$ denotes the domain of the finite ordered set of Boolean $\{b_1, \cdots, b_n\}$. A valuation of the variables in V is a mapping ν which maps every variable $v \in V$ to a value $\nu(v)$ in $\mathcal{D}_v$.

A finite (resp. infinite) word over $\mathcal{D}_C$ (where $C = I \cup O$) is a finite (resp. infinite) sequence $\sigma = \eta_1 \cdot \eta_2 \cdots \eta_n$ where $\forall i \in [1,n] : \eta_i$ is a tuple of values of variables in $C = I \cup O$. For convenience where necessary, each element η_i is considered to be a pair (η_I, η_O), where η_I is a valuation of all the variables in I, and η_O is a valuation of all the variables in O. The set of finite (resp. infinite) words over $\mathcal{D}_C$ is denoted by $\mathcal{D}_C^*$ (resp. $\mathcal{D}_C^\omega$). The *length* of a finite word σ is n, denoted $|\sigma|$. The empty word over $\mathcal{D}_C$ is denoted by ε_C, or ε when clear from the context. $\mathcal{D}_C^+$ denotes $\mathcal{D}_C^* \setminus \{\varepsilon\}$. The *concatenation* of two words σ and σ' is denoted by $\sigma \cdot \sigma'$. A word σ' is a *prefix* of a word σ, denoted as $\sigma' \preccurlyeq \sigma$, whenever there exists a word σ'' such that $\sigma = \sigma' \cdot \sigma''$; conversely σ is said to be an *extension* of σ'.

Given an input-output word $\sigma = (x_1, y_1) \cdot (x_2, y_2) \cdots (x_n, y_n) \in \mathcal{D}_C^*$, the input word obtained from σ is denoted by σ_I where $\sigma_I = x_1 \cdot x_2 \cdots x_n \in \mathcal{D}_I^*$ is the projection on inputs ignoring outputs. Similarly, the output word obtained from σ is denoted by σ_O where $\sigma_O = y_1 \cdot y_2 \cdots y_n \in \mathcal{D}_O^*$ is the projection on outputs.

Given a word σ and $i \in [1, |\sigma|]$, $\sigma_{[i]}$ denotes the element at index i in σ. Given a word σ and two integers i, j s.t. $1 \leq i \leq j \leq |\sigma|$, the subword of σ from index i to j is denoted as $\sigma_{[i \cdots j]}$. Given an n-tuple of symbols $e = (e_1, \cdots, e_n)$, for $i \in [1,n]$, $\pi_i(e) = e_i$ denotes the projection of e on its i-th element. The operator π_i is naturally extended to words of n-tuples of symbols to produce the word formed by the concatenation of the projections on the i^{th} element of each tuple.

In every tick, the EI manager first examines the input from the environment (in this context, the drone), and later the output from the controller (the base station). The overall output of the EI manager in every tick is an input-output event. We introduce function IO which is used to treat an input word $\sigma_I \in \mathcal{D}_I^*$, and an output word $\sigma_O \in \mathcal{D}_O^*$ as in input-output word in $(\mathcal{D}_I \times \mathcal{D}_O)^*$.

Function IO: Given an input word $\sigma_I = x_1 \cdot x_2 \cdots x_n \in \mathcal{D}_I^*$ and an output word $\sigma_O = y_1 \cdot y_2 \cdots y_n \in \mathcal{D}_O^*$ s.t. $|\sigma_i| == |\sigma_o|$, $IO(\sigma_i, \sigma_o) = (x_1, y_1) \cdot (x_2, y_2) \cdots (x_n, y_n)$.

A property φ over C defines a set $\mathcal{L}(\varphi) \subseteq \mathcal{D}_C^*$. A program $\mathcal{P} \models \varphi$ iff $\mathcal{L}(\mathcal{P}) \subseteq \mathcal{L}(\varphi)$. In this paper, properties are formally defined as VDTA.

A. Policy Specification

We adopt Valued Discrete Timed Automata (VDTA), a language for expressing policies over cyber-physical systems [24]. A VDTA can be seen as an automaton with a finite set of locations, a finite set of discrete clocks used to represent time evolution, and external input (resp. output channels) called "external variables" which are used for representing system data. They model the data from the monitored system (resp. environment) read from the input (resp.) channels in every tick. In a VDTA, time evolves synchronously: that is, the system executes as a series of discrete *logical ticks* where each tick takes exactly one transition [25]. In the semantics of VDTA, each transition will be associated with values of external variables. Compared with Discrete Timed Automata (DTA), which have only boolean inputs and outputs, the increased expressiveness of valued external variables in VDTA enables more sophisticated policies to be created.

B. VDTA Syntax and Semantics

Let $X = \{x_1, \ldots, x_k\}$ be a finite set of integer variables representing discrete clocks. A *valuation* for a clock variable x of X is an element of $\mathbb{N}$, that is a function from x to $\mathbb{N}$. The set of valuations for the set of clocks X is denoted by χ. For $\chi \in \mathbb{N}^X$, $\chi + 1$ (which captures the ticking of the digital clock) is the valuation assigning $\chi(x) + 1$ to each clock variable x of X. Given a set of clock variables $X' \subseteq X$, $\chi[X' \leftarrow 0]$ is the valuation of clock variables χ where all the clock variables in X' are assigned to 0.

Definition III.1 (Syntax of VDTAs)**.** An VDTA is a tuple $\mathcal{A} = (L, l_0, F, X, I, O, \Delta)$ where:

- L is a finite non-empty set of locations, with $l_0 \in L$ the initial location, and $F \subseteq L$ the set of active locations[1];
- X is a finite set of discrete clocks;
- I is the set of input channels;
- O is the set of output channels;
- Δ is a finite set of transitions, and each transition $t \in \Delta$ is a tuple $\left(l, G, A^X, l'\right)$ also written $l \xrightarrow{G(I,O),A^X} l'$ such that,
  - $l, l' \in L$ are respectively the origin and target locations of the transition;
  - $G = G^D \wedge G^X$ is the guard where
    - $G^D = G^I \wedge G^O$ where
      - G^I is a computable predicate over inputs i.e., conjunction of constraints of the form $f1(I) \sharp f2(I)$, where $f1$ and $f2$ are computable functions over input variables, and $\sharp \in \{<, \leq, =, \geq, >, \neq\}$;
      - G^O is a computable predicate over inputs and outputs i.e., conjunction of constraints of the form $f1(I \cup O) \sharp f2(I \cup O)$, where $f1$ and $f2$ are computable functions over input and output variables (but requiring at least one of the output variables as an argument), and $\sharp \in \{<, \leq, =, \geq, >, \neq\}$;
    - G^X is a clock constraint over X defined as conjunctions of constraints of the form $x \sharp c, x \sharp f1(I), x \sharp f2(I \cup O)$ where $x \in X$ and and $c \in \mathcal{N}$, $f1(I)$ is a computable predicate over input variables, $f2(I \cup O)$ (requiring at least one of the output variables as an argument) is a computable predicate over input and output variables, and $\sharp \in \{<, \leq, =, \geq, >, \neq\}$;
  - $A^X \subseteq X$ is the set of clocks to be reset.

We now provide two examples to demonstrate valued variables and discrete clocks respectively in VDTA.

Example III.2 (VDTA with valued variables)**.** *The signal alteration attack VDTA $\varphi_{AlterLocation}$ in Figure 5a has a set of locations $L = \{l_0, l_1, l_v\}$, with active locations $F = \{l_0, l_1\}$. l_0 is also the initial location. The VDTA has the set of input variables $I = \{\mathsf{CONNECTED}\}$ of type Boolean and the set of output variables $O = \{\mathsf{CONFIG}, \mathsf{ALT}, \mathsf{NORTH}, \mathsf{EAST}\}$ of types {Boolean, unsigned, integer, integer} respectively. In an VDTA, a transition can have guards involving input/output variables, clocks, and functions over input/output variables. For example, the transition from l_1 to l_v happens when* CONFIG $=$ *true and any of* ALT, NORTH, *and* EAST *are* ***not*** *equal to 10, 20, or 20 respectively. This implies that the drone is being configured by the base-station to values that differ from the attackers desire.*

[1]Locations are usually termed *accepting* or *non-accepting* however in our attacker modelling case attacks are active in *accepting* locations, and if in a *non-accepting* location, the attack has failed.

Example III.3 (VDTA with discrete clock)**.** *The signal injection attack VDTA $\varphi_{InjectLand}$ in Figure 5b has a set of locations $L = \{l_0, l_1, l_v\}$, with active locations $F = \{l_0, l_1\}$. l_0 is also the initial location. The policy has a set of discrete clocks $X = \{t\}$. The VDTA has an empty set of input variables $I = \{\}$ as is an output policy, and the set of output variables $O = \{\mathsf{LAND}\}$ of type Boolean. In this VDTA, the transitions can have guards involving output variables and clocks. For example, the transition from l_0 to l_v happens when the* LAND *signal is false and clock $t \geq T_A$. The attack time, T_A, is the time after which the signal* LAND *is required to keep the attack active, hence the transition to l_v if* LAND *is not present, and l_1 if* LAND *is present.*

1) Semantics for VDTA: Let $\mathcal{A} = (L, l_0, F, X, I, O, \Delta)$ be a VDTA. The semantics of $\mathcal{A}$ is a timed transition system, where a state consists of a location, and valuations of clocks X. Each transition is associated with values of external variables in C.

Definition III.4 (Semantics of VDTAs)**.** The semantics of $\mathcal{A}$ is a timed transition system $[\![\mathcal{A}]\!] = (Q, q_0, Q_F, \Gamma, \rightarrow)$, defined as follows:

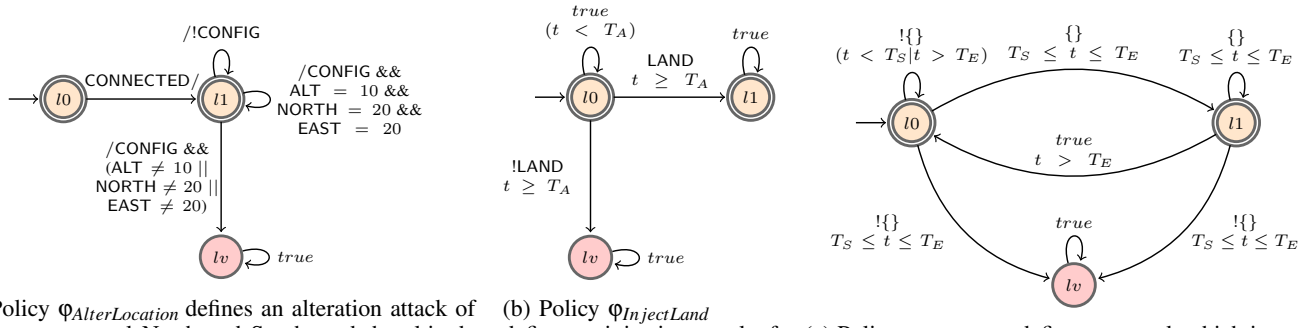
- $Q = L \times \mathbb{N}^X$, is the set of states of the form $q = (l, \chi)$ where $l \in L$ is a location, χ is a valuation of clocks;
- $Q_0 = \{(l_0, \chi_{[X \leftarrow 0]})\}$ is the set of initial states;
- $Q_F = F \times \mathbb{N}^X$ is the set of active states;
- $\Gamma = \{\eta \mid \eta \in \mathcal{D}_C\}$ is the set of transition labels;
- $\rightarrow \subseteq Q \times \Gamma \times Q$ the transition relation is the smallest set of transitions of the form $(l, \chi) \longrightarrow \eta \langle l', \chi' \rangle$ such that $\exists \left(l, G, A^X, l'\right) \in \Delta$, with $G^X(\chi + 1, \eta) \wedge G^D(\eta)$ evaluating to true, and $\chi' = (\chi + 1)[A^X \leftarrow 0]$.

A *run* ρ of $[\![\mathcal{A}]\!]$ from a state $q \in Q$ over a *trace* $w = \eta_1 \cdot \eta_2 \cdots \eta_n$ is a sequence of moves in $[\![\mathcal{A}]\!]$: $\rho = q \xrightarrow{\eta_1} q_1 \cdots q_{n-1} \xrightarrow{\eta_n} q_n$, for some $n \in \mathbb{N}$. A run is accepted if it starts from the initial state $q_0 \in Q$ and ends in an accepted state $q_n \in Q_F$.

Definition III.5 (Deterministic (complete) VDTA)**.** A VDTA $\mathcal{A} = (L, l_0, F, X, I, O, \Delta)$ with its semantics $[\![\mathcal{A}]\!]$ is said to be a *deterministic* VDTA whenever for any location l and any two distinct transitions $\left(l, g_1, A_1^X, l_1'\right) \in \Delta$ and $\left(l, g_2, A_2^X, l_2'\right) \in \Delta$ with the same source l, the conjunction of guards $g_1 \wedge g_2$ is unsatisfiable. $\mathcal{A}$ is *complete* whenever for any location $l \in L$ the disjunction of the guards of the transitions leaving l evaluates to *true*.

C. VDTA Enforcers

An enforcer monitors and corrects both the input and output of a system according to a given policy φ. The details of how the bidirectional enforcement mechanism for VDTA is defined and algorithms of how to synthesise an enforcer from a policy specified as VDTA are provided in [12], [26]. Here we model drone attacks of jamming, injection, and alteration, as VDTA. We can synthesise enforcers using these exiting approaches, which are then placed between the environment and controller, as illustrated in Figure 2b. We term these as attack enforcers.



(a) Policy $\varphi_{AlterLocation}$ defines an alteration attack of distances to travel North and South, and the altitude to fly at.

(b) Policy $\varphi_{InjectLand}$ defines an injection attack of the LAND signal

(c) Policy $\varphi_{JamAllTime}$ defines an attack which jams all control and status signals for a specified time period

Figure 5. A range of VDTA policies which define attacks on the drone system

Table I
ATTACK ENFORCER TRACE FOR $\varphi_{InjectLand}$

Tick	Controller Output	Clock t	$t \leq T_A$	Enforced Output	Transitions
0	{}	0	*true*	{}	$l0$
1	{}	1	*true*	{}	$l0 \rightarrow l0$
2	{}	2	*true*	{}	$l0 \rightarrow l0$
3	{}	3	*false*	{LAND}	$l0 \rightarrow l1$

The enforcer must keep the policy satisfied, to ensure the attack remains active, and so it will examine the updated external variables (input and output variables) each tick, and will transform any that are non-accepting that would cause the attack to fail.

Example III.6 (VDTA Attack Enforcer)**.** *Consider the VDTA policy explained in Example III.3 and illustrated in Figure 5b. This policy defines an injection attack which occurs when the policy clock reaches T_A. The synthesised attack enforcer would, to ensure the policy remains active, ensure the output* LAND *was true before or as the clock reaches T_A. An example trace, where $T_A = 3$, is provided in Table I where the controller does not provide the output in tick 3, so the enforcer injects it.*

In the following section, we introduce VDTA attack policies for drones and provide an example of bidirectional enforcement.

IV. DRONE ATTACKS WITH RUNTIME ENFORCERS

A. Individual Attacks

As discussed earlier, attacks on drone systems can be grouped into: jamming, injection, and alteration of drone status and/or base station control signals. We present policies that produce time based jamming, time based injection, and persistent alteration attacks. Attack parameters are able to be set by the attacker to achieve their desired impact.

1) Jamming: In Figure 5c the policy $\varphi_{JamAllTime}$ is illustrated. Designed to jam all control and status signals between the drone and base station, this policy consists of 3 locations; $l0$ and $l1$ active locations, and lv a failed trap location. A discrete clock, t, is used to start and end the attack. This policy is defined with empty input output ({} where *Boolean* variables are $false$ and *valued* variables are *0*) and any input or output (!{} where one or more *Boolean* variables is $true$ and/or one or more *valued* variable is not *0*) . Starting in $l0$ the policy accepts any control or status signal (!{}) while the clock is less than or greater than the defined start (T_S) and end (T_E) times. When the clock, t, is greater than or equal to the start time (T_S), any input or output (!{}) would cause a transition to the violation location, lv, and an empty input output ({}) results in a transition to $l1$. In $l1$, any input or output (!{}) causes a transition to the violation location, unless the clock is greater than the end time (T_E), when the transition back to $l0$ is taken.

To avoid transitioning to the violation location, the synthesised enforcer will suppress any input or output signal from being passed between the base station and drone between the start and stop times. This behaves as a signal jammer active between the start T_S and stop times T_E.

Example IV.1 (Bidirectional VDTA Attack Enforcer)**.** *Consider the Policy $\varphi_{JamAllTime}$ VDTA explained above and illustrated in Figure 5c. An example trace, where $T_S = 2$ and $T_E = 3$, is provided in Table II where signals from the base station and drone are suppressed.*

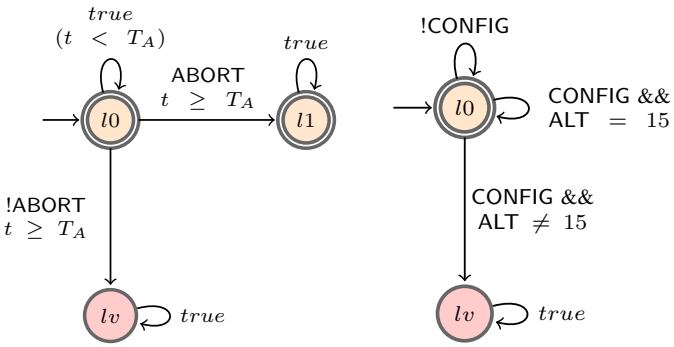
2) Injection: In Figure 6a the policy $\varphi_{InjectAbort}$ is illustrated. This policy consists of two active locations, $l0$ and $l1$, and one violating trap location, lv. There are no input channels $I = \{\}$ and output channel $O = \{$ABORT$\}$ of type *boolean*. The policy starts in $l0$, where any event while the clock, t, is less than the attack time, causes the self loop to remain in $l0$. When the clock, t, is greater than or equal to the attack time (T_A), the absence of the ABORT signal would cause a transition to the violation location, lv. Alternatively, with the presence of the ABORT signal, the policy would transition to the active location, $l1$.

When the clock, t, reaches the attack time (T_A), if not present from the base station, the enforcer will inject the ABORT signal, to prevent transition to the violation location. This behaves as an injection attack which occurs at a specified time.

3) Alteration: In Figure 6b the policy $\varphi_{AlterAltitude}$ is illustrated. This simple policy consists of a single active location

Table II
ATTACK ENFORCER TRACE FOR $\varphi_{JamAllTime}$

Tick	Drone Input	Clock t	$t < T_S$	$t > T_E$	Enforced Input	Base Station Output	Enforced Output	Transitions
0	{}	0	*true*	*false*	{}	{}	{}	$l0$
1	{CONNECTED}	1	*true*	*false*	{CONNECTED}	{CONFIG, ALT = 10}	{CONFIG, ALT = 10}	$l0 \rightarrow l0$
2	{}	2	*false*	*false*	{}	{RUN}	{}	$l0 \rightarrow l1$
3	{}	3	*false*	*false*	{}	{RUN}	{}	$l1 \rightarrow l1$
4	{}	4	*false*	*true*	{}	{RUN}	{RUN}	$l1 \rightarrow l0$



(a) Policy $\varphi_{InjectAbort}$ defines an attack which injects an abort signal after a specified time

(b) Policy $\varphi_{AlterAltitude}$ defines a configuration alteration attack

Figure 6. Injection and alteration attacks expressed as VDTA

$l0$, and a violating trap location lv. There are no input channels, $I = \{\}$, and the output channels $O = \{\mathsf{CONFIG}, \mathsf{ALT}\}$ which are of types $\{$*Boolean*, *unsigned integer*$\}$. Initially in the active location $l0$, the policy takes self loops when the CONFIG signal is absent, or when the CONFIG signal is present and ALT is present with a value of 15. This reflects the desire for the altitude to be set to 15 meters. If the CONFIG signal is present and the value is not equal to 15, the policy would transition to the violation location lv.

To avoid the transition to the violation location the synthesised enforcer would set the ALT value to 15 whenever it is present. This behaves as a man-in-the-middle attack where any ALT signal is set to a value of 15, resulting in the drone operating at 15 meters above ground altitude.

V. ENFORCER INTERCHANGE FOR SOPHISTICATED ATTACKS

A. Problem Definition

In the previous section, we introduced a set of attack policies which alone, are able to interfere with the drone system, however are limited to a single attack enforcer at any time. If individual attacks were combined by some means, far more sophisticated attacks could be launched. These should incorporate the status of drone and controller allowing a range of attacks for different missions to be combined.

In this section, we introduce enforcer interchange to launch complex attacks. First, individual attacks must be enabled/-suspended, so we extend VDTA enforcers to be suspendable. Second, we introduce enforcer interchange, to combine suspendable VDTA enforcers.

B. Activating/Suspending Attack Enforcers Dynamically

In order to choreograph complex attacks, the attack enforcers need to be activated and suspended during system operation based on the desired attack and the current status of the drone and base station.

To evaluate guards, increment discrete clocks, and take any location transitions, the VDTA must be activated via an activation signal. If not activated, the VDTA is suspended, and therefore it's location and any clocks are frozen and do not increment. In plain language, the enforcer attack is turned off, not enforcing inputs or outputs. This is further elaborated using Remark V.1.

Remark V.1. *As discussed, attack policies need to be activated/suspended dynamically. Here we describe the mechanism for this. Let $\varphi_{AlterLocation}$ be the policy for a location attack, as introduced in Example III.2 Let $x \in \mathcal{D}_I$ denote the input in a particular tick.*

- *If the attack will be active (not suspended) in that particular tick, the state of the policy $\varphi_{AlterLocation}$ will be updated by consuming the event $(x,y) \in \mathcal{D}_C$, evaluating the transition relation Δ, and updating the location as appropriate.*
- *If the attack will be suspended in a given tick, for updating of the policy $\varphi_{AlterLocation}$ w.r.t. the input $x \in \mathcal{D}_I$ that is observed in that tick, the output of the attack policy in that tick is considered to be invalid/empty (denoted as $!\alpha$). For every location in $\varphi_{AlterLocation}$, for all $x \in \mathcal{D}_I$ we consider implicit self transitions with event $(!\alpha)$, allowing the policy to remain in the same location when it is suspended.*

Thus, the output domain $\mathcal{D}_O$ will be will be considered as $\mathcal{D}_O \cup \{!\alpha\}$.

Example V.2. *An example of a suspendable policy is presented in Figure 7. This is a modified location altering attack $\varphi_{AlterLocation}$ presented first in Figure 5a. If the policy is in location l_0, while the policy is suspended ($!\alpha$), the input* CONNECTED *is ignored. As a result, the implicit self loop (shown as a dashed line) in l_0 is taken instead of advancing to location l_1.*

C. Enforcer Interchange (EI)

We consider the system (drone and base station) as a *grey-box*, since the EI Manager is designed based on the missions

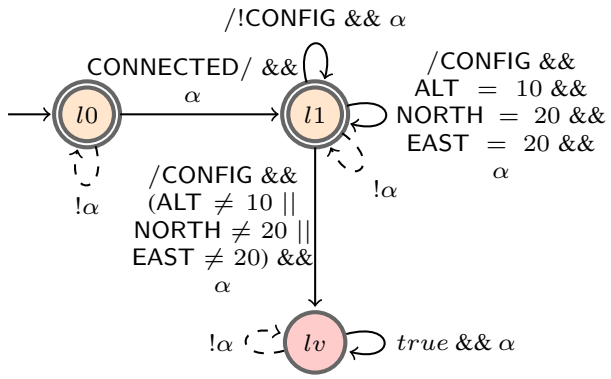


Figure 7. Suspendable $\varphi_{AlterLocation}$

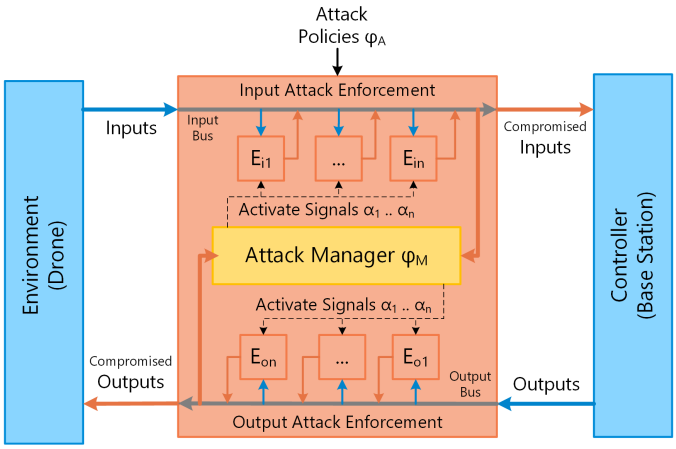


Figure 8. Context of EI Manager

the drone is configured for, and I/O signals are known and understood, but the internals of the drone and base station are unknown.

For the synthesis of the EI Manager, the user (a malicious actor) provides a set of policies, one policy per attack defined as VDTA, and a further policy specifying an attack manager, also defined as a VDTA. The system diagram for an EI Manager is illustrated in Figure 8.

Given a set of attack policies denoted as $\varphi_A = \{\varphi_1, \cdots, \varphi_n\}$ where n is the number of attacks, $\forall i \in 1 \cdots n : \varphi_i \subseteq \mathcal{D}_C^*$ is the policy corresponding to attack i.

Definition V.3 (Attack Manager)**.** Given φ_M specifying the attack manager defined as a VDTA $\mathcal{A}_{\varphi_M} = (L, l_0, F, X, I, O, \Delta)$ with semantics $(Q, q_0, Q_F, \Gamma, \rightarrow)$, extended with A, an n-tuple of Boolean activation labels in each location. The attack manager is $AM_{\varphi_m} : D_C^* \rightarrow \mathbb{B}_1 \times \mathbb{B}_2 \cdots \times \mathbb{B}_n$. Let $\sigma \in (D_C)^*$, and let $q \in Q$ be the state reached in the VDTA AM_{φ_m} upon σ, $AM_{\varphi_m}(\sigma) = (\alpha_1, \alpha_2, \cdots, \alpha_n)$, where $A_q = (\alpha_1, \alpha_2, \cdots, \alpha_n)$ is the Boolean n-tuple in the location corresponding to state q.

The attack manager executes as defined in Algorithm 1. In every reaction (tick), the environment input to the EI Manager is $x \in \mathcal{D}_I$.

The EI Manager reads the input channels x (i.e., input from

Algorithm 1 Enforcer Interchange Manager

$n \leftarrow$ number of enforcers
$q_M \leftarrow q_{M0}$
$\alpha \leftarrow A_{q_{M0}}$
$t_M \leftarrow 0$
while *true* **do**
  $x \leftarrow$ read_input()
  for E_i^j where $\forall j \in 1 \cdots n$ **do** *// Input Enforcement*
    $x_{cE_i^j} \leftarrow \mathsf{E_i^j}\left(x_{E_i^j}, \alpha^j\right)$
  end for
  controller(x_c)
  $y \leftarrow$ read_output()
  for E_o^j where $\forall j \in 1 \cdots n$ **do** *// Output Enforcement*
    $y_{cE_o^j} \leftarrow \mathsf{E_o^j}\left(x_{cE_o^j}, y_{E_o^j}, \alpha^j\right)$
  end for
  for E^j where $\forall j \in 1 \cdots n$ **do** *// Update Enforcer states*
    if α^j **then**
      $q_{E^j} \leftarrow q''_{E^j}$ where $q_{E^j} \xrightarrow{(x_c, y_c)} q''_{E^j}$
    end if
  end for
  // Update Attack Manager state
  $q_M \leftarrow q''_M$ where $q_M \xrightarrow{(x_c, y_c)} q''_M$
  $\alpha \leftarrow A_{q_M}$ *// Update activation tuple*
  $t_M \leftarrow t_M + 1$ *// Increment clocks*
  environment(y_c)
end while

the environment) along the internal input bus, appropriate channels are diverted to relevant enforcers (synthesised as per [12], [26]). The attack manager AM_{φ_m} provides the set of *Boolean* activate signals ($A_q = (\alpha_1, \cdots, \alpha_n)$) enabling/suspending the appropriate attack enforcers based on the location reached in φ_M. Attack policies may be defined for subsets of the complete set of input/output channels such that multiple attacks can be active simultaneously and enforcers will not edit shared signals.

Input enforcers then execute with the inputs $x_{Ei} \subseteq x$ that are of relevance for that particular policy/enforcer. Any suspended enforcer(s) return the input unaltered. Activated enforcers perform any edits necessary to keep their attack from failing, as explained in Section III-C. Once all n input attack enforcers have processed the input, the input bus now carries compromised input x_c which is read by the attack manager and exposed to the controller.

The controller executes, and produces the output $(y) \in \mathcal{D}_O$ given input x_c. The output y is taken along the output bus. The *unchanged* set of activate signals (A_q) is forwarded to the output attack enforcers. Again, similarly to input attacks, activated output enforcers execute with relevant outputs $y_{E_i} \subseteq y$, and perform any edits necessary to keep their attack from failing. With all n output attack enforcers having executed, the output bus carries compromised output y_c which is read by the attack manager, and exposed to the environment. The final input-output event for the tick is then (x_c, y_c) which then

updates the attack manager state and the activation signals.

Remark V.4 (Composition for Simultaneously Active Enforcers)**.** *Figure 8 illustrates the attack enforcers share input and output buses, however, the attack policies and attack manager must be designed such that any* ***simultaneously active*** *enforcers have independent input/output channels. This is to ensure they do not both enforce a signal simultaneously. Note that it can be desirable for some attack policies to share input and output channels, but the attack enforcers cannot be active simultaneously. The attack manager policy allows different enforcers to be active for different situations. In future work, we will assess enforcer interchange separately as a method of composition where simultaneously active enforcers alter shared channels.*

Example V.5. *Consider the drone system, programmed for three missions (flight test, relocation, and delivery). In Figure 9 we illustrate the attack manager policy which enables and disables attacks to interfere with each mission.*

The policy consists of locations $L = \{Idle, Conn, Alt, Loc, Climb, Jam, AtLoc, Land, Land2, StealD, StealP, End\}$ *with initial location* $l_0 = Idle$*, and clocks,* $X = \{t\}$*. The policy is defined over input channels* $x = \{\mathsf{CONNECTED}, \mathsf{AT\_ALT}, \mathsf{AT\_LOC}, \mathsf{LANDED}\}$ *of types* $\{boolean, boolean, boolean, boolean\}$ *which refer to* $\{drone\ connected, drone\ at\ configured\ altitude, drone\ at\ configured\ location, drone\ landed\}$*.*

Output channels $y = \{\mathsf{MISSION}, \mathsf{RUN}, \mathsf{LAND}, \mathsf{RETURN}, \mathsf{END}\}$ *of types* $\{integer, boolean, boolean, boolean, boolean\}$ *which refer to* $\{configure\ for\ mission\ (1)\ flight\ test\ (2)\ relocation\ or\ (3)\ delivery, execute\ the\ configured\ mission, execute\ landing, return\ to\ base, end\ mission\}$*.*

Five attack policies are managed by this attack manager, $\{AlterAltitude, AlterLocation, JamLand, InjectLand, JamAbort\}$*. These attack policies are defined in earlier sections. Activation tuples are* $A_{Alt} = \{True, False, False, False, False\}$*,* $A_{Loc} = \{False, True, False, False, False\}$*,* $A_{Jam} = \{False, False, True, False, False\}$*,* $A_{AtLoc} = \{False, False, False, True, True\}$*, in all other locations* $A = \{False, False, False, False, False\}$*.*

If the base station requests a flight test, $\mathsf{MISSION} = 1$*, the attack manager will firstly activate the attack which alters the configured altitude and then will jam signals once the drone reaches altitude until the clock, t, reaches JamTime.*

If the base station then requests a delivery, $\mathsf{MISSION} = 2$*, the manager will activate the location attack which sets the drone to fly to the attacker's location where the package is intercepted. Once the drone reaches the location, to ensure it lands and base station doesn't abort the mission, the attacks to jam the* ABORT *signal and inject the* LAND *signal are activated. The drone will then deliver the package to the attacker.*

If the base station then requests the drone relocates, $\mathsf{MISSION} = 3$*, the attack manager will follow the same actions as for* $\mathsf{MISSION} = 2$*, though this time the drone will not return to base and so could be stolen by the attacker.*

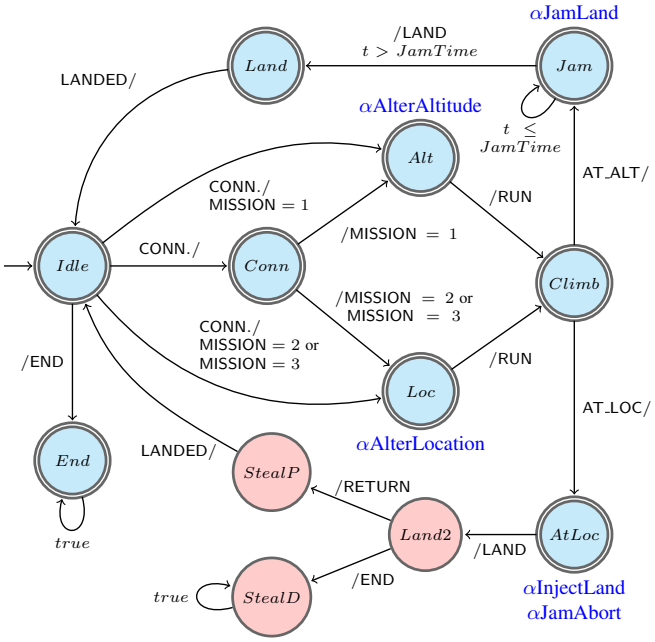


Figure 9. An enforcer interchange attack manager policy which choreographs attacks to interfere with take off, intercept packages, and steal a drone.

VI. Results

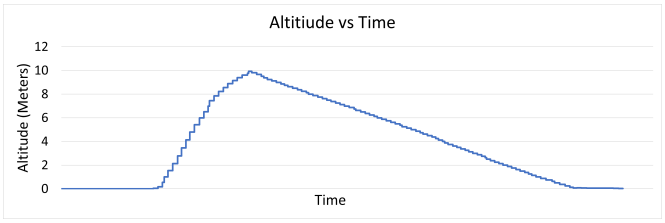
To evaluate the impact of our enforcer attacks and the combined EI attacks, we consider the quadcopter simulator, introduced in Section II-C, tasked with one of three missions: flight test, delivery, and relocation. When evaluating individual attacks we use the flight test mission and consider the profile of the drone's altitude over time. For the combined EI attack, we see all three missions execute and consider the profile of altitude, longitude, and latitude.

A. Enforcer Attacks

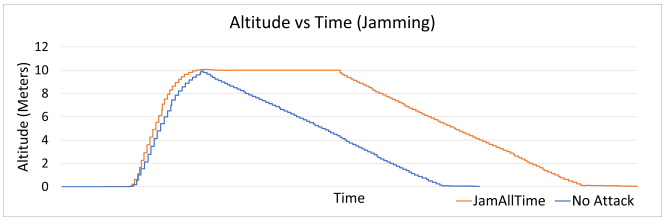
1) Flight Test Profile: As a baseline to compare our individual attacks, we use the flight test mission, where the drone is instructed to climb to a defined altitude then descend to the ground. For the results the configured altitude is 10m. The chart in Figure 10a illustrates the altitude (meters above ground level) across time as the drone climbs and then descends.

2) Jamming Attack: In Section 5c we introduced $\varphi_{JamAllTime}$ which blocks all control and status signals for a defined time period. As illustrated in Figure 10b the drone remains at 10 meters altitude for a longer duration while the attack is active. The base station's control signal, requesting the drone to land, is unable to reach the drone while the attack is active. In this example, only a mild inconvenience is likely, but extending the duration of the attack could result in the drone running out of battery and crash landing.

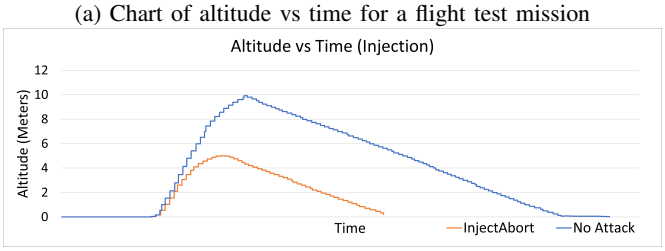
3) Injection Attack: In Section 6a we introduced $\varphi_{InjectAbort}$ which injects a control signal to the drone requesting it to abort the mission at a defined time. As illustrated in Figure 10c the flight test mission is not



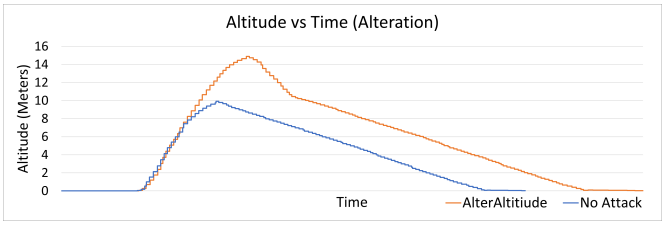
(a) Chart of altitude vs time for a flight test mission



(b) With a jamming attack ($\varphi_{JamAllTime}$)



(c) With a control signal injection ($\varphi_{InjectAbort}$) attack



(d) With a control signal alteration ($\varphi_{AlterAltitude}$) attack.

Figure 10. Charts of altitude vs time for a flight test mission without attacks and then with various attacks. Note the different Y axis scale for sub figure (d).

accomplished as the drone doesn't reach full altitude and returns to the ground prematurely. This is an example attack that may be used on a rogue drone, to ground it before it can execute it's malicious mission.

4) Alteration Attack: In Section 6b we introduced $\varphi_{AlterAltitude}$ which modifies the configuration control signal to the drone setting the altitude to 15 meters above ground level. As illustrated in Figure 10d the drone climbs above the expected altitude (of 10m), this could breach airspace clearances and conflict with other air traffic.

B. Enforcer Interchange Attacks

We now consider a sophisticated attack using our Enforcer Interchange policy illustrated in Figure 8. In these results, the drone is sequentially requested to execute a flight test, a delivery, and a relocation.

This requires five attack enforcers: AlterAltitude, AlterLocation, JamLand, InjectLand, and JamAbort. The policy for AlterAltitude is previously defined in Figure 6b. The policy for AlterLocation is similar to AlterAltitude additionally with values defined for the distance North and East to travel. The policies for JamLand and JamAbort follow the structure of JamAll, as previously illustrated in Figure 5c, but only violate when the signal LAND or respectively ABORT are present. The policy for InjectLand follows the structure of InjectAbort, illustrated in Figure 6a, with the LAND signal substituted for the ABORT signal.

1) Altitude Profile: With no attacks present the drone executes the requested flight test (to 10m), delivery (at 20m altitude), and relation (at 15m altitude). This is illustrated in the altitude over time profile in Figure 11. The second and third peaks represent the drone's travel to and from the delivery location.

With the enforcer interchange attack manager present, the drone's altitude profile is altered. The AlterAltitude attack sets the altitude to 5m instead of the requested 10m. The

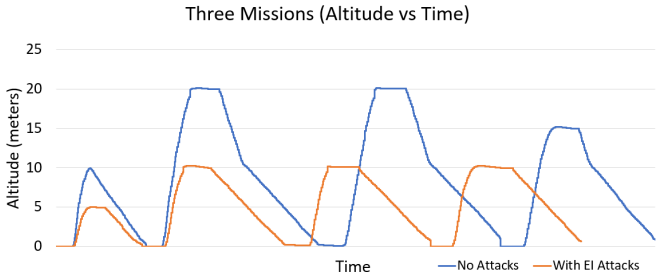


Figure 11. Charts of altitude vs time for three consecutive missions without and with EI attacks

AlterLocation attack sets the altitude to 10m instead of the requested 20m (for delivery missions) and 15m (for relocation missions). This is illustrated in Figure **??**. Note the axes scales are matched to Figure 11, which shows the lower altitudes and a faster completion of the missions.

2) Location Profile: The delivery and relocation missions result in the drone travelling to different geographical locations. We illustrate this in Figure 12. These results are presented in latitude and longitude as the drone simulator uses real world mapping.

Without attacks active the drone first travels to the South-East delivery location, then returns before travelling to the due West location to relocate. With the attack manager present, the locations are replaced with the attackers location, this results in both the delivery and relocation travelling to the North-East where the attacker is positioned.

VII. Related Work

Modelling cyber security attacks is a broad field with various high level approaches [27] to understand the attacks. The attack modelling taxonomy in [28] provides an approach more focused on the cross domain impacts of exploiting CPS systems. These approaches are high level and therefore do not

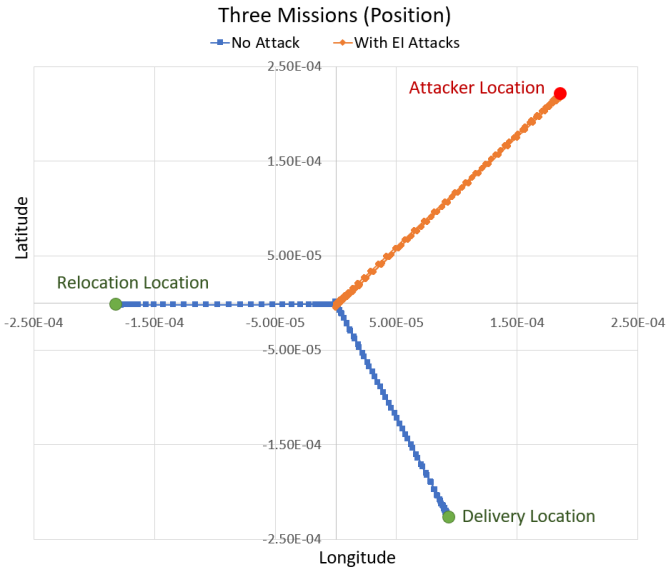


Figure 12. Chart of position for three consecutive missions.

consider implementing and demonstrating such attacks. Our work is a level lower, as we model specific attacks, identified during high level modelling. Additionally, our formal approach can be synthesised to demonstrate the attack and it's impact on a target system.

The aeronautical industry is safety critical. The application of *runtime assurance* to turbofan engines is considered by NASA in [29] and to autonomous aeronautical systems in [30]. The drone industry is inconsistently regulated, and the high overhead of formal validation can be avoided by system designers. The result is an industry with many exploitable vulnerabilities [6], which are found in examples like the analysis of a DJI drone in [5].

Existing work in *runtime assurance* [9], specifically in *runtime enforcement*, comprises of non-reactive techniques [21]–[23], [31], which are unsuitable for our drone application. Our work is related to the reactive class of runtime enforcement techniques [12], [32], [33].

These rely on low-overhead wrappers, which mediate between the environment and the controller of a reactive system, to ensure that the system operates safely at all times by ensuring that all user specified policies hold. Whenever the input and output of the system leads to non-compliance, the *enforcer* alters the input / output streams appropriately.

Early enforcers for reactive systems [32] were uni-directional. More recently, bi-directional enforcement has been developed for both industrial processes [24] and medical devices [12].

Our inspiration comes from edit-automata [10], which performs *suppression*, *insertion*, and *editing* actions for streams, which are generated by transformational systems. These actions are analogous to drone attack actions such as *jamming*, *injection*, and *alteration*. However, edit automata are not expressive enough for complex drone attack policies, as they are not suitable for CPS. In particular, they lack any notion of timing and the need to ensure that the reactivity of the system could be attacked. Also, runtime enforcement, in general, has been used for ensuring safe system operation rather than being used for studying the ethical hacking of drones using formal methods.

Hence, our work considers the flip-side of runtime enforcement, using malicious actors. To be able to model many complex attack scenarios, we propose two innovations, which are lacking in Runtime Enforcement (RE) literature. First, we use an expressive automata called VDTA to express attack policies. We then generate a set of enforcers from these policies, where the enforcers are composed using a global policy, called the *attack manager policy*. This is used for suitably orchestrating enforcers to launch complex attacks. For performing this, we use, for the first time, the concept of *suspension* from synchronous languages [34], [35]. This generalises the usual enforcement actions of known enforcers such as editing. Hence, we term the current framework *enforcer interchange*.

Enforcer interchange is also inspired by earlier concepts in synchronous programming such as *Mode Automata* [36] and real-time systems i.e., *Modechart* [37]. *Mode Automata*, however, lacks a direct notion of time unlike VDTA. Also, *Mode Automata* and *Modechart* have no concept of activation/suspension. In our work, the extension of VDTA with the activation/suspension effectively *freezes* a subset of the attack policies in any given state.

VIII. Conclusions

The cyber-security of unmanned aerial drones is of increasing concern [6]. However, a formal framework for attacker modelling and launching of attacks is lacking in the literature. To this end, we leverage runtime enforcement, combined with the idea of *suspension* from synchronous programming to develop a new formal framework.

We introduce the *enforcer interchange* framework. This allows us to create sophisticated and adaptive attacks on drones. We demonstrate this using a drone simulator that is able to be tasked with a range of missions. Individual runtime enforcer attacks were presented, then combined using an *enforcer interchange* manager which used different attacks based on the situation and mission. Future attacks can be choreographed with different enforcer interchange managers, making use of, and extending, the suite of attack models. This provides a formal framework, for the first time, for validating drone systems against malicious interference by leveraging formal methods.

This work has some limitations. First, while we believe that *enforcer interchange* generalises known RE frameworks for CPS which are limited to some form of editing actions only, we are yet to formally study the power of this relative to other classes of automata such as edit automata. Second, we are yet to study defense mechanisms for mitigation.

Source Access

The source codes for the drone communication case study, including the attack enforcers and enforcer interchange manager, are available online under the MIT license at https://github.com/PRETgroup/drone-attack-enforcers.

References

[1] R. Alur, *Principles of cyber-physical systems*. MIT Press, 2015.

[2] D. Hambling. (2021) This record-breaking shanghai drone display is a show of technological strength. https://www.forbes.com/sites/davidhambling/2021/04/06/why-this-record-breaking-drone-display-in-shanghai-is-a-show-of/-technological-strength/.

[3] R. Kellermann, T. Biehle, and L. Fischer, "Drones for parcel and passenger transportation: A literature review," *Transportation Research Interdisciplinary Perspectives*, vol. 4, p. 100088, 3 2020.

[4] (2022) Wing. [Online]. Available: https://wing.com/

[5] F. Trujano, B. Chan, G. Beams, and R. Rivera, "Security analysis of dji phantom 3 standard," *Massachusetts Institute of Technology*, 2016.

[6] J.-P. Yaacoub, H. Noura, O. Salman, and A. Chehab, "Security analysis of drones systems: Attacks, limitations, and recommendations," *Internet of Things*, vol. 11, p. 100218, 2020.

[7] Y. Zeng, R. Zhang, and T. J. Lim, "Wireless communications with unmanned aerial vehicles: Opportunities and challenges," *IEEE Communications Magazine*, vol. 54, no. 5, pp. 36–42, 2016.

[8] X. Lin, R. Wiren, S. Euler, A. Sadam, H.-L. Määttänen, S. Muruganathan, S. Gao, Y.-P. E. Wang, J. Kauppi, Z. Zou *et al.*, "Mobile network-connected drones: Field trials, simulations, and design insights," *IEEE Vehicular Technology Magazine*, vol. 14, no. 3, pp. 115–125, 2019.

[9] M. Clark, X. Koutsoukos, J. Porter, R. Kumar, G. Pappas, O. Sokolsky, I. Lee, and L. Pike, "A study on run time assurance for complex cyber physical systems," Air Force Research Laboratory, Vanderbilt University, Iowa State University, University of Pennsylvania, Galois Inc., Tech. Rep., 2013.

[10] J. Ligatti, L. Bauer, and D. Walker, "Edit automata: Enforcement mechanisms for run-time security policies," *International Journal of Information Security*, vol. 4, no. 1, pp. 2–16, 2005.

[11] D. de Niz, B. Andersson, and G. Moreno, "Safety enforcement for the verification of autonomous systems," in *Autonomous Systems: Sensors, Vehicles, Security, and the Internet of Everything*, vol. 10643. International Society for Optics and Photonics, 2018, p. 1064303.

[12] S. Pinisetty, P. S. Roop, S. Smyth, N. Allen, S. Tripakis, and R. V. Hanxleden, "Runtime enforcement of cyber-physical systems," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 5s, pp. 178:1–178:25, Sep. 2017.

[13] O. Westerlund and R. Asif, "Drone hacking with raspberry-pi 3 and wifi pineapple: Security and privacy threats for the internet-of-things," in *2019 1st International Conference on Unmanned Vehicle Systems-Oman (UVS)*. IEEE, 2019, pp. 1–10.

[14] A. H. Abunada, A. Y. Osman, A. Khandakar, M. E. H. Chowdhury, T. Khattab, and F. Touati, "Design and implementation of a rf based anti-drone system," *2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies, ICIoT 2020*, pp. 35–42, 2 2020.

[15] P. Valianti, S. Papaioannou, P. Kolios, and G. Ellinas, "Multi-agent coordinated close-in jamming for disabling a rogue drone," *IEEE Transactions on Mobile Computing*, 2021.

[16] S.-H. Seo, B.-H. Lee, S.-H. Im, and G.-I. Jee, "Effect of spoofing on unmanned aerial vehicle using counterfeited gps signal," *Journal of Positioning, Navigation, and Timing*, vol. 4, no. 2, pp. 57–65, 2015.

[17] A. J. Kerns, D. P. Shepard, J. A. Bhatti, and T. E. Humphreys, "Unmanned aircraft capture and control via gps spoofing," *Journal of Field Robotics*, vol. 31, no. 4, pp. 617–636, 2014.

[18] A. Belous and V. Saladukha, "Hardware trojans in electronic devices 3.1 hardware trojan programs in telecommunication systems 3.1.1 trojans in network equipment," pp. 209–275, 2020. [Online]. Available: https://doi.org/10.1007/978-3-030-47218-4_3

[19] M. A. Rahman, M. T. Rahman, M. Kisacikoglu, and K. Akkaya, "Intrusion detection systems-enabled power electronics for unmanned aerial vehicles," *2020 IEEE CyberPELS, CyberPELS 2020*, 10 2020.

[20] S. Pinisetty, P. S. Roop, S. Smyth, N. Allen, R. V. Hanxleden, S. Tripakis, R. von Hanxleden, S. Pinisetty, P. S. Roop, S. Smyth, N. Allen, S. Tripakis, and R. von Hanxleden, "Runtime enforcement of cyber-physical systems *," *Runtime Enforcement of Cyber-Physical Systems. 1, 1, Article*, vol. 1, 2017. [Online]. Available: https://doi.org/10.1145/nnnnnnn.nnnnnnn

[21] F. B. Schneider, "Enforceable security policies," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 1, pp. 30–50, 2000.

[22] Y. Falcone, L. Mounier, J.-C. Fernandez, and J.-L. Richier, "Runtime enforcement monitors: composition, synthesis, and enforcement abilities," *FMSD*, vol. 38, no. 3, pp. 223–262, 2011.

[23] J. Ligatti, L. Bauer, and D. Walker, "Run-time enforcement of nonsafety policies," *ACM Trans. Inf. Syst. Secur.*, vol. 12, no. 3, pp. 19:1–19:41, Jan. 2009.

[24] H. Pearce, S. Pinisetty, P. S. Roop, M. M. Y. Kuo, and A. Ukil, "Smart I/O modules for mitigating cyber-physical attacks on industrial control systems," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 7, pp. 4659–4669, 2020.

[25] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone, "The synchronous languages 12 years later," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 64–83, Jan 2003.

[26] H. Pearce, S. Pinisetty, P. S. Roop, M. M. Y. Kuo, and A. Ukil, "Smart i/o modules for mitigating cyber-physical attacks on industrial control systems," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 7, pp. 4659–4669, 2020.

[27] H. Al-Mohannadi, Q. Mirza, A. Namanya, I. Awan, A. Cullen, and J. Disso, "Cyber-attack modeling analysis techniques: An overview," *Proceedings - 2016 4th International Conference on Future Internet of Things and Cloud Workshops, W-FiCloud 2016*, pp. 69–76, 10 2016.

[28] M. Yampolskiy, P. Horvath, X. D. Koutsoukos, Y. Xue, and J. Sztipanovits, "Taxonomy for description of cross-domain attacks on cps," *Proceedings of the 2nd ACM international conference on High confidence networked systems*, 2013.

[29] J. D. Schierman, D. A. Neal, E. Wong, and A. Chicatelli, "Runtime assurance protection for advanced turbofan engine control," *AIAA Guidance, Navigation, and Control Conference, 2018*, 1 2018. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2018-1112

[30] J. D. Schierman, M. D. Devore, N. D. Richards, and M. A. Clark, "Runtime assurance for autonomous aerospace systems," *https://doi.org/10.2514/1.G004862*, vol. 43, pp. 2205–2217, 9 2020. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/1.G004862

[31] S. Pinisetty, Y. Falcone, T. Jéron, H. Marchand, A. Rollet, and O. Nguena Timo, "Runtime enforcement of timed properties revisited," *FMSD*, vol. 45, no. 3, pp. 381–422, 2014.

[32] R. Bloem, B. Könighofer, R. Könighofer, and C. Wang, "Shield synthesis: Runtime enforcement for reactive systems," in *Tools and Algorithms for the Construction and Analysis of Systems*, C. Baier and C. Tinelli, Eds. Berlin, Heidelberg: Springer, 2015, pp. 533–548.

[33] M. Wu, J. Wang, J. Deshmukh, and C. Wang, "Shield synthesis for real: Enforcing safety in cyber-physical systems," in *2019 Formal Methods in Computer Aided Design (FMCAD)*, 2019, pp. 129–137.

[34] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. Le Guernic, and R. De Simone, "The synchronous languages 12 years later," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 64–83, 2003.

[35] S. A. Edwards, "An esterel compiler for large control-dominated systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 2, pp. 169–183, 2002.

[36] F. Maraninchi and Y. Remond, "Mode-automata: About modes and states for reactive systems," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 1381, pp. 185–199, 1998. [Online]. Available: https://link.springer.com/chapter/10.1007/BFb0053571

[37] F. Jahanian, R. Lee, and A. K. Mok, "Semantics of modechart in real time logic," in *Proceedings of the Twenty-First Annual Hawaii International Conference on System Sciences. Volume II: Software track*, vol. 2. IEEE Computer Society, 1988, pp. 479–480.