FLARE — Monitoring for the Regulatory Requirements of a Drone Case Study

Sean Fenech[1][0009−0008−9960−3376], Christian Colombo[1][0000−0002−2844−5728], Gordon Pace[1][0000−0003−0743−6272], and Axel Curmi[1][0000−0002−9463−2070]

University of Malta, Msida MSD2080, Malta
{sean.fenech.23,christian.colombo,gordon.pace,axel.curmi}@um.edu.mt

Abstract. As regulatory requirements for software systems in the European Union continue to evolve, there is growing pressure to embed mechanisms into deployed systems that ensure both operational trustworthiness and legal accountability. Frameworks such as the AI Act and the Cyber Resilience Act introduce obligations related to cybersecurity, incident response, transparency, and auditability, particularly for high-risk and autonomous systems. These demands go beyond traditional verification and increasingly call for runtime components capable of monitoring behaviour, detecting non-compliance, and preserving forensic evidence.

In this work, we present FLARE, a runtime verification tool that combines automated monitoring with tamper-evident logging to support regulatory compliance. Building on existing runtime verification techniques, FLARE enables the construction of both a system harness that monitors live interactions with the environment and flags policy violations; and a forensic node, capable of recording verifiable logs. We demonstrate the application of FLARE on a waste-identification and localisation drone, a cyber-physical system subject to multiple legal and safety constraints. Our case study shows how FLARE can support legal and operational requirements while introducing minimal overhead, providing a practical path towards compliance-aware software instrumentation.

Keywords: runtime verification · regulatory compliance · forensic node.

1 Introduction

The regulatory demands for the transparency, security and auditability of digital systems — particularly those deployed in safety- or mission-critical domains — are becoming increasingly more stringent. Within the European Union, the *AI Act* and the *Cyber Resilience Act (CRA)* (amongst other technology regulation) highlight the growing expectation that deployed systems should not only behave correctly but should also be built to support investigation and accountability in the event of a failure or breach.

As part of this shift, we are seeing regulatory bodies starting to provide requirements for the technical architecture of trustworthy digital systems. One of the early adopters of such an approach is the *Malta Digital Innovation Authority (MDIA)* and the *Technology Assurance and Recognition Framework (TARF)* [23] with associated proposed guidelines for assessing AI systems [24]. For AI systems registered under the TARF, the proposed guidelines require two core architectural components that systems are required to include: (i) a *system harness*; and (ii) a *forensic node*. The system harness wraps around the main application logic to enable real-time monitoring of system-environment interactions, helping detect policy violations or anomalies during operation. The forensic node, on the other hand, serves as a tamper-evident event logger, storing runtime data securely to support audits or legal compliance checks after the fact. Together, these components form the backbone of systems that are not only functionally sound, but verifiably accountable.

In this paper we explore how runtime verification (RV) — a formal method typically used for checking temporal properties during system execution — can be adapted to support compliance with these regulatory requirements. RV has traditionally focused on monitoring and potentially intervening with the system-under-scrutiny in a manner that aligns closely with the system harness component requirements. By incorporating tamper-evident storage in its execution, RV can also provide a strategically-placed forensic node component.

In this paper, we introduce FLARE, a runtime verification tool that couples Larva (a Java-based RV tool) with SealFS, a file system offering tamper-evident logging. The result is a tool that not only checks compliance with runtime properties but also produces tamper-evident audit trails, allowing it to fulfil the roles of both a system harness and a forensic node.

To validate this approach, we apply FLARE to a *waste-identification and localisation drone* — a cyber-physical system that operates under a mix of safety, environmental, and data protection constraints. We demonstrate how FLARE can monitor and record key behaviours relevant to regulatory compliance, including geofencing, speed and altitude constraints, signal strength, and privacy concerns, using a risk-based approach to selective event logging.

The remainder of this paper is structured as follows: Section 2 gives an overview of the regulatory landscape, and Section 3 discusses the combination of the underlying technologies — Larva for runtime monitoring and SealFS for tamper-evident logging. Section 4 describes a waste detection and localisation project, detailing how the system harness and forensic node requirements were realised for this case study. Next, we present our empirical evaluation, including performance and overhead measurements in Section 5. Finally, we discuss related work and limitations in Section 6 and conclude in Section 7.

2 The Regulatory Landscape

The need for the regulation of digital technologies has been increasingly recognised in the past years. Before then, high-risk domains had their own domain-specific legislation to address risk of failure. For instance, in the case of medical devices, regulation going back to the 1990s was applicable to software components of such devices, which was then made more explicit and stringent in the 2010s e.g. the European Union's Medical Devices Regulation[1]. Regulation of software used in aviation can be traced back to the 1980s,[2] and was strengthened over time. In contrast, we are now starting to see legislation addressing digital solutions in general or for more specific technologies. At an EU level, one finds, for instance, the EU Cybersecurity Act [16], the NIS2 Directive [17], the EU AI Act [18], and more recently, the EU Cyber Resilience Act [19]. One can also find such regulation appearing at a national level, such as the *Innovative Technology Arrangement Act* set up in Malta [21, 14], covering blockchain and distributed ledger technologies but later widened to cover the whole class of critical systems.

The definition of AI in the EU AI Act is rather wide: *"machine-based system designed to operate with varying levels of autonomy and that may exhibit adaptiveness after deployment and that, for explicit or implicit objectives, infers, from the input it receives, how to generate outputs such as predictions, content, recommendations, or decisions that can influence physical or virtual environments"* [18][3] The aim of the regulation is that AI systems are safe and respect fundamental rights whilst promoting trustworthy innovation, and takes a risk-based approach to regulatory requirements: (i) *prohibited artificial intelligence practices* as identified in the Act include: systems using subliminal techniques, ones which distort the behaviour of persons on the basis of their age, disability or a specific social or economic situation, use of biometric data to categorise individuals, inferring personal information such as race, political opinions, etc., some uses of real-time remote biometric identification systems in publicly accessible spaces, and AI systems to infer emotions of a natural person in workplace areas; (ii) *high-risk AI systems* and their operators regulated under specific stringent requirements and obligations include: AI in critical infrastructure, education, essential private and public services, law enforcement; and (iii) *certain AI systems* such as chatbots and content-generation systems being subject to rules covering transparency, market placement, monitoring, surveillance and enforcement.

Various data-centric obligations arise from this new suite of regulations, including ones which address functional correctness and ones which address legal compliance. Given the importance of technical auditing and to enable replayability and availability of evidence in case of failures, the AI Act specifically requires logging capabilities for high-risk systems which are *"designed to ensure a level of traceability of the AI system's operation that is appropriate to the intended purpose of the system, the level of risk and the obligations laid down in this Regulation"* (see Article 19: Automatically Generated Logs[4] and Article 12: Record-Keeping[5]). Similar constraints arise in the Cyber Resilience Act (CRA): Annex I, Section 1(3d)[6], requires that, *"where ap-*

[1] Regulation (EU) 2017/745.

[2] For instance, ED-12 (EUROCAE) was a de facto standard for certifying airborne software required at a national level by a number of countries in Europe in the 1980s.

[3] In 2025, the European Commission has published more extensive guidelines on the definition of AI systems spanning over 13 pages [20].

[4] https://artificialintelligenceact.eu/article/19/

[5] https://artificialintelligenceact.eu/article/12/

[6] https://www.european-cyber-resilience-act.com/Cyber_Resilience_Act_Annex_1.html

plicable, products with digital elements shall protect the integrity of stored, transmitted or otherwise processed data, personal or other, commands, programs and configuration against any manipulation or modification not authorised by the user, as well as report on corruptions".

The control objectives to be used in conformity assessments for the EU AI Act are still under development. Thus, we will be referring to the AI regulatory guidelines set up in Malta in 2019 [24], where specific control objectives were proposed [26], allowing us to concretely explore the use of runtime verification and other formal methods in a regulatory context.

It is worth adding that the AI Act also entails the setting up of AI regulatory sandboxes to allow for *"the development of tools and infrastructure for testing, benchmarking, assessing and explaining dimensions of AI systems relevant for regulatory learning, such as accuracy, robustness and cybersecurity as well as measures to mitigate risks to fundamental rights, environment and the society at large"* [18]. Such a technical assurance sandbox was set up in Malta in 2020 [28], which also allows us to consider the impact of the use of our approach within that context.

The approach adopted in the regulatory guidelines developed in Malta is closely aligned with that of the EU AI Act, focusing on the trustworthiness of AI solutions through regulatory processes and obligations, as well as technical audits or assessments against control objectives. These cover a variety of aspects of the system under review, ranging from the processes in place (e.g. auditing of design and development processes) and technical assessment (e.g. functional correctness and measures mitigating data and bias risks including ones potentially arising post deployment), to appropriate support measures (e.g. appropriate measures in place to address risks such as cybersecurity and the handling of personal data, and compliance engines).

Of particular interest to this paper are control objectives covering components intended to support monitoring of the live system, including the *system harness* and the *forensic node*[7]:

System harness: The guidelines make it a requirement that an AI system should include an *ITA Harness*[8] [25]. The harness must surround the underlying system (see Fig. 1), providing a safety net (i) enabling the monitoring environmental interaction to ensure compliance with the documented expected behaviour; (ii) enabling identification of anomalies it detects e.g. out-of-bounds inputs and outputs. Note that the harness encompasses not only the core system (the dotted box and the bottom box), but also the data collection and processing engines (the top two boxes), as well as the training process (if applicable). This enables its use, not only to enable assessment of the system, but also to collect information and detect anomalies elsewhere e.g. detecting potential bias during the data collection phase. It is worth adding that the control objectives require the technical auditors to review that the implementation of the harness is as specified in the regulatory guidelines i.e. collects all relevant information and events faithfully.

Forensic node: The Forensic Node is intended to be a data repository to store all relevant events and information during the runtime of the AI-ITA in real-time, and in a secure tamper-proof manner. In conjunction with the system harness, this allows for offline assessment of the system's control effectiveness during audits and would support legal compliance investigations. Although the official guidelines architectural diagram (Fig. 1) does not show the forensic node, it is mentioned that it would typically feed off the system harness or be an integral part of it.

3 Runtime Verification and Trustable Logs

In the context of safety- and mission-critical software, runtime verification (RV) [10] has emerged as a compromise between full-state but non-scalable formal verification, and practical but non-exhaustive testing. It bridges the gap between formal requirements and the concrete implementation by automatically synthesising runtime monitor specifications into executable code.

As its history spanning two decades has shown[9], runtime verification has proved itself useful in securing a variety of software systems which could be classified as ITAs ranging from self-driving cars [38], IoT [37],

[7] A more thorough overview of the approach adopted by Malta can be found in [15].

[8] *Innovative Technology Arrangement* (ITA) term broadly refers to digital systems encompassed by the legislation, including AI systems deployed in critical areas.

[9] https://runtime-verification.github.io/events — international event series held annually since 2001.

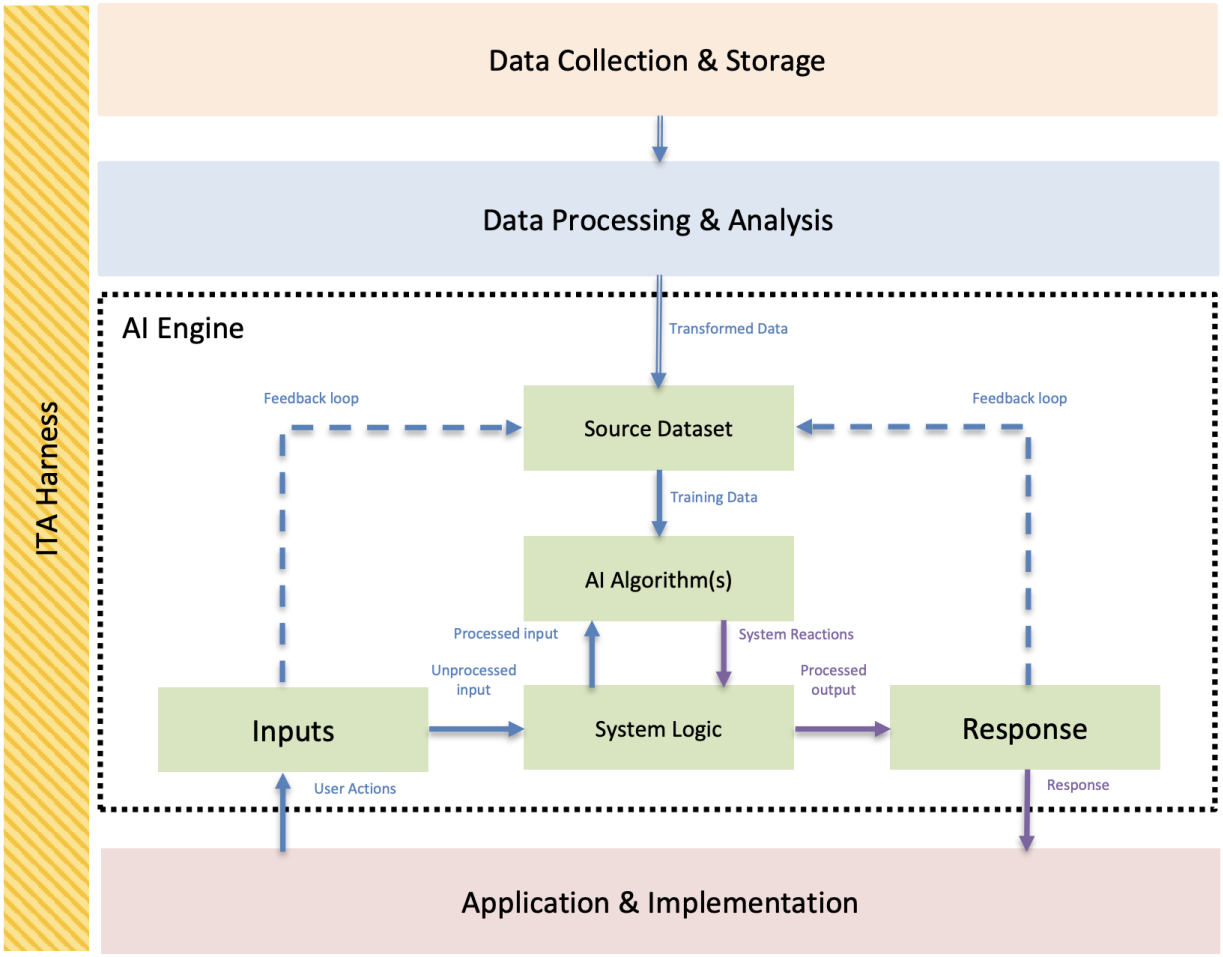

Fig. 1: ITA Harness around an AI system (taken from [25]).

robotics [35], smart contracts [2], and not least AI systems [6]. Inspired by these works, we turn to RV in the context of new regulatory challenges and adapt an existing tool for the job.

3.1 Larva

In this paper, we build on Larva [12, 8], a runtime verification tool which natively supports the monitoring of Java code through AspectJ[10] instrumentation. This is achieved by weaving additional monitoring code into the application at compile/load time, enabling the interception of method calls and other relevant events without modifying the source code directly. Behavioural correctness properties are specified using Larva scripts that capture a textual representation of DATE automata [11]. These symbolic automata communicate via channels, support timers, and may include verification-state variables. Listing 1.1 shows an example of a drone safety property, specifying that after 5 seconds or more of significant acceleration, the camera must never face away from the direction of acceleration (depicted in Fig. 2).

Line 2 declares any necessary variables, in this case consisting of a single timer object. Lines 3–9 identify the traced method calls that trigger state transitions and initialise timer objects. Lines 11–19 define the states of the automaton, identifying the starting, normal and bad ones. Lines 20–36 specify the state transitions, with each transition also qualified by an event, a guard condition, and the action performed (within `[]` and separated by \\). Intuitively, the automaton starts in state `LowAcc` and moves to `Acc` upon a high acceleration

[10] https://www.eclipse.org/aspectj/

```
GLOBAL {
  VARIABLES { Clock c; }
  EVENTS {
    HA() = {...} %% High Acceleration
    LA() = {...} %% Low Acceleration
    L() = {...} %% Looking
    LAw() = {...} %% Looking Away
    Clock() = {c@5}
  }
  PROPERTY cameraProperty {
    STATES {
      BAD { Acc5sLW }
        %% Low Acc and Looking Away
      NORMAL {
        Acc %% Accelerating
        Acc5s %% Accelerating for >5s
      }
      STARTING { LowAcc } %% Low Acceleration
    }
    TRANSITIONS {
      LowAcc -> Acc [HA\\c1.reset();]
      LowAcc -> LowAcc [L\!look\look=true;]
      LowAcc -> LowAcc [LAw\look\look=false;]

      Acc -> LowAcc [LA\\c1.off();]
      Acc -> Acc5s [Clock\look\]
      Acc -> Acc5sLW [Clock\!look\]
      Acc -> Acc [L\!look\look=true;]
      Acc -> Acc [LAw\look\look=false;]

      Acc5s -> LowAcc [LA\\]
      Acc5s -> Acc5sLW [LAw\\look=false;]

      Acc5sLW -> Acc5s [L\\look=true;]
      Acc5sLW -> LowAcc [LA\\]
    }
  }
}
```

Listing 1.1: Larva script for: *After 5 seconds or more of significant acceleration, the camera must never face away from the direction of acceleration.*

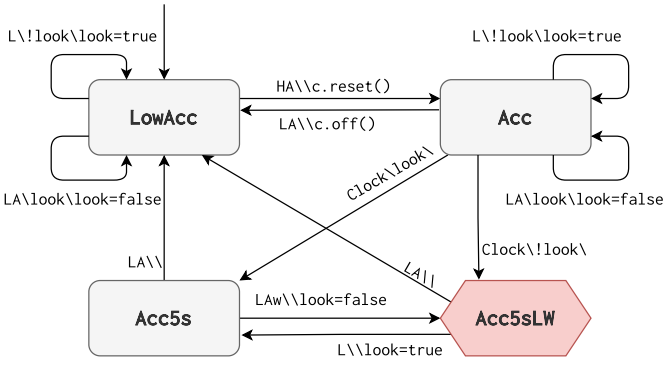

Fig. 2: Visual representation of the property.

event. The clock is reset so that it stores the length of time in which the drone has been accelerating. If 5 seconds elapse, the clock triggers the transition to the `Acc5s` state if its field of vision is aligned with the moving direction, or bad state `Acc5sLW` if otherwise.

Since DATEs allow unrestricted Java code as transition guards and actions, their expressivity is equivalent to that of Java. If used appropriately, DATE specifications should support modular design complementing the main system implementation, potentially sharing underlying modules. As such, DATEs can also express complex properties requiring statistical reasoning or learning-based anomaly detection.

Importantly, the choice of the events, which trigger automata transitions, should reflect points in the system execution where monitoring checks should be carried out. Selecting the right points of entry for the monitor is crucial to access the relevant data context for the property at hand, as well as to detect any violations as early as possible. Through AspectJ, compiled monitoring code is automatically injected into the system code, avoiding manual modification of the underlying system.

3.2 Tamper-Evident Logging

Runtime verification is tightly coupled with logging because it frequently operates directly on system logs [3] and the output it generates could also constitute important log material [22]. In this sense, RV can serve as a gatekeeper of the forensic node, acting as an event collector, filter, processor, and enhancer. This natural fit is the motivation for considering the integration of RV with tamper-evident logging.

At the core of tamper-evident logging lies the principle of forward integrity [4], which ensures that once a log entry is committed, it cannot be modified or deleted without detection —even if future keys or the system itself are compromised at any privilege level. This is typically achieved through a variety of cryptographic primitives such as hash chains [33]. Importantly, such systems must strike a balance between computational cost, hardware dependencies, and compatibility with existing infrastructure—particularly in resource-constrained and/or online environments.

SealFS [36, 29] is a stackable Linux file system, authenticating data as it is written to log files using a one-time-use keystream. By supporting append-only operations, SealFS ensures that once data is written, it cannot be altered without detection. A copy of the keystream is stored securely on an external system that is disconnected after setup. This is then used to verify the integrity of the logs when required, e.g. for forensic analysis purposes.

3.3 Runtime Verification with Tamper-Evident Logging

Building on the existing RV tool Larva, we have modified its open source compiler to have it automatically generate monitors whose output is committed to a tamper-evident file. We call this FLARE since it provides Forensic Logs for Auditing Runtime Events.

The tool's architecture, depicted in Fig. 3, closely follows that of Larva with the additional machinery to provide out-of-the-box tamper-evident logging. The process of using the tool is split into two stages: compilation and runtime execution, corresponding to the upper and lower parts of Fig. 3 respectively.[11] Note that in the diagram, black arrows represent the creation or passing of a file or process. Yellow arrows represent data being consumed and generated at runtime, and purple arrows represent procedures relevant to SealFS.

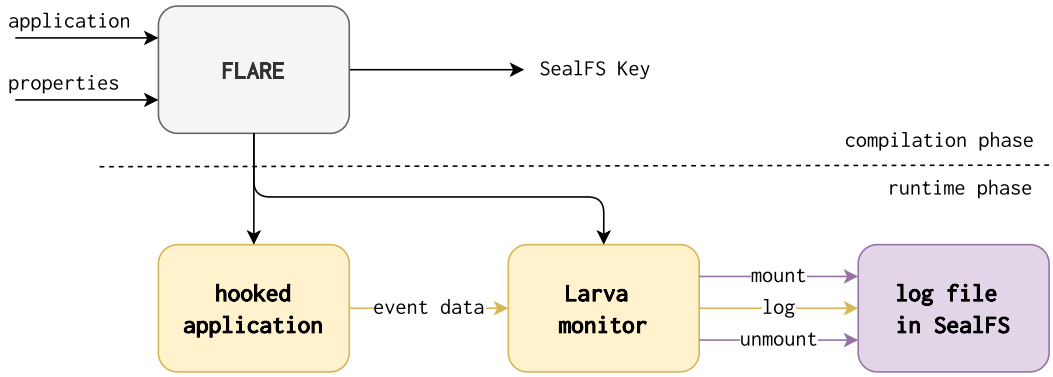

Fig. 3: FLARE combines Larva with SealFS.

Stage 1: Compilation. Starting off with the application to be monitored and the monitoring script (properties), the user uses the FLARE compiler to generate: (i) the hooked application, i.e., the application plus AspectJ files, which extracts events from the application, (ii) the monitor itself with abilities to setup tamper-evident logging via SealFS, and (iii) the cryptographic key used by SealFS to ensure tamper resistance. This is later also used to verify the logs.

[11] The tool and the script with the properties can be accessed from https://github.com/SeanFenech/FLARE.

Stage 2: Runtime Execution. Executing the FLARE-generated code, two things happen: (i) Upon initialisation, the monitor starts by mounting SealFS[12]. (ii) The hooked application automatically starts sending events to the monitor as they occur.

As the monitor starts receiving events, it starts processing them as happens with normal monitoring. In the background, a separate thread is launched which is responsible for writing the logs produced by FLARE to a file under the SealFS mount-point. This approach enables the log writing to be asynchronous to its processing, as the latter must keep up with incoming events from the target application or otherwise slow the application's progress. The log-writing thread can then take priority when events are less frequent or require less computational effort to process.

Upon application exit, the monitor unmounts SealFS and the logs remain available for the user to access and/or verify them whenever necessary.

3.4 Tamper-Evident Runtime Verification for Compliance

System Harness To satisfy the requirements set out in the regulatory guidelines, a system harness must *"safeguard the operation of the AI-ITA by actively checking that the user actions (inputs), system reactions (outputs), and any other behaviour is within the expected boundaries of the operation of the AI-ITA and any exceptions trigger the appropriate failure modes"*.

Larva, and by extension FLARE, can act as this harness since it provides:

Active checking As a runtime verification tool, FLARE runtime-monitors the system's state and events, including inputs, outputs, and its behaviour more generally. Specification of expected operational boundaries can be defined through formal properties. Sitting at a higher level of abstraction, such formal specifications are automatically compiled into code to minimise the possibility of bugs being introduced if implementing the checks by hand.

Appropriate automated triggering Specific reactions can be programmed alongside the specification to trigger system steering code in response to policy violations. This can include, but is not limited to, triggering failure modes as mentioned in the quoted document. Importantly, such reactions need to be timely, depending on the context.

Forensic Node To qualify as a forensic node under the regulatory framework, a component has to satisfy three conditions[13]:

Completeness *"All relevant events and data are recorded faithfully in real-time."*

A strict interpretation of completeness — recording all events — is rarely feasible, especially in constrained or resource-sensitive environments such as drones. In such cases, a risk-based approach could be necessary to select which event streams need to be logged and at which frequency.

Soundness Data is *"stored in a tamper-proof and accurate manner"*.

Tamper-evident storage is achieved through the use of SealFS, which provides strong cryptographic guarantees that logged data has not been altered. SealFS allows for verification of the integrity of log files — a core requirement in any forensic investigation.

Availability *"Processes are in place to ensure timely access to this information."*

Logs must be accessible and verifiable in a timely fashion, for example when needed for a forensic investigation. This aspect will be evaluated empirically later in the paper.

Having established FLARE as a valid candidate to meet regulatory requirements, the next section introduces the case study, explaining the risk elements that need to be handled by the assurance components.

[12] Note that root privileges are required for this operation. However, this still does not allow for the application to tamper with logs written to SealFS without invalidating them.

[13] See page 6 of the Guidelines [27]

4 Waste Detection and Localisation Case Study

Manual litter collection can be a costly process involving a number of steps: litter needs to be visually located (possibly from some distance) by having personnel searching for it within an area under consideration; after which, the litter needs to be precisely located (possibly requiring a vehicle), picked up, and sorted according to type (paper, plastic, glass, etc).

From such a traditional perspective that assumes no use of technology, efforts in litter collection are typically focused and optimised for urban contexts for a number of practical reasons emanating from the fact that rural areas are typically unstructured, variable, and spread over a relatively large area: (i) visual detection is easier when litter lies on a uniform surface, such as roads or pavements; (ii) infrastructure typically found in an urban context limits the areas where litter might be, thus providing a smaller search space; (iii) litter collection is harder when access to particular areas is limited, particularly for vehicles.

Novel and recent developments in AI provide an opportunity to address the challenges described above to carry out sustainable, efficient, and large-scale litter collection in rural areas. An ongoing project, Aerial Waste Identification and Geolocation System (AWIGS), aims to use aerial images captured from different altitudes to automate the search and geolocation process for litter in a vast area. Referring to Fig. 4, the drone sends telemetry data and footage to the controller which in turn displays the video feed to the human operator. The data is further forwarded to a cloud server from where it is processed (using image processing techniques from [32, 31]). Litter geolocation information is subsequently forwarded to human litter collectors. Even though the process currently relies on humans to collect litter, this setup still brings along various technical, safety, and regulatory risks and challenges.

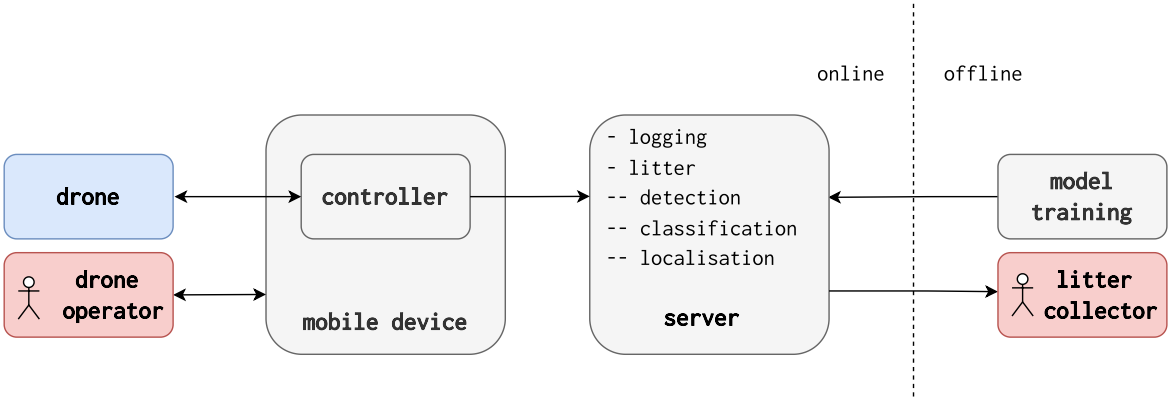

Fig. 4: The setup of the AWIGS project without monitoring.

In [13], the authors have identified a number of risks organised under the headings data, functionality, regulatory, and safety. Based on these risks, we have derived a number of properties to be monitored, constituting the system harness.

Regulatory risk: UAV regulations Following Maltese drone regulations, it must be ensured that the drone remains below the maximum height (60 meters) and that it remains within its predefined flight area (and outside the restricted zones). Furthermore, the drone's speed should not exceed a particular threshold, depending on the altitude — lower threshold at lower altitudes. Similarly, the threshold is lowered when the remote control (RC) signal is weak since this poses a greater risk as the operator may not have sufficient control over the drone.

Safety risk To mitigate safety risks, it must be ensured that the drone remains in range from the takeoff location, and that the signal strength remains above an acceptable threshold. The battery cell consistency and temperature should also be monitored to detect early warning signs of wear. Wind speed is a major factor in safe drone navigation. For this reason, this needs to be monitored, giving particular attention to high wind speeds or sudden large changes.

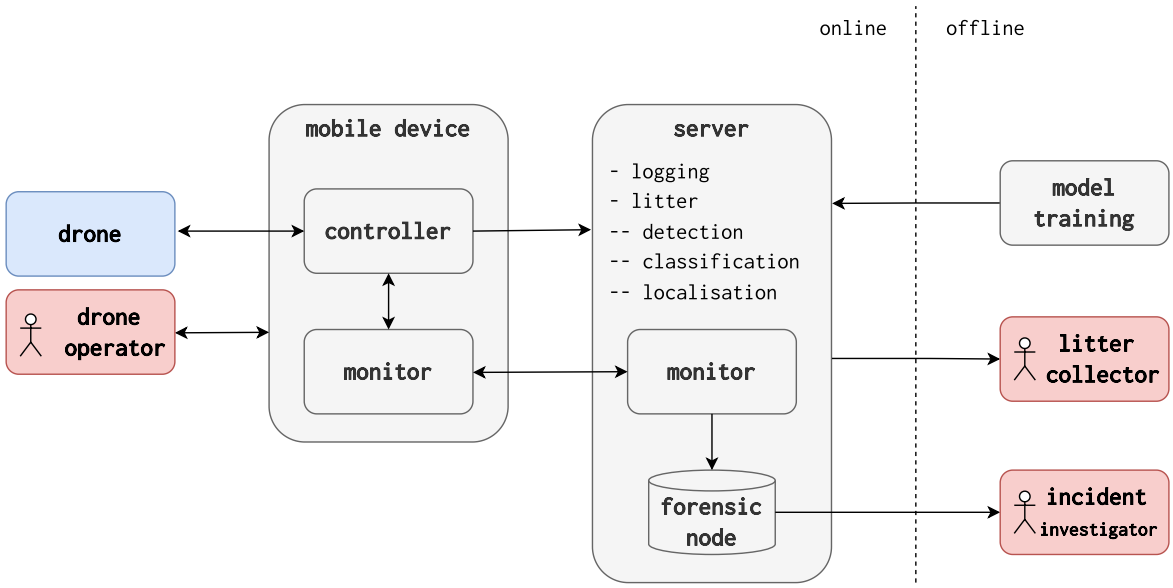

Fig. 5: The setup of the AWIGS project with monitoring components and the forensic node.

Data risk To ensure correct detection and classification of litter, even before training begins, it must be ensured that the training and evaluation datasets have sufficient variety and size. This would involve checking the overall dataset sizes and their distribution across different categories. Required data includes the dataset itself, the counts of objects per category, as well as the various environments (e.g., beach, forest, etc.) where the litter is found. This applies both to initial training and eventual re-training of the models.

Regulatory risk: Privacy concerns When the drone scans areas for litter, there is a significant probability of capturing identifiable data of individuals or vehicles within the drone's field of view. To avoid non-compliance to privacy regulations, such elements need to be detected and anonymised, for example through blurring/masking people and vehicles with solid colour boxes.

4.1 Monitoring Setup

Taking the above into consideration, we proceed to propose a modified AWIGS setup which incorporates the system harness and the forensic node. Given the time-sensitive nature of monitor feedback, the monitor should receive the data stream with minimal delay. For this reason, as shown in Fig. 5, we split the monitor into two parts: the monitor which handles the drone receives the feed directly from the controller, by-passing the server, and being able to give immediate feedback to the human drone operator. On the other hand, the monitor which handles AI components can operate on the server, dealing with more expensive, yet less time-sensitive checks. The two monitoring components would interact with each other: in one direction, from the mobile device to the server to log the monitor output in the forensic node. In the other direction, actionable insights from the server-side monitor can be sent back to the controller. For example, one could suggest that the operator takes the drone lower or revisits a particular area at a slower speed if the footage was not clear enough for a reliable classification.

As expected, safety and regulatory risks related to UAV regulations tend to be more time sensitive. What follows is a list of such properties that are ideally checked directly on the mobile device linked to the controller:

- High wind speed check
- Battery monitoring in relation to charge and health status
- GPS and remote control signal strength monitoring

- Speed check in relation to altitude and signal strength
- Position monitoring for inbound altitude, coordinates, and distance from the controller
- Ensuring that during consistent acceleration the drone's camera axis matches that of the drone acceleration direction.

FLARE code needed to monitor such properties is identical to that required by Larva as provided in the example in Section 3.1 (corresponding to the latter property in the list above).

While we could use FLARE to actively intervene (e.g. overriding the human operator in certain exceptional scenarios), in the current phase of the project, the system does not yet support direct automated feedback. Instead, the tool acts as a watchdog: upon detecting non-compliance, FLARE issues alerts to the human operator, who remains the only agent authorised to modify system behaviour. As noted in [25], ultimately the *"harness is not intended to be a replacement for human-level checks"*. Rather, it provides a safety net that detects and communicates anomalies, ensuring operational awareness and supporting legal defensibility.

Concerning data risk and privacy concerns, such properties are typically less clearcut to monitor. For example, the data risk can be kept under watch through proxy measures, e.g. dataset distribution throughout ongoing training updates. Tackling privacy concerns adds another layer of uncertainty; while the harness would be able to ensure that the algorithm used has a high accuracy score, it wouldn't be able to check with certainty that there are no privacy issues without using alternative models (which would themselves not be 100% theoretically accurate). These properties are not yet implemented but are planned to be added once the project reaches the corresponding development phase.

4.2 Forensic Node Data Collection

To avoid storing excessive amounts of data on the forensic node, a risk-based approach has been adopted, taking into consideration the properties being monitored and assessing both the likelihood and impact of potential system failures or legal breaches. Through this exercise, we have identified the set of events necessary to support a meaningful forensic investigation, should the need for one arise. For instance, in the context of operator control risk, critical data includes RC connection strength, speed, and altitude. Events related to these parameters are logged at higher frequencies, while lower-risk parameters (e.g. GPS signal strength above thresholds) are recorded less frequently (see Table 1). The balance between extra overhead and the quantity of data collected can vary greatly, even within different contexts of our use case. For our proof of concept we have chosen the frequencies of 1Hz and 0.1Hz based on the feedback we received from the domain experts. This balancing act ensures that the system remains responsive and efficient while still capturing essential evidence for post-incident analysis.

Event	**Condition**	**Frequency (Hz)**
Position update	alt $>$ 60m, in restricted area, $>$115m away from takeoff	1
	otherwise	0.1
Battery update	Low battery health, or charge $<$ 20%	1
	otherwise	0.1
	Takeoff/landing	F[14]
RC Signal update	Low RC signal strength	1
	otherwise	0.1
GPS Signal update	Low GPS signal strength	1
	otherwise	0.1

Table 1: Events and the frequency at which they are logged

[14] F is the frequency at which flight control signals are received by the controller — in our case $F = 10/s$.

The FLARE code snippet in Listing 1.2 shows events being received from the controller and their parameters being stored within the monitor (see lines 3–15). At second and ten-second intervals a number of data fields are logged (see lines 17–24). Similarly, at critical events such as takeoff and landing, data is logged as frequently as the system allows (see lines 26–27).

FLARE offers the possibility of encrypting the data stored in the logfile. However, this is optional as this is not a requirement of the forensic node. In our case study we have not used this feature.

```
1  ...
2  TRANSITIONS {
3      normal -> normal [dataAltitude\\
4          l_altitude = altitude;
5          lt_altitude = time;
6      ]
7      normal -> normal [dataGPS\\
8          l_satellites = satellites;
9          lt_satellites = time;
10         l_gpsLevel = gpsLevel;
11         lt_gpsLevel = time;
12     ]
13     normal -> normal [goodCellConsistency\\d_goodCellConsistency = true;]
14     normal -> normal [badCellConsistency\\d_goodCellConsistency = false;]
15     normal -> normal [veryBadCellConsistency\\d_goodCellConsistency = false;]
16     ...
17     normal -> normal [dataClock1\\
18         logData(d_location, d_rc, d_gps, d_battery);
19         dc1.reset();
20     ]
21     normal -> normal [dataClock10\\
22         logData(true, true, true, true);
23         dc10.reset();
24     ]
25     ...
26     normal -> normal [takeOff\\logData(true, false, false, false);]
27     normal -> normal [landing\\logData(true, false, false, false);]
28 }
```

Listing 1.2: Larva script for property: *To log the relevant data at the right frequencies.*

4.3 Lessons Learnt from Implementation

The main implementation process involved the encoding of the identified properties into DATEs. Once DATEs were formatted into a complete Larva script, these were ready for compilation by FLARE. From that stage, the compilation process automatically generates the necessary code. While there is a learning curve when using the tool for the first time, our experience has shown that this moderate. Perhaps what remains a significant challenge is that the testing process of the RV process itself is not typically automated as industrial Java systems. Indeed, the way the monitor integrates with the system through AspectJ and its output in the dedicated text file, makes automated testing cumbersome to set up. For this reason, testing was carried out manually. Beyond the actual implementation of the requirements, a significant step in this project was the capturing of the requirements themselves. As our previous experiences with runtime verification applications in industry have shown [9], bridging this gap comes with its own set of challenges. To facilitate communication, we organised several meetings with the domain experts to ensure that the requirements truly reflected the intended semantics. Such meetings were also crucial in understanding the risks involved and, subsequently, in establishing the right frequency of events to be processed and logged for each type (as expressed in Table 1). Crucially, we aimed to ensure that the risk appetite of the domain experts is correctly translated into the corresponding numbers.

We estimate the number of hours used by the runtime verification engineer (the main author of this paper) as listed in Table 2.

Table 2: Number of hours used by the runtime verification engineer.

Task	Number of Hours
Designing properties and communicating with domain experts	40
Expressing properties as DATEs and writing Larva script	25
Testing and debugging	20
Validation with domain experts and the resulting fine-tuning	15

5 Empirical Evaluation

Increased security always comes with a tradeoff. Given the additional cryptographic operations carried out by FLARE, one would expect an overhead penalty over the original tool Larva. Furthermore, since a major concern for a system harness is its ability to react in an appropriately timely manner, one would also need to assess whether this is adequate for drone monitoring. Secondly, since the monitoring output is being used as a forensic node, in line with the requirements, we need to ensure that data is accessible without undue delay.

5.1 Measuring Monitoring Latency

The aim of our experiment is to have a clear indication of the additional latency introduced by the new monitoring tool. Using Larva as a baseline, we analyse the performance of FLARE on actual drone flight data. For this reason we use a simulated telemetry generation process, triggering events with 100ms delay between each data record. The experiment consisted of processing 153,226 records which translates to a little over 4 hours (number of records multiplied by 100ms) worth of data from the drone. The experiment was run on a virtual machine running Ubuntu 22.04.5 LTS, with the 5.15.0-119-generic kernel, through VirtualBox 7.0.20 and Vagrant 2.4.1, given 4GB RAM and 4/8 CPUs. The virtual machine ran on an Intel i7-8565U CPU 1.80GHz, 4 Cores, 8 Logical Processors, with 8GB DDR4 RAM.

Given the 100ms delay between each data record received from the drone, the small additional processing time required by FLARE was not discernable, both tools took 4.26 hours (reflecting the time taken to simulate all the events) with FLARE taking 11ms longer. The reason for this small difference is that the 100ms delays allows enough time for FLARE to catch up after each event.

Therefore, to obtain a measurable difference between the monitoring tools, we also simulated the monitoring process without any delays. The results are shown in Table 3. As expected due to the cryptographic operations involved, FLARE took more time to process the data: a 38% increase. The data generated by the monitor increased by 15% namely due to the metadata added to the logfile.

To explore how the monitoring overhead increases as the data increases, we ran both versions on the data available twice. While this could vary depending on the properties monitored, in this case, the monitoring time and the data generated grew linearly (i.e., doubled). More precisely, the monitoring time for Larva increased by 57% while the monitoring time for FLARE increased by 93%. As expected, the data generated by the monitors doubled exactly, i.e., a 100% increase.

Table 3: Monitoring data and SealFS verification for different setups.

Setup	Larva	FLARE	Larva (×2)	FLARE (×2)
Total Monitoring Time (s)	43.41	59.89	68.24	115.30
Average Monitoring Time per Sample (μs)	283	391	223	376
Data generated (MB)	218.47	250.86	437.02	501.78
SealFS verification (s)	N/A	31.56	N/A	62.82

Since our main concern is the latency introduced through monitoring, we also divided the monitoring time by the number of samples processed. The result is in the order of hundreds of microseconds — leading

to an insignificant increase in latency. Of course one would expect these numbers to be bigger if monitoring is carried out on a mobile device, but even with an order of magnitude higher, latency would still be a modest one.

Finally, due to the forensic node requirement of timely access to the stored data, we consider this element for our implementation. Reading from our forensic node is not different from reading a file on a standard operating system. However, verifying the integrity of the data requires processing time as shown in Table 3. For example 500MB of data took a little over one minute to verify. To ensure that the verification time remains manageable (depending on the context), one could have separate log files, for example one log file for each hour of operation. In our case, using simple proportion, one hour of operation would require around 8 seconds to verify.

5.2 Discussion

The results presented above suggest that FLARE introduces manageable overheads in terms of performance and storage, even when integrated with cryptographic logging via SealFS. In our experiments, the system handled over four hours' worth of drone telemetry with negligible differences in the performance, demonstrating that compliance-aware monitoring and forensic capabilities can be introduced without undermining the responsiveness or reliability of real-time systems.

The linear relationship observed between data size and verification time also supports the practical viability of forensic node deployment, particularly when logs are segmented appropriately. In our case, 500MB of data could be verified in just over one minute, suggesting that the forensic verification process scales well and can meet the availability requirements defined by the regulatory guidelines.

That said, the current implementation is not without its limitations:

Key depletion. SealFS consumes a finite keystream for tamper-evident logging. Although sufficient for our case study, this could become a bottleneck in long-running or high-frequency logging scenarios. Future versions could integrate ratcheting-based key regeneration [29] to address this.

Crash resilience. If the system crashes between event occurrence and log writing, that event may be lost, and the monitor's state reset. Future iterations could explore persistent monitor state snapshots to support recovery.

Meta-event visibility. Events involving the monitoring infrastructure itself — e.g., system reboots, monitor initialisation, or failures — currently fall outside the visibility scope of FLARE. These may need to be captured via a complementary "meta-level" forensic mechanism to ensure completeness of the forensic narrative.

6 Related Work

Runtime verification allows the monitoring of system behaviour against formalised specifications. While the combination of RV with tamper-evident storage has been previously discussed [1, 7], to the best of our knowledge, FLARE is the first open-source tool that provides an integrated pipeline supporting both runtime monitoring and verifiable logging out-of-the-box.

That said, several other RV tools could, in principle, be extended to fulfil the system harness role described in regulatory frameworks. One notable example is JavaMOP [5], which supports the specification of monitoring properties using multiple logic formalisms. Crucially, JavaMOP also supports a corrective process, enabling not just passive observation but also the execution of predefined actions in response to violations — a capability also offered by Larva and required to instantiate an active system harness. With additional support for secure logging (e.g. via integration with SealFS or another cryptographically-verifiable backend), JavaMOP could similarly provide a sound basis for building compliance-aware systems.

Beyond SealFS, tamper-evident logging has been implemented in various contexts through a number of tools such as EmLog [34], Custos [30], and QED[15]. These tools are generally focused on ensuring the integrity of recorded data, but do not provide mechanisms for high-level behavioural monitoring or compliance

[15] https://github.com/BBVA/qed

reasoning. As such, they serve well as forensic storage layers but do not meet the broader objectives of a system harness as described in regulatory frameworks.

The combination of Larva and SealFS represents one possibility out of many others (e.g., those mentioned above). For the purposes of the proof of concept implementation presented in this work, the chosen tools ticked all the boxes.

7 Conclusions and Future Work

In this paper, we presented FLARE, a runtime verification tool designed to meet emerging regulatory requirements through tamper-evident logging. By integrating runtime verification with SealFS-based storage, FLARE offers a pragmatic solution for achieving compliance with frameworks such as the EU AI Act and the Cyber Resilience Act, particularly in the context of cyber-physical systems like drones. Our case study demonstrates that runtime verification, when combined with secure logging, can provide a lightweight yet robust layer of assurance, improving auditability, forensic capabilities, and regulatory alignment without imposing prohibitive overheads.

The experimental results show that the additional monitoring overhead introduced by FLARE remains manageable, and the approach scales linearly with the data generated. We also highlight important considerations for deploying forensic nodes in practice, including the need for risk-based data prioritisation and timely access to verified logs.

As the project evolves, we plan to implement the data-focused properties, particularly for the subsystems involving image processing, where balanced training data and privacy pose additional challenges. Looking ahead, we can also consider extending FLARE with code attestation capabilities to further strengthen trust guarantees. Furthermore, expanding the tool's validation across multiple case studies will help assess its generalisability to a broader range of AI-driven and safety-critical systems.

Ultimately, FLARE represents a step towards bridging the gap between formal methods research and the real-world demands of regulated digital technologies.

Acknowledgments. We gratefully acknowledge support from the Aerial Waste Identification and Geolocation System (AWIGS) project, funded by Xjenza Malta and led by Dylan Seychell, with key feedback from Matthias Bartolo and Gabriel Hili.

References

1. Abela, R., Colombo, C., Curmi, A., Fenech, M., Vella, M., Ferrando, A.: Runtime verification for trustworthy computing. In: Proceedings of the Third Workshop on Agents and Robots for reliable Engineered Autonomy, AREA@ECAI 2023, Krakow, Poland, 1st October 2023. EPTCS, vol. 391, pp. 49–62 (2023). https://doi.org/10.4204/EPTCS.391.7
2. Azzopardi, S., Ellul, J., Pace, G.J.: Monitoring smart contracts: Contractlarva and open challenges beyond. In: Runtime Verification (RV 2018). Lecture Notes in Computer Science, vol. 11237, pp. 113–137 (2018). https://doi.org/10.1007/978-3-030-03769-7_8
3. Barringer, H., Groce, A., Havelund, K., Smith, M.H.: An entry point for formal methods: Specification and analysis of event logs. In: Proc. Workshop on Formal Methods for Aerospace (FMA). Electronic Proceedings in Theoretical Computer Science, vol. 20, pp. 16–21 (2009). https://doi.org/10.4204/EPTCS.20.2
4. Bellare, M., Yee, B.S.: Forward integrity for secure audit logs. Tech. Rep. CS98-580, Department of Computer Science and Engineering, University of California at San Diego (November 1997)
5. Chen, F., Rosu, G.: Java-mop: A monitoring oriented programming environment for java. In: Halbwachs, N., Zuck, L.D. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3440, pp. 546–550. Springer (2005). https://doi.org/10.1007/978-3-540-31980-1_36
6. Cheng, C., Nührenberg, G., Yasuoka, H.: Runtime monitoring neuron activation patterns. In: Design, Automation and Test in Europe (DATE). pp. 300–303 (2019). https://doi.org/10.23919/DATE.2019.8714971
7. Colombo, C., Curmi, A., Abela, R.: Rvsec: Towards a comprehensive technology stack for secure deployment of software monitors. In: Proceedings of the 7th ACM International Workshop on Verification and Monitoring at Runtime Execution, VORTEX 2024, Vienna, Austria, 19 September 2024. pp. 13–18. ACM (2024). https://doi.org/10.1145/3679008.3685542

8. Colombo, C., Pace, G.J.: Runtime verification using LARVA. In: RV-CuBES 2017. An International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools, September 15, 2017, Seattle, WA, USA. Kalpa Publications in Computing, vol. 3, pp. 55–63. EasyChair (2017). https://doi.org/10.29007/N7TD
9. Colombo, C., Pace, G.J.: Industrial experiences with runtime verification of financial transaction systems: Lessons learnt and standing challenges. In: Bartocci, E., Falcone, Y. (eds.) Lectures on Runtime Verification - Introductory and Advanced Topics, Lecture Notes in Computer Science, vol. 10457, pp. 211–232. Springer (2018). https://doi.org/10.1007/978-3-319-75632-5_7
10. Colombo, C., Pace, G.J.: What is Runtime Verification, pp. 9–15. Springer International Publishing (2022). https://doi.org/10.1007/978-3-031-09268-8_2
11. Colombo, C., Pace, G.J., Schneider, G.: Dynamic event-based runtime monitoring of real-time and contextual properties. In: Cofer, D.D., Fantechi, A. (eds.) Formal Methods for Industrial Critical Systems, 13th International Workshop, FMICS 2008, L'Aquila, Italy, September 15-16, 2008, Revised Selected Papers. Lecture Notes in Computer Science, vol. 5596, pp. 135–149. Springer (2008). https://doi.org/10.1007/978-3-642-03240-0_13
12. Colombo, C., Pace, G.J., Schneider, G.: LARVA — safer monitoring of real-time java programs (tool paper). In: Seventh IEEE International Conference on Software Engineering and Formal Methods, SEFM 2009, Hanoi, Vietnam, 23-27 November 2009. pp. 33–37. IEEE Computer Society (2009). https://doi.org/10.1109/SEFM.2009.13
13. Colombo, C., Pace, G.J., Seychell, D.: Runtime verification and AI: addressing pragmatic regulatory challenges. In: Bridging the Gap Between AI and Reality - Second International Conference, AISoLA 2024, Crete, Greece, October 30 - November 3, 2024, Proceedings. Lecture Notes in Computer Science, vol. 15217, pp. 225–241. Springer (2024). https://doi.org/10.1007/978-3-031-75434-0_16
14. Ellul, J., Galea, J., Ganado, M., McCarthy, S., Pace, G.J.: Regulating blockchain, dlt and smart contracts: a technology regulator's perspective. ERA Forum **21**, 209–220 (2020). https://doi.org/10.1007/s12027-020-00617-7
15. Ellul, J., Pace, G.J., McCarthy, S., Sammut, T., Brockdorff, J., Scerri, M.: Regulating artificial intelligence: a technology regulator's perspective. In: ICAIL '21: Eighteenth International Conference for Artificial Intelligence and Law, São Paulo Brazil, June 21-25, 2021. pp. 190–194. ACM (2021). https://doi.org/10.1145/3462757.3466093
16. European Union: Regulation (EU) 2019/881 of the European Parliament and of the Council of 17 April 2019 on ENISA (the European Union Agency for Cybersecurity) and on information and communications technology cybersecurity certification and repealing Regulation (EU) (2019)
17. European Union: Directive (EU) 2022/2555 of the European Parliament and of the Council of 14 December 2022 on measures for a high common level of cybersecurity across the Union, amending Regulation (EU) (2022)
18. European Union: Proposal for a Regulation of the European Parliament and of the Council laying down harmonised rules on artificial intelligence (Artificial Intelligence Act) and amending certain Union legislative acts (2024)
19. European Union: Regulation (EU) 2024/2847 of the European Parliament and of the Council of 23 October 2024 on horizontal cybersecurity requirements for products with digital elements and amending Regulations (EU) (2024)
20. European Union: Approval of the content of the draft Communication from the Commission — Commission Guidelines on the definition of an artificial intelligence system established by Regulation (EU) 2024/1689 (AI Act) (2025)
21. Government of Malta: Innovative Technology Arrangements and Services Act (2018)
22. Havelund, K., Joshi, R.: Experience with rule-based analysis of spacecraft logs. In: Artho, C., Ölveczky, P.C. (eds.) Formal Techniques for Safety-Critical Systems - Third International Workshop, FTSCS 2014, Luxembourg, November 6-7, 2014. Revised Selected Papers. Communications in Computer and Information Science, vol. 476, pp. 1–16. Springer (2014). https://doi.org/10.1007/978-3-319-17581-2_1
23. Malta Digital Innovation Authority: Technology assessment recognition framework: Guidelines for applicants, assessors and other stakeholders, G-SPG-012 Rev. 1, 2023
24. Malta Digital Innovation Authority: AI ITA Guidelines (October 2019), https://mdia.gov.mt/app/uploads/2024/07/AI-ITA-Guidelines-03OCT19.pdf
25. Malta Digital Innovation Authority: AI ITA Nomenclature (October 2019), https://mdia.gov.mt/app/uploads/2024/07/AI-ITA-Nomenclature-03OCT19.pdf
26. Malta Digital Innovation Authority: AI System Auditor Control Objectives (October 2019), https://www.mdia.gov.mt/wp-content/uploads/2022/11/AI-ITA-SA-Control-Objectives-03OCT19.pdf
27. Malta Digital Innovation Authority: Forensic Node Guidelines (September 2019), https://mdia.gov.mt/app/uploads/2022/11/Forensic-Node-Guidelines.pdf
28. Malta Digital Innovation Authority: Technology Assurance Sandbox v2.0 Programme Guidelines (June 2020), https://www.mdia.gov.mt/wp-content/uploads/2022/11/MDIA-Technology-Assurance-Sandbox-TAS-Programme-Guidelines.pdf
29. Muzquiz, G.G., Soriano-Salvador, E.: Sealfsv2: combining storage-based and ratcheting for tamper-evident logging. Int. J. Inf. Sec. **22**(2), 447–466 (2023). https://doi.org/10.1007/S10207-022-00643-1

30. Paccagnella, R., Datta, P., Hassan, W.U., Bates, A., Fletcher, C.W., Miller, A., Tian, D.: Custos: Practical tamper-evident auditing of operating systems using trusted execution. In: 27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020. The Internet Society (2020). https://doi.org/10.14722/ndss.2020.24065
31. Pisani, D., Seychell, D., Schembri, M.: Detecting litter from aerial imagery using the SODA dataset. In: 2024 IEEE 22nd Mediterranean Electrotechnical Conference (MELECON) (2024). https://doi.org/10.1109/MELECON56669.2024.10608507
32. Schembri, M., Seychell, D.: Small object detection in highly variable backgrounds. In: 2019 IEEE 11th International Symposium on Image and Signal Processing and Analysis (ISPA). pp. 32–37 (2019). https://doi.org/10.1109/ISPA.2019.8868719
33. Schneier, B., Kelsey, J.: Secure audit logs to support computer forensics. ACM Transactions on Information and System Security **2**(2), 159–176 (May 1999). https://doi.org/10.1145/317087.317089
34. Shepherd, C., Akram, R.N., Markantonakis, K.: EmLog: Tamper-resistant system logging for constrained devices with TEEs. CoRR **abs/1712.03943** (2017). https://doi.org/10.48550/arXiv.1712.03943
35. Shivakumar, S., Torfah, H., Desai, A., Seshia, S.A.: Soter on ros: A run-time assurance framework on the robot operating system. In: Runtime Verification (RV 2020). Lecture Notes in Computer Science, vol. 12399, pp. 184–194 (2020). https://doi.org/10.1007/978-3-030-60508-7_10
36. Soriano-Salvador, E., Muzquiz, G.G.: Sealfs: Storage-based tamper-evident logging. Comput. Secur. **108**, 102325 (2021). https://doi.org/10.1016/J.COSE.2021.102325
37. Yahyazadeh, M., Hussain, S.R., Hoque, E., Chowdhury, O.: PatrIoT: Policy assisted resilient programmable IoT system. In: Runtime Verification (RV 2020). Lecture Notes in Computer Science, vol. 12399, pp. 151–171 (2020). https://doi.org/10.1007/978-3-030-60508-7_8
38. Zapridou, E., Bartocci, E., Katsaros, P.: Runtime verification of autonomous driving systems in carla. In: Runtime Verification (RV 2020). Lecture Notes in Computer Science, vol. 12399, pp. 172–183 (2020). https://doi.org/10.1007/978-3-030-60508-7_9