Form Methods Syst Des
DOI 10.1007/s10703-017-0275-x

R2U2: monitoring and diagnosis of security threats for unmanned aerial systems

Patrick Moosbrugger[1] · **Kristin Y. Rozier**[2] · **Johann Schumann**[3]

Abstract We present R2U2, a novel framework for runtime monitoring of security properties and diagnosing of security threats on-board Unmanned Aerial Systems (UAS). R2U2, implemented in FPGA hardware, is a real-time, Realizable, Responsive, Unobtrusive Unit for runtime system analysis, now including security threat detection. R2U2 is designed to continuously monitor inputs from on-board components such as the GPS, the ground control station, other sensor readings, actuator outputs, and flight software status. By simultaneously monitoring and performing statistical reasoning, attack patterns and post-attack discrepancies in the UAS behavior can be detected. R2U2 uses runtime observer pairs for Linear and Metric Temporal Logics for property monitoring and Bayesian networks for diagnosis of system health during runtime. We discuss the design and implementation that now enables R2U2 to handle security threats and present simulation results of several attack scenarios on the NASA DragonEye UAS.

Work supported in part by NASA ARMD 2014 I3AMT Seedling Phase I, NNX12AK33A; and NASA Autonomy Operating System (AOS) 8042-018286_NASA NNX14AN61A. A preliminary version of this work was reported by Johann Schumann, Patrick Moosbrugger, and Kristin Y. Rozier. "R2U2: Monitoring and Diagnosis of Security Threats for Unmanned Aerial Systems." In Proceedings of the 15th International Conference on Runtime Verification (RV15), Springer-Verlag, Vienna, Austria, September 22–25, 2015.

✉ Patrick Moosbrugger
moosbrugger@cps.tuwien.ac.at

Kristin Y. Rozier
KYRozier@iastate.edu

Johann Schumann
Johann.M.Schumann@nasa.gov

[1] Vienna University of Technology, Treitlstrasse 3, Vienna, Austria

[2] Iowa State University, Ames, IA, USA

[3] SGT, Inc., NASA Ames, Moffett Field, Moffett Field, CA, USA

Published online: 12 April 2017

Keywords Runtime monitoring · Metric temporal logic · Linear temporal logic · Bayesian networks · FPGA · Security · Unmanned aerial systems · GPS spoofing

1 Introduction

Unmanned Aerial Systems (UAS) will soon become ubiquitous. From toy quadcopters, to industrial aircraft for delivery, to utility craft for crop dusting, to police and fire fleets for public safety, to drones for military operations, UAS vary widely in terms of weight, size, and complexity. Although the hardware technology has significantly advanced in the past years, there are still considerable issues to be solved before this wide range of UAS can be shown to operate safely, particularly when autonomy is required. Perhaps the biggest concern is the integration of UAS into the National Air Space (NAS), where they have to seamlessly blend into the crowded skies and obey Air Traffic Control commands without endangering other aircraft or lives and property on the ground [6].

A related topic, which has been vastly neglected so far, is security [36]. All sensors and software set up to ensure UAS safety are useless if a malicious attack can cause the UAS to crash, be abducted, or cause severe damage or loss of life. Often, live video feeds from military UAS are not encrypted, so people on the ground, with only minimal and off-the-shelf components, could see the same images as the remote UAS operator [50]. In 2011, Iran allegedly abducted a CIA drone by jamming its command link and spoofing its GPS. Instead of returning to the CIA base, the UAS was directed to land in Iranian territory [8]. Even large research UAS worth millions of dollars are controlled via unencrypted RF connections; most UAS communicate over a large number of possible channels [14], relying on the assumption that "one would have to know the frequencies" to send and receive data.

There are multiple reasons for these gaping security holes: most UAS flight computers are extremely weak with respect to computing power. Thus, on-board encryption is not possible, especially for larger data volumes as produced, for example, by on-board cameras. Another reason is that a lot of UAS technology stems from the Hobby RC area, where security is of low concern. Finally, security aspects have only played a minor role in FAA regulation to date [11].

On a UAS, there are multiple attack surfaces: the communication link, sensor jamming or spoofing, exploitation of software-related issues, and physical attacks like catching a UAS in a net. While on-going work, e.g., the DARPA HACMs project [10], is demonstrating considerable success at designing secure UAS software systems that thwart insecure interactions by design, a UAS must necessarily sense and interact with its environment on some level to enable effective operation. Even with secure-by-design software running on-board, such interactions must be monitored and analyzed during runtime to ensure system-level correct operation. Therefore, in this paper, we focus on the detection of communication, sensor, and software-related security threats, but do not elaborate on attack prevention or possible mitigation strategies. Though design-time verification and validation activities can secure a number of attack surfaces, an actual attack will, most likely, happen while the UAS is in the air. We therefore propose the use of dynamic monitoring, threat detection, and security diagnosis.

In order to minimize impact on the flight software and the usually weak flight computer, R2U2 is implemented on dedicated FPGA hardware. This no-overhead implementation is designed to uphold the FAA requirements of REALIZABILITY and UNOBTRUSIVENESS.

Previously, we developed our on-board monitoring and diagnosis framework R2U2 for system health management of hardware-only components and developed implementations to detect hardware failures [12,42,46]. We defined and proved correct our FPGA temporal logic observer encodings [42] and our Bayesian network (BN) encodings [12], which comprise R2U2's underlying health model. We also envisioned a compositional building-block framework for integration with other diagnosis technologies that also analyzed software components [46]; in a preliminary version of this work [44] and in this paper, we follow up on that idea by providing the first implementation of R2U2 that includes software components, via monitoring software variable values passed over the system bus. Here we focus on new types of patterns related to on-board security whereas our previous work explored safety-related aspects of system health management. Specifically, we recognize that some off-nominal behaviors, when analyzed as individual events, may be either innocuous or indicative of a problem, but we can configure R2U2 to recognize patterns of possibly off-nominal events followed by some delayed behaviors that, taken as a whole, indicate a security problem.

Here, we extend R2U2 to enable the dynamic monitoring of the flight software via the the communication stream, along with sensor values, for indications of a malicious attack on the autopilot and, even more importantly, to be able to quickly and reliably detect post-attack behavior of the UAS. We add sophisticated signal processing blocks to the R2U2 framework, including, for example, moving average, or Fast Fourier Transformation, which can be applied directly to the incoming data stream of the monitored system. These filters are a major benefit towards bridging the gap between theoretical examples and real-world applications where signals are often very noisy. The temporal and probabilistic health models and their FPGA implementations are suited for fast detection and diagnosis of attacks and post-attack behavior. The separate FPGA implementation of a security extension to R2U2 described in this paper is highly resilient to attacks, being an isolated hardware entity and programmed using VHDL.

Our contributions include:

- extending R2U2 in a couple of ways for better enabling on-board, real-time detection of attack scenarios and post-attack behavior
  - extending R2U2 from monitoring of safety properties of hardware [12,42] to integrating hardware and software bus traffic monitoring for security threats
  - adding more sophisticated signal processing blocks to the R2U2 framework, such as FFT and moving average, that broaden its reasoning capabilities
- detection of *attack patterns* rather than component failures;
- ensuring monitoring and reasoning are isolated from in-flight attacks; our FPGA implementation provides a platform for secure and independent monitoring and diagnosis that is not re-programmable in-flight by attackers;
- demonstrating R2U2 via case studies on a real NASA DragonEye UAS; and
- implementing a novel extension of R2U2 that we release to enable others to reproduce and build upon our work: http://temporallogic.org/research/RV15.html.

The rest of this paper is structured as follows. Section 2 provides background information on our UAS platform, the open-source flight software, and the R2U2 framework. Section 3 is devoted to our approach of using temporal logic observers and BN diagnostic reasoning for detection of security threats and post-attack UAS behavior. In Sect. 4, we will illustrate our approach with several small case studies on attacks through the ground control station (GCS), attempts to hijack a UAS through an attacker GCS, and GPS spoofing. Finally, Sect. 5 discusses future work and concludes.

1.1 Related work

Existing fault diagnosis techniques can also be used to detect cyber attacks, since they can be regarded as faults in the system [23]. There exist a number of tools and approaches (both commercial and open) for fault detection, diagnosis, and recovery that are used in the aerospace industry. FACT[1] is a research project to develop modeling approaches and real-time, embeddable fault-diagnostic algorithms for system health management applications. They use temporal failure propagation graph models, whereas R2U2 uses temporal logic specifications in combination with probabilistic Bayesian reasoning. FACT has a "certifiability" constraint similar to our UNOBTRUSIVENESS property that states that a monitor should not create a need for re-certification of the system under test. SamIam[2] and Hugin Expert [3] Bayesian Networks are well known for fault detection, diagnosis, and decision making. They offer reasoning capabilities using probabilistic models. Due to their computationally intense algorithms they have not been used extensively for health management. In R2U2, however, we use the SamIam tool for the graphical modeling of our Bayesian Network models. In Runtime Verification with Particle Filtering [19] the authors use dynamic Bayesian networks to model the program, the program monitor, their interaction, and their observations. They provide a method to control the tradeoff between uncertainty and overhead. In R2U2, we currently do not consider gaps although they could occur due to unsuccessful transmissions from the system under test to the R2U2 monitors.

NASA projects uses three model-based diagnosis tools for on-board analysis of related system platforms. TEAMS-RT[4] is a real-time embedded diagnostics and reasoning tool based on models with specified cause-effect dependencies. It is targeted toward integrated health management (IHM) applications. TEAMS-RT is executed as a process on the target device, whereas R2U2 is executed on dedicated FPGA hardware in order to minimize overhead on the system under test. HyDE[5] is a model-based tool written in C++ to diagnose discrete faults in stochastic hybrid systems. Similar to R2U2, it uses probabilistic reasoning for diagnosis. Livingstone2[6] is a software system written in C++ for diagnosis and recovery that uses artificial intelligence. It analyzes the system's state and recommends commands or repair actions. Similar to R2U2, it uses an external GUI to support model creation. Livingstone2 provides a real-time interface to provide sensor readings to the monitoring system, which is similar to what we achieve using R2U2's preprocessing.

Software monitoring techniques can also be specialized for reasoning about real-time avionics. Copilot [37] is a domain-specific language tailored to generating C monitors in constant time and constant space based on a Haskell specification. It is designed to perform real-time monitoring on distributed safety-critical systems. Compared to R2U2, Copilot does not have reasoning capabilities, which we require in order to perform attack detection based on probabilistic models.

Several previous works have utilized on-board FPGAs as a performable platform for runtime verification of past-time speciications. BusMOP [30,35] uses a non-intrusive approach for runtime monitoring similar to R2U2 by means of passive instrumentation on a communication bus. The tool can automatically generate monitors that are executed on an FPGA,

[1] http://www.isis.vanderbilt.edu/projects/FACT.

[2] http://reasoning.cs.ucla.edu/samiam/.

[3] http://www.hugin.com/.

[4] https://www.teamqsi.com/products/teams-rt/.

[5] http://ti.arc.nasa.gov/tech/dash/diagnostics-and-prognostics/hyde-diagnostics/.

[6] http://ti.arc.nasa.gov/opensource/projects/livingstone2/.

thereby achieving zero runtime overhead. Compared to R2U2, it is limited to the past-time fragment of LTL but supports extended regular expressions as well. Heffernan et al. [13] present an on-chip, real-time runtime verification tool that supports past-time LTL specifications, whereas R2U2 also supports future-time specifications, which can be more natural to humans [26]. Similar to R2U2, they implement the monitor on a System-on-Chip FPGA. The system under test is executed on the same chip, whereas for R2U2 we use a separate hardware platform dedicated exclusively for monitoring. They follow a non-invasive approach where they do not require additional instrumentation. This is achieved by tapping into the memory bus of the system under test and extracting the events from the observed memory read and write transactions. Todman et al. [53] present an approach for in-circuit monitoring of designs implemented in reconfigurable hardware. Their monitors can be evaluated at the execution speed of the system under test. Their implementation is tailored to high-speed in-circuit applications where the system under test has to be executed on the same FPGA. This is not an option for a system like a UAS that consists of multiple hardware components and usually a flight computer running the control algorithms in software. Also, compared to R2U2, their monitors must be added to the design during compile time. However, they argue that they are working on a partly reconfigurable architecture, where they can change parts of the design, while others continue to execute. Their implementation supports past-time temporal logic specifications. Selyunin et al. [48] use IBM's brain-inspired TrueNorth spiking neuron model for runtime monitoring. The model is capable of performing a variety of deterministic or stochastic tasks, such as Boolean/arithmetic operations, filtering, or convolution similar to R2U2's signal processing blocks. The model's parameters can be configured to recognize MTL operators. They translate the monitors from a C++ description into synthesizable HDL code using High-Level Synthesis (HLS) from Xilinx. Compared to R2U2, changes in the specification require a re-synthesis of the FPGA implementation. Although they support future-time MTL specifications by translating them into equisatisfiable past-time formulas, their implementation violates our responsiveness requirement, since the evaluation of such a formula is delayed by the size of the formula.

P2V [24] is a PSL to Verilog compiler that generated monitors from assertions inserted into the target system. Any changes in these assertions require a re-synthesis and re-programming of the FPGA.

There are also several FPGA-based approaches that reason about signal temporal logic (STL) specifications. Jaksic et al. [16] present algorithms for generating hardware runtime monitors implemented in FPGAs. Unlike R2U2, changes of the specification require re-synthesizing the FPGA implementation. They use STL assertions that allow the specification of complex timing relations between digital and analog events. Compared to our MTL specifications, their STL specifications consider dense time. From an application point of view, the signals we intend to monitor are usually sampled to discrete time signals with an appropriate sampling frequency. Their STL models can consider real-valued signals, whereas our MTL monitors require Boolean input signals. However, using our preprocessing, we can formalize similar models. For example, instead of specifying intervals directly using STL specifications, our preprocessing can "Booleanize" intervals on real-valued signals using thresholds before we analyze them in our MTL specifications. Hence, assuming an appropriate sampling frequency, it is possible with R2U2 to specify similar models in many cases, as for example, the "stabilizability property" from [26]. In [49] the authors generate STL monitors from a SystemC implementation. Their monitors can either be used in high-level simulations, or synthesized to FPGA monitors, which can be deployed and run within the same FPGA as the design under test. Hence, their monitors are non-compliant with our unobtrusiveness requirement since they alter the system under test, potentially requiring a re-certification.

Similar to R2U2, their FPGA-based monitors can monitor operations in real time. However, their approach is focused on developing monitors that can be integrated in different phases of the electronic chip development cycle. In R2U2 we focus on monitoring complete systems like UAS, consisting of several subsystems. Donze et al. [9] present work related to our signal processing in the frequency domain. They contribute a specification formalism called time-frequency logic (TFL) for real-valued signal properties in the frequency domain of real-valued signals. To that end, they extend STL to include operators that calculate the Short-Time Fourier Transformed (STFT) coefficients for a parameterizable frequency of an arbitrary input signal. Since the STL semantics are defined relative to dense-time signals, they define their extension in the continuous domain. R2U2 follows a more practical approach where we compute a finite set of frequencies using the Fast Fourier Transform (FFT) algorithm. We do not define a new formalism, but use an arbitrary configuration specific to our tool. Their monitoring machinery, however, works in a similar way to ours: the raw input signals are preprocessed in order to obtain the spectral signals of interest. The resulting signals can then be used similarly to all other input signals in STL (or MTL in our case).

Some security-monitoring frameworks have also been adapted for running on-board UAS. TeStID [2], ORCHIDS [33], and MONID [32] are intrusion detection systems that use temporal logic to specify attack patterns. These security-monitoring frameworks are targeted at IT systems and infrastructure. Kim et al. [21] analyze the vulnerabilities of UAS to cyber attacks. Their work focuses on the identification of possible attack surfaces and an analysis of the post-attack behaviors. Similar to this paper, they identify, for example, malicious parameters on the control loops, denial of service attacks between the GCS and the UAS, as well as spoofing attacks on the sensors, as potential attack surfaces of a UAS. They conduct numerical analysis to study the post-attack behavior of the UAS. They propose a theoretic system architecture robust to cyber attacks, by adding additional layers of encryption and authentication, and by augmenting a cybersecurity monitoring component called a supervisor. The supervisor's role is to detect and isolate abnormal or malicious activity within the autopilot system. They rely on the assumption that the behavior of the UAS would change as a result of an attack in such a way that their supervisor can detect it. R2U2, on the other hand, tries to use patterns that correlate information from different subsystems in order to detect unknown attacks, similar to intrusion detection systems for IT infrastructures. Compared to R2U2, their architecture is a theoretic approach; they do not provide any information on an implementation. Kwon et al. [23] discuss security for cyber-physical systems against deception attacks. They propose an approach that can determine the vulnerability level of a CPS and derive conditions that allow an attack to deceive a monitoring system. Similar to our work, they consider both the sensor components and the control units of the CPS, and demonstrate the approach by simulating a deception attack on the navigation system of a UAS. Javaid et al. [17] also analyze cybersecurity threats for UAS. They simulate the effects of attacks that usually end in a crash, focusing on identifying different existing attack surfaces and vulnerabilities rather than focusing on runtime detection or post-attack analysis.

2 Background

Although our approach can be used for multiple types of UAS, for this paper, we consider a simple and small UAS platform, the NASA DragonEye (Fig. 1a). With a wingspan of 1.1m it is small, but shares many commonalities with larger and more complex UAS. Figure 1b shows a high-level, generic UAS architecture: the UAS is controlled by an on-board flight

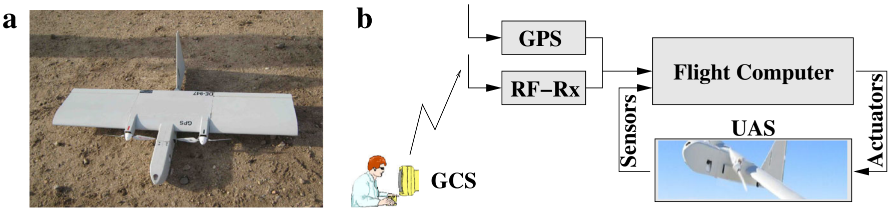


Fig. 1 **a** Photo of NASA DragonEye. **b** High level system architecture of a small UAS

computer running the flight software (FSW). It receives measurements from various sensors, like barometric pressure and airspeed, GPS, compass readings, and readings from the inertial measurement unit (IMU). Based upon this information and a flight plan, the FSW calculates the necessary adjustments of the actuators: elevator, rudder, ailerons, throttle. A ground control station (GCS) computer transmits commands and flight plans to the UAS, and receives and displays UAS telemetry information. This link, however, has a very low bandwidth and is not reliable. For fully autonomous missions, there is no link between the UAS and the GCS.

2.1 Flight software

Our example system uses the open-source FSW "APM:Plane" [3], which does not contain any security features like command or data encryption for the GCS-UAS link per default. This architecture allows us to easily carry out white-box experiments and to study the relationship between attacks and post-attack behavior.

Despite its simplicity, we selected this FSW because it very closely resembles the architecture of both similarly small and larger, more complex UAS. Even large commercial and military UAS do not have encrypted command links, or their encryption is rather weak and can easily be compromised [50]. Therefore, we assume in this paper that the command link is an attack surface.

2.2 R2U2

Developed to continuously monitor system and safety properties of a UAS in flight, our real-time R2U2 (Realizable, Responsive, and Unobtrusive Unit) has been implemented on an FPGA (Field Programmable Gate Array) [12]. Hierarchical and modular models within this framework are defined using Metric Temporal Logic (MTL) [22] and mission-time Linear Temporal Logic (LTL) [42] for expressing temporal properties and Bayesian networks (BN) for probabilistic and diagnostic reasoning. An extension of the framework to incorporate Prognostics Models is described in [45]. In the following, we give a high-level overview over the R2U2 framework and its FPGA implementation. For details on temporal reasoning, its implementation, and semantics the reader is referred to [12,42,47].

2.2.1 Temporal logic observers

LTL and MTL formulas consist of propositional variables, the logic operators $\wedge$, $\vee$, $\neg$, or $\rightarrow$, and temporal operators to express temporal relationships between events. For LTL formulas p, q, we have $\Box p$ (Always p), $\Diamond p$ (Eventually p), $\mathscr{X}\, p$ (NextTime p), $p\,\mathscr{U}\,q$ (p Until q), and $p\mathscr{R}q$ (p Releases q). Their formal definition and concise semantics is

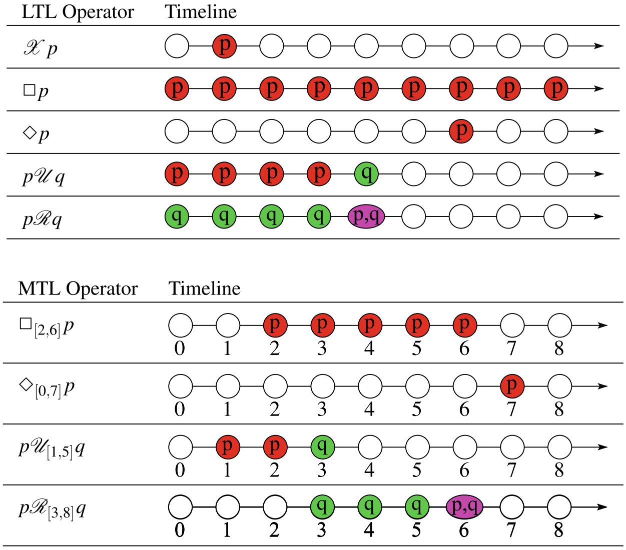


Fig. 2 Pictorial representation of LTL temporal operators and MTL operators

given in [42]. On an informal level, given Boolean variables p, q, the temporal operators have the following meaning (see also Fig. 2):

Always p ($\Box p$) means that p must be *true* at all times along the timeline.
Eventually p ($\Diamond p$) means that p must be *true* at some time, either now or in the future.
NextTime p ($\mathscr{X} p$) means that p must be *true* in the next time step; in this paper a time step is a tick of the system clock aboard the UAS.
p Until q ($p \mathscr{U} q$) signifies that either q is *true* now, at the current time, or else p is *true* now and p will remain *true* consistently until a future time when q must be *true*. Note that q must be *true* sometime; p cannot simply be *true* forever.
p Releases q ($p \mathscr{R} q$) signifies that either both p and q are *true* now or q is *true* now and remains *true* unless there comes a time in the future when p is also *true*, i.e., $p \wedge q$ is *true*. Note that in this case there is no requirement that p will ever become *true*; q could simply be *true* forever. The Release operator is often thought of as a "button push" operator: pushing button p triggers event $\neg q$.

For MTL, each of the temporal operators are accompanied by upper and lower time bounds that express the time period during which the operator must hold. Specifically, MTL includes the operators $\Box_{[i,j]}\, p$, $\Diamond_{[i,j]}\, p$, $p\, \mathscr{U}_{[i,j]}\, q$, and $p\, \mathscr{R}_{[i,j]}\, q$, where the temporal operator applies over the interval between time i and time j, inclusive (Fig. 2).

Additionally, we use a *mission bounded* variant of LTL [42] where these time bounds are implied to be the start and end of the mission of a UAS. Throughout this paper, time steps refer to ticks of the system clock. So, a time bound of [0, 7] would designate the time bound between 0 and 7 ticks of the system clock from now. Note that this bound is relative to "now" so that continuously monitoring a formula $\Diamond_{[0,7]}\, p$ would produce *true* at every time step t for which p holds anytime between 0 and 7 time steps after t, and *false* otherwise.

For the purpose of real-time dynamic monitoring, R2U2 provides two kinds of observers: asynchronous observers and synchronous observers. Each formula is encoded twice: once as an asynchronous observer and once as a synchronous observer. Asynchronous observers directly follow the LTL semantics and provide a precise valuation of the formula (True or False) as early as the result can be determined, together with valuation time, which is the time for which that result is known. An example output from an asynchronous observer is (True, 1). If the True/False valuation of the formula cannot be completed at the current time, the observer returns (␣, ␣) designating output is delayed until a later time. This output repeats until there is enough information to complete the valuation of the formula. For some specification φ for each clock tick an asynchronous observer may resolve φ for clock ticks prior to the current time n if the information required for this resolution was not available until n. For example, for the formula $\Box_{[2,6]}p$ in Fig. 2 the asynchronous observer would return, at time $t = 6$, the value True with valuation time $t = 0$.

In order to provide time-triggered real-time information about the currently-known valuation of a formula, we also utilize synchronous observers [42], which can have values of False, True, or Maybe, and which produce results at each time step. For our previous example $\Box_{[2,6]}p$, a synchronous observer would, for time $t = 1$, report False. For time steps between $t = 2$ and $t = 5$ it would report Maybe, because at that point in time, the formula cannot be decided yet, but there is still the possibility that it may become True. Pairing an asynchronous observer with a synchronous observer for observing a future-time property enables us to simultaneously be precise, enable intermediate decisions about formula valuations when we don't have complete information to determine whether it will pass or fail, and utilize the time-triggered synchronous observer results as event triggers for other system operations. (We do also allow past-time formulas; these are encoded once because formula valuation can always be completed immediately for the current time since we have already seen all of the data required to evaluate the formula.)

2.2.2 *Bayesian networks for health models*

In many situations, temporal logic monitoring might find several violations of security and safety properties. For example, a certain system state might have been caused by an attack or by a bad sensor; we can use the combination of property violations to determine the current situation. In order to be able to disambiguate the root causes, the R2U2 framework uses Bayesian networks (BN) for diagnostic reasoning. BNs are directed acyclic graphs, where each node represents a statistical variable. They are well-established in the area of diagnostic and health management (e.g., [29,34]). Conditional dependencies between the different statistical variables are represented by directed edges; local conditional probabilities are stored in the Conditional Probability Table (CPT) of each node [12,43,47]. R2U2 evaluates posterior probabilities, which reflect the most likely root causes at each time step. Note that we do not use dynamic BNs, which are capable of reasoning about time, because all temporal aspects are being dealt with by the efficient temporal observers described above. Our decision allows us to maintain a clear separation between temporal and probabilistic aspects in a health model and to use efficient algorithms for each of these aspects.

For our health models, we are using BNs of a general structure as shown in Fig. 3. Outputs of the temporal observers or discrete sensor signals are clamped to the "sensor" and "command" nodes of the Bayesian network as observable. In this paper, the nodes are shaded and their names start with "S_." Since sensors can fail, they have (unobservable) health nodes ("H_S" in Fig. 3) attached. As priors, these health nodes can contain information on how reliable the sensor component is, e.g., by using a Mean Time To Failure (MTTF) metric. In contrast to sensor nodes, command nodes "C" are used to convey modes or inputs, that are assumed to be *true*, to the BN.

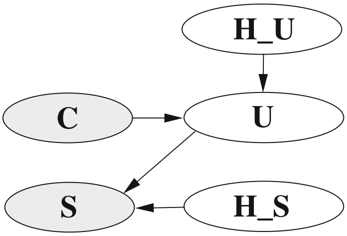


Fig. 3 Prototypical structure of a BN for health management

Unobservable nodes U may describe the behavior of the system or component as it is defined; they are influenced by the sensors or other behavior information. Often, such nodes are used to define a mode or state of the system. For example, it is likely that the UAS is climbing if the altimeter sensor says “altitude increasing.” Such (desired) behavior can also be affected by faults, so behavior nodes have health nodes attached. For modeling details see [43]. In our health management system, we, at each time step, calculate the posterior probabilities of the BN’s health nodes, given the sensor and command values $\mathbf{e}$ as evidence. The probability $Pr(H\_S = good|\mathbf{e})$ gives an indication of the status of the sensor or component. Reasoning in real-time avionics applications requires aligning resource consumption of diagnostic computations with tight resource bounds [31]. We are therefore using a representation of BNs that is based upon arithmetic circuits (AC). These are directed acyclic graphs where leaf nodes represent indicators λ and parameters θ while all other nodes represent addition and multiplication operators. Figure 4 shows the AC for our simple BN in Fig. 3. Posterior marginals are calculated by first executing a bottom-up pass with the λ indicators clamped to the evidence, followed by a top-down path over the AC. Except for a final division, only floating point additions and multiplications are used, making the reasoning efficient and resource-bounded. Thus, AC reasoning provides predictable real-time performance [7,29] because bounding the resources and operations required enables reliable predictions about the real-time performance of each such calculation.

2.3 FPGA implementation

R2U2 is implemented in FPGA hardware to provide an independent component for the monitoring of the UAS flight hardware (Fig. 1b). Figure 5a shows the high level architecture. Signals from the flight computer and communication buses are transmitted to the R2U2 monitoring device that continuously monitors these signals using signal processing, temporal logic observers and Bayesian networks. The current instantiation is implemented on an Adapteva Parallella Board [1], which is a credit-card sized, low-cost platform, featuring a Xilinx Zynq xc7z010 or xc7z020 FPGA (Fig. 5b). R2U2 can also utilize the platform’s ARM A9 processing system and a 16 or 64 core coprocessor if additional computation power is required. For instance, filters, or mathematical operations on signals can also be performed in software.

An onboard Ubuntu Linux installation facilitates interfacing, debugging, and logging of the monitoring data.

2.4 Data extraction from the UAS

The R2U2 approach uses passive instrumentation by tapping into the UAS’s communication buses similarly to BusMOP [30,35], Heffernan et al. [13], or Reinbacher et al. [41]. Most peripherals of the UAS are connected to the flight computer using a standard protocol over

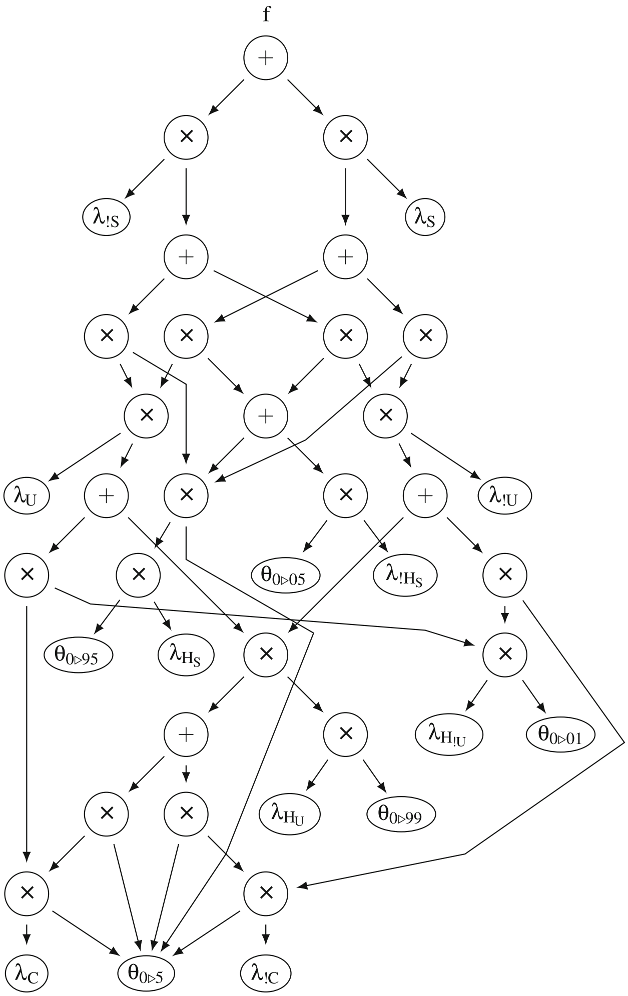

f
$\lambda_{!S}$
λ_S
λ_U
$\lambda_{!U}$
$\theta_{0\triangleright 05}$
$\lambda_{!H_S}$
$\theta_{0\triangleright 95}$
λ_{H_S}
$\lambda_{H_{!U}}$
$\theta_{0\triangleright 01}$
λ_{H_U}
$\theta_{0\triangleright 99}$
λ_C
$\theta_{0\triangleright 5}$
$\lambda_{!C}$

Fig. 4 BN representation as arithmetic circuit (from [12])

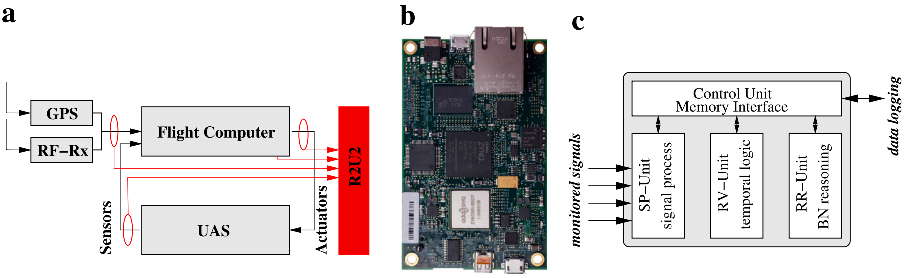
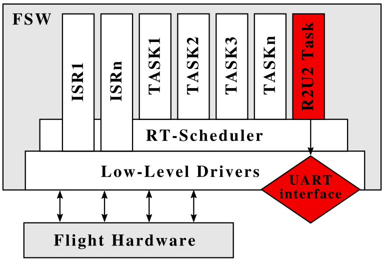


Fig. 5 **a** High-level architecture of R2U2. **b** Adapteva Parallella: our R2U2 hardware platform. **c** Main FPGA architecture

Fig. 6 Low overhead flight software instrumentation



some attached wires. For example, the GPS receiver is connected to the flight computer using a UART connection. Tapping into such an electrical signal can be achieved using an electrical decoupling technique by means of an opto-isolator circuit. Such a non-intrusive approach for generating data traces is required to avoid re-certification of the FSW. The Zynq System-on-Chip FPGA on the Parallella board offers various interfaces like, for example, I^{2}C, SPI, or a dual 12-bit on-chip analog-to-digital converter that support monitoring signals from different sources. Many of the test flights we performed for this paper were performed using software-in-the-loop simulations. Therefore, we did not monitor those signals electrically. Instead, we monitored internal FSW signals and forwarded them using a UART interface.

The NASA DragonEye FSW uses a scheduler that supports multiple tasks with priority assignments. Instrumentation of existing tasks or interrupt service routines (ISR) can add overhead, alter the tasks' timing behavior, and even lead to violations of their associated worst case execution times. Therefore, we create a dedicated R2U2 monitoring task with the lowest priority in order to avoid interference with the normal operation of the FSW (Fig. 6).

The R2U2 task collects the data from the FSW and transmits it via a non-blocking UART transmission to the R2U2 hardware platform. A CRC32 checksum is used for every transmission in order to ensure data integrity. The task does not perform any calculations since all necessary calculations are performed directly by the signal processing unit on the dedicated R2U2 hardware platform. Thus, the computation overhead on the flight computer caused by R2U2 is minimized.

Since the R2U2 task runs at the lowest priority, the transmission might be skipped or interrupted. Unsuccessful transmissions are reported at later R2U2 task executions. The R2U2 framework on the receiver side expects a periodic transmission from the FSW. Thus, delayed or missing transmissions are detected and logged. That way, the framework detects even a starvation of the R2U2 task. Since this instrumentation approach is tailored to low computation overhead it is currently not capable of determining the cause of such a problem. This would require a more intrusive approach as, for example, instrumenting the scheduler, or instrumenting higher priority tasks.

An alternative approach appears in [52], where the authors use Hidden Markov Models (HMM) to calculate the probability of a (non-probabilistic) temporal property being satisfied using an execution trace that contains gaps. A missing transmission can be seen as such a gap in the received event sequence. A statistical model of the monitored system can then be used to fill these gaps in order to calculate the probability of a property being satisfied. In [4] the authors take it a step further by combining the runtime verification with state estimation, and with overhead control. Monitoring events are enabled and disabled for each monitor as needed, in order to maintain the overhead at a specified target level. Based on probabilistic models they predict the likelihood of a property violation and allocate a larger fraction of the target overhead to the respective (critical) monitor instances in order to increase

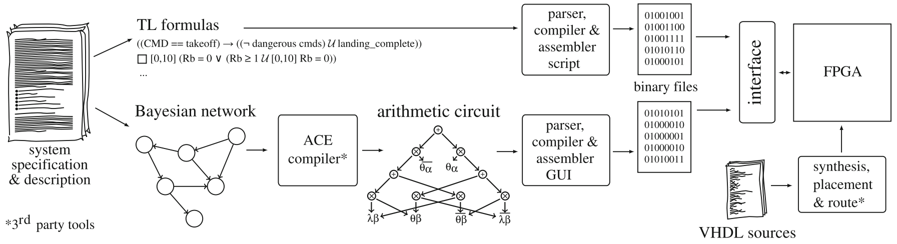

TL formulas
((CMD == takeoff) → ((¬ dangerous cmds) $\mathcal{U}$ landing_complete))
□ [0,10] (Rb = 0 ∨ (Rb ≥ 1 $\mathcal{U}$ [0,10] Rb = 0))
...
parser, compiler & assembler script
01001001
01001100
01001111
01010110
01000101
binary files
interface
FPGA
system specification & description
Bayesian network
ACE compiler*
arithmetic circuit
$\overline{\theta\alpha}$
$\theta\alpha$
$\lambda\beta$
$\theta\beta$
$\overline{\theta\beta}$
$\overline{\lambda\beta}$
parser, compiler & assembler GUI
01010101
01000010
01000001
01000010
01010011
synthesis, placement & route*
VHDL sources
*3rd party tools

Fig. 7 R2U2 software tool chain. Adopted from [12]

the probability of violation detection. However, we have designed R2U2 to be modular and extensible to a variety of avionics systems by reporting only the known valuations of our temporal specifications, separately from any probabilistic estimations. In this way, R2U2 is adaptable to aid different estimation algorithms for diagnostics and prognostics, e.g., with varying levels of conservatism.

2.5 Tool chain

Our system health models are constructed based upon information extracted from system requirements, sensor schematics, and specifications of expected behaviors, which are usually written in natural language. Details will be presented in the following sections. Our software tool chain takes temporal logic specifications, signal processing definitions and parameters, as well as the Bayesian network, compiles each part of the model suitable for FPGA and/or software execution, and produces a binary configuration file. Figure 7 shows the necessary steps for the processing of temporal and BN specifications.

2.6 Scalability

The Xilinx Zynq xc7z020 FPGA offers 53,200 LUTs and 106,400 registers. The Parallella board has a default resource consumption of 3,732 slice registers and 3,117 slice lookup tables due to the interface to an onboard Epiphany coprocessor. The additional resource consumption for the R2U2 design can be controlled by instantiating an arbitrary number of parallel hardware-based Bayesian network computing blocks (algorithm discussed in [40, 47]). We instantiate a single Runtime Verification (RV-Unit) within the FPGA, although this unit can also be parallelized as we discussed in [39]. The implementation in this paper uses 128 input signals, though this number could be extended for other implementations. The limits depend on the bandwidth of the UART transmission to the FPGA.

$$\text{data-bandwidth-required} = \text{number-of-inputs} \cdot \text{size-of-inputs} \cdot \text{sampling-frequency} \quad (1)$$

As can be seen in Eq. 1, the maximum number of inputs depends on the size of the inputs, the sampling frequency, and the bandwidth of the interface. For the case studies in this paper we use a bandwidth of 115.2 kbps and a sampling frequency of 5 Hz. Thus, the bandwidth enables to transmit a maximum of 2,090 Bytes (16,720 binary signals) within a single evaluation cycle (provided that a stop-bit, a start-bit, and a parity-bit is used).

Table 1 and Fig. 8 illustrate the maximum design operation frequency and the resource consumption within the FPGA in terms of the used slice registers and the slice lookup tables (LUT) depending on different numbers of parallel Bayesian computing blocks. These numbers are independent of the size and structure of the health models.

Table 1 R2U2 FPGA resource usage with hardware-based Bayesian reasoning

# Computing blocks	# Slice registers[a]	# Slice lookup tables[b]	Maximum design frequency [MHz]
0	6,971	7,240	100.583
1	12,076	10,009	86.022
2	14,287	11,187	87.009
3	16,469	12,564	86.438
4	20,829	15,288	86.994
10	31,737	21,971	86.133
20	53,559	34,381	–

[a] 106,400 slice registers are available on the Xilinx Zynq XC7Z020
[b] 53,200 slice lookup tables are available on the Xilinx Zynq XC7Z020

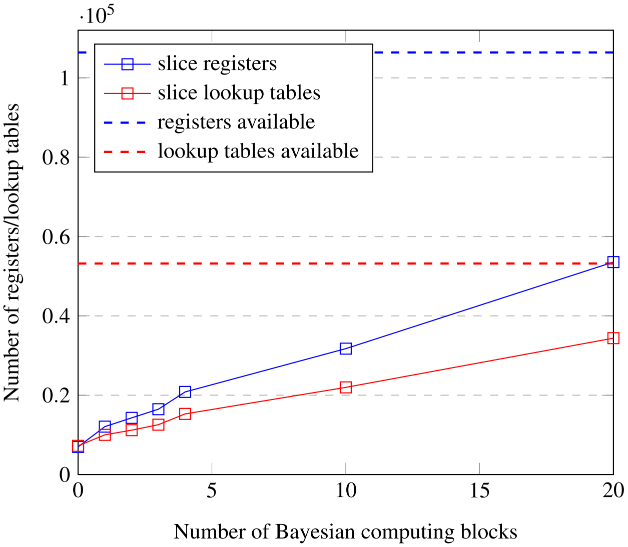


Fig. 8 R2U2 FPGA slice register/lookup table usage

Table 2 R2U2 FPGA resource usage with/without hardware-based atomic checker

Atomic checker	# Computing blocks	# Slice registers	# Slice lookup tables
Software based	1	12,076	10,009
FPGA based	1	43,504	45,046

A significant improvement in the FPGA resource consumption can be achieved by moving the hardware-based atomic proposition checkers into software. Table 2 compares the resource consumption of a design with software-based, and FPGA-based atomic checkers. Both designs again use 128 binary input signals and a single Bayesian computing block.

Although all R2U2 processing components are available in the FPGA configuration, equivalent software implementations of the signal processing unit (SP-unit) and the BN reasoning

(RR-Unit) have been used for the experiments in this paper. The R2U2 software implementations can be used for tasks where the ARM processing system offers sufficient computational power. This offers further scalability by reducing the number of required FPGA resources and therefore, enabling a higher number of parallel computation blocks within the FPGA. Besides, the software implementation facilitates fast model development and testing.

This hybrid software/FPGA architecture allows a customizable configuration depending on the given task. The maximum sampling frequency of the UAS depends on the following properties:

- Type and number of the preprocessing modules
- Complexity and number of the temporal logic observers
- Complexity of the Bayesian reasoning network
- For each of the above: the selected target for evaluation (e.g., FPGA, ARM-core, Epiphany coprocessor)
- For each of the above: the number of parallel computing block instantiations
- Maximum operation frequency of the execution target
- Available resources of the execution target (e.g., registers, memory)
- Current load of the execution target
- Interface bandwidth
- Number and size of the input signals

3 Our approach to threat detection

For our approach, we consider the "system" UAS (as depicted in Fig. 1b) to be a complex feedback system. Commands, GPS readings, and measurements of the sensors are processed by the FSW on the flight computer to calculate new values for the actuators, and to update its internal status. We assume that all signals to the flight computer (e.g., ground control station commands or GPS signals) are received by on-board RF receivers. In this paper, we assume that all malicious attacks are attempted via wireless communication during flight. [7]

With our R2U2 framework, we continuously monitor inputs from ground control and GPS and can identify many attack mechanisms and surfaces. Typical examples include denial-of-service, sending of illegal or dangerous commands, or jamming of the GPS receiver. Because, in most cases, information about the communication does not suffice to reliably identify an attack scenario, additional supporting information is necessary. This will be obtained from the analysis of post-attack behavior of the UAS. Any successful attack on the UAS will result in some unusual and undesired behavior of the UAS.

Monitoring the system inputs and analyzing the post-attack behavior are not independent from each other so we have to model their interactions within our R2U2 framework. Typically, a certain input pattern followed by a specific behavior characterizes an attack. For example, a strong oscillation in the aircraft movement that was triggered by an unusual GCS command indicates an attack (or an irresponsible pilot). Similarly, transients in GPS signals followed by subtle position movements could be telltales of a GPS spoofing attack. Figure 5 shows how our R2U2 framework monitors the various inputs going into the UAS system (GCS and GPS), as well as sensor/actuator signals and status of the flight software for post-attack analysis. We next consider modeling for attacks and post-attack behavior, loosely following [21].

[7] We do not model attack scenarios via compromised flight software.

3.1 Attack monitoring

As all attacks are initiated through the GCS or GPS inputs, we monitor the following attack surfaces. Because of zero-day attack mechanisms, this list will always be incomplete [5]. Note that the occurrence of such a situation does not mean that an actual attack is happening; other reasons like unusual flight conditions, transmission errors, or faulty hard- or software might be the reason.

Ill-formatted and illegal commands should not be processed by the FSW. Such commands could result from transmission errors or might be part of an attack. If such commands are received repeatedly a denial-of-service attack might be happening.

Dangerous commands are properly formatted but might cause severe problems or even a crash depending on the UAS mode. For example, a "reset-FSW" command sent to the UAS while in the air, will, most certainly, lead to a crash of the UAS because all communication and system parameters are lost. Thus, in all likelihood, this command indicates a malicious attack. Other dangerous commands are, for example, the setting of a gain in the control loops during flight. However, there are situations where such a command is perfectly legal and necessary.

Nonsensical or repeated navigation commands could point to a malicious attack. Although new navigation waypoints can be sent to a UAS during flight to update its mission, repeated sending of waypoints with identical coordinates, or weird/erroneous coordinates might indicate an attack.

Transients in GPS signals might be signs of GPS spoofing or jamming. Because the quality of UAS navigation strongly depends on the quality of the received GPS signals, sudden transients in the number of available satellites, or signal strength and noise ratios (Jamming-to-Noise Sensing [14]) might indicate a GPS-based attack.

It should be noted that these patterns do not provide enough evidence to reliably identify an attack. Only in correspondence with a matching post-attack behavior are we able to separate malicious attacks from unusual, but legal command sequences. We therefore also monitor UAS behavior.

3.2 System behavior monitoring

Our R2U2 models for monitoring post-attack behavior obtain their information from the UAS sensors, actuators, and the flight computer. In our current setting, we do not monitor those electrical signals directly, but obtain their values from the FSW. This simplification, however, prevents our current implementation from detecting a crash of the flight software initiated by a malicious attack. Nevertheless, since R2U2 is executed on a dedicated hardware platform, it can detect a reboot of the FSW, or an unusual delay of the monitored events that might be an indication of a crash of the FSW. With our R2U2 framework we are able to monitor the following UAS behaviors, which might (or might not be) the result of a malicious attack.

Oscillations of the aircraft around any of its axes hampers the aircraft's performance and can lead to disintegration of the plane and a subsequent crash. Pilot-induced oscillations (PIO) in commercial aircraft have caused severe accidents and loss of life. In a UAS such oscillations can be caused by issuing appropriate command sequences or by setting gains of the control loops to bad values. Oscillations of higher frequencies can cause damage due to vibration or can render on-board cameras inoperative.

Deviation from flight path: In the nominal case, a UAS flies from one waypoint to the next via a direct path. Sudden deviations from such a straight path could indicate some unplanned or possibly unwelcome maneuver. The same observation holds for sudden climbs or descents of the UAS.

Sensor Readings: Sudden changes of sensor readings or consistent drift in the same direction might also be part of a post-attack behavior. Here again, such behavior might have been caused by, for example, a failing sensor.

Unusual software behavior like memory leaks, increased number of real-time failures, or illegal numerical values can possibly point to an on-going malicious attack. In the case of software, such a behavior might be a post-attack behavior or the manifestation of the attack mechanism itself. Therefore, security models involving software health are the most complex ones.

3.3 R2U2 models

We capture the specific patterns for each of the attack and behavior observers with temporal logic and Bayesian networks. We also use these mechanisms to specify the temporal, causal, and probabilistic relationships between them. As a high-level explanation, an attack is detected if a behavioral pattern B is observed some time after a triggering attack A has been monitored. Temporal constraints ensure that these events are actually correlated. So, for example, we can express that an oscillation of the UAS ($osc = true$) occurs between 100 and 200 time steps after the control loop parameters have been altered ($param\_change = true$). The time step is determined by the sampling interval we use to sample data from the UAS, which is 200 milliseconds (5 Hz) in our experiments. Any trace satisfying the following formula could indicate an attack.

$$param\_change \wedge \Diamond_{[100,200]} osc$$

3.4 Modeling variants and patterns

The combination of signal processing, filtering, past-time and future-time MTL, and Bayesian reasoning provides a highly expressive medium for formulating security properties. Further generality can be achieved by grouping related indicators. For example, we can define groups of dangerous commands, unusual repeated commands, or events:

$$\begin{aligned}
&\textbf{dangerous\_cmds} &&= cmd\_reset \vee cmd\_calibrate\_sensor \vee cmd\_disarm \ldots \\
&\textbf{unusual\_cmds\_airborne} &&= cmd\_get\_params \vee set\_params \vee get\_waypoints \ldots \\
&\textbf{unusual\_cmds\_periodic} &&= cmd\_nav\_to \vee cmd\_mode\_change \vee invalid\_packet\_rcvd
\end{aligned}$$

This enables us to directly use these preprocessed groups in temporal formulas and feed them into a BN, thereby supporting simple reuse of common patterns and allowing us to create more comprehensive security models. The following example demonstrates how we use such patterns to specify that there shall be no dangerous commands between takeoff and landing.

$$\Box[(CMD == \text{takeoff}) \rightarrow ((\neg\, \textbf{dangerous\_cmds})\ \mathcal{U}\ landing\_complete)]$$

3.5 Bayesian networks for security diagnosis

Most models of attack monitoring and post-attack behavior are capable of indicating that there might have been an attack, but cannot reliably detect one as such, because the observed patterns could have been caused by a sensor failure, for example. However, we can use the Bayesian network (BN) reasoning engine of R2U2 to perform better probabilistic diagnosis.

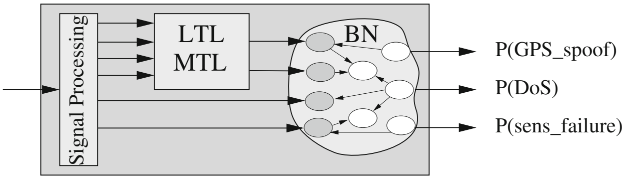


Fig. 9 Threat detection with R2U2 models. We combine signal preprocessing, temporal logic observers and Bayesian networks for our health and security models. The BN outputs the probability of an attack for a specific attack scenario or pattern based on the observed data stream from the UAS

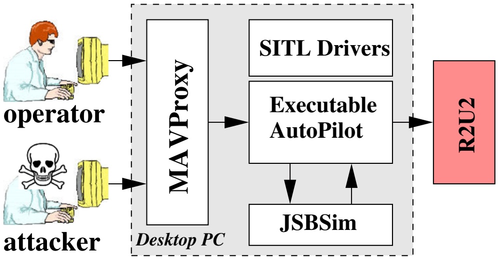
For details of Bayesian R2U2 models see [47]. The results of all the temporal observers are provided as inputs to the observable nodes (shaded in Fig. 9) of the BN. The internal structure of the BN then determines how likely a specific attack or failure scenario is. Prior information on how likely a certain monitor fires helps to disambiguate the diagnosis. For example, a sudden change in measured altitude could be attributed to a failing barometric altimeter, a failing laser altimeter, a failing GPS receiver, or a GPS spoofing attack. In order to determine the most likely root cause, additional information about recently received commands, or the signal strength of the GPS receiver can be used. So, transients in GPS signal strength with otherwise healthy sensors (i.e., measured barometric and laser altitude coincide) make an attack more likely. On the other hand, strongly diverging readings from the on-board altitude sensors make a sensor failure more likely. With prior information added to the BN, we can, for example, express that the laser altimeter is much more likely to fail than the barometric altimeter or the GPS sensor. Also, GPS transients might be more likely in areas with an overall low signal strength. Since a BN is capable of expressing, in a statistically correct way, the interrelationships of a multitude of different signals and outputs of temporal observers, R2U2 can provide best-possible attack diagnosis results as sketched in Fig. 9.

4 Experiments and results

Our experiments can be run either in a software-in-the-loop (SITL) simulation, the Ironbird (processor-in-the-loop) setup, or directly on the UAS.

In our SITL setup (Fig. 10), the FSW is executed on a Desktop PC. Also, the UAS flight behavior is simulated on the Desktop computer by the open source JSBSim [18] flight dynamics model. All hardware components are emulated by SITL low-level drivers, which enables us to inject the desired behavior without the risk of damaging the aircraft during a real test flight. The operator's GCS is connected to the simulated UAS via an open source

Fig. 10 Software-in-the-loop (SITL) test setup. The FSW and the flight dynamics of the UAS are simulated on a Desktop PC; the produced data traces are forwarded to our R2U2 framework running on an Adapteva Parallella Board



MAVLink proxy [27]. We also connect a second GCS to the proxy in order to simulate the attackers injected MAVLink packets.

Most of the experiments in this paper were executed in a software-simulation, or on our Ironbird processor-in-the-loop setup, which consists of the original UAS flight computer hardware components in a laboratory environment. In all configurations, the produced data traces were forwarded via a UART transmission to the R2U2 framework running on an Adapteva Parallella Board [1].

The health models and their parameters like, for example, thresholds, or timings for the following case studies were created manually based on the log files of our test flights. Hence, especially obtaining suitable values for the conditional probability tables for the Bayesian reasoning network was time consuming and posed a great challenge. Some of the values, especially those related to hardware components can be derived from their reliability according to their datasheet. Other values might be obtained using analysis of statistical data from previous flights. For the experiments in this paper, we mainly used educated guesses based on assumptions as, for example, possible attacker behaviors. For each case study we performed several manually designed real-time test flights that contained our attack scenarios. Also, the test flights contained scenarios where we tried to provoke false warnings by mimicking similar behavior caused by random or environmental conditions. We then tried to validate and refine our models and their parameters, mainly following a trial and error approach. In this paper we present the results of a single demonstrative test flight for each case study. We plan to improve this time consuming approach in the future by using some semi-automated or automated techniques based on machine learning. Besides, we plan to develop benchmarks for the models in order to evaluate their reliability regarding false positives and negatives.

4.1 Dangerous MAV commands

In addition to commands controlling the actual flight, the MAVLink protocol allows the user to remotely setup and configure the aircraft. In particular, parameters that control the feedback loops inside the FSW can be defined, as they need to be carefully adjusted to match the flight dynamics of the given aircraft. Such commands, which substantially alter the behavior of the UAS can, when given during flight, cause dangerous behavior of the UAS or a potential crash. In 2000, a pilot of a Predator UAS inadvertently sent a command “Program AV EEPROM” while the UAS was in the air. This caused all FSW parameters and information about communication frequencies to be erased on the UAS. Communication to the UAS could not be reestablished and the UAS crashed causing a total loss $3.7M [54]. If parameters for the FSW control loops are set to extreme values during flight, the aircraft can experience oscillations that could lead to disintegration of the UAS and subsequent crash. Therefore, such commands might be welcome targets for a malicious attack.

In this experiment, we set up our R2U2 to capture and report such dangerous behavior. Our security model consists of two parts: (a) detection that a potentially dangerous MAV command has been issued, and (b) that a dangerous behavior (in our case, oscillation around the pitch axis) occurs. Each of the parts seen individually does not give an indication of an attack: MAV commands to change parameters are perfectly legal in most circumstances. On the other hand, oscillations can be caused by turbulence, aircraft design, or the pilot (pilot-induced-oscillations). Only the right temporal combination of both pieces of information allows us to deduce that a malicious attack has occurred: after receiving the “set parameter” command, a substantial and persistent oscillation must follow soon thereafter. In our model we use the specification $\Box(\Diamond_{[0,2000]} C \wedge \Diamond_{[0,2000]}(\Box_{[0,300]} O))$, where O is the occurrence of

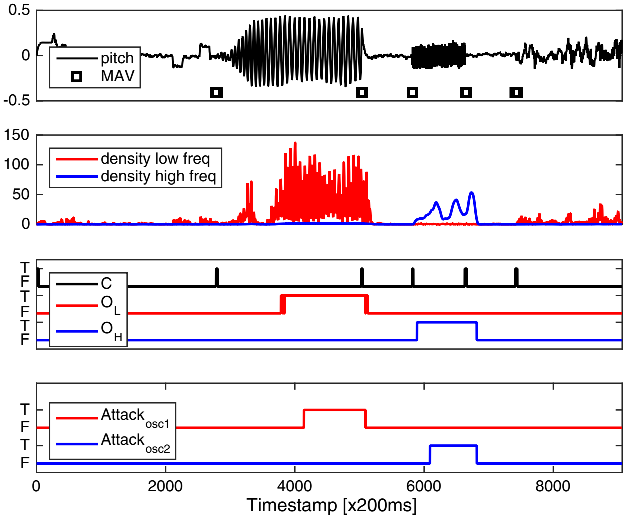


Fig. 11 UAS behavior after malicious setting of gain parameters

oscillations and C the event of receiving a "set parameter" command. We require that the oscillation persists for at least 300 time steps and is separated by the command by not more than 2000 time steps.

The event C can be directly extracted from the stream of received MAV commands; oscillations can be detected with the help of a Fast Fourier Transform (FFT) on the pitch, roll, or yaw angular values. Figure 11 shows how such an attack occurs. The top panel shows the UAS pitch as well as the points in time when a "set-parameter" command has been received (small boxes). Caused by a malicious command (setting pitch gain extremely high) issued at around $t = 2800$, a strong low-frequency up-down oscillation appears in the pitch axis. That excessive gain is turned off at around $t = 5100$ and the oscillation subsides. Shortly afterwards, at $t = 5900$, a malicious setting of a damping coefficient causes smaller oscillations but at a higher frequency. This oscillation ramps up much quicker and ends with resetting that parameter. In the second panel, two elements of the power spectrum obtained by an FFT transform of the pitch signals are shown. The signals, which have been subjected to a low-pass filter clearly indicate the occurrence of a low (red) and high (blue) frequency oscillation. The third panel shows the actual Boolean inputs for R2U2: "set-parameter received" C (black), "Low-frequency-oscillation" O_L (red), and "high-frequency-oscillation" O_H (blue). The bottom panel shows valuations of formulas $\Box(\Diamond_{[0,2000]} C \wedge \Diamond_{[0,2000]}(\Box_{[0,300]} O_L))$ and $\Box(\Diamond_{[0,500]} C \wedge \Diamond_{[0,500]}(\Box_{[0,200]} O_H))$ as produced by the R2U2 monitor. On the latter property the maximal lead time of the malicious attack has been set to only 500 time steps to reduce the number of false alarms, because the high-frequency oscillation ramps up almost immediately. We estimate that 10 person-hours were spent writing, debugging, and revising the two temporal logic properties used for this experiment and approximately 30 hours were spent on experimental setup and simulation.

4.2 DoS hijack attack

Attackers continuously find new ways to break into and compromise a system. Therefore, it is challenging to account for every possible attack scenario, since there can always be an

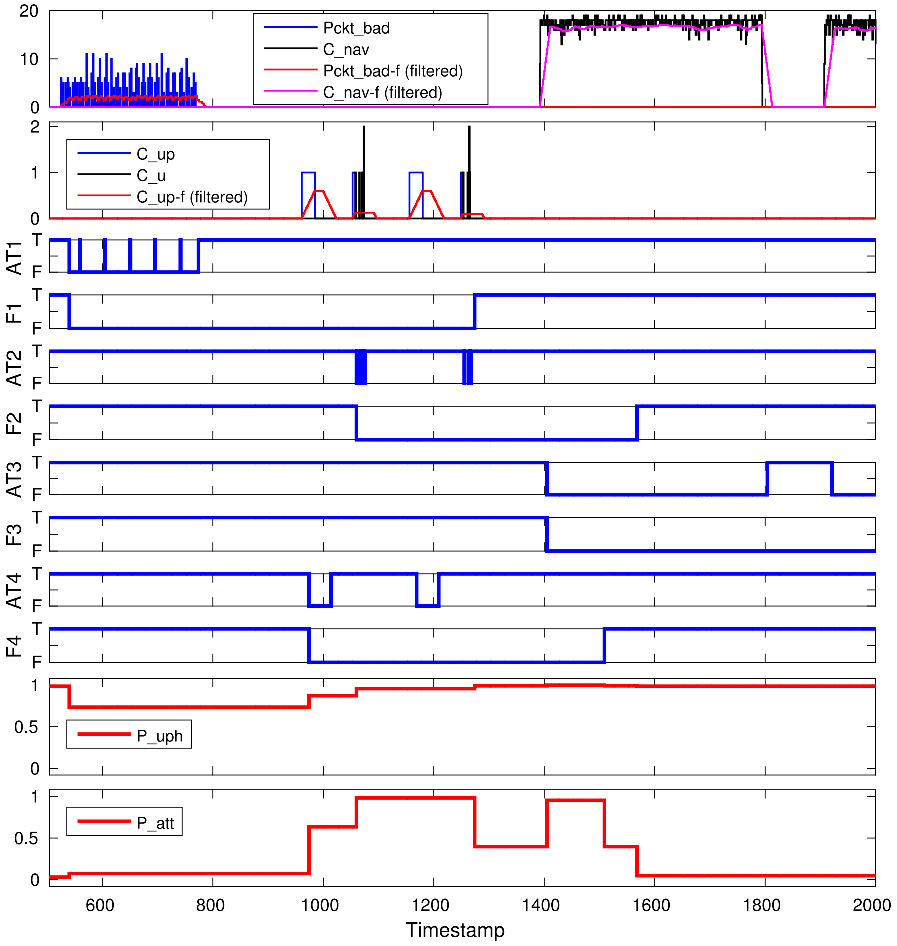


Fig. 12 Signal traces (*top panel*), valuations of temporal formulas (*middle panel*), and posterior probabilities of the health nodes (*bottom panel*). P_{uph} refers to the probability for the node *H_uplink* to be in state "uplink healthy." P_{att} refers to the probability for the node *H_attack* to be in the state "under attack"

unforeseen loophole. The following experiment shows how our R2U2 framework can detect an intrusion without the need of an explicit security model for each specific scenario. Here, we will look at possible indicators that can be grouped into patterns as described earlier.

In our simulation we initiate a sophisticated attack to hijack the UAS by first trying to establish a link from the attacker's GCS to the UAS. Because the attacker has to cope with issues like an incorrect channel, a different version of the protocol, or link encryption, a large number of bad command packets will be received within a short time frame.

The top panel of Fig. 12 shows such a typical situation with a high number of bad packets $Pckt\_bad$ (blue). Given the rate of bad packets R_b, we can detect such an occurrence of bad packets by means of temporal formulas like, for example:

$$F_{Bad\_Pckt\_Rate} \equiv \Box_{[0,10]}(R_b = 0 \vee (R_b \geq 1\ \mathcal{U}_{[0,10]} R_b = 0))$$

The formula $F_{Bad\_Pckt\_Rate}$ states that no more than one bad packet is received within each time interval of length 10 time steps.

Next, an attacker could try to gather information about the UAS, e.g., by requesting aircraft parameters or trying to download the waypoints using the MAVLink protocol.

This activity is shown in the second panel of Fig. 12 between time step 800 and 1300. The black spikes represent unusual or dangerous commands C_u, whereas the blue spikes represent the occurrence of commands that are unusual when called periodically C_{up}. This group of commands can be legal during setup or after landing, but might indicate an ongoing attack when received airborne. In order to detect this, we can use temporal formulas like, for example:

$$F_{Cu\_airborne} \equiv \Box((CMD == \text{takeoff}) \rightarrow \ (\neg\, C_u)\ \mathcal{U}\ landing\_complete)$$

Which states that no unusual command should be received after takeoff until the UAS has landed.

Finally, an attacker may flood the communication link in a way similar to a Denial of Service (DoS) attack by sending continuous requests to navigate to the attacker's coordinates, combined with requests $C_{homeloc}$ to set the home location of the UAS to the same coordinates. This phase of the attacks results in a continuously high number of navigation commands C_{nav} starting around $t = 1400$ as shown in the top panel of Fig. 12. Also, an attacker could flood the channel with random commands in order to disrupt the operator's attempts to regain control. For example, an attacker could repeatedly send commands to activate the UAS's autopilot mode in order to disrupt switching from autopilot to manual operator control, which would again result in a high number of unusual periodically called commands C_{up}.

The simulation showed that even if the attack was detected by the UAS operator, all attempts to change the UAS to its original course would immediately be overwritten by the attacker's high-rate navigation commands. Due to the altered home coordinates, any attempt of the UAS to return to the launch site would fail as well. Rather it would fly to the target location desired by the attacker.

4.2.1 Signal preprocessing

For attack detection, we start by defining the preprocessing for the input signals $Pckt_{bad}$, C_{nav}, and C_{up}. In this scenario, we filter the input signals by continuously calculating the discrete convolution of the input signal with a rectangular filter window of arbitrary length N, which is defined as $N = 10$ for signal $Pckt_{bad}$, $N = 20$ for signal C_{nav}, and $N = 40$ for signal C_{up}. Figure 12 shows the results $Pckt_{bad\text{-}f}$ and $C_{nav\text{-}f}$ in the top panel, and $C_{up\text{-}f}$ in the second panel. The smoothed signals can now be used in our temporal formulas, which in turn can simplify them. For example, the detection of a continuously high number of bad packets can be achieved by comparing the signal $Pckt_{bad\text{-}f}$ to an arbitrary threshold.

4.2.2 Temporal formulas

We specify several temporal formulas in order to construct our pattern. These formulas can either explicitly detect the occurrence of a particular command like, for example, C_u, monitor a particular signal like, for example, $Pckt_{bad\text{-}f}$, or monitor previously-defined groups like, for example, $C_{up\text{-}f}$, $C_{nav\text{-}f}$. We define the atomic propositions for the attack pattern detection model as follows:

$$
\begin{aligned}
AT_1 &\equiv Pckt_{bad\text{-}f} < \theta_1\\
AT_2 &\equiv C_u = 0\\
AT_3 &\equiv C_{nav\text{-}f} \leq \theta_3\\
AT_4 &\equiv C_{up\text{-}f} \leq \theta_4
\end{aligned}
$$

where $\theta_1 = 1.5, \theta_3 = 10$, and $\theta_4 = 0.3$.

Next, we define our temporal formulas in the form of:

$$F_n \equiv \Box_{[roi_{n-l}, roi_{n-u}]} AT_n \quad (n \in 1, 2, 3, 4)$$

where the time interval $[roi_{n-l}, roi_{n-u}]$ determines the region of interest (lower bound roi_{n-l}, upper bound roi_{n-u}) for an event to be relevant for the pattern. We chose this particular form of the temporal formulas for the purpose of demonstration, however, R2U2 models are not limited to this simple structure.

$$
\begin{aligned}
F_1 &\equiv \Box_{[0,500]} AT_1\\
F_2 &\equiv \Box_{[0,300]} AT_2\\
F_3 &\equiv \Box_{[0,300]} AT_3\\
F_4 &\equiv \Box_{[0,300]} AT_4
\end{aligned}
$$

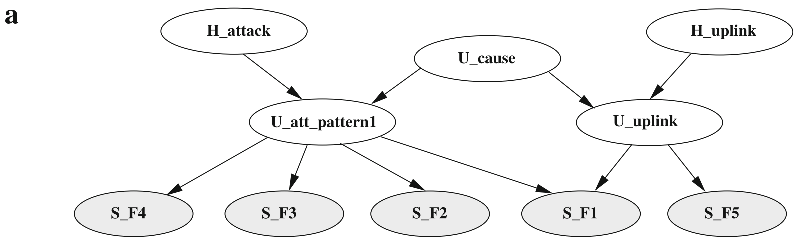
The formula F_1 states that the filtered number of bad packets (average over 10 timestamps) has never exceeded 1.5 within the last 500 timestamps. Formula F_2 states that there shall be no unusual or dangerous commands (previously defined group unusual_cmds_airborne) within the last 300 timestamps. Formula F_3 states that the filtered number of received navigation commands (average over 20 timestamps) has never exceeded 10 within the last 300 timestamps. Finally, formula F_4 states that the filtered number of commands from the previously defined group unusual_cmds_periodic (average over 40 timestamps) has never exceeded 10 within the last 300 timestamps.

The middle panel in Fig. 12 shows the results of the evaluation. The formulas F_1, F_2, F_3, and F_4 are not reliable indicators of an ongoing attack if viewed individually. Only by considering the overall pattern, we can calculate the probability for an ongoing attack. In order to accomplish this, we feed the results of these formulas into a Bayesian network for probabilistic reasoning.

4.2.3 Bayesian network for attack pattern detection

Our BN for the security model is shown in Fig. 13a, with its corresponding CPT (B). The BN models the status of the uplink communication subsystem (node *U_uplink*), the attack pattern (node *U_attack_pattern1*), and their correlation via node *U_cause*. We evaluate the posterior probabilities of the health nodes *H_uplink* to estimate the health of the communication subsystem, and *H_attack* to estimate the likelihood of an ongoing attack that matches the pattern. The sensor nodes S_F_n are mapped to the outputs of the corresponding temporal observers F_n that provide the evidence for the evaluation of the BN health nodes *H_uplink* and *H_attack*. The sensor node S_F_5 represents the state of the uplink signal power Sig_{pwr}. For this simulation, we did not include the signal power. Therefore, we set the node artificially to the state "signal OK." However, the effect of the node can be seen in Table 3, where we present an interpretation of different scenarios. The results of the health estimation are shown in the bottom panel of Fig. 12.

Initially, the probability for node *H_uplink* to be in state "uplink healthy" P_{uph} is around 0.99, whereas the probability for node *H_attack* to be in the state "under attack" P_{att} is roughly 0.03, given the evidence of the temporal logic observers. At around timestamp



b

S_F1:

U_uplink	uplink good		uplink bad	
U_att_pattern1	no pattern match	pattern match	no pat. match	pat. match
bad packets	0.03	0.8	0.5	1.0
no bad packets	0.97	0.2	0.5	0.0

U_att_pattern1:

U_cause	attack		uplink	
H_attack	OK	under attack	OK	under attack
no pat. match	1.0	0.0	1.0	1.0
pattern match	0.0	1.0	0.0	0.0

U_uplink:

H_uplink	uplink healthy		uplink unhealthy	
U_cause	attack	uplink	attack	uplink
uplink good	0.7	1.0	0.0	0.0
uplink bad	0.3	0.0	1.0	1.0

S_F2-4:

U_att_pattern1	no pattern match	pattern match
bad state detected	0.03	0.5
OK	0.97	0.5

S_F5:

U_uplink	uplink good	uplink bad
signal bad	0.03	0.5
signal OK	0.97	0.5

U_cause:

attack	0.5
uplink	0.5

Fig. 13 **a** Bayesian network for attack pattern detection. **b** Conditional probability tables for the BN. The prior for the health node H_attack is 0.5 (OK) and 0.5 (under attack), and for node H_uplink 0.7 (uplink healthy) and 0.3 (uplink unhealthy)

$t = 550$, F_1 indicates a high number of bad packets that causes a drop of P_{uph} to around 0.74, and a slight increase of P_{att} to 0.08. At around $t = 980$, F_4 indicates the occurrence of an unusual high number of commands C_{up} resulting in a sudden increase of P_{att} (0.63), while the health of the uplink P_{uph} also increases (0.87). This is due to the fact that the earlier detected high number of bad packets is likely to be related to the attack, rather then caused by a bad uplink. This effect is increased when F_2 detects the occurrence of dangerous commands C_u at around $t = 1060$. At this stage, an attack is almost certain ($P_{att} = 0.98$), while the uplink seems to be healthy ($P_{uph} = 0.96$). At around $t = 1280$, F_1 evaluates to *true* again. This is because more than 500 timestamps have elapsed since the last occurrence of bad packets. Thus, the bad packets detected in the beginning have left the region of interest for F_1 and are not considered to be related to the current events any more. This results in a drop of P_{att} to 0.4 and an increase of P_{uph} to 0.99. At $t = 1400$, F_3 indicates a high number of navigation commands C_{nav}. This causes P_{att} to increase again to 0.95, while the effect on P_{uph} is negligible. Once the events detected by F_4 and F_2 leave the region of interest, P_{att} decreases again ($t = 1510$ and $t = 1570$).

Table 3 Interpretation of different scenarios. (P_{att} and P_{uph} are rounded)

F_1	F_2	F_3	F_4	F_5	P_{att}	P_{uph}	interpretation
$Pckt_{bad}$	C_u	C_{nav}	C_{up}	Sig_{pwr}	%	%	
🟩	🟩	🟩	🟩	🟥	2	35	Uplink might be bad
🟩	🟩	🟩	🟩	🟩	3	99	Everything is ok
🟥	🟩	🟩	🟩	🟥	3	26	Uplink is bad
🟩	🟥	🟩	🟩	🟩	5	99	Dangerous airborne command detected
🟥	🟩	🟩	🟩	🟩	8	74	Random bad packets
🟩	🟥	🟥	🟩	🟩	40	99	Something is going on, might be an attack
🟥	🟥	🟩	🟩	🟩	63	87	Attention, UAS could be under attack
🟩	🟥	🟥	🟥	🟥	70	83	Attack is very likely! also, Sig_{pwr} is bad
🟩	🟥	🟥	🟥	🟩	95	99	Danger! UAS under attack!
🟥	🟥	🟩	🟥	🟩	98	96	Danger! UAS under attack!
🟥	🟥	🟥	🟥	🟩	100	96	Danger! UAS under attack!
🟥	🟥	🟥	🟥	🟥	100	43	UAS under attack and uplink might be bad!

The results show that our model is capable of detecting malicious attacks on a UAS by means of generic specified attack pattern online. Our R2U2 model correlates these attack patterns with the relevant UAS subsystems and their health models. This allows us to distinguish between random occurrences of attack indicators, subcomponent failures, or an actual attack in a quantitative fashion.

The simulation of this DoS attack scenario showed that besides crashing the UAS intentionally, there was no simple way for the UAS operator to prevent this kind of hijacking. In particular, for autonomous missions, where the UAS is flying outside the operator's communication range, it is essential that the UAS is capable of detecting such an attack autonomously. R2U2 enables such an attack detection in order to enable adequate countermeasures.

We estimate that 10 person-hours were spent writing, debugging, and revising the four temporal logic properties and the filters used for this experiment and approximately 25 hours were spent on experimental setup and simulation.

4.3 GPS spoofing

GPS plays a central role in the control of autonomous UAS. Typically, a flight plan for a UAS is defined as a list of waypoints, each giving a target specified by its longitude, latitude, and altitude. The FSW in the UAS then calculates a trajectory to reach the next waypoint in sequence. In order to accomplish this, the UAS needs to know its own position, which it obtains with the help of a GPS receiver. Due to limited accuracy, only GPS longitude and latitude are used for navigation; the UAS's current altitude is obtained using the barometric altimeter.

For the control of UAS attitude, the UAS is equipped with inertial sensors. Accelerometers measure current acceleration along each of the aircraft axes; gyros measure the angular velocity for each axis. Integration of these sensor values yields relative positions and velocities. These data streams are produced at a very fast rate and are independent from the outside interference but very noisy. Thus, the inertial sensors alone cannot be used for waypoint navigation. Therefore, the FSW uses an Extended Kalman Filter (EKF) to mix the inertial signals with the GPS position measurements. If the inertial measurements deviate too much from the GPS position, the filter is reset to the current GPS coordinates.

Several methods for attacking the GPS-based navigation of a UAS are known, including GPS jamming and GPS spoofing. In a jamming attack, the signals sent from the GPS satellites are drowned out by a powerful RF transmitter sending white noise. The UAS then cannot receive any useful GPS signals anymore and its navigation must rely on compass and dead reckoning. Such an attack can cause a UAS to miss its target or to crash. A more sophisticated attack involves GPS spoofing. In such a scenario, an attacker gradually overpowers actual GPS signals with counterfeit signals that have been altered to cause the UAS to incorrectly estimate its current position. That way, the UAS can be directed into a different flight path.

This type of attack became widely known when Iran allegedly used GPS spoofing to hijack a CIA drone and forced it to land on an Iranian airfield rather than its base [8,51]. Subsequently, researchers from the University of Texas at Austin successfully demonstrated how a $80M yacht at sea, [8] as well as a small UAS can be directed to follow a certain pattern due to GPS spoofing [20]. Because civil GPS signals are not encrypted it is always possible to launch a GPS spoofing attack. For such an attack, only a computer and a commercially available GPS transmitter is necessary.

We can employ our R2U2 framework to detect realistic GPS spoofing attacks like the common attack scenarios described in [20]; whereas that paper discusses attack detection in theory we demonstrate it via hardware-in-the-loop simulation on-board our IronBird UAS. Here we focus on attack detection; techniques to avoid or mitigate GPS spoofing are beyond the scope of this paper.

Our developed R2U2 model monitors the GPS and the inertial navigation information online. For our experimental evaluation, we defined a UAS mission that flies, at a fixed altitude, toward the next waypoint south-south-west of the current UAS location as shown in Fig. 14 (blue). When spoofing occurs, the attacker modifies the GPS signal in such a way that it tricks the UAS into believing it is still flying a direct route as expected. In reality, however, the UAS is actually veering off to reach a target point defined by the attacker (red). A severe and increasing discrepancy can be observed as the effect of the attack.

As the actual position (ground truth) is not available to the on-board FSW, R2U2 reasons about relationships with alternate signals that convey similar information. The inertial navigation unit produces an error or offset signal that reflects the deviation between the current position observed by GPS and the inertial sensors. The two top panels of Fig. 15c show these offset signal for North (EKF_N) and East (EKF_E) orientation during this mission. These offset signals can become substantially large during the actual spoofing period, when the GPS locations are gradually moved to the attacker's target (starting at timestamp $t = 2000$). The center panels show the output stream of the four temporal logic observers:

$$\begin{aligned} F_{North1} &\equiv |EKF_N| \geq \theta_{N1} \\ F_{North2} &\equiv |EKF_N| \geq \theta_{N2} \\ F_{East1} &\equiv |EKF_N| \geq \theta_{E1} \\ F_{East2} &\equiv |EKF_N| \geq \theta_{E2} \end{aligned}$$

where θ_{N1}, θ_{N2}, θ_{E1}, and θ_{E2} are arbitrary spoof detection thresholds (dashed).

Small offsets might be observed more frequently due to environmental conditions. Hence, we use two thresholds for each signal to add robustness to our model in the sense, that a larger offset represents a more unusual (bad) state than a smaller offset. This can help to reduce the number of false warnings, if we assume that a higher deviation represents a higher likelihood of an attack.

The absolute amount $|EKF_{\{N,E\}}|$ of the values $EKF_{\{N,E\}}$ are evaluated by R2U2's preprocessing.

[8] http://www.ae.utexas.edu/news/623-humphreys-research-group.

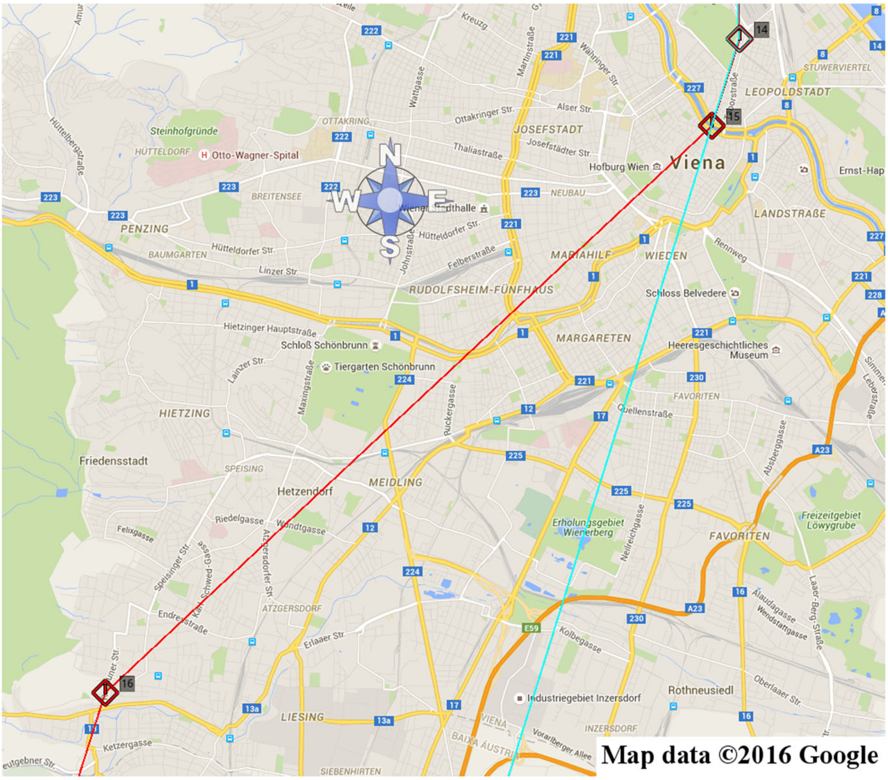
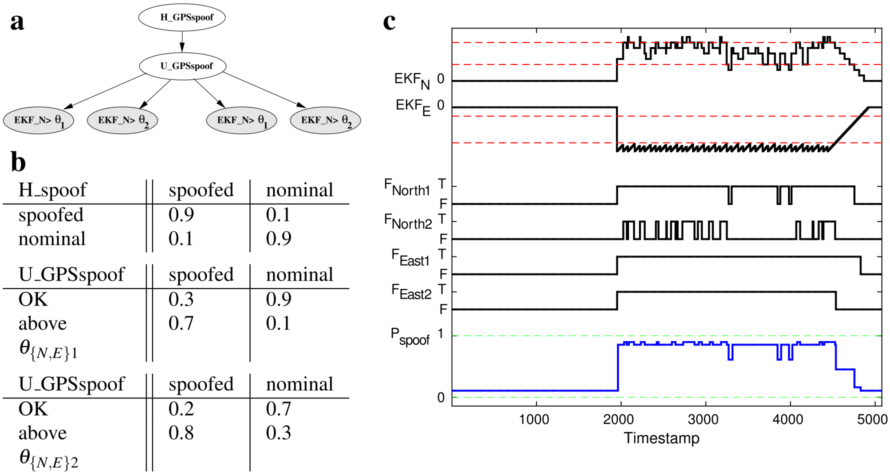


Fig. 14 Desired UAS mission: a straight trajectory south-south-west to the next waypoint (*blue*); GPS spoofing causes the UAS to veer off to reach a target point defined by the attacker (*red*). (Color figure online)



H_spoof	spoofed	nominal
spoofed	0.9	0.1
nominal	0.1	0.9

U_GPSspoof	spoofed	nominal
OK	0.3	0.9
above $\theta_{\{N,E\}1}$	0.7	0.1

U_GPSspoof	spoofed	nominal
OK	0.2	0.7
above $\theta_{\{N,E\}2}$	0.8	0.3

Fig. 15 **a** Bayesian network for diagnosis of GPS spoofing. **b** Conditional probability tables for selected nodes of the network. CPT values for each direction (north, east) are the same. The prior for the health node is (0.5, 0.5). **c** Set of traces that indicate GPS spoofing. *Top panel* shows the input signals EKF_N and EKF_E. The *middle panel* shows the Boolean signals after preprocessing. P_{spoof} is the output of the Bayesian network and shows the probability of a GPS spoof attack occurring

We use a simple BN as shown in Fig. 15a in order to calculate the probability of an attack. The node *U_GPSspoof* represents the unobservable state whether or not the UAS is under attack. Each of the four sensor nodes correspond to one of the four temporal observers (F_{North1}, F_{North2}, F_{East1}, F_{East2}). We use the node *H_GPSspoof* in order to evaluate the likelihood of an attack based on the evidence provided by the temporal observers.

The bottom panel of Fig. 15c shows the results of the evaluation. It can be seen that a high probability of a GPS attack is calculated during the attack. Again, these signals are not individually absolute indicators that an attack has happened. Flying in areas with weak GPS coverage, for example, in a mountainous or urban environment, could produce similar signals. Therefore, in our R2U2 models, we aim to take into account other observation patterns and use a Bayesian network for probabilistic reasoning. Information supporting the hypothesis of an attack could include prior loss of satellite locks, transients in GPS signals, or other types of attacks. In the case of the captured CIA drone, an Iranian engineer claimed to have jammed the drone's communications link in order to force the drone into an autopilot mode and then initiated the GPS spoofing attack [51].

According to Psiaki et al. [38], there are two main approaches to get a victim to lock onto a false signal. In one scenario, an attacker provides a powerful fake signal and forces a re-acquisition of the victim's tracking by jamming the signal. The victim's GPS receiver is then likely to lock onto the stronger fake signal. The other approach is to Doppler-match the false signals to the *true* signal and increase power gradually until the tracking loop of the victim is captured. In both cases, such an attack most likely affects the power and/or the noise of the signal. Hence, various spoofing detection methods are based on signal power monitoring as for example, absolute power monitoring, signal-to-noise ratio (SNR or C/N_0) monitoring, or received power variations versus receiver movement [15,25,28].

Our simulations did not include an authentic spoofed SNR simulation. However, our signal preprocessing in combination with temporal observers allow us to model and detect such behaviors of the signals like for example, a constantly increasing signal power, or a sudden drop of the SNR. Thus, R2U2 we plan to implement such signal power based spoofing detection mechanisms in our future work.

We estimate that 10 person-hours were spent on model development and approximately 45 hours were spent on experimental setup and simulation.

5 Conclusion

We have extended our Realizable, Responsive, Unobtrusive Unit (R2U2) to enable real-time monitoring and diagnosis of security threats. This includes the ability to reason about complex and subtle threats utilizing indicators from both the UAS system and its software. We present a novel hybrid monitoring architecture consisting of software- and hardware-based modules where we utilize the features of a System-on-Chip FPGA. Our self-contained, embedded implementation on-board a standard, flight-certifiable FPGA meets stated FAA requirements and efficiently recognizes both individual attack indicators and attack patterns, adding a new level of security checks not available in any previous work. Case studies on-board a real NASA DragonEye UAS provide a promising proof-of-concept of this new architecture.

The myriad directions now open for future work include considering software instrumentation to enable more FSW-related compromises and doing hardware-in-the-loop simulation experiments to detect these. We plan to extend this technology to other, more complex UAS and beyond, to other types of aircraft and spacecraft with different configurations and capabilities. A major bottleneck of the current R2U2 is the manual labor required to develop and test every temporal logic formula and BN; we are currently considering methods for making this a semi-automated process to better enable future extensions.

Acknowledgements Open access funding provided by TU Wien (TUW). Funding was provided by Ames Research Center (Grant No. NNX14AN61A, NNX12AK33A)

References

1. Adapteva: The parallella board, https://www.parallella.org/board
2. Ahmed A, Lisitsa A, Dixon C (2013) Testid: a high performance temporal intrusion detection system. In: ICIMP 2013, The eighth international conference on internet monitoring and protection, pp 20–26
3. Ardupilot.com: APM:Plane, Open source fixed-wing aircraft UAV, http://plane.ardupilot.com
4. Bartocci E, Grosu R, Karmarkar A, Smolka SA, Stoller SD, Zadok E, Seyster J (2013) Adaptive runtime verification. In: Proc RV 2012, LNCS, vol 7687. Springer. doi:10.1007/978-3-642-35632-2_18
5. Bilge L, Dumitras T (2012) Before we knew it: An empirical study of zero-day attacks in the real world. In: Proceedings of the 2012 ACM conference on computer and communications security. pp 833–844. CCS '12, ACM, New York, NY, USA, doi:10.1145/2382196.2382284
6. Bushnell D, Denney E, Enomoto F, Pai G, Schumann J (2013) Preliminary recommendations for the collection, storage, and analysis of UAS safety data. Technical Report NASA/TM-2013-216624, NASA Ames Research Center
7. Chavira M, Darwiche A (2005) Compiling Bayesian networks with local structure. In: Proceedings of the 19th international joint conference on artificial intelligence (IJCAI), pp 1306–1312
8. Christian Science Monitor: RQ-170 GPS Spoofing (2011), http://www.csmonitor.com/World/Middle-East/2011/1215/Exclusive-Iran-hijacked-US-drone-says-Iranian-engineer-Video
9. Donzé A, Maler O, Bartocci E, Nickovic D, Grosu R, Smolka S (2012) On temporal logic and signal processing. In: Proc ATVA 2012, LNCS, vol 7561. Springer. doi:10.1007/978-3-642-33386-6_9
10. Fisher K (2014) Using formal methods to enable more secure vehicles: DARPA's HACMS program. In: Proceedings of the 19th ACM SIGPLAN international conference on functional programming. pp 1–1. ICFP '14, ACM, New York, NY, USA, doi:10.1145/2628136.2628165
11. GAO (2015) Air traffic control: FAA needs a more comprehensive approach to address cybersecurity as agency transitions to nextgen. Tech. Rep. GAO-15-370, United States Government Accountability Office (04 2015), http://www.gao.gov/assets/670/669627.pdf
12. Geist J, Rozier KY, Schumann J (2014) Runtime observer pairs and bayesian network reasoners on-board FPGAs: Flight-certifiable system health management for embedded systems. In: Proceedings of Runtime verification - 5th international conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Springer, pp 215–230. doi:10.1007/978-3-319-11164-3_18
13. Heffernan D, Macnamee C, Fogarty P (2014) Runtime verification monitoring for automotive embedded systems using the iso 26262 functional safety standard as a guide for the definition of the monitored properties. IET Softw 8(5):193–203
14. Humphreys T (2012) Statement on the vulnerability of civil unmanned aerial vehicles and other systems to civil GPS spoofing. University of Texas at Austin (July 18, 2012)
15. Jafarnia-Jahromi A, Broumandan A, Nielsen J, Lachapelle G (2012) GPS vulnerability to spoofing threats and a review of antispoofing techniques. Int J Navig Obs 2012:12702. doi:10.1155/2012/127072
16. Jaksic S, Bartocci E, Grosu R, Kloibhofer R, Nguyen T, Nickovic D (2015) From signal temporal logic to FPGA monitors. In: 2015 ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE), pp 218–227
17. Javaid AY, Sun W, Devabhaktuni VK, Alam M (2012) Cyber security threat analysis and modeling of an unmanned aerial vehicle system. In: Proceedings of international conference on technologies for Homeland security (HST), 2012 IEEE, pp 585–590
18. JSBSim: JSBSim - open source flight dynamics model, http://jsbsim.sourceforge.net

19. Kalajdzic K, Bartocci E, Smolka SA, Stoller SD, Grosu R (2013) Runtime verification with particle filtering. In: Proc RV 2013, LNCS, vol 8174. Springer, pp 149–166
20. Kerns AJ, Shepard DP, Bhatti JA, Humphreys TE (2014) Unmanned aircraft capture and control via GPS spoofing. J Field Robot 31(4):617–636
21. Kim A, Wampler B, Goppert J, Hwang I, Aldridge H (2012) Cyber attack vulnerabilities analysis for unmanned aerial vehicles. In: Proceedings of Infotech@Aerospace 2012. doi:10.2514/6.2012-2438
22. Koymans R (1990) Specifying real-time properties with metric temporal logic. Real-time Syst 2(4):255–299
23. Kwon C, Liu W, Hwang I (2013) Security analysis for cyber-physical systems against stealthy deception attacks. In: 2013 American control conference, pp 3344–3349
24. Lu H, Forin A (2007) The design and implementation of P2V, an architecture for zero-overhead online verification of software programs. Tech. Rep. MSR-TR-2007-99, Microsoft Research, http://research.microsoft.com/apps/pubs/default.aspx?id=70470
25. Magiera J, Katulski R (2015) Detection and mitigation of GPS spoofing based on antenna array processing. J Appl Res Technol 13(1): 45–57, http://www.sciencedirect.com/science/article/pii/S1665642315300043
26. Maler O, Nickovic D, Pnueli A (2008) Checking temporal properties of discrete, timed and continuous behaviors. In: Pillars of Computer Science, LNCS, vol 4800. Springer, pp 475–505. doi:10.1007/978-3-540-78127-1_26
27. MAVProxy: A UAV ground station software package for mavlink based systems, http://tridge.github.io/MAVProxy
28. McMilin E, De Lorenzo DS, Walter T, Lee TH, Enge P (2014) Single antenna GPS spoof detection that is simple, static, instantaneous and backwards compatible for aerial applications. In: Proceedings of the 27th international technical meeting of the satellite division of the institute of navigation (ION GNSS+ 2014), Tampa, FL. pp 2233–2242
29. Mengshoel OJ, Chavira M, Cascio K, Poll S, Darwiche A, Uckun S (2010) Probabilistic model-based diagnosis: an electrical power system case study. IEEE Trans Syst Man Cyberne Part A Syst Hum 40(5):874–885
30. Meredith PO, Jin D, Griffith D, Chen F, Roşu G (2012) An overview of the MOP runtime verification framework. Int J Softw Tools Technol Transf 14(3):249–289
31. Musliner D, Hendler J, Agrawala AK, Durfee E, Strosnider JK, Paul CJ (1995) The challenges of real-time AI. IEEE Computer 28, 58–66. http://citeseer.comp.nus.edu.sg/article/musliner95challenges.html
32. Naldurg P, Sen K, Thati P (2004) A temporal logic based framework for intrusion detection. In: FORTE, LNCS, vol. 3235, Springer, pp 359–376
33. Olivain J, Goubault-Larrecq J (2005) The orchids intrusion detection tool. In: CAV, LNCS, vol. 3576, Springer, pp 286–290
34. Pearl J (1985) A constraint propagation approach to probabilistic reasoning. In: UAI, AUAI Press, pp 31–42
35. Pellizzoni R, Meredith P, Caccamo M, Rosu G (2008) Hardware runtime monitoring for dependable COTS-based real-time embedded systems. In: RTSS 2008, IEEE, pp 481–491
36. Perry S (2015) Subcommittee hearing: Unmanned aerial system threats: exploring security implications and mitigation technologies. Committee on homeland security, http://homeland.house.gov/hearing/subcommittee-hearing-unmanned-aerial-system-threats-exploring-security-implications-and
37. Pike L, Goodloe A, Morisset R, Niller S (2010) Copilot: a hard real-time runtime monitor. In: Proc RV 2010, LNCS, vol 6418, Springer, pp 345–359
38. Psiaki ML, Humphreys TE (2016) GNSS spoofing and detection. Proc IEEE 104(6):1258–1270
39. Reinbacher T, Geist J, Moosbrugger P, Horauer M, Steininger A (2012) Parallel runtime verification of temporal properties for embedded software. In: MESA, pp 224–231
40. Reinbacher T (2013) Analysis of embedded real-time systems at runtime. Ph.D. thesis, Technische Universität Wien, Austria
41. Reinbacher T, Brauer J, Horauer M, Steininger A, Kowalewski S (2014) Runtime verification of microcontroller binary code. Sci Comput Program 80:109–129. doi:10.1016/j.scico.2012.10.015
42. Reinbacher T, Rozier KY, Schumann J (2014) Temporal-logic based runtime observer pairs for system health management of real-time systems. In: Tools and algorithms for the construction and analysis of systems - 20th international conference, TACAS 2014, Grenoble, France, LNCS, vol 8413, pp 357–372. Springer. doi:10.1007/978-3-642-54862-8_24
43. Schumann J, Mbaya T, Mengshoel OJ, Pipatsrisawat K, Srivastava A, Choi A, Darwiche A (2013) Software health management with Bayesian networks. In: Innovations in System and Software Engineering, vol 9. Springer, pp 271–292

44. Schumann J, Moosbrugger P, Rozier KY (2015) R2U2: monitoring and diagnosis of security threats for unmanned aerial systems. In: Proc RV 2015, LNCS, vol 9333. Springer, Cham. pp 233–249. doi:10.1007/978-3-319-23820-3_15
45. Schumann J, Roychoudhury I, Kulkarni C (2015) Diagnostic reasoning using prognostic information for unmanned aerial systems. In: Proceedings of the 2015 annual conference of the prognostics and health management society (PHM2015)
46. Schumann J, Rozier KY, Reinbacher T, Mengshoel OJ, Mbaya T, Ippolito C (2013) Towards real-time, on-board, hardware-supported sensor and software health management for unmanned aerial systems. In: Proceedings of the 2013 annual conference of the prognostics and health management society (PHM2013)
47. Schumann J, Rozier KY, Reinbacher T, Mengshoel OJ, Mbaya T, Ippolito C (2015) Towards real-time, on-board, hardware-supported sensor and software health management for unmanned aerial systems. Int J Progn Health Manag 6(21):1–27
48. Selyunin K, Nguyen T, Bartocci E, Nickovic D, Grosu R (2016) Monitoring of MTL specifications with IBM's spiking-neuron model. In: 2016 Design, automation test in europe conference exhibition (DATE), pp 924–929
49. Selyunin K, Nguyen T, Bartocci E, Grosu R (2016) Applying runtime monitoring for automotive electronic development. In: Proc RV 2016, LNCS, vol 10012. Springer, pp 462–469. doi:10.1007/978-3-319-46982-9_30
50. Shachtman N, Axe D Most U.S. drones openly broadcast secret video feeds. Wired (10 2012), http://www.wired.com/2012/10/hack-proof-drone/
51. Shepard DP, Bhatti JA, Humphreys TE (2012) Drone hack. GPS World 23(8):30–33
52. Stoller SD, Bartocci E, Seyster J, Grosu R, Havelund K, Smolka SA, Zadok E (2012) Runtime verification with state estimation. In: Proc RV 2011, LNCS, vol 7186. Springer, pp 193–207. doi:10.1007/978-3-642-29860-8_15
53. Todman T, Stilkerich S, Luk W (2015) In-circuit temporal monitors for runtime verification of reconfigurable designs. In: Proceedings of the 52nd annual design automation conference. pp 50:1–50:6. DAC '15, ACM, New York, NY, USA, doi:10.1145/2744769.2744856
54. U.S. Air Force: Aircraft Accident Investigation: RQ-1L, S/N 96-3023. AIB Class A Aerospace mishaps (September 2000), http://usaf.aib.law.af.mil/ExecSum2000/RQ-1L_Nellis_14Sep00.pdf