R2U2: Monitoring and Diagnosis of Security Threats for Unmanned Aerial Systems*

Johann Schumann[1], Patrick Moosbrugger[1], and Kristin Y. Rozier[2]

[1] SGT, Inc., NASA Ames, Moffett Field, CA, USA, Johann.M.Schumann@nasa.gov, Patrick.Moosbrugger@technikum-wien.at
[2] University of Cincinnati, OH, USA, Kristin.Y.Rozier@uc.edu

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 security threat detection.
R2U2 is designed to continuously monitor inputs from GPS and the ground control station, sensor readings, actuator outputs, and flight software status. By simultaneously monitoring and performing statistical diagnosis, attack patterns and post-attack discrepancies in the UAS behavior can be detected. R2U2 uses runtime observer pairs for linear temporal logic for property monitoring and Bayesian networks for diagnosis of security threats. We present results of simulations of several attack scenarios on the NASA DragonEye UAS running ArduPilot flight software.

1 Introduction

Unmanned Aerial Systems (UAS) are starting to permeate many areas in everyday life. From toy quadcopters, to aircraft for delivery, crop dusting, public safety, and military operations, UAS of vastly different weight, size, and complexity are used. Although the hardware technology has significantly advanced in the past years, there are still considerable issues to be solved before UAS can be used safely. Perhaps the biggest concern is the integration of UAS into the national airspace (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 [4].

A related topic, which has been vastly neglected so far, is security [24]. 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. There are numerous examples of system- and software-related UAS incidents. Often, video feeds from military UAS flying were 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 [30]. In 2013, the Iran allegedly abducted a CIA drone by jamming its command link and spoofing GPS. Instead of returning back to the CIA base, the UAS was directed to land in Iranian territory [5]. Video footage retrieved from that UAS later [12] provided additional evidence of that abduction. Even large research UAS worth millions of dollars are controlled via unencrypted RF connections; most UAS communicate over a large number of possible channels [9], relying on the assumption that "... one would have to know the frequencies" to send and receive data.

* This work was supported in part by NASA ARMD 2014 I3AMT Seedling Phase I, NNX12AK33A and the Austrian Josef Ressel Center (VECS).

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 [7]. In addition, the Automatic Dependent Surveillance - broadcast (ADS-B), which will be a cornerstone for Next Generation Air Traffic Control, does not provide any authentication [25].

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. In this paper, we focus on the communications, sensor, and software-related security threats. 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 this paper, we extend our on-board monitoring and diagnosis framework R2U2. Originally developed for system health management, R2U2 dynamically monitors software and sensor traffic on-board complex systems [26,8,28]. The underlying health model is specified using metric and linear temporal logics (MTL and LTL, respectively) and Bayesian networks (BN). This combination of specification paradigms allows the user to concisely express temporal relationships between sensor and software status and signals as well as to perform probabilistic diagnostic reasoning.

In order to minimize impact on the flight software and the usually weak flight computer, R2U2 is implemented using FPGA hardware. This no-overhead implementation is designed to uphold the FAA requirements of REALIZABILITY and UNOBTRUSIVENESS. To our knowledge, there are only two previous embedded hardware monitoring frameworks capable of analyzing formal properties: P2V [15] and BusMOP [23,19]. However, P2V is a PSL to Verilog compiler that violates our UNOBTRUSIVENESS requirement by instrumenting software. Like R2U2, BusMOP can monitor COTS peripherals, achieving zero runtime overhead via a bus-interface and an implementation on a reconfigurable FPGA. However, BusMOP violates our REALIZABILITY requirement by reporting only property failure and handling only past-time logics whereas we require early-as-possible reporting of future-time temporal properties passing and intermediate status updates. BusMOP can require up to 4 clock cycles from any event that triggers a property resolution to executing the corresponding handler, violating our RESPONSIVENESS requirement; R2U2 always reports in 1 clock cycle. BusMOP also violates UNOBTRUSIVENESS by executing arbitrary user-supplied code on the occurrence of any property violation.

We extend R2U2 to enable the dynamic monitoring of the flight software, the communication stream, and 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. The temporal and probabilistic health models and its FPGA implementation 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 a separate hardware entity and pro-

grammed using VHDL. Javaid et al. [10] also analyze cyber security threats for UAS. They simulated the effects of attacks that usually ended in a crash, focusing on identifying different existing attack surfaces and vulnerabilities rather than focusing on runtime detection or post-attack analysis. TeStID [2], ORCHIDS [21] and MONID [20] are Intrusion Detection Systems which use temporal logic to specify attack patterns. These security monitoring frameworks are targeted at IT systems and infrastructure.

Our contributions include:

- extending R2U2 from monitoring of safety properties of hardware [26,8], integrating hardware and software bus traffic monitoring for security threats,
- enabling on-board, real-time detection of attack scenarios and post-attack behavior,
- designing more expressive runtime reasoning than previous approaches, which is needed for catching subtle security properties; this includes temporal logic formula patterns and BN reasoning for probabilistic diagnostics and root cause analysis,
- detection of *attack patterns* rather than a specific, isolated attack or incident
- isolating monitoring and reasoning our implementation 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 monitors and BN dignostic reasoning for detection of security threats and post-attack UAS behavior. In Section 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 GSC, and GPS spoofing. Finally, Section 5 discusses future work and concludes.

2 Background

For this paper, we consider a simple and small UAS platform, the NASA DragonEye (Figure 1A). With a wingspan of 45in it is a small UAS, but it 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 computer and 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. For autonomous missions, there is no link between the UAS and the GCS.

Our example system uses the open-source FSW ”APM:Plane” [3], which does not contain any security features like command and data encryption for the GCS-UAS link per default. We nevertheless selected this FSW because it very closely resembles the

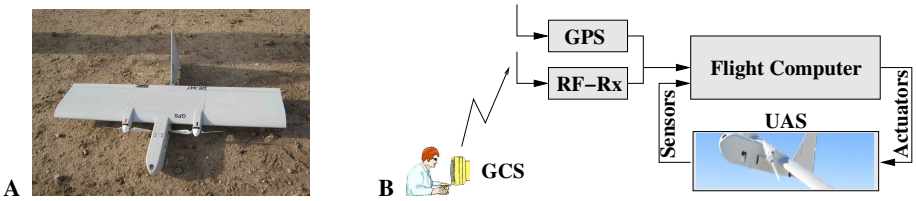

Fig. 1. A: Photo of NASA DragonEye. B: High level system architecture of a small UAS.

architecture of similar and even larger, more complex UAS. This architecture allows us to easily carry out white-box experiments and to study the relationship between attacks and post-attack behavior. Results of our studies can be carried over to highly secure and resilient flight software.

2.1 R2U2

Developed to continuously monitor system and safety properties of an UAS in flight, our real-time R2U2 (REALIZABLE, RESPONSIVE, and UNOBTRUSIVE Unit) has been implemented on an FPGA (Field Progammable Gate Array) [26,8]. Health models within this framework [28,29] are defined are defined using Metric Temporal Logic (MTL) [26] for expressing temporal properties and Bayesian Networks (BN) for probabilistic and diagnostic reasoning.

Temporal Logic Monitors MTL formulas consist of propositional variables, logic operators like $\wedge$, $\vee$, $\neg$, or $\rightarrow$, and temporal operators to express temporal relationships between events. For MTL formulas p, q, we have $\Box p$ (ALWAYS p), $\Diamond p$ (EVENTUALLY p), $\mathcal{X}p$ (NEXTTIME p), $p\mathcal{U}q$ (p UNTIL q), and $p\mathcal{R}q$ (p RELEASES q) with their usual semantics [26]. 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 $\Box_{[i,j]}\, p$, $\Diamond_{[i,j]}\, p$, $p\,\mathcal{U}_{[i,j]}\, q$, and $p\,\mathcal{R}_{[i,j]}\, q$ where the temporal operator applies over the interval between time i and time j, inclusive, and time steps refer to ticks of the system clock.

Bayesian Networks for Health Models In many situations, temporal logic monitoring might come up with 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. 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. BNs are well established in the area of diagnostic and health management (e.g., [22,18]). 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 [8,27,29]. R2U2 evaluates posterior probabilities, which reflect the most likely root causes at each time step.

2.2 FPGA Implementation

R2U2 is implemented in FPGA hardware. Figure 2 shows the major components: the *control subsystem*, the *signal processing and filtering system* (SP), the *runtime verification* (RV) unit, and the *runtime reasoning* (RR) unit. The control subsystem establishes

the communication link to the external world to load health models and to receive health results.

Continuous signals obtained, using a read-only interface, from the flight computer and from communications between the flight computer and the payload and sensor system. These signals are filtered and discretized in the SP unit to obtain streams of propositional variables. The RV and RR units comprise the proper health management hardware: RV monitors MTL properties using pairwise observers [26]. After the temporal logic formulas have been evaluated, the results are transferred to the runtime reasoning (RR) subsystem, where the compiled Bayesian network is evaluated to yield the posterior marginals of the health model [8].

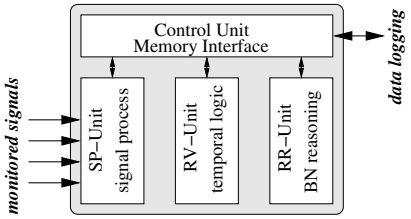

Fig. 2. Principled R2U2 implementation on FPGA.

3 Our Approach to Threat-Detection

For our approach, we consider the "system" UAS (as depicted in Figure 1B) as a complex feedback system. Commands, GPS readings as well as measurements of the sensors are processed by the FSW on flight computer to calculate new values for the actuators (e.g., throttle, aileron, rudder, or elevator commands) and to update its internal status.

In this paper, we assume that all malicious attacks are attempted during flight.[3] Furthermore, all external inputs to the UAS are received via a wireless link from the ground control station and a GPS transmitter only. Spoofing of the compass sensor, for example, via a strong magnetic field is outside the scope of R2U2. All attacks considered in our study involve the wireless link from the ground control station or the GPS transmitter.

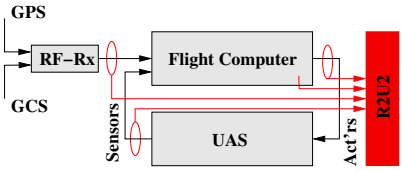

Fig. 3. High-level architecture of R2U2

With our R2U2 framwork, we continuously monitor both inputs and are capable of identifying some 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, this information does not suffice to reliably identify an attack scenario, additional supporting information is necessary. This will be obtained from the analysis from 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 the post-attack behavior are the two major tasks of R2U2. Both monitoring tasks, however, are not independent from each other and thus we have to model these interactions within our R2U2 framework. Typically, a certain

[3] We do not model attack scenarios via compromised FSW.

input pattern, followed by a specific behavior characterizes an attack. For example, a strong oscillation in the aircraft movement, which was triggered by a an unusual GCS command indicates an attack (or an irresponsible pilot). Similarly, transients in GPS signals followed by subtle position movements would be telltales of a GPS spoofing attack.

Figure 3 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. In the following, we will discuss attack monitoring and post-attack behavior monitoring, loosely following [14].

3.1 Attack Monitoring

As all attacks are initiated through the GCS or GPS inputs, we will monitor the following attack surfaces. Because of zero-day attack mechanisms, this list will always be incomplete.[4] 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 a malicious 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 of the mode the UAS is in. For example, a "reset-FSW" command sent to the UAS, while in the air, will, most likely lead to a crash of the UAS, because all communication and system parameters are lost. Thus, in all likelihood, this command has been issued during 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 way-points can be sent to the UAS during flight to update its mission, repeated sending of way-points with identical coordinates, or weird/erroneous coordinates might be an indicator for a malicious attack.

Transients in GPS signals: since the quality of UAS navigation strongly depends on the quality of the received GPS signals, sudden transients of signal strength and noise ratios (Jamming-to-Noise Sensing [9]) or the number of available satellites might give an indication of GPS spoofing or jamming.

It should be noted that these pattern 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.

[4] `http://dl.acm.org/citation.cfm?id=2382284`

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 the values of the variables carring their pre-processed sensor readings from the FSW. This simplification, however, prevents our current implementation from detecting a crash of the flight software initiated by a malicious attack. 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 subsequent crash. Pilot-induced oscillations (PIO) in commerical aircraft have caused several severe accidents and loss of life. Due to the increased loads on wings or control surfaces, parts of the aircraft can be literally ripped off. In a UAS such oscillations can be caused by issueing 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 inoperational.

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

Sensor Readings: Sudden changes of sensor readings or consistent drift 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, 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

With temporal logic and BN, the specific patterns for each of the attack and behavior monitors are captured. We also use these mechanisms to specify the temporal, causal, and probabilistic relationsships 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. Obviously, there are temporal constraints to ensure that these events are actually correlated. So, for example, an oscillation of the UAS occurs between 100-200 timesteps after the control loop parameters have been altered.

A typical formula would look like:

$$A \rightarrow WHATEVER(B)$$

KRISTIN: some formula(s) here

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.

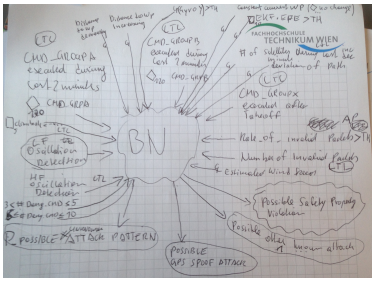

Fig. 4. Using TL+BN Network to detect threats

Opens up many variants for concise and easy specification:

- FT logic: A results in B soon
- PT: if we observe B and A has happened not too far in the past
- "soft" probabilistic logic with BN

specification patterns – see Kristin's email

Another variant can be achieved by grouping related input signals. For example, we can define groups of dangerous commads, unusual repeated commands or events like:

$\mathbf{dangerous\_cmds} = cmd\_reset \vee cmd\_calibrate\_sensor \vee cmd\_disarm \ldots$
$\mathbf{unusual\_cmds\_after\_takeoff} = cmd\_get\_params \vee set\_params \vee get\_waypoints \ldots$
$\mathbf{unusual\_periodic\_events} = cmd\_navigate\_to \vee cmd\_mode\_change \vee invalid\_packet\_received \ldots$

This enables us to directly use these preprocessed groups in temporal formulas and feed them into a BN, therby supporting simple reuse of common patterns and assist 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.

$\square((CMD == \text{takeoff}) \rightarrow \square \neg \mathbf{dangerous\_cmds}\, \mathcal{U}\, land\_complete)$

Kristin, does this make sense?

4 Experiments and Results

4.1 Experimental Setup

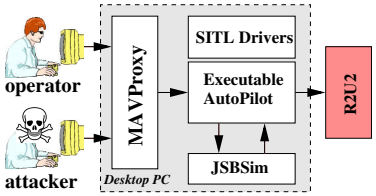

Fig. 5. R2U2 sitl test setup

Our experiments can be run either directly on the UAS, on our Ironbird processor-in-the-loop setup, which consists of the original UAS hardware components in a LAB environment or in a software-in-the loop simulation. In all configurations, the produced data-traces are forwarded via a UART transmission to the R2U2 framewok running on the Parallella Board [1], a credit-card sized, low-cost, COTS SoC-FPGA platform, where the actual monitoring is performed inside the FPGA. An Ubuntu Linux installation on the parallella board is used for the interface configuration and signal pre-processing, hence, inputs from other sources like I^2C, SPI, analog, etc. can be configured by loading the appropriate Linux kernel drivers. An overview of the software-in-the-loop simulation is shown in figure 5. The UASs flight behavior is simulated by connecting it to the open source JSBSim [11] flight dynamics model. The 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 testflight. The operator's GCS is connected to the UAS via an open source MAVLink proxy [17]. We also connect a second GCS to the proxy in order to simulate the attackers injected MAVLink packets. Similar to the Ironbird or the UAS test-setup, the produced traces are streamed directly to the parallella board by means of a UART cable.

4.2 Dangerous MAV Commands

The Arduino flight software uses a standardized communication protocol to send commands to the UAS and to receive data from it: the MAVLink protocol [16]. Each MAVLink command consists of a command identifier followed by a fixed number of 0 or more parameters.

In addition to commands controlling the actual flight, the MAVLink protocol allows the user to set-up 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. It is obvious that such commands, which substantially alter the behavior of the AC can, when given during flight, cause dangerous behavior of the UAS and 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 were erased on the UAS. Communication to the UAS could not be reestablished and the UAS crashed causing a total loss $3.7M [6].

Although, many of those commands in our flight software can only be issued on the ground using a wired connection between the GCS and the flight computer, most gain parameters can be set even during flight for setup and fine tuning. If parameters for the control loops are set to extreme values, the aircraft can experience oscillations that could lead to disintegration of the UAS and subsequent crash. Therefore, such commands, sent in midair, 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, pitch oscillation) 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, e.g., to adjust for a specific load of the UAS. 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 command (or a very stupid pilot) caused the dangerous oscillations. Our model uses the specification FIX FORMULA

$$\square((MAV\_cmd == \text{set-parameter}) \rightarrow \Diamond_{[0,1200]} oscillation\_detected)$$

Oscillations can be detected by executing a Fast Fourier Transform (FFT) on the pitch measurements and monitoring if a certain element of the power spectrum is above a threshold.[5]

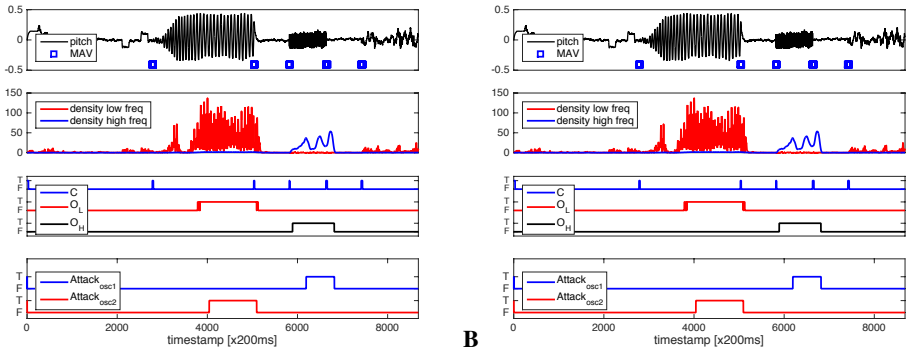

Fig. 6. UPDATED FIGSA: UAS behavior after malicious setting of gain parameters. B: Oscillations have not been caused by an attack.

Figure 6A 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 issued. At around $t = 3000$, a strong low-frequency oscillation appears that ends around $t = 5000$. Shortly afterwards, a high-frequency oscillation occurs (around $t = 6000 - 7000$). Low (red) and high (blue) frequency elements of the power spectrum (second panel from top) clearly indicate the oscillations, compared to the regular noise in the pitch. The third panel contains the Boolean inputs for R2U2: "set-parameter received", "Low-frequency-oscillation", and "high-frequency-oscillation". The bottom panel shows the outputs of the R2U2 monitor, indicating two separate attacks. Figure 6B shows, for comparison, the same pitch oscillations, but no "set-parameter" commands have received. Thus, the R2U2 does not find evidence for a malicious attack using the attack surface just described.

[5] Oscillations around the other UAS axes are generated in a similar manner.

4.3 DoS hijack attack

Attackers continiously find new ways to break into and compromise systems. Hence, it is challenging to account for every possible attack scenario, since there can always be a non foreseen loophole.

This testcase demonstrates, how our framework can detect an intrusion due to different indications without explicitly write a security model for a particular attack by means of using grouping patterns as described earlier.

In our simulation we initiate a sophisticated attack to hijack the AC by trying to establish a link from the attackers GCS to the UAS, which results in many bad packets as can be seen in figure 7 between the timestamps 500 and 1000. These can be caused by different reasons like an incorrect channel, protocol, or protocol-version, encryption. In order to detect this, we can use for exampe a formula like: S_1: The number of bad packets N_b^R is low, no more than one bad packet every 10 seconds.

$\square_{[0,10]}(N_b^R = 0 \vee (N_b^R \geq 1 \, \mathcal{U}_{[0,10]} N_b^R = 0))$ Kristin, this is copied from RV2014 - does this make sense?

Next, an attacker would try to gather information about the AC, e.g., by requesting the AC parameters or download the waypoints and parameters through the MAVLink protocol, which is represented as spikes between timestamp 1000 and 1300. For detecting this, we can use our eralier defined input groups unusual_cmds_after_takeoff.

S_2: After takeof, there shall be no unusual commands after takeoff until the AC has landed. $\square((CMD == \text{takeoff}) \rightarrow \square \neg \, unusual\_cmds\_after\_takeoff \, \mathcal{U} \, land\_complete)$

Kristin, does this make sense?

Finally, the attacker is flooding the communication link in a type of DoS attack by sending continious requests to navigate to the attackers coordinates, combined with requests to set the UASs home location to the same coordinates. The result can be seen in the occurance of continiously high number of navigation requests starting around timestamp 1400. For the detection, we write formulas, either explicitly detecting an unusual period of navigational commands (S_3) or a group of earlier defined unusual periodic commands (S_4).

S_3: There shall be no continious navigation requests for more than 30 timestamps.

$$\square_{[0,30]} cmd\_navigate\_to\_coordinates$$

Kristin, does this make sense?

S_4: There shall be no continious unusual periodic events for more than 60 timestamps.

$$\square_{[0,60]} unusual\_periodic\_events$$

Kristin, does this make sense?

The formulas S_1, S_2, S_3 or S_4 are no reliable indication for an ongoing attack if viewed individually. However, if they are combined by feeding them into our BN, we can calculate a high possibility for an ongoing attack.

Our simulation revealed, that even this type of attack is detected by the operator eventually, all attempts e.g. to change the

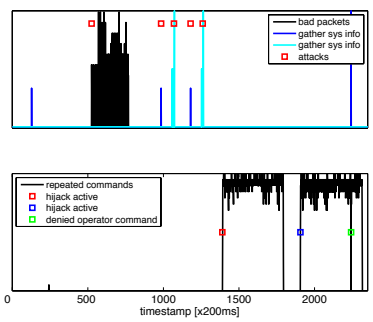

Fig. 7. UAS DoS hijack results

UAS to its original course will immediately be overwritten by the attackers navigation commands at a high rate. Even if the AC tries to return to the launch position either because of an empty battery, or as result of an operators attempted countermeasure, it would fly to the attackers location due to the altered home coordinates. Hence, the simulation of this scenario revealed, that besides crashing the AC intentionally, there was no simple way to prevent the DoS hijacking from the operators GCS. Also, especially in cases where the AC is flying outside the operators communication range, it is desirable for the AC to be capable of detecting such an attack autonomously.

4.4 GPS Spoofing

GPS plays a central role in the control of autonomous UAS. Typically, a flight mission for a UAS is defined by a list of waypoints, each giving a target specified by the tuple (longitude, latitude, 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 by with the help of a GPS receiver. Due to limited accuracy, only GPS longitude and latitude are used for navigation, the altitude is obtained using the barometric means of a pitot tube.

For its short-term (inner loop) control for aircraft attitude, the UAS is equipped with inertial sensors. Accelerometers measure current acceleration in each of the aircraft axes, gyros measure the angular rate for each axis. Integration of these sensor values can be used to obtain relative positions and velocities. These data come with a very fast rate, are independent from the outside but very noisy. So these signals cannot be used for waypoint navigation. Thus the FSW uses a Kalman Filter to mix the inertial signals with the GPS signals. 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: GPS jamming and GPS spoofing. In a jamming scenario, 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 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 the Iranians alledgedly used GPS spoofing to highjack a CIA drone and forced it to land on an Iranian airfield rather than its base [5,31] Subsequently, researchers from the University of Texas at Austin

successfully demonstrated how a $80M Yacht at Sea,[6], as well as a small UAV can be directed to follow a certain pattern due to GPS spoofing [13].

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.

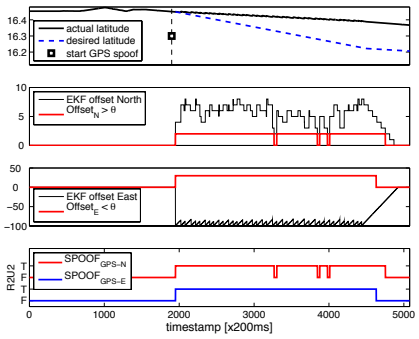

Fig. 8. GPS spoofing

We developed an R2U2 model that is able to detect certain kinds of GPS spoofing. This model monitors the quality of the GPS signal and the inertial navigation information. For our experimental evaluation, we defined a UAS mission, which flies at a fixed altitude toward the next waypoint, which is south-south-west of the current UAS location. When spoofing occurs, the attacker modifies the GPS signal in such a way that tricks the UAS into believing it is still flying in a straight line toward the next waypoint, when the UAS is actually veering off to reach a target point as defined by the attacker. Figure 8 shows the relevant signals during this mission. Here, we focus on the latitude as observed by the UAS. The top panel shows the point of the spoofing attack and the trace for the temporal development of the UAS longitude as observed by the UAS (blue) and the actual UAS position (green). 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, monitoring focuses on alternate signals that convey a similar information. The inertial navigation unit produce an error or offset signal that reflects the difference between the current position observed by GPS and by the inertial sensors. The next two panels of Figure **??** shows that these offset signals can become substantially large during the actual spoofing period, when the GPS locations are gradually moved to the attacker's target. These signals are discretized and fed into the R2U2 model. The bottom panel shows the output of the temporal observers. In order to avoid false alarms, we present a flight path that corresponds to the trajectory of the UAS during the spoofing attack. However, this mission is flown by using waypoints accordingly and no attack occurs. Thus, the GPS spoof alarm should not become active.

In order to protect UASs from attacks against command link jamming, they are sometimes put in a complete autonomous mode, accepting no further external commands. [9] Ironically, what was intended to be a security measure could inhibit the operator's attempts to recover a UAS during such an attack. However, R2U2 enables the AC to detect an ongoing attack autonomously in order to enable adequate countermeasures.

Regarding the supposed CIA drone capture example, an Iranian engineer claimed to have jammed the drone's communications link in order to force the drone into an autopilot mode and initiate the attack. [31] Similar to the hijack example in section 4.3, the

[6] http://www.ae.utexas.edu/news/features/humphreys-research-group

spoofing detection can be improved by creating more sophisticated detection patterns for our reasoning unit, where such a preceding communication loss or other information like transients in GPS signals are taken into account.

5 Conclusion

We have extended our REALIZABLE, RESPONSIVE, UNOBTRUSIVE Unit 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 hardware and software. Our 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 check 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 synthesize 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.

References

1. Adapteva: The parallella board, https://www.parallella.org/board
2. Ahmed, A., Lisitsa, A., Dixon, C.: Testid: A high performance temporal intrusion detection system. In: ICIMP 2013, The Eighth International Conference on Internet Monitoring and Protection. pp. 20–26 (2013)
3. Ardupilot.com: APM:Plane, Open source fixed-wing aircraft UAV, http://plane.ardupilot.com
4. Bushnell, D., Denney, E., Enomoto, F., Pai, G., Schumann, J.: Preliminary Recommendations for the Collection, Storage, and Analysis of UAS Safety Data. Technical Report NASA/TM-2013-216624, NASA Ames Research Center, December (2013)
5. 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
6. Force, U.S.A.: 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
7. GAO: 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
8. Geist, J., Rozier, K.Y., Schumann, J.: Runtime observer pairs and bayesian network reasoners on-board FPGAs: Flight-certifiable system health management for embedded systems. In: Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings. pp. 215–230 (2014), http://dx.doi.org/10.1007/978-3-319-11164-3_18

9. Humphreys, T.: Statement on the vulnerability of civil unmanned aerial vehicles and other systems to civil GPS spoofing. University of Texas at Austin (July 18, 2012) (2012)
10. Javaid, A.Y., Sun, W., Devabhaktuni, V.K., Alam, M.: Cyber security threat analysis and modeling of an unmanned aerial vehicle system. In: Homeland Security (HST), 2012 IEEE Conference on Technologies for. pp. 585–590. IEEE (2012)
11. JSBSim: JSBSim - open source flight dynamics model, http://jsbsim.sourceforge.net
12. Karimi, N.: Iran drone capture claim: State tv airs images allegedly extracted from U.S. aircraft (video). The World Post (02 2013), http://www.huffingtonpost.com/2013/02/07/iran-drone-capture-claim_n_2636745.html
13. Kerns, A.J., Shepard, D.P., Bhatti, J.A., Humphreys, T.E.: Unmanned aircraft capture and control via GPS spoofing. Journal of Field Robotics 31(4), 617–636 (2014)
14. Kim, A., Wampler, B., Goppert, J., Hwang, I., Aldridge, H.: Cyber attack vulnerabilities analysis for unmanned aerial vehicles. Infotech@ Aerospace (2012)
15. Lu, H., Forin, A.: The design and implementation of p2v, an architecture for zero-overhead online verification of software programs. Tech. Rep. MSR-TR-2007-99, Microsoft Research (August 2007), http://research.microsoft.com/apps/pubs/default.aspx?id=70470
16. MAVLink: Micro air vehicle protocol, https://github.com/mavlink
17. MAVProxy: A UAV ground station software package for mavlink based systems, http://tridge.github.io/MAVProxy
18. Mengshoel, O.J., Chavira, M., Cascio, K., Poll, S., Darwiche, A., Uckun, S.: Probabilistic model-based diagnosis: An electrical power system case study. IEEE Trans. on Systems, Man and Cybernetics, Part A: Systems and Humans 40(5), 874–885 (2010)
19. Meredith, P.O., Jin, D., Griffith, D., Chen, F., Roşu, G.: An overview of the mop runtime verification framework. International Journal on Software Tools for Technology Transfer 14(3), 249–289 (2012)
20. Naldurg, P., Sen, K., Thati, P.: A temporal logic based framework for intrusion detection. In: FORTE, LNCS, vol. 3235, pp. 359–376. Springer (2004)
21. Olivain, J., Goubault-Larrecq, J.: The orchids intrusion detection tool. In: CAV, LNCS, vol. 3576, pp. 286–290. Springer (2005)
22. Pearl, J.: A constraint propagation approach to probabilistic reasoning. In: UAI. pp. 31–42. AUAI Press (1985)
23. Pellizzoni, R., Meredith, P., Caccamo, M., Rosu, G.: Hardware runtime monitoring for dependable COTS-based real-time embedded systems. RTSS pp. 481–491 (2008)
24. Perry, S.: Subcommittee hearing: Unmanned aerial system threats: Exploring security implications and mitigation technologies. Committee on Homeland Security (March 2015), http://homeland.house.gov/hearing/subcommittee-hearing-unmanned-aerial-system-threats-exploring-security-implications-a
25. Prince, B.: Air traffic control systems vulnerabilities could make for unfriendly skies [black hat]. Security Week (July 2012), http://www.securityweek.com/air-traffic-control-systems-vulnerabilities-could-make-unfriendly-skies-black-hat
26. Reinbacher, T., Rozier, K.Y., Schumann, J.: Temporal-logic based runtime observer pairs for system health management of real-time systems. In: Ábrahám, E., Havelund, K. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8413, pp. 357–372. Springer (2014), http://dx.doi.org/10.1007/978-3-642-54862-8_24

27. Schumann, J., Mbaya, T., Mengshoel, O.J., Pipatsrisawat, K., Srivastava, A., Choi, A., Darwiche, A.: Software health management with Bayesian networks. Innovations in Systems and Software Engineering 9(2), 1–22 (2013)
28. Schumann, J., Rozier, K.Y., Reinbacher, T., Mengshoel, O.J., Mbaya, T., Ippolito, C.: 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) (October 2013)
29. Schumann, J., Rozier, K.Y., Reinbacher, T., Mengshoel, O.J., Mbaya, T., Ippolito, C.: Towards real-time, on-board, hardware-supported sensor and software health management for unmanned aerial systems. International Journal of Prognostics and Health Management p. to appear (October 2015)
30. 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/
31. Shepard, D.P., Bhatti, J.A., Humphreys, T.E.: Drone hack. GPS World 23(8), 30–33 (2012)

6 TODO's

```
2. Writing assignments (ALL)   -- writing ONLY
     * abstract:   done
     * Introduction: done
     * Related work: done
     * Contributions: done
     * Background: done; KYR can do polishing/shortening if needed
     * Approach: material there; needs writing and formulation: Sang/Kristin
     * Experimental:  done; needs polishing, figure
     * Experimental:  dangerous commands/oscillations: done: needs formula and updated fi
     * Experimental: Hijack:  material and text there; needs polishing/streamlining. Figu
     * Experimental: GPS spoofing   TODO   Patrick/Johann
     * Conclusions:  Kristin (in progress)

FIGS and STUFF:

* Fig 2B: necessary????
* Fig 3: wrapfigure
* Fig 4A ?????
* Fig 4B: Johann
* Fig 5: update w/formulas
* Fig 6: Johann
* Fig 7: ?????? but interesting
* hijack example, mchtest du da auch eine "matlab" grafik machen?
Falls ja, das habe ich die rtr2u2 daten vom testflug als .csv und .odt
in 13_* eingecheckt.
Die interessanten CSV Spalten wren (beginnend bei 0):

11: "illegal packets received" (Indication for brute force decryption)
12: "Unusual periodic commands" (like mode changes - Indication of
ongoing attack)
14,15,16,17: "Unusual commands during flight"

* Fig 8: Add start time for spoofing

\begin{verbatim}
3. READING/POLISHING:
     * Intro:   DONE
     * Related work: DONE
     * Background: done?
     * Approach: Johann
     * Experimental: Sang/Kristin
     * conclusions: Kristin (in progress)
```