THE NATIONAL UNIVERSITY
of SINGAPORE


School of Computing
Computing 1, 13 Computing Drive, Singapore 117417

**TRA1/17**

# *Security Analysis of Unmanned Aircraft Systems*

**Manh-Dung Nguyen, Naipeng Dong and Abhik Roychoudhury**

*January 2017*

# T e c h n i c a l  R e p o r t

## Foreword

*This technical report contains a research paper, development or tutorial article, which has been submitted for publication in a journal or for consideration by the commissioning organization. The report represents the ideas of its author, and should not be taken as the official views of the School or the University. Any discussion of the content of the report should be sent to the author, at the address shown on the cover.*

Mohan KANKANHALLI
Dean of School

# Security Analysis of Unmanned Aircraft Systems

Manh-Dung Nguyen, Naipeng Dong, Abhik Roychoudhury

National University of Singapore, Singapore

**Abstract.** Security is a big concern in widely adopting security critical systems, such as Unmanned Aerial Vehicles (UAV). To ensure security of a system, the first step is to identify the required security properties as well as the potential attacks, i.e., security requirements. We identify a set of security requirements for UAV systems which is more complete and in more details than existing works. To facilitate formal analysis of a system against the set of requirements, we propose algorithms to automatically generate attack trees from the requirement set for the developers and designers to have a better understanding of the potential risks of a UAV system. Moreover, we propose an algorithm to automatically generate attack models from the attack tree and associate each attack model with the expected security properties. Given a UAV system model representing the honest behavior of participants, we are able to verify whether the system suffers from some potential attacks, by feeding the automatically generated attack models and the UAV system model to a model checker to verify the associated security properties.

## 1 Introduction

With the development of technology related to automatic machine systems, such as energy, communication, computer vision, sensing and information processing, navigation and path planning, the unmanned vehicle systems have become prosperous. Popular unmanned vehicle systems include Unmanned Aerial Vehicle (UAV), unmanned ground vehicle, such as autonomous car, autonomous underwater vehicle, etc. Among them, the UAV system is an important branch, which attracts huge amount of expenditures in the past decade, as an integral part of military and government operations [23]. Recently, civil applications of UAV systems have drawn great attention in both academia and industry. A broad range of applications have been found in various areas, such as, detecting illegal narcotics [23], flooding impact studies [12] and forest firefighting operations [23]. There is also increasing demand for commercial use of UAVs. For instance, Amazon's project *Prime Air* [1], Google's project *Wing* [6], Flirtey and NASA's project *Let's Fly Wisely* [2] and DHL and Airbus's project *Skyways* [3] aim at designing the future delivery system using UAVs.

Along with the bright future of UAVs is the big concern on the security of UAV systems, especially for the civil and commercial UAVs, as various types of vulnerabilities and attacks have been identified (e.g., [17,20,18,14,21]). Ensuring security has becoming increasingly necessary and important for the wide adoption of UAV systems.

To do so, the first step is to precisely define the meaning of security in UAV systems, that is, to specify a set of security requirements. However, during the literature study, we observe that 1) each work describes only a small part of security requirements for specific systems, 2) the security requirements of UAV systems are specified in various formats at different levels of detail with different assumptions. To ensure the security of a system, all possible attacks shall be taken into account, as one attack could break the entire system. Therefore, we aim to identify a set of rather complete security requirements for the general UAV systems and provide a unique format that allows us to specify security requirements originally in different formats.

With the set of identified security requirements, we can analyze whether a UAV system is secure. Compared to informal analysis such as testing, formal analysis has shown its strength in finding subtle and counter-intuitive flaws due to its preciseness and has been successfully applied to analyze security and reliability of various systems. To formally analyze a system, we need to formally specify the system as well as the security requirements. A security requirement consists of two parts: the attack behavior and the desired security property under the attack. We automatically generate the attack behavior from the set of security requirements and use reachability and Linear Temporal Logic (LTL) properties to model the corresponding security property.

In more details, we first collect a set of potential attacks on UAV systems. By analyzing the relations of the attacks, we identify the atomic attack behaviors such that an existing attack is a combination of a set of atomic attacks. We represent all the possible combinations of the atomic attacks in a formal specification – the *attack tree*, which is a graphical view of all possible attack behaviors and thus provides great convenient for non-security-experts to understand all the potential risks of a UAV system. The relations of the attacks in the tree facilitate the formal modeling of the complex attack behaviors. We propose algorithms to automatically generate an attack tree from the set of security requirements, and then automatically generate the attack models from the attack tree and associate each attack model with the expected security properties. Each attack is modeled using the formal language CSP# and each security property is formalized as either a reachability or a temporal property. Feeding the system model, attack model and the desired property, we use the model checker Process Analysis Toolkit (PAT) [28] to verify whether the system satisfies the property under the attack.

Furthermore, we validate our approach by analyzing the security of the Paparazzi UAV system [5], especially communication links between its components.

Our contributions are threefold:

- We identify a set of security requirements of UAV systems. The security requirement set is rather complete in the sense that various subcomponents in a general UAV system are considered, instead of focusing on part of the system or a specific system. In addition, new security requirements are identified by considering all combination of atomic attacks.

– We propose algorithms to automatically derive attack trees from the set of security requirements, and then automatically generate attack models with the corresponding security properties from the attack trees.
– We validate our approach by performing a case study on the Paparazzi UAV system and provide suggestions to improve the security of that system.

The remaining part of this paper is organized as follows. Section 2 presents a brief survey of UAV systems and their security. Next, we illustrate an overview of our method in Section 3. Each part of the method is discussed in details in the next three sections (Section 4 - 5 - 6). Then, in Section 7, we present the case study and the analysis results. Finally, Section 8 concludes the paper and discusses about the future works.

## 2 Background

In this section, we show the general structure of a typical UAV system and discuss the existing works on security requirements of UAV systems.

### 2.1 General Structure of UAV Systems

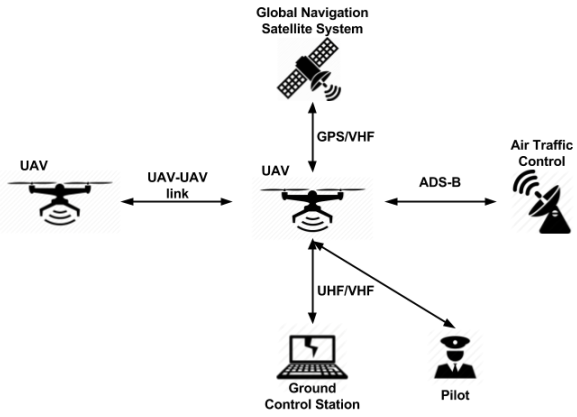


**Fig. 1.** General structure of a UAV system

Fig.1 shows a general UAV system involving various components, such as UAVs, Ground Control Station (GCS), Air Traffic Control (ATC) and Global Navigation Satellite System (GNSS). As the central component of the system, UAVs communicate with other components via different types of communication channels, for example, radio communication, UAV – UAV and satellite link. The GCS controls a UAV in flight, uploads new mission commands and setting parameters, collects and processes information of the UAV. In some systems, system staffs and pilots are required to handle critical tasks and interrupt the system in emergency situations.

### 2.2 State-of-art for Security Requirements in UAV Systems

As security analysis of UAV systems is still in its infancy, there are only a few works that addressed security requirements of UAV systems and thus we include the security requirements for aircraft system in general. We discuss the security requirements in the following aspects: identification methods, representation and concrete requirements.

*Security Requirements Identification.* Andrew J. Kornecki *et al.* [19] identify safety and security requirements of the gateway of the Next Generation Air Traffic Management system (NextGen) by using the Fault Tree Analysis technique. Ahmad Y. Javaid *et al.* [17] use Cyber-Security Threat Model to express various security threats that compromise confidentiality, integrity and availability of a UAV system. This work also performs risk analysis determining the likelihood, impact and risk of each threat.

*Security Requirements Representation.* Security requirements of aircraft systems in existing works are expressed in various formats, such as in natural languages (e.g., [19,27,13]) and in tree form (e.g., [17]). In addition, security requirement format can be template and tabular models (e.g., use cases), graphical notations (e.g., UML models and business process models) and formal specifications (e.g., Z and algebraic specifications).

*Security Requirements of Aircraft Systems.* In [19], five security requirements focusing on potential hazards related to potential threats impacting communication of the gateway of the NextGen system are specified in natural language. In the survey [27], Martin Strohmeier *et al.* summarize the vulnerabilities and attacks including severity and complexity on Automatic Dependent Surveillance Broadcast (ADS-B) which is currently the communications protocol of modern air traffic control. Then, several security primitives in air traffic control systems are explicitly identified. Raja Naeem Akram *et al.* discuss the adversary model for UAVs fleets and define a list of security requirements deriving from functional requirements [13].

In the brief literature study of UAV systems and their security requirements, we observe the following points.

- Security requirements are often implicit and specific to a particular system.
- Security standards, e.g., Common Criteria for Information Technology Security Evaluation [10] are too general to be directly applied for formal analysis.
- Security requirements are specified at different levels of details with different assumptions.
- Security requirements are expressed in various formats. Thus, it is challenging to combine existing requirements from different sources to make a complete set of security requirements.

Therefore, we aim to identify a set of explicit security requirements for general UAV systems and specify them in a format such that it provides insights of the relations of various existing security requirements.

## 3 Overview

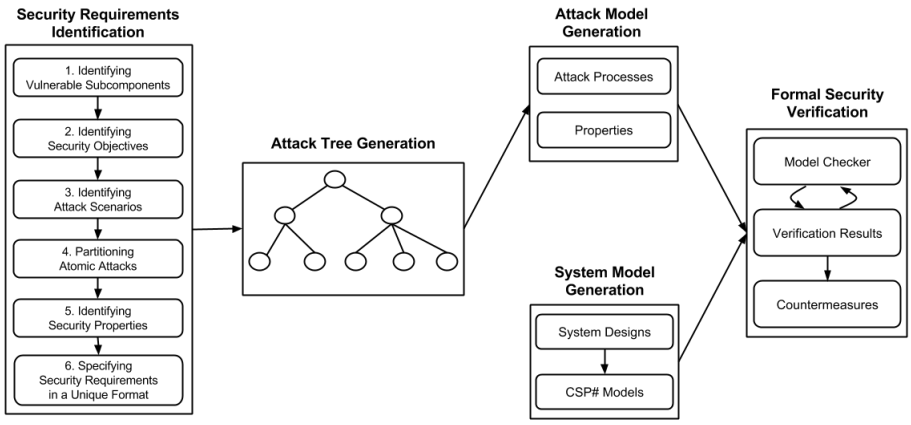


**Fig. 2.** An overview of our methodology

Fig. 2 shows an overview our approach which contains the following three main processes: security requirement identification, attack tree generation and attack model generation.

### 3.1 Security Requirements Identification

UAV systems are 1) complex as they consist of a large number of autonomous subcomponents and 2) dynamic as the subcomponents may vary according to the unpredictable environment. To study security requirements of a general UAV system, we first identify the subcomponents of the UAV systems (Step 1 in Fig. 2) that may be potentially vulnerable. In addition to addressing the complexity and dynamicity of the UAV systems, this step also provides flexibility to extend the security requirements by adding subcomponents, and allows the reuse of the security requirements to other unmanned systems, like unmanned ground systems and unmanned underwater systems, by replacing some subcomponents.

For each subcomponent, we identify both the required security guarantee (Step 2 in Fig. 2) and the attack behaviors that can be applied to that subcomponent (Step 3 in Fig. 2), assuming the environment of the UAV systems may be malicious. The corresponding relation between the attack and the security guarantee is kept in security properties (Step 5 in Fig. 2), with the intuition that the security objectives hold under this attack.

The set of collected attack behaviors may have complicated relations, for instance, two attacks may have the same results, one attack may rely on the other, one attack may overlap with another on a smaller attack action, etc. To better represent the relations between attacks, we partition each attack as smaller attack steps (Step 4 in Fig. 2), such that each attack is a combination of atomic attacks. The exhaustive combination of all atomic actions may also help

to identify a new attack that is not in the collected attack set. In addition, this enables the automatic attack tree generation and attack model generation.

Finally, we represent each security requirement in a unique format, such that all security requirements and attacks from different sources can be specified (Step 6 in Fig. 2) and from which the attack tree can be automatically generated.

### 3.2 Attack Tree Generation

An attack tree [25] is conceptual diagram showing how a system may be compromised. In an attack tree, the root is the system and other nodes show the sub-attacks. The children nodes of one intermediate node have either "AND" relation or "OR" relation. In our case, the UAV systems are the target and thus being the root of the tree.

Since we categorize the set of security requirements according to the vulnerable subcomponents, the root naturally contains a set of children nodes representing the subcomponents and their combinations. The relations between the children nodes are "OR" relations, since whenever there is a subcomponent suffering from an attack, the entire system suffers from the attack too. For each node representing a subcomponent, there is a set of potential attacks, which forms the grand children nodes of the root. The relations between the grand children nodes are "OR" relations as well, since as long as one attack is possible, the subcomponent suffers from an attack. Each attack node may be a combination of many atomic attacks. Thus, each attack node has further children nodes where the leaf nodes are the atomic attacks. The relations between the further children nodes can be "AND" or "OR", depending on the specific behavior of the attacks.

### 3.3 Attack Model Generation

Given the attack tree representing all potential attacks, we generate the attack model for each attack. The basic steps are as follows: first, we model the atomic attacks (the leaf nodes) as a set of atomic processes, which are often very short, containing only a few events. For each attack (an intermediate node having a subtree), we construct a process representing the attack from the set of atomic processes corresponding to the leaf nodes in the subtree. The construction procedure is essentially representing a parent node by combining the children nodes' processes with "AND" or "OR" relations. "AND" relation is modeled by the interleaving of the two processes and "OR" relation is captured by the choice relation of two processes in CSP#. Along with the attack model, we automatically generate the associated security properties, by union the security properties of the corresponding leaf nodes.

Finally, given a system model, we combine it with a generated attack model in parallel to represent the system running in an untrusted environment. By feeding the combined model into a model checker, we can verify whether the corresponding security properties are satisfied. If the model does not satisfy a security property, the model checker reports a counter-example showing the sequence of attack actions, which helps to develop a counter-measure.

## 4 Security Requirements

In this section, we present the results of each step in the security requirement identification process.

***Step 1 – Identifying Vulnerable Subcomponents.*** We have identified the following subcomponents: the UAVs, the GCS, the pilot and communication channels among all the subcomponents.

***Step 2 – Identifying Security Objectives.*** We have identified four general classes of security objectives: availability, confidentiality, authentication and integrity. Availability ensures that all of the services are always available. Confidentiality guarantees that secret information or sensitive data is never revealed to unintended entities. Authentication ensures that an attacker cannot masquerade any honest participant to gain unauthorized access to the system resources. Finally, integrity prevents an adversary to manipulate critical data by insertion, deletion or modification.

***Step 3 – Identifying Attack Scenarios.*** Unlike security objectives, security attacks are events that have a negative effect on system security by unauthorized accessing and modifying of secret data, information, services, networks and devices of the systems. Different subcomponents have different vulnerabilities and thus are targets of different attacks. For each subcomponent, we collected and identified a set of potential attacks [17,21,16,20,18,27].

**Table 1.** Potential atomic attacks on UAV systems

| Subcomponent | Atomic Attacks | Security Objectives |
|---|---|---|
| Channel UAV-GCS | [id1] Jamming | Availability |
| | [id2] Flooding | |
| | [id3] Denial of Service | |
| | [id4] Buffer Overflow | |
| | [id5] Eavesdropping | Confidentiality Authentication |
| | [id6] Identity Spoofing | |
| | [id7] Hijacking | |
| | [id8] Protocol Analysis | |
| | [id9] Message Injection | Integrity |
| | [id10] Replay Attack | |
| | [id11] Message Modification | Integrity |
| | [id12] Message Deletion | Integrity |
| Channel UAV-ATC | [id13] Jamming | Availability |
| | [id14] Eavesdropping | Confidentiality Authentication |
| | [id15] Message Injection | Integrity |
| | [id16] Message Modification | Integrity |
| | [id17] Message Deletion | Integrity |
| Channel UAV-GNSS | [id18] GPS Jamming | Availability |
| | [id19] GPS Spoofing | Confidentiality Integrity |
| UAV | [id20] Battery Exhaustion Attack | Availability |
| | [id21] Fuzzing Attack | |
| | [id22] Physical Component Warning Suppression | |
| GCS | [id23] Malware | Confidentiality Availability Integrity Authentication |
| | [id24] Virus | |
| | [id25] Keyloggers | |
| | [id26] Trojans | |
| | [id27] System Vulnerabilities | Confidentiality Availability Authentication |
| | [id28] Attacks via USB Connection | Confidentiality Integrity |
| | [id29] Denial of Service | Availability |
| Pilot | [id30] Social Engineering | Confidentiality Authentication |
| | [id31] Wrong Acts | Confidentiality |
| | [id32] Surveillance Attack | Confidentiality |

***Step 4 – Partitioning Atomic Attacks.*** To represent the attacks in a uniformed manner, we break the attacks into atomic attack events shown in Table 1.

In the attack breaking procedure, we also analyze the impacts of each attack, such as, what the attacker can achieve using the attacks. If two attacks have the same attacking results, for example, both jamming the signals and blocking the communication violate the availability of the satellite services, we annotate that the two attacks have the same goal.

***Step 5 – Identifying Security Properties.*** A security property states what the system must hold when an attack occurs in nature language, providing intuition to each security requirement identified by an attack and a corresponding security objective. For example, given an attack "Jamming" where the attacker aims to break the communication link between UAVs and the GCS, and a security objective "Availability", the following property needs to be hold – "The communication is available even if an attacker tries to jam the channel UAV – GCS". In a security property, the security objectives are specified in more details, providing more information when applying the security requirement in a concrete system. Such additional information depends on specific systems and thus is only provided in nature language.

***Step 6 – Specifying Security Requirements in a Unique Format.*** We use a tuple of the following form to represent the above mentioned elements of a security requirement: $\langle id, subcomponent, atomicattack, objective, samegoal, property \rangle$.

- *id*. The *id* distinguishes different security requirements.
- *subcomponent*. Different vulnerable subcomponents of UAV systems are represented in *subcomponent* (Step 1).
- *atomicattack*. It corresponds to the atomic attack name (Step $3-4$).
- *objective*. This element denotes security objectives that may be violated (Step 2). In addition, *objective* can be used to determine what security properties UAV systems need to satisfy.
- *samegoal*. To achieve a specific goal, such as breaking the functionality or gaining access to sensitive information of a *subcomponent*, the attacker can perform various types of attacks, thus we classify a set of atomic attacks leading to the same attacker's goal in *samegoal* (Step 4).
- *property*. To ensure the security of UAV systems, *property* must be satisfied in the presence of *atomicattack* (Step 5).

The set of security requirements can be found in [8]. Fig. 3 shows an example of one security requirement of the channel between UAVs and the GCS in JSON format representing the tuple discussed above.

```
{
  "id": "id1",
  "subcomponent": "Channel UAV-GCS",
  "atomicattack": "Jamming",
  "objective": "Availability",
  "samegoal": [ "id2", "id3", "id4" ],
  "property": "The communication is available even if an attacker tries to
      jam the channel UAV-GCS"
}
```

**Fig. 3.** Example of JSON format of security requirements

## 5 Attack Tree Generation

To visualize all the attacks that can be derived from the set of security requirements, we use the attack tree to graphically represent the attacks and their relations. Fig. 4 shows an attack subtree that is automatically generated from a subset of security requirements of the channel between UAVs and the GCS. The attack subtree of each subcomponent of UAV systems can be found in [8].

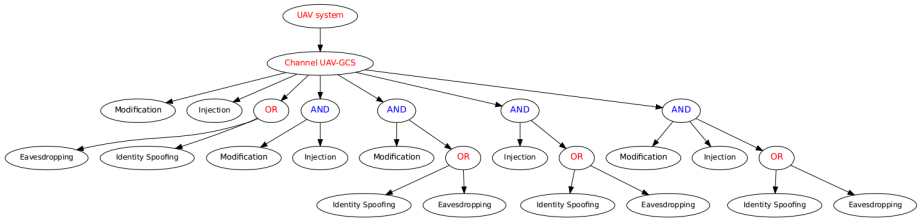


**Fig. 4.** Example of an attack subtree from 4 security requirements

Algorithm 1 takes the set of well-formated security requirements in JSON format as input and builds an attack tree from top to bottom. In more details, the main goal of the attackers is the entire UAV system and thus we add a "root" node with label "UAV system".

**Algorithm 1:** Attack Tree Generation from Security Requirements

**Data**: The set of security requirements in JSON format $secReqs[]$
**Result**: An attack tree $attackTree$

$attackTree \leftarrow \emptyset$;
$setSubcomps[] \leftarrow$ getSubcomps($secReqs[]$); `// Collect a set of subcomponents`
$setNodes[] \leftarrow \emptyset$; `// Store leaf nodes and OR intermediate nodes`
$attackTree \leftarrow$ addNode($root$);
**foreach** $subcomponent \in setSubcomps[]$ **do**
    $attackTree \leftarrow$ addNode($subcomponent$); addEdge($root, subcomponent$);
    `// Leaf nodes in black`
    **foreach** $secReq \in secReqs[]$ **do**
        **if** $secReq.samegoal == \emptyset$ **then**
            $attackTree \leftarrow$ addNode($secReq$); addEdge($subcomponent, secReq$);
            $setNodes[] \leftarrow secReq$;
    `// Map a security requirement to a set of nodes sharing the same goal to avoid redundant OR nodes, e.g., [‘id1’:{‘id1’, ‘id2’, ‘id3’}, ‘id4’:{‘id4’, ‘id5’}]`
    $sr2SameGoal[] \leftarrow$ getSameGoal($secReqs[]$);
    `// OR intermediate nodes in red`
    **foreach** $sr \in sr2SameGoal[]$ **do**
        $attackTree \leftarrow$ addNode($orNode$); addEdge($subcomponent, orNode$);
        **foreach** $node \in sr2SameGoal[sr]$ **do**
            $attackTree \leftarrow$ addNode($node$); addEdge($orNode, node$);
        $setNodes[] \leftarrow orNode$;
    `// Generate all the combinations, e.g., [[‘id1’, ‘id2’], [‘id1’, ‘OR1’], [‘id2’, ‘OR1’], [‘id1’, ‘id2’, ‘OR1’]]`
    $combineNodes[] \leftarrow$ makeCombinations($setNodes[]$);
    `// AND intermediate nodes in blue`
    **foreach** $combinedNode \in combineNodes[]$ **do**
        **if** len($combinedNode$) $> 1$ **then**
            $attackTree \leftarrow$ addNode($andNode$); addEdge($subcompnent, andNode$);
            **foreach** $node \in combinedNode$ **do**
                $attackTree \leftarrow$ addNode($node$); addEdge($andNode, node$);
**return** $attackTree$;

For each subcomponent that is collected from security requirements, we add a "subcomponent" node as a child node of the root node. Note that, we distinguish two kinds of intermediate nodes by its label color (e.g., red for OR nodes and blue for AND nodes). Then, we construct a subtree of each "subcomponent" node by considering three cases as follows:

- *Leaf nodes in black.* This kind of nodes represents atomic attacks that can be performed separately in a specific way.
- *OR intermediate nodes in red.* We add OR intermediate nodes to represent all atomic attacks sharing the same goal as defined in *samegoal* of each security requirement.
- *AND intermediate nodes in blue.* To cover all potential attacks that can happen to UAV systems, AND intermediate nodes are used to represent all possible combinations of the two previous types of nodes.

## 6 Formal Verification and Attack Model Generation

With the applicable attacks from the hostile environment, we formally verify whether a system satisfies the desired security objectives. We model the behavior of the system and the attackers using the CSP# formal language which is supported by the model checker PAT, because it allows not only high-level operations but low-level procedural codes in C#, which is useful for modeling non-trivial data and functions.

The syntax of a subset of the CSP# [28] is as follows.

$$P, Q ::= Stop \mid Skip \mid a \rightarrow P \mid P\,;Q \mid P \mid\mid\mid Q \mid P \,\Box\, Q \mid P \parallel Q \mid P \,\triangleleft b \triangleright\, Q \mid$$
$$c!e \rightarrow P \mid c?e \rightarrow P \mid e\{assign\_exp\} \rightarrow P.$$

*Stop* denotes that the process is in the state of deadlock; *Skip* denotes a process which terminates successfully; $a \rightarrow P$ describes an object which first engages in the event $a$ and then behaves exactly as described by $P$; $P\,;Q$ describes the case where $P$ and $Q$ execute in sequence; $P \,\Box\, Q$ denotes the general choice between $P$ and $Q$; $P \mid\mid\mid Q$ denotes that $P$ and $Q$ run concurrently without barrier synchronization, where $\mid\mid\mid$ denotes interleaving; $P \parallel Q$ denotes a process consisting of two parallel processes – the two processes execute concurrently and are synchronized with the same communication events; $P \,\triangleleft b \triangleright\, Q$ denotes the conditional choice – if the value of $b$ is true then it behaves like $P$ otherwise like $Q$; $c!e \rightarrow P$ sends a value $e$ through channel $c$ and then behaves like process $P$; $c?e \rightarrow P$ receives a value through channel $c$ and stores the value in variable $x$ and then the behavior is like $P$; $e\{assign\_exp\} \rightarrow P$ denotes that a sequence of assignments may be attached to an event by one atomic operation.

### 6.1 Attack Model Generation

Differing from the system model which depends on the target system, the attack models can be constructed from the generated attack tree in the previous section. That is, we generate CSP# processes that model each node in the attack tree. Since there are many attack combinations, manually generating the attack

processes are tedious work. We propose Algorithm 2 which takes the attack tree, the modeling of the atomic attacks as well as a set of atomic attacks as inputs and generates the attack process combining all the selected atomic attacks.

This algorithm uses a bottom-up Breadth-First Search (BFS) to traverse the tree. For a set of sibling nodes $n_1, \ldots, n_n$ with the same parent, denoted as processes $P_{n_1}, \ldots, P_{n_n}$ respectively, if the nodes have "AND" relation, the parent node is the process that combines $P_{n_1}, \ldots, P_{n_n}$ using the interleaving operation $|||$, meaning that in order to achieve the parent node, all the sibling nodes are satisfied; if the nodes have "OR" relation, the parent node is represented as a process in which $P_{n_1}, \ldots, P_{n_n}$ are combined with the operation $\Box$, capturing that as long as one of the sibling node is satisfied, the parent node is satisfied. Formally,

- *AND relation.* $P_{parent} = P_{n_1} ||| \ldots ||| P_{n_n}$
- *OR relation.* $P_{parent} = P_{n_1} \Box \ldots \Box P_{n_n}$

**Algorithm 2:** Attack Model & Property Generation

**Data**: A set of attack scenarios $attacks[]$ which are leaf nodes of the attack tree. $attack_i$ is modeled as the process $P_{attack_i}$ and has a set of properties $Q_{attack_i}[]$.
**Result**: The attack process $P_{attack}$ & the set of properties $Q_{verify}[]$

```
1  unvisitedAttacks[] ← ∅ // The set of attacks that have not been visited
2  currentAttack ← ∅ // The attack currently being examined
3  currentParent ← ∅ // The parent node of the current attack
   // Initialize attack processes & properties for all nodes in the attack tree
4  initProcessProp(attacks[]);
5  foreach attack ∈ attacks[] do
6  |  unvisitedAttacks.enqueue(attack);
7  while unvisitedAttacks[] ≠ ∅ do
8  |  currentAttack ← unvisitedAttacks.dequeue();
9  |  currentParent ← getParent(currentAttack);
10 |  if currentParent ≠ ∅ then
11 |  |  P_currentParent ← P_currentAttack;
12 |  |  Q_currentParent[] ← Q_currentAttack[];
13 |  |  currentChilds[] ← getChilds(currentParent);
14 |  |  if getRelation(currentParent) == AND then
   |  |  |  // AND relations
15 |  |  |  foreach andChild ∈ currentChilds[] do
   |  |  |  |  // Assume that all child attacks must be performed at the same time to
   |  |  |  |     ensure attack success
16 |  |  |  |  if andChild ≠ currentAttack then
17 |  |  |  |  |  P_currentParent = (P_currentParent ||| P_andChild);
18 |  |  |  |  |  Q_currentParent[].union(Q_andChild[]);
19 |  |  |  |  |  unvisitedAttacks.remove(andChild);
20 |  |  else
   |  |  |  // OR relations
21 |  |  |  foreach orChild ∈ currentChilds[] do
22 |  |  |  |  if orChild ∈ unvisitedAttacks[] then
23 |  |  |  |  |  P_currentParent = (P_currentParent □ P_orChild);
24 |  |  |  |  |  Q_currentParent[].union(Q_orChild[]);
25 |  |  |  |  |  unvisitedAttacks.remove(orChild);
26 |  |  unvisitedAttacks.enqueue(currentParent);
27 |  else
   |  |  // Root node
28 |  |  P_attack ← P_currentAttack;
29 |  |  Q_verify[] ← Q_currentAttack[];
30 |  |  unvisitedAttacks.remove(currentAttack);
31 return P_attack & Q_verify[];
```

### 6.2 Attack Property Generation

Recall that each attack has a set of related security properties. When constructing a combined attack, we need to construct the related security properties. Along with the attack model generation in Algorithm 2, we employ the set union operator for updating the set of security properties of the parent node. In the case of "AND" relation, the security properties of a parent node are the union of the security properties of the children nodes. In the case of "OR" relation, the sibling nodes have the same set of security properties, since they achieve the same security results. If the sibling nodes with "OR" relation have different security properties, we union them for the parent node but with a notation on each security property denoting the applicable attack, e.g., the choice of the sibling attack for that security property. Finally, the set of properties of the root node contains all properties that we need to verify to enhance system security.

Now, given a process $Systems$ modeling a UAV system and chose one generated attack process $P_{attack}$, we have a set of security properties $Q_F = \{q_1, q_2, ..., q_n\}$. We verify whether the system is secure against this attack by verifying the following formula:

$$(Systems \parallel P_{attack}) \vDash q_i, \forall\, q_i \in Q_F.$$

## 7 Case Study

### 7.1 Overview

Paparazzi UAV system [5] is an open-source drone hardware and software project for multiple types of unmanned aircraft. Recently, it has been used by various institutes for scientific research. Fig. 5 shows the global overview of the system.

The airborne segment consists of an aircraft, its hardware and all embedded software to control the flight. The ground station includes a laptop, a radio modem and ground software that are used to prepare, monitor and analyze the flight. There are two types of communication links and protocols between the ground and airborne segments. The first one is a bi-directional wireless data link including both telemetry (down link) and telecontrol (up link), which is used in the autonomous mode. The second one is a safety link that is used to provide manual control of the aircraft.


**Fig. 5.** Overview of the Paparazzi UAV system [5]

### 7.2 System Modeling

The Paparazzi UAV system is very large and complex and thus cannot be modeled completely in the paper. In our simple model in Fig. 6, the Paparazzi system consists of three main components, including the UAV, the GCS and the

Joystick. Each main component of the system is modeled as a CSP# process. Specifically, the UAV is devided into several subcomponents, such as the Flight Plan Processor, the Data link Radio Modem and the Radio Control Receiver, and each subcomponent is also modeled as a CSP# process. In addition, we model the data link and the safety link as the synchronous CSP# channels.

Since MAVLink [4] – a lightweight, header-only message marshaling library for micro air vehicles is added to Paparazzi autopilot [11], we use MAVLink protocol as the data link between UAV and the Paparazzi Center. The UAV can send messages to the GCS to update its current status (e.g., battery status, system status, position) or request and retrieve a mission including a list of waypoints via the data link. From the structure of MAVLink packet, the MAVLink messages have the following structure $SYSID.MSGID.DATA$ when we abstract away the unnecessary information. Note that, $SYSID$ is used to identify the drone who sends or is expected to receive the messages.

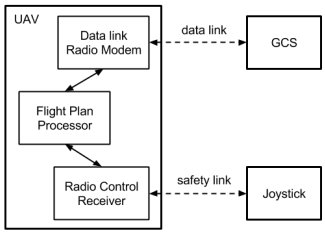


**Fig. 6.** Paparazzi UAV system models in CSP#

In autonomous mode, the UAV follows a pre-programmed flight plan. The UAV behavior in this mode is modeled in the *FlightPlanProcessor* process. In this process, we define a variable *uavStatus* to represent the elementary operations of the flight plan, e.g., WAIT_GPS, GEO_INIT, TAKE_OFF, LAND. In manual mode, we model the joystick that sends basic motion commands to the UAV via the safety link, e.g, ROLL, THROTTLE, PITCH to manually control the drone. Finally, the complete model of the Paparazzi UAV system is as the following:

$$Paparazzi() = UAV() \;|||\; GCS() \;|||\; Joystick();$$

### 7.3 Attack Modeling & Properties

We illustrate the formal verification in this case study using the network attacks. Recall that the channels may suffer from the following atomic attacks: eavesdropping all messages, injecting, deleting and modifying unencrypted messages on network. We show how these atomic attacks together with their related properties are modeled as follows.

*Eavesdropping attack.* The attacker learns all the messages including the *SYSID*. If we would like to know whether the $SYSID$ is leaked, we assign *true* to the variable *kSYSID* when the attacker knows it and query the following assertion.

$$\begin{aligned}
&P_eavesdropping() = datalink?ur_sysid.ur_msgid.ur_data \\
&\quad \rightarrow eavesdropping\{\ att_sysid = ur_sysid\ ; att_msgid = ur_msgid; \\
&\qquad\qquad att_data = ur_data\ ; kSYSID = true;\ \} \\
&\quad \rightarrow P_eavesdropping(); \\
&\#define\ propEveasdropping(kSYSID\ ==\ false); \\
&\#assert\ Total_systems() \models [] propEveasdropping;
\end{aligned}$$

*Message modification attack.* After knowing about the *SYSID* of the target drone, the attacker can control that vehicle by capturing the target waypoint sent from the GCS, modifying it and then sending the modified packet to the UAV such that the drone will go to the attacker position. We model the attack and query whether the UAV can be controlled as follows.

$$
\begin{aligned}
&P\_modification() = \\
&\quad datalink?gwp\_sysid.gwp\_msgid.gwpr\_payload \rightarrow modification\{ \\
&\quad if(gwp\_sysid == UAV\_ID \;\&\&\; gwp\_msgid == WAYPOINT)\{ \\
&\qquad gwps\_payload = att\_waypoint;\}\} \\
&\quad \rightarrow datalink!att\_sysid.WAYPOINT.gwps\_payload \rightarrow P\_modification(); \\
&\#define\ propModification(recvWP\ !=\ att\_waypoint); \\
&\#assert\ Total\_systems() \models [] propModification;
\end{aligned}
$$

*Message injection attack.* Attackers also can inject a fake waypoint into the data link such that the drone will go the wrong way.

$$
\begin{aligned}
&P\_injection() = \\
&\quad [att\_sysid == UAV\_ID]\ datalink!att\_sysid.WAYPOINT.fake\_waypoint \\
&\quad \rightarrow P\_injection(); \\
&\#define\ propInjection(recvCorrectWP\ ==\ true); \\
&\#assert\ Total\_systems() \models [] propInjection;
\end{aligned}
$$

Note that, the total system is modeled as the following:

$$Total\_systems() = Paparazzi() \parallel P\_attack();$$

where $P\_attack()$ is either an atomic attack defined as above or a combination of the above atomic attacks. Due to space limit, we do not list all the attack processes, and we only show a few properties which are modeled as assertions. The system model, the attack models and properties can be found in [9].

### 7.4 Verification Results

We use PAT [28] to check the system model against several attack models to analyze the security of the simplified Paparazzi UAV system. By using our approach, we can verify that the Paparazzi UAV system is vulnerable to well-known networking attacks [22], since the communication channels like the MAVLink protocol version 1.0 is unauthenticated and unencrypted.

One possible countermeasure is to employ encryption mechanisms and authentication techniques on communication channels, for example RC5 [15], Caesar Cipher [24] or AES Block Cipher [26]. Furthermore, a security-enhanced version of the MAVLink protocol, named Secure MAVLink (SMAVLink) [7], should be considered to be integrated into the Paparazzi UAV system.

## 8 Conclusion

In this work, we identify a set of atomic attacks for UAV systems, from which all potential attacks can be derived. The potential attacks are automatically generated and represented in a tree form. To facilitate formal verification of UAV systems, we automatically generate the attack models and associate them with the corresponding security properties. An obvious next step is to apply our approach to verify designs of other real unmanned systems (e.g., the Skyways project) in early stages to enhance their security and safety.

**Acknowledgments** The authors would like to thank Jin-Song Dong for useful discussions. We also thank our collaborator of the Skyways project, *Airbus*, to provide part of the security and reliability requirements of the UAV systems.

## References


1. Amazon Prime Air, https://www.amazon.com/b?node=8037720011
2. Australian startup Flirtey and NASA team to execute the first FAA-approved drone deliveries in America, https://goo.gl/P2Ma8T
3. Forget self-driving cars: Airbus wants to make self-FLYING taxis - and it could begin tests of its first prototype next year, https://goo.gl/DLxXk5
4. MAVLink Protocol, http://qgroundcontrol.org/mavlink/start
5. Paparazzi UAV System, https://wiki.paparazziuav.org/wiki/Main_Page
6. Project Wing, https://www.solveforx.com/wing/
7. Secure MAVLink (SMAVLink), https://goo.gl/5AXqic
8. Security requirements & attack trees of UAV systems, www.comp.nus.edu.sg/~dungnguy/nfm2017/requirements.zip
9. System model & attack models in CSP#, https://goo.gl/hIzHUc
10. The Common Criteria, http://www.commoncriteriaportal.org/
11. The module datalink of the Paparazzi UAV system, https://goo.gl/qiS2Dq
12. UAS Projects, https://goo.gl/JPBOnB
13. Akram, R., Bonnefoi, P.F., Chaumette, S., Markantonakis, K., Sauveron, D.: Secure autonomous uavs fleets by using new specific embedded secure elements. In: IEEE TrustCom-16 (2016)
14. Benchoff, B.: Hijacking drones with a MAVLink exploit (2014), http://resources.infosecinstitute.com/hack-proof-drones-possible-hacms-technology/
15. Butcher, N., Stewart, A., Biaz, S.: Securing the MAVLink Communication Protocol for Unmanned Aircraft Systems
16. Hartmann, K., Steup, C.: The vulnerability of UAVs to cyber attacks-An approach to the risk assessment. In: Cyber Conflict (CyCon), 2013 5th International Conference on. pp. 1–23. IEEE (2013)
17. Javaid, A.Y., Sun, W., Devabhaktuni, V.K., Alam, M.: Cyber security threat analysis and modeling of an unmanned aerial vehicle system. IEEE (2012)
18. Kim, A., Wampler, B., Goppert, J., Hwang, I., Aldridge, H.: Cyber attack vulnerabilities analysis for unmanned aerial vehicles. Infotech@ Aerospace (2012)
19. Kornecki, A.J., Liu, M.: Fault tree analysis for safety/security verification in aviation software. Electronics 2(1), 41–56 (2013)
20. Mansfield, K., Eveleigh, T., Holzer, T.H., Sarkani, S.: Unmanned aerial vehicle smart device ground control station cyber security threat model. IEEE (2013)
21. Mansfield, K.M., Eveleigh, T.J., Holzer, T.H., Sarkani, S.: DoD comprehensive military unmanned aerial vehicle smart device ground control station threat model. Tech. rep., DTIC Document (2015)
22. Marty, J.A.: Vulnerability analysis of the mavlink protocol for command and control of unmanned aircraft. Tech. rep., DTIC Document (2013)
23. Paczan, N.M., Cooper, J., Zakrzewski, E.: Integrating Unmanned Aircraft into NextGen Automation Systems. Tech. rep. (2012)
24. Rajatha, B., Ananda, C., Nagaraj, S.: Authentication of MAV communication using Caesar Cipher cryptography. IEEE (2015)
25. Schneier, B.: Attack Trees: Modeling Security Threats. Dr.Dobb's Journal (1999)
26. Sparrow, R.D., Adekunle, A.A., Berry, R.J., Farnish, R.J.: A novel block cipher design paradigm for secured communication. In: 2016 Annual IEEE Systems Conference (SysCon) (2016)
27. Strohmeier, M., Lenders, V., Martinovic, I.: On the security of the automatic dependent surveillance-broadcast protocol. IEEE Communications Surveys & Tutorials 17(2), 1066–1087 (2015)
28. Sun, J., Liu, Y., Dong, J.S.: PAT Language Syntax and Semantics