CS 6600 Project Report: Unmanned Aircraft System

Matthew Whitesides and Bruce M. McMillin
Department of Computer Science
Missouri University of Science and Technology, Rolla, MO 65409-0350

Abstract—The abstract goes here.

1 Infrastructure

1.1 Infrastructure Description

In its simplest form, an unmanned aircraft system (UAS), is an aircraft system that operates without an onboard pilot. UAS are either controlled remotely through some form of wireless radio communication, semi-autonomously in conjunction with a remote pilot, or fully autonomously using some form of computational intelligence as navigation. These intelligent crewless vehicles have many potential uses in the defense sector, including surveillance, strategic mission execution, aerial sustainability support, and training systems, to name a few. These aircraft fall under the umbrella of cyber-physical systems (CPS), merging the intelligent navigation processes, system health monitoring, and communication with the physical mobile aerial vehicle.

We will design a proposed infrastructure security policy for the Boeing MQ-25 unmanned aircraft system, but it will also apply to similar mission support drones. The MQ-25 is an unmanned aircraft system designed for the U.S. Navy, and it provides autonomous refueling capability for the Boeing F/A-18 Super Hornet, Boeing EA-18G Growler, and Lockheed Martin F-35C fighters. This capability extends the combat range of the supported aircraft, seamlessly and semi-anonymously navigating to the plane, refueling, and returning to base. MQ-25 is the first unmanned aircraft to support aerial refueling another aircraft and is currently in the flight test phase of development [1], making it the perfect system to analyze security impacts for current and future unmanned aircraft systems.

1.1.1 Infrastructure Security Policy

Our infrastructure security policy breaks down the various actions a system user can perform using the following terms.

- *Subject*: Any entity that contains the proper rights can request the UAS perform operations, access objects, or grant rights to another subject.
- *Object*: An entity that is part of the UAS functionality or data that does not have control over another entity.
- *Rights*: A property assigned to a subject that defines its right to access an object or grant permissions to another subject.

Table 1 describes the rights and their associated functionality. Table 2 breaks down the subject roles involved in operating the UAS during a refueling mission. Table 3 contains the access control matrix (ACM) showing each Subject's rights over the objects.

TABLE 1
Description of rights over objects in the UAS.

Right	Description
Owns (O)	The owner of the given object.
Read (R)	Can observe the given object.
Write (W)	Can modify the given object.
Execute (E)	Can execute the functionality of the given object.
Grant (G)	Can grant a given right to another subject.
Control (C)	Can control a given system object.
Delete (C)	Can delete a given object or right.
Create (C)	Can create a new subject or object.

1.2 HRU

The Harrison, Ruzzo, Ullman security model (HRU) establishes a finite set of mono-operational procedures our system can perform on subjects and objects. Given our set of rights and ACM, we will establish a set of commands available to the system that acts upon the subjects and objects in the system. The commands will consist of mono-operational modifications and pre-condition checks. Therefore given these sets, we can discover if any rights leakages can occur.

The following shows the basic HRU commands related to our UAS mission. A fundamental UAS refueling mission follows these basic steps.

1) UAS is verified flight-ready by the MC.
2) The PC plans the mission.
3) The PC and IP execute the mission.
4) During the flight, the PC and IP execute the ANC and RO as needed.
5) After the refueling operation, the UAS returns to base, and the flight is debriefed.
6) PC, IP, or MC download the flight data from the UAS flight recorder.

TABLE 2
Description of actor subject roles during a UAS refueling mission.

Subject	Description
Pilot Commander (PC)	The primary remote pilot of the UAS during the mission.
Instructor Pilot (IP)	Assists the Pilot Commander and can pilot the UAS if given permission from the PC or MC.
Maintenance Crew (MC)	Handles work orders created by the PC, IP, or FDA, responsible for the maintenance of the UAS.
Flight Data Admin (FDA)	Handles and analyses all mission flight data.
External Contractor (Bad Actor) (EC)	Has a similar job to the MC however only has read rights to the FED.

TABLE 3
Initial UAS Refueling Mission Access Control Matrix

	PC	*IP*	*MC*	*FDA*	*EC*	**ANC**	**RO**	**FED**	**RTD**	**FRS**
Pilot Commander (PC)	O,R,W	R,W	R,W	R,W	R,W	O,R,W,E,G,C	O,R,W,E,G,C	R,W,E,G,C	R,W,E,G,C	R,W,E,G,C
Instructor Pilot (IP)	R	O,R,W	∅	∅	∅	R,W,E,C	R,W,E,C	R,E	R,E	R,E
Maintenance Crew (MC)	∅	∅	O,R,W	R	R	∅	∅	R,E	R,E	R,E
Flight Data Admin (FDA)	∅	∅	R	O,R,W	O,R,W	∅	∅	O,R,W,E,G,C	O,R,W,E,G,C	O,R,W,E,G,C
Maintenance Contractor (Bad Actor) (EC)	∅	∅	∅	∅	O,R,W	∅	∅	∅	R	∅
Autonomous Navigation Control (ANC)	∅	∅	∅	∅	∅	R,W,E,C	R	R	R	∅
Refueling Operation (RO)	∅	∅	∅	∅	∅	∅	R,W,E,C	R	R	∅
Flight Engine Data (FED)	∅	∅	∅	∅	∅	∅	∅	R,W,E,C	∅	∅
Refueling Tank Data (RTD)	∅	∅	∅	∅	∅	∅	∅	∅	R,W,E,C	∅
Flight Record System (FRS)	∅	∅	∅	∅	∅	∅	∅	∅	∅	R,W,E,C

TABLE 4
ACM After Create Flight Record

	PC	*IP*	*MC*	*FDA*	*EC*	**ANC**	**RO**	**FED**	**RTD**	**FRS**	**FR**
PC	O,R,W	R,W	R,W	R,W	R,W	O,R,W,E,G,C	O,R,W,E,G,C	R,W,E,G,C	R,W,E,G,C	R,W,E,G,C	R,W,E,G,C
IP	R	O,R,W	∅	∅	∅	R,W,E,C	R,W,E,C	R,E	R,E	R,E	R,E
MC	∅	∅	O,R,W	R	R	∅	∅	R,E	R,E	R,E	R,E
FDA	∅	∅	R	O,R,W	O,R,W	∅	∅	O,R,W,E,G,C	O,R,W,E,G,C	O,R,W,E,G,C	O,R,W,E,G,C
EC	∅	∅	∅	∅	O,R,W	∅	∅	∅	R	E,W	∅
ANC	∅	∅	∅	∅	∅	R,W,E,C	R	R	R	∅	∅
RO	∅	∅	∅	∅	∅	∅	R,W,E,C	R	R	∅	∅
FED	∅	∅	∅	∅	∅	∅	∅	R,W,E,C	∅	∅	∅
RTD	∅	∅	∅	∅	∅	∅	∅	∅	R,W,E,C	∅	∅
FRS	∅	∅	∅	∅	∅	∅	∅	∅	∅	R,W,E,C	∅
FR	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	R,W,E,C

7) A flight record (FR) is created that contains flight tracking information, engine usage data, and various UAS health status.
8) This flight record is uploaded by the PC, IP, MC, or EC to the record-keeping system.
9) Mission is completed.

With this in mind, we have the following basic HRU commands available for post-flight maintenance (steps 5 - 9). We will then show how improper use of these commands can lead to a rights leakage with our external contractor standing in as our "bad actor" gaining a leak of Integrity and confidentiality rights beyond our initial ACM.

The first command is a generic *Grant r Rights* that allows a subject to grant any right they have over a subject/object to another subject/object.

```
command grant_r_right(r, o, p, q)
  if grant in A[p, o] and r in A[p, o]
  then
    enter r into A[q, o];
end
```

Next we have command *Make Owner* allowing a subject p to make another subject q the owner of a object o they currently have owner rights over.

```
command make_owner(p, q, o)
  if owns in A[p, o]
  then
    enter owns into A[q, o];
end
```

When the *PC* and *IP* return from a flight they or a *MC* will read the flight data and create a new object **Flight Record (FR)** holding the flight data, available to subjects who have **FED** access. After running this command, our

ACM would transition to a new state similar to Table 4, representing the subjects and objects involved in the create flight record procedure. This command will also give rights to the FRS to control the flight record data once uploaded.

```
command create_flight_record(p)
  if create in A[p, FED]
  then
    create object FR;
    enter own into A[p, FR];
    enter delete into A[p, FR];
    enter read into A[p, FR];
    enter grant into A[p, FR];
end
```

After creating a flight record which may be done by a PC, IP, or MC, they may choose to give the task of processing and uploading the flight record to an external maintenance contractor and execute the following command to provide them with access to the record. In the command *Grant Flight Record Access*, *p* is the subject granting the right to subject *q*, for the *fr* flight record.

```
command grant_flight_record_access(p, q,
    fr)
if own in A[p, fr]
then
  enter read into A[q, FR];
  enter write into A[q, FR];
  enter execute into A[q, FR];
end
```

Finally, the flight record needs to be uploaded to our flight record system using the following *Upload Flight Record* command with *p* being the subject executing the command and *fr* being flight record to upload.

```
command upload_flight_record(p, fr)
if own in A[p, fr] and read in A[p, FRS]
then
  enter read into A[FRS, FR];
  enter write into A[FRS, FR];
  enter execute into A[FRS, FR];
  enter control into A[FRS, FR];
end
```

Similar to *Create Flight Record* a subject may need to delete or update a flight record from the system.

```
command
    delete_flight_record_from_system(p, FR)
if delete in A[p, FR]
then
  delete read from A[FRS, FR];
  delete write from A[FRS, FR];
  delete execute from A[FRS, FR];
  delete control from A[FRS, FR];
end
```

If any modifications need to be made to the FRS, the following command begins an update transaction and ends one for a given subject and flight record.

```
command update_flight_record_system(p, FR)
if own in A[p, FR]
  then
    enter read into A[p, FED];
    enter read into A[p, FR];
    enter write into A[p, FED];
    enter write into A[p, FR];
  end
```

1.3 Rights Leakages

1.3.1 Confidentiality

For our example of a confidentiality attack, we simulate a scenario where our *External Contractor* is a bad actor seeking leaked rights beyond the initial ACM utilizing the following HRU commands available for the UAS.

The following sequence of commands is typical among a mission debriefing process.

```
create_flight_record(Maintenance Crew
    (MC));
grant_flight_record_access(Maintenance
    Crew, Maintenance Contractor (EC),
    Flight Record (FR));
upload_flight_record(Maintenance
    Contractor (EC), Flight Record (FR));
```

However, when running *upload_flight_record*, the EC will find they do not have "own" rights over the record, which is required so that they will go back to the original MC and as to grant them execute privileges to the FRS system. The MC will execute the commands to make EC have permission to upload the FR.

```
make_owner(MC, EC, FR);
command grant_r_right(execute, FRS, MC,
    EC);
```

This command, unfortunately, will lead to a leak as the contractor (EC) now has the execute privileges over the FRS, which they did not initially and is not intended and can lead to other flight records confidential information being exposed to the EC. Table 4 shows the leaked rights in red.

1.3.2 Integrity

In an attempt at an integrity leak, our EC may seek to modify existing data and, to get permission to do so, may attempt the following sequence.

```
create_flight_record(Maintenance Crew
    (MC));
upload_flight_record(Maintenance
    Contractor (EC), Flight Record (FR));
grant_flight_record_access(Maintenance
    Crew, Maintenance Contractor (EC),
    Flight Record (FR));
make_owner(MC, EC, FR);
update_flight_record_system(EC, FR);
```

However, upon reviewing the FR, the MC notices an issue in the data and wants the EC to update the flight record system. Unfortunately, *Update Flight Record System* checks only for rights to the given flight record and not the initial systems, therefore, giving the EC *write* access to the FRS system, which could lead them modifying the FRS data ruining its integrity. Table 4 shows the result of executing the confidentiality and integrity leaks.

Fig. 1. Initial Iteration T-G Model Graph

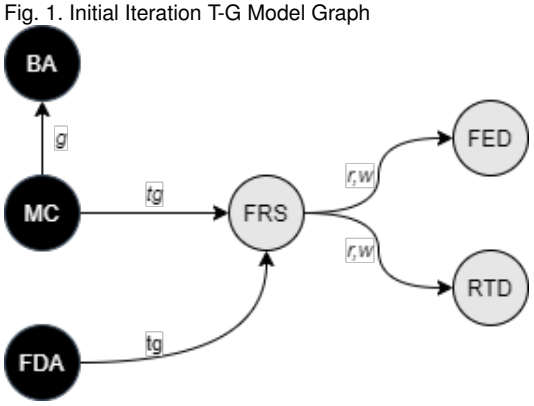
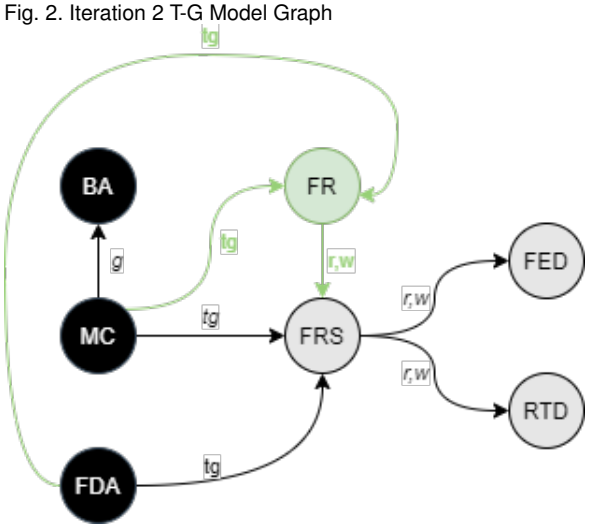


Fig. 2. Iteration 2 T-G Model Graph



2 Survey

2.1 Take-Grant

Next, we take a look at the Take-Grant (TG) model interpretation of our rights leakages. This models our HRU example in terms of a TG model where the *Flight Record* created object has *read, write* rights over the *Flight Record System* that our **Maintenance Contractor (EC)** can utilize to gain *read, write* access to the *Flight Engine Data*.

Our bad actor could achieve this leaked through the following TG commands.

1) **MC** creates object **FR**.
2) **MC** grants (*t* to **FR**) to **EC**.
3) **EC** takes (*r,w* to **FRS**) from **FR**.
4) **EC** takes (*r,w* to **FED**) from **FRS**.

Figure 1 shows the initial state of rights among actors and objects in the **FRS** system.

Figure 2 shows the state of rights after the **MC** runs the HRU command *create_flight_record(MC)* which a new **FR** object.

Figure 3 shows the state after the **EC** leads the **MC** into giving them *t* rights over **FR** they can exploit the system to take our laked rights to **FED** through the **FRS**. This would be established after the command *grant_flight_record_access(MC, EC, FR)*.

Fig. 3. Iteration 3 T-G Model Graph

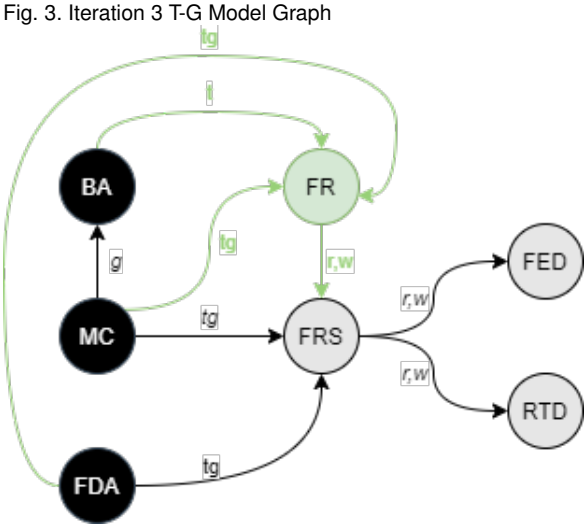
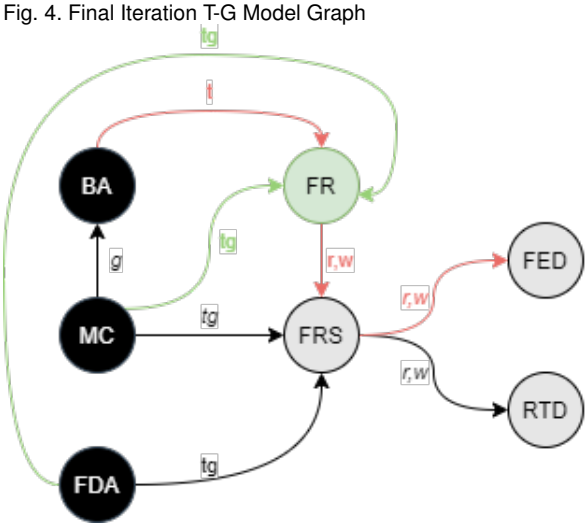


Fig. 4. Final Iteration T-G Model Graph



Therefore allowing EC to execute *update_flight_record_system(EC, FR)* which leaks access to the FED as shown by the state in Figure 4.

2.2 Bell-LaPadula

The Bell-LaPadula Model (BLP) focuses on establishing access control confidentiality through security levels and categories. BLP focuses on two simple rules in a security policy within a given ACM. The simple security property states a subject can not read an object at a higher security level. The star property states that a subject may not write to any object at a lower security level.

For our BLP implementation, we first define the security classifications for our subjects and objects. Given we are dealing with military records, it makes sense to use the standard government security levels. In addition to the security level, subjects and objects will fall into categories based upon the appropriate work unit. Table 5 shows the security clearance levels and the subjects/objects at the given levels. Table 6 shows the work unit categories of each subject/object.

TABLE 5
Security Classifications for the UAS

Clearance Level	Subjects	Objects
Top Secret (TS)	PC, FDA	ANC, RO
Secret (S)	IP	FRS, RTD
Confidential (C)	MC, EC	FED, FR
Unclassified (UC)		

TABLE 6
Work Unit Categories for the UAS

Category	Subjects	Objects
Flight Ops (FO)	PC,IP	ANC,RO
Maintenance Ops (MO)	PC,IP,MC,FDA	FED,RTD,FR,FRS
Flight Data Ops (FDO)	PC,IP,FDA,EC	FED,FR,FRS

Now that we've established security classifications for our subjects and objects, we can see how the system and security policy could still allow leakage and further theft of flight record data.

```
create_flight_record(Maintenance Crew
    (MC));
grant_flight_record_access(Maintenance
    Crew, Maintenance Contractor (EC),
    Flight Record (FR));
upload_flight_record(Maintenance
    Contractor (EC), Flight Record (FR));
make_owner(MC, EC, FR);
update_flight_record_system(EC, FR);
```

1) **MC** executes *create flight record* creating flight record object **FR**.
    - This write is allowed as the object **FR** created has clearance level of *Confidential* which is greater than or equal to **MC's** level of *Confidential* and **FR** dominates **MC** by categories.
2) **MC** executes *upload flight record* granting the object **FRS** access to the flight record object **FR**.
    - This write is allowed as the object **FRS** created has clearance level of *Secret* which is greater than **FR's** level of *Confidential* and **FRS** dominates **MC** by categories.
3) **MC** executes *grant flight record access* granting **EC** access flight record object **FR**.
    - This read is allowed as the subject **EC** created has clearance level of *Confidential* which is greater than or equal to **FR's** level of *Confidential* and **MC** dominates **EC** by categories.
4) **MC** executes *make owner* making **EC** have owner rights over flight record object **FR**.
    - This write is allowed as the subject **EC** created has clearance level of *Confidential* which is greater than or equal to **FR's** level of *Confidential* and **FR** dominates **EC** by categories.
5) **EC** executes *update flight record system* allowing **EC** write to the flight record system object **FRS**.
    - This write is allowed as the subject **EC** created has clearance level of *Confidential* which is less than or equal to **FRS's** level of *Secret* and **FRS** dominates **EC** by categories.

As you can see, while this system does keep our bad actor **EC** from reading the confidential flight records from the flight record system, it does not prevent an integrity attack of writing up to the **FRS** and **FR**.

REFERENCES

[1] A. Erwin, and J. Gibson, Navy, Boeing Make Aviation History with MQ-25 Becoming the First Unmanned Aircraft to Refuel Another Aircraft, Accessed on: Sept. 1, 2021. [Online]. Available: https://www.boeing.com/defense/mq25/

Matthew Whitesides Master's Student at Missouri University of Science and Technology.