THALES

Building a future we can all trust

FIRST WORLDWIDE HACKING DEMO OF AN IN-ORBIT SATELLITE

TRADING AN EXPERIMENT FOR A WHOLE OPS-SAT

Cyber Solutions by Thales

AGENDA

- The Team
- The Context
- Experimenters’ side (The Good)
- Attackers’ side (The Bad)
- Post Exploitation (The Ugly)
- Key takeaways
- Mitigation strategies

THE TEAM

In order of appearance:

- Brian: Cyber Security Evaluator @ Thales ITSEF
- Quentin: Reverse Engineer @ Thalium
- Guillaume: Reverse Engineer @ Thalium
- Arnaud: Reverse Engineer @ Thalium

- **Thalium**: Thales laboratory dedicated to cyberdefense, offensive security, vulnerabilities assessment and Red Team activities
- **Thales ITSEF**: Thales' Information Technology Security Evaluation Facility, specialized in independent security evaluation of components and embedded systems

A BIT OF CONTEXT

- Thales's offensive cybersecurity team took part in the Hack CYSAT 2023 challenge
- **Objective:** identify vulnerabilities on-board OPS-SAT that could enable malicious actors to disrupt satellite mission operations
- The results of the challenge will be used to:
  - Tighten satellite security and its on-board applications
  - Improve the cyber resilience of space systems
  - Support the long-term success of space programmes

THALES
Building a future we can all trust

Experimenter's access to OPS-SAT
starring: The Good, An Innocent Experimenter

Cyber Solutions by Thales

DEVELOPPING AN OPS-SAT EXPERIMENT 101

- Experiments on OPS-SAT run on the SEPP*
- Via the Nanosat Mission Operations Framework (NMF), an experiment can use of a range of services:
  - Camera, GPS, ADCS, …
  - Ground ↔ space communication
- **For starters, you just want to take some pretty pictures using the satellite:**
  1. Wait for the ADCS to be available (other experiments may be using it)
  2. Point the satellite along your target direction
  3. Take a picture with the camera

** Satellite Experimental Processing Platform*

DEVELOPPING AN OPS-SAT EXPERIMENT 101

- What you expect:

DEVELOPPING AN OPS-SAT EXPERIMENT 101

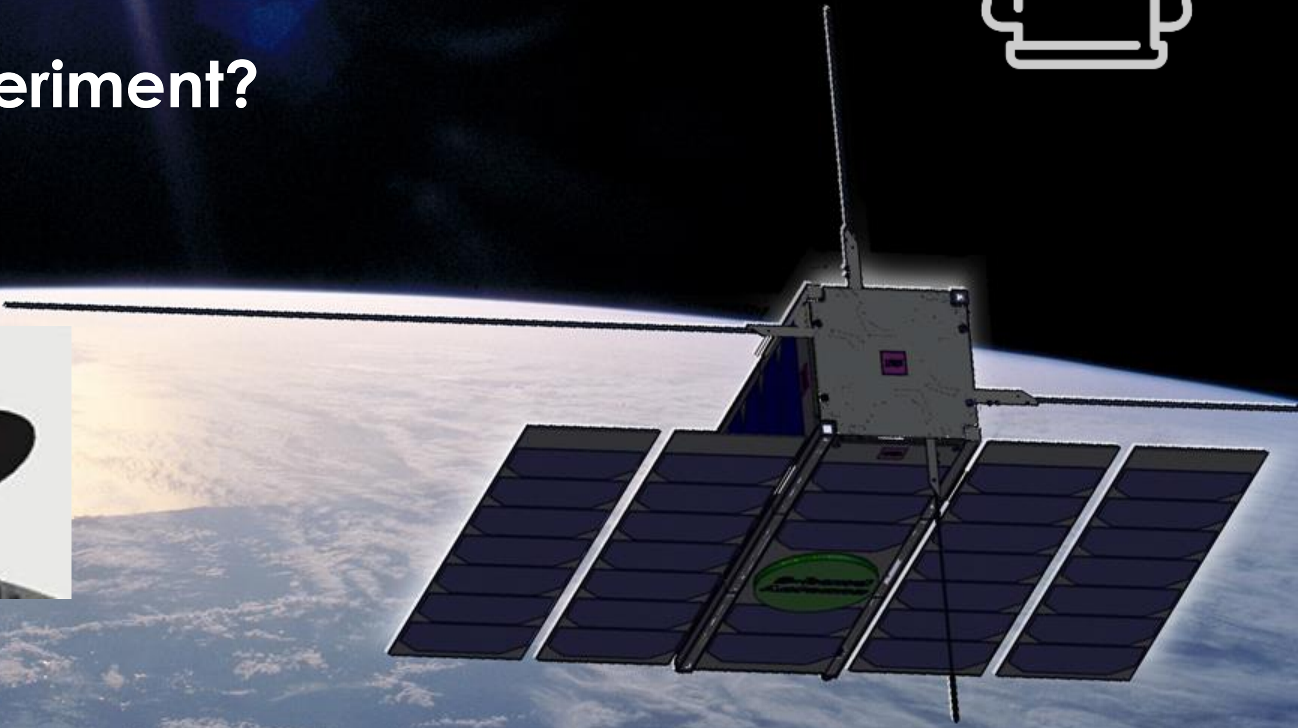
- What you actually get:

YOU WANT TO DEVELOP AN OPS-SAT APPLICATION

What happened?

THALES
Building a future we can all trust

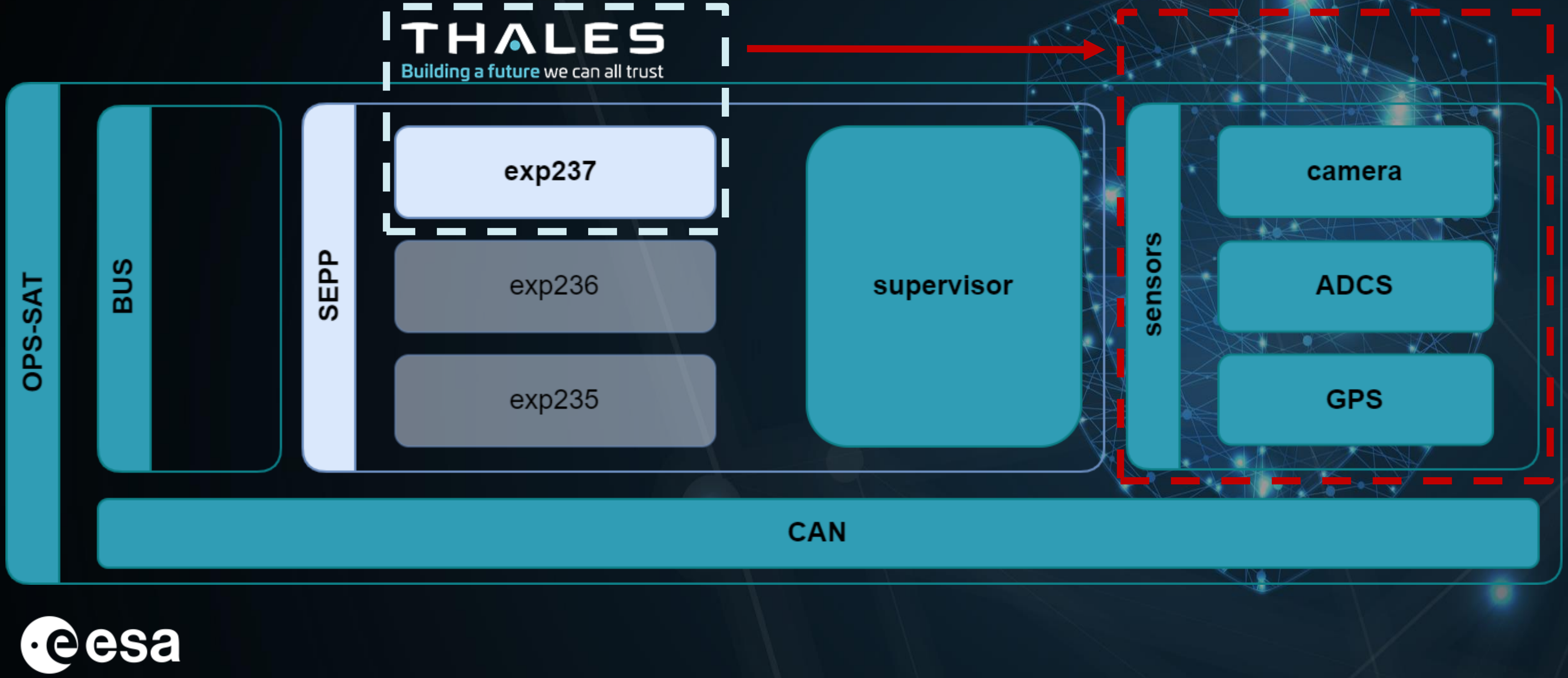
A malicious experiment?
starring: The Bad



ATTACKER'S OBJECTIVES

- **Take control** of OPS-SAT's sensors & actuators, for:
  - Disinformation: tamper with camera images, falsify sensor readings
  - Destruction: damage the platform and disrupt the mission
- Stay **undetected**
  - Our malicious code should not be detectable before upload on the satellite

THALES DEMO OBJECTIVE : TAKING CONTROL OF THE SENSORS



PROBLEM #1: STAY UNDETECTED

- Our experiment app relies on the supervisor to access OPS-SAT services
- But our app goes through a review process before running on the real satellite
- **How to evade this?** → Find a way to dynamically execute shell commands

- **Good starting point**: experiments can communicate with ground apps directly
- **Possible vectors**:
  - Abuse a command execution feature: existing (*CommandExecutor*) or ad-hoc
  - Leverage a vulnerability to exploit it: **existing (NMF*)** or ad-hoc

**Nanosat Mission Operations Framework used for the development of OPS-SAT experiments*

STAY UNDETECTED: DESERIALIZATION VULNERABILITY

- We submitted an innocuous-looking app

Derived from a sample NMF app: `hello-world-simple`

- It contains no **overtly** malicious code

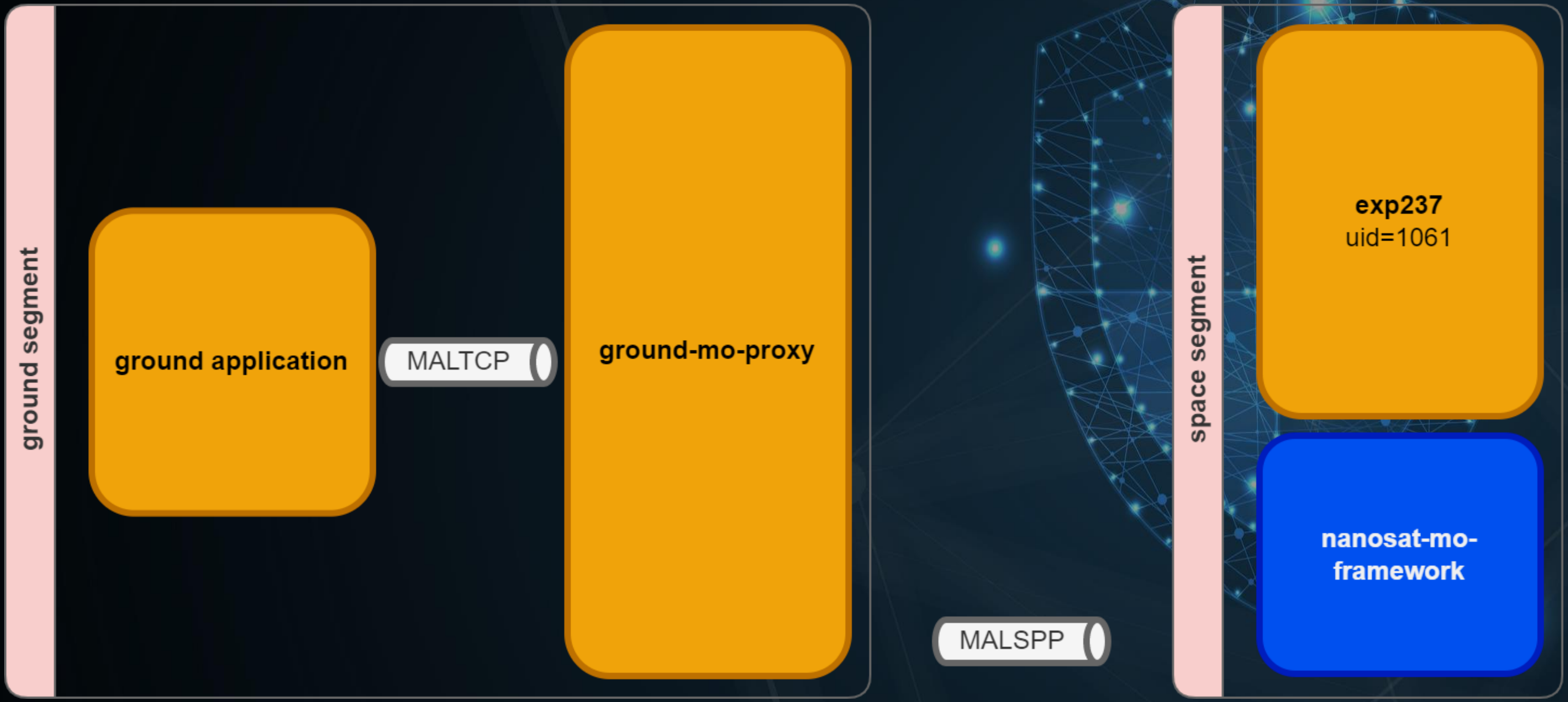
But there's a slight twist:

```
new Parameter("Dummy parameter", 1, /*…*/)
```

This exposes a vulnerability in NMF: **« unsafe Java deserialization »**

(a call to `readObject` *with attacker-controlled data)*

STAY UNDETECTED: GROUND APP COMMUNICATES WITH SPACE APP



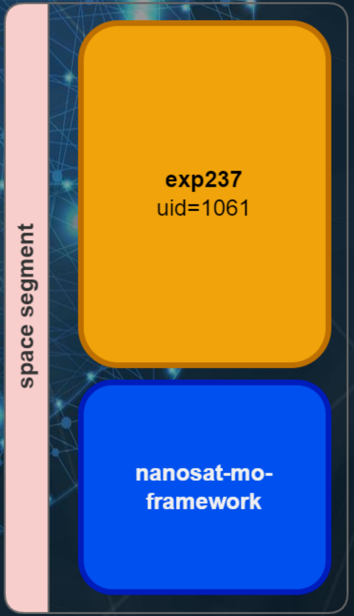
STAY UNDETECTED: LEVERAGING THE SAMPLES CODE BASE

NMF class

…

Blob parameter

```
public class MCAdapter extends SimpleMonitorAndControlAdapter {
  public void initialRegistrations(MCRegistration registrationObject) {
    ParameterDefinitionDetailsList pddl = new ParameterDefinitionDetailsList();
    IdentifierList names = new IdentifierList();

    pddl.add(new ParameterDefinitionDetails("The sent data", (byte) 1, ...));
    names.add(new Identifier("Data"));
    registrationObject.registerParameters(names, pddl);
  }

  public Serializable onGetValueSimple(String name) {
    AttributeValue aval = new AttributeValue();
    aval.setValue(new UInteger(1234));
    return aval;
  }

  public boolean actionArrivedSimple(String name, Serializable []srlzbls, Long l) {
    throw new UnsupportedOperationException("Not supported yet.");
  }

  public boolean onSetValueSimple(String name, Serializable srlzbl) {
    throw new UnsupportedOperationException("Not supported yet.");
  }
}
```



STAY UNDETECTED: JAVA DESERIALIZATION VULNERABILITY IN NMF

Receive the blob

…

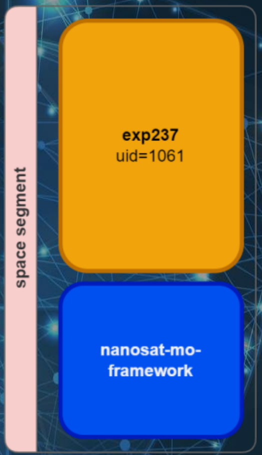
Unserialize!

esa.mo.nmf.MonitorAndControlNMFAdapter

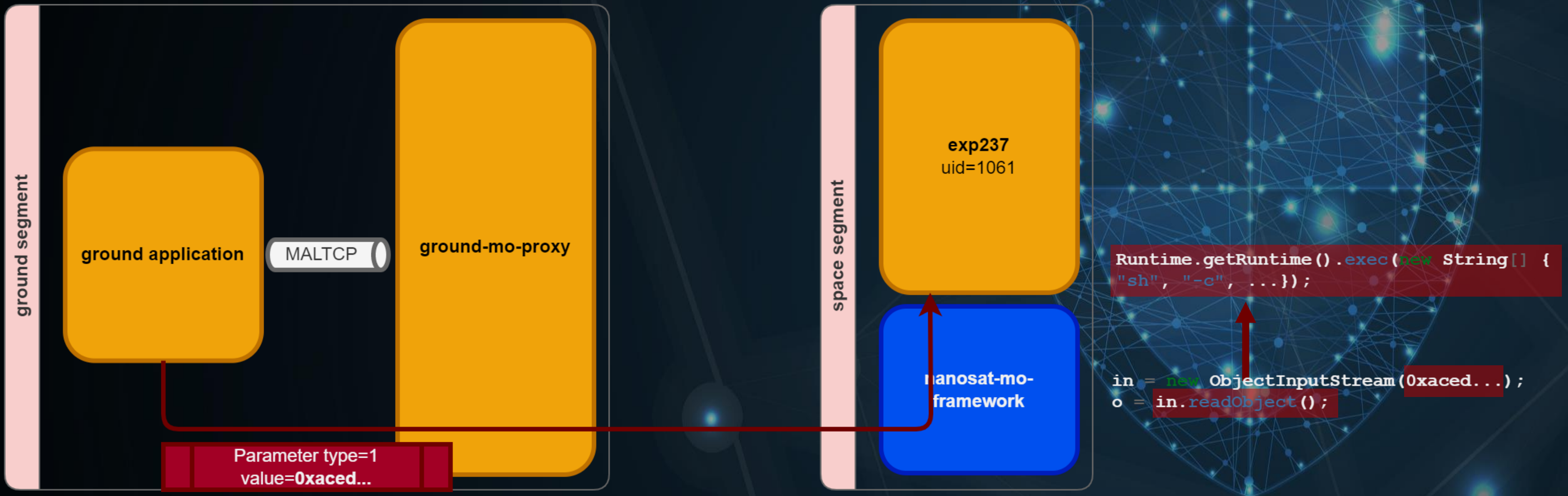
```java
public abstract class SimpleMonitorAndControlAdapter extends MonitorAndControlNMFAdapter implements
    SimpleMonitorAndControlListener {

    @Override
    public UInteger actionArrived(Identifier identifier, AttributeValueList attributeValues,
        Long actionInstanceObjId,
        boolean reportProgress, MALInteraction interaction) {
        Serializable[] values = new Serializable[attributeValues.size()];

        for (int i = 0; i < attributeValues.size(); i++) {
            AttributeValue attributeValue = attributeValues.get(i);

            if (attributeValue.getValue() instanceof Blob) {
                try {
                    values[i] = HelperAttributes.blobAttribute2serialObject(
                     (Blob) attributeValue.getValue());
                } catch (IOException ex) {
                    values[i] = attributeValue; // ...
                }
            } else {
                values[i] = attributeValue;
            }
        }
```

esa.mo.helpertools.helpers.HelperAttributes

```java
public static Serializable blobAttribute2serialObject(Blob obj) {
    if (obj == null) {
        throw new IllegalArgumentException("The Blob must not be null.");
    }
    ByteArrayInputStream bis = null;
    Object o = null;
    try {
        bis = new ByteArrayInputStream(obj.getValue());
        ObjectInput in = null;
        try {
            in = new ObjectInputStream(bis);
            o = in.readObject();
```



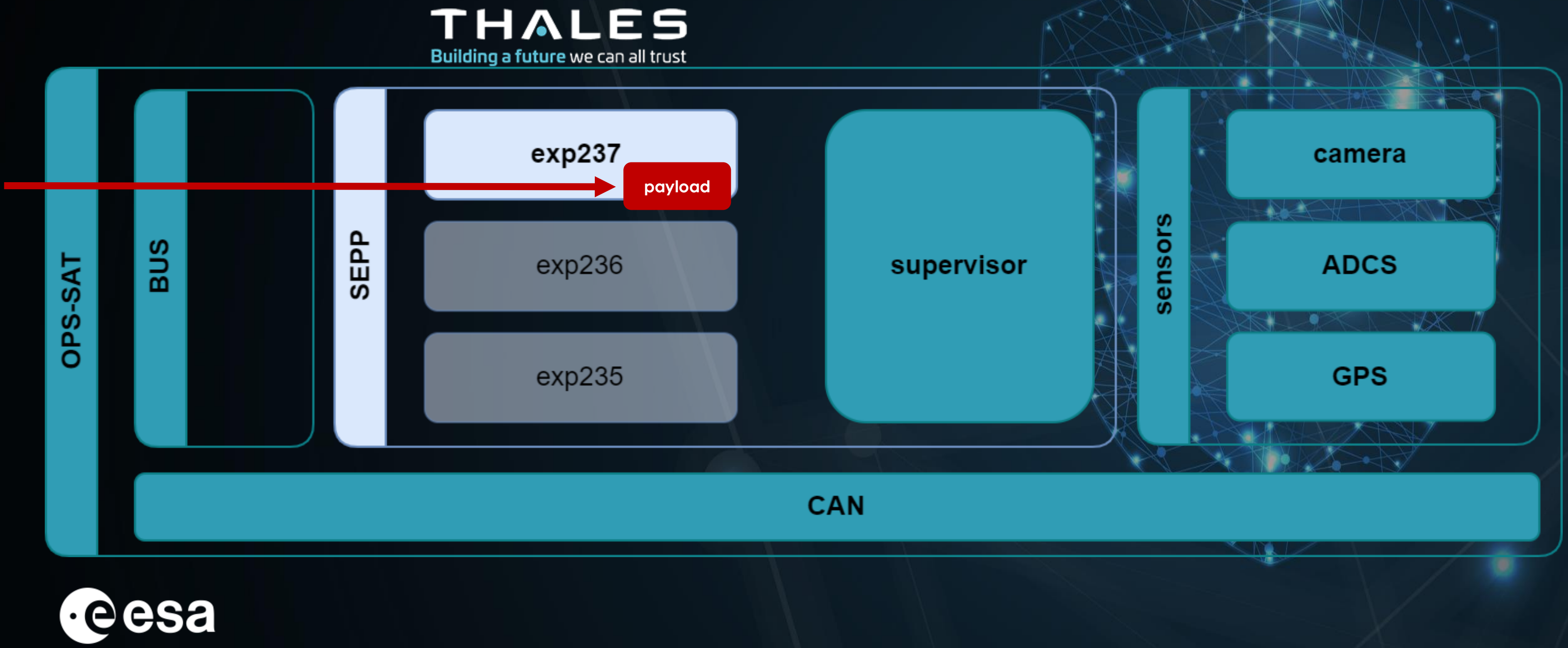
STAY UNDETECTED: EXECUTE ARBITRARY COMMANDS AS EXP237



STAY UNDETECTED: SUCCESS!

- We leveraged this vulnerability to design a **covert channel**, in cooperation with ESA
  - The exploit is sent from a ground app we developed: a parameter is sent to our space app
  - The malicious parameter payload is routed to space
  - Once received by our app, it triggers arbitrary code execution under the identity of our app
  - Yet this code doesn't appear in the binary files submitted for our experiment

STAY UNDETECTED: UNRESTRICTED PAYLOADS UPLOAD



PROBLEM #2: TAKING CONTROL OF THE SEPP

- Our app runs as an unprivileged Linux user
- It has no direct access to sensors and actuators, but though the supervisor
- **How to take control of them?**

- **Good starting point**: being root yields full privileges over the whole system
- **Possible vectors:**
  - Find system configuration issues
  - Exploit a 1-day vulnerability either user-space or kernel
  - **Find homebrew daemons running as root**

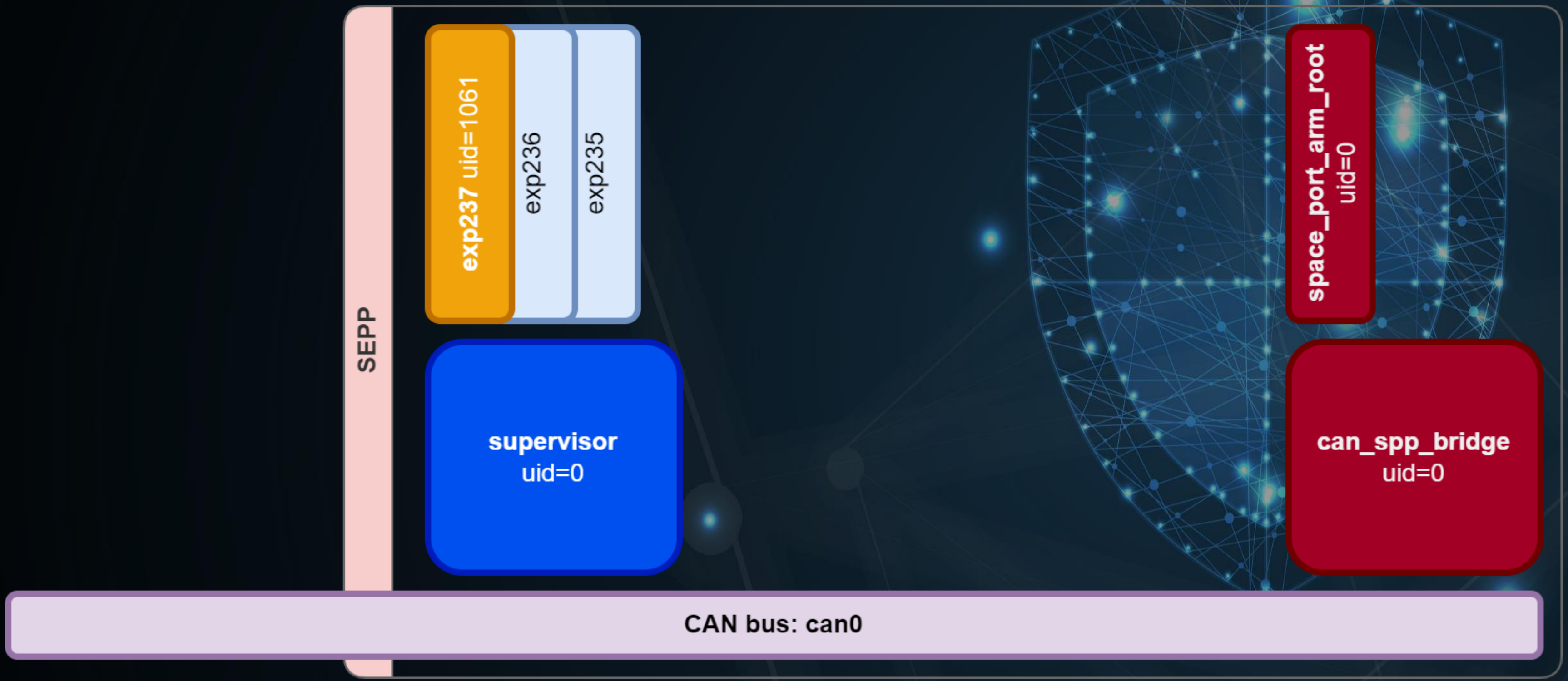
TAKING CONTROL: PRIVILEGE ESCALATION FROM USER TO ROOT

- The SEPP's supervisor controls access to the sensors for NMF apps
  - It runs as **root**
  - To take control of the sensors, we take control of their gatekeeper: the **supervisor**
  - To do so, we need to **escalate our privileges** from our user to root

- **There's an intriguing service running on the SEPP**: `space-shell-root`
  - We grabbed the binary & **reverse engineered** it
  - It's a client that decodes then **executes as root** whatever command it receives…
  - Anyone can talk on the CAN bus, including unprivileged apps

- **Thus… any app can send commands for the `space-shell-root` to run as root ☺**
  (this is OPS-SAT-specific, not NMF-related)

TAKING CONTROL: CAN BUS VULNERABILITY



TAKING CONTROL: NICE LOOKING FEATURE!

```
while (true) {
  int32_t n_received = receiveData(&received, 0xfa, r0_5, 1);
  if (n_received > 0) {
    char* ciphered_ptr = &received;
    char* const xor_ptr = &XOR_KEY;
    while (true) {
      k = xor_ptr[0];
      ciphered_ptr[0] = (k ^ ciphered_ptr[0]);
      if (ciphered_ptr == &received[(n_received - 1)]) break;
      xor_ptr += 1;
    }
    received[n_received] = 0;
    signal(SIGCHLD, 0x1 /* SIG_IGN */);
    pid_t child = fork();
    if (child == 0) {
      execl("/bin/sh", "/bin/sh", "-c", &received);
      exit(0);
      /* no return */
    }
  }
  ...
```

TAKING CONTROL: NICE LOOKING FEATURE!

Received Data

…

XORed with key

…

Executed as root!

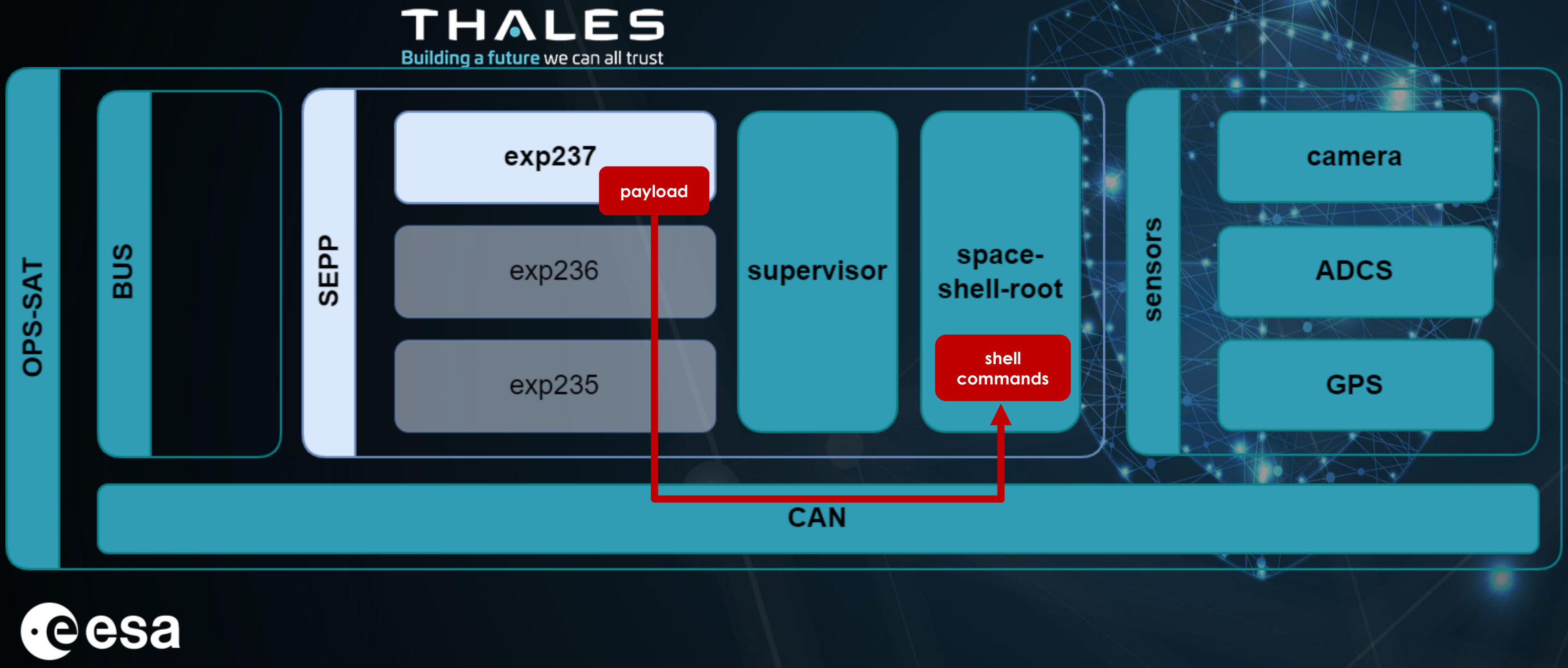
```
while (true) {
 int32_t n_received = receiveData(&received, 0xfa, r0_5, 1);
 if (n_received > 0) {
   char* ciphered_ptr = &received;
   char* const xor_ptr = &XOR_KEY;
   while (true) {
     k = xor_ptr[0];
     ciphered_ptr[0] = (k ^ ciphered_ptr[0]);
     if (ciphered_ptr == &received[(n_received - 1)]) break;
     xor_ptr += 1;
   }
   received[n_received] = 0;
   signal(SIGCHLD, 0x1 /* SIG_IGN */);
   pid_t child = fork();
   if (child == 0) {
     execl("/bin/sh", "/bin/sh", "-c", &received);
     exit(0);
     /* no return */
   }
 }
 ...
```

space_port_arm_root
uid=0

can_spp_bridge
uid=0

CAN bus: can0

TAKING CONTROL: ARBITRARY CODE EXECUTION AS ROOT



PROBLEM #3: PERSISTENCE

- Our app escalated as root
- **How to ensure persistent effects** on sensors and actuators ?

- **Good starting point:** apps use the NMF framework

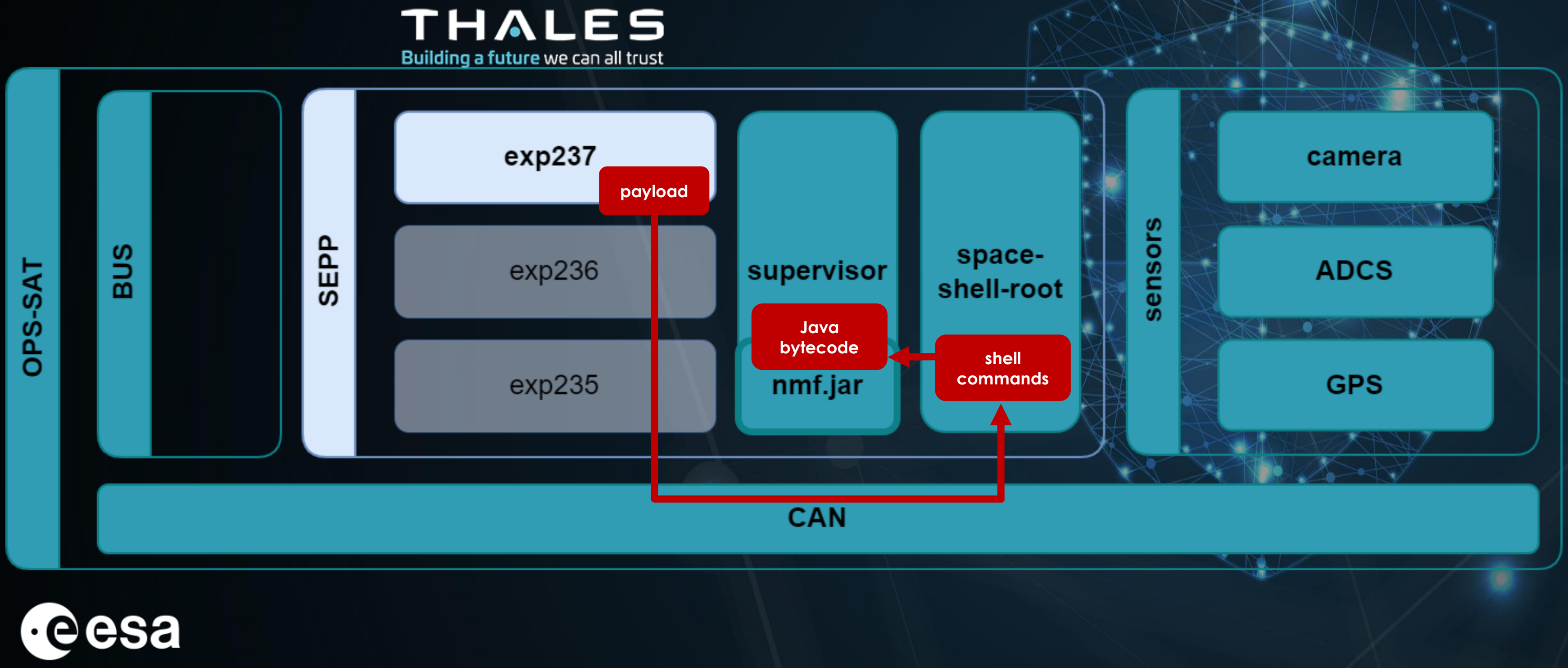
- **Possible vectors:**
  - Inject into **a library** or an executable file
  - Configure a new job or a new service

PERSISTENCE: Injection of a jar library

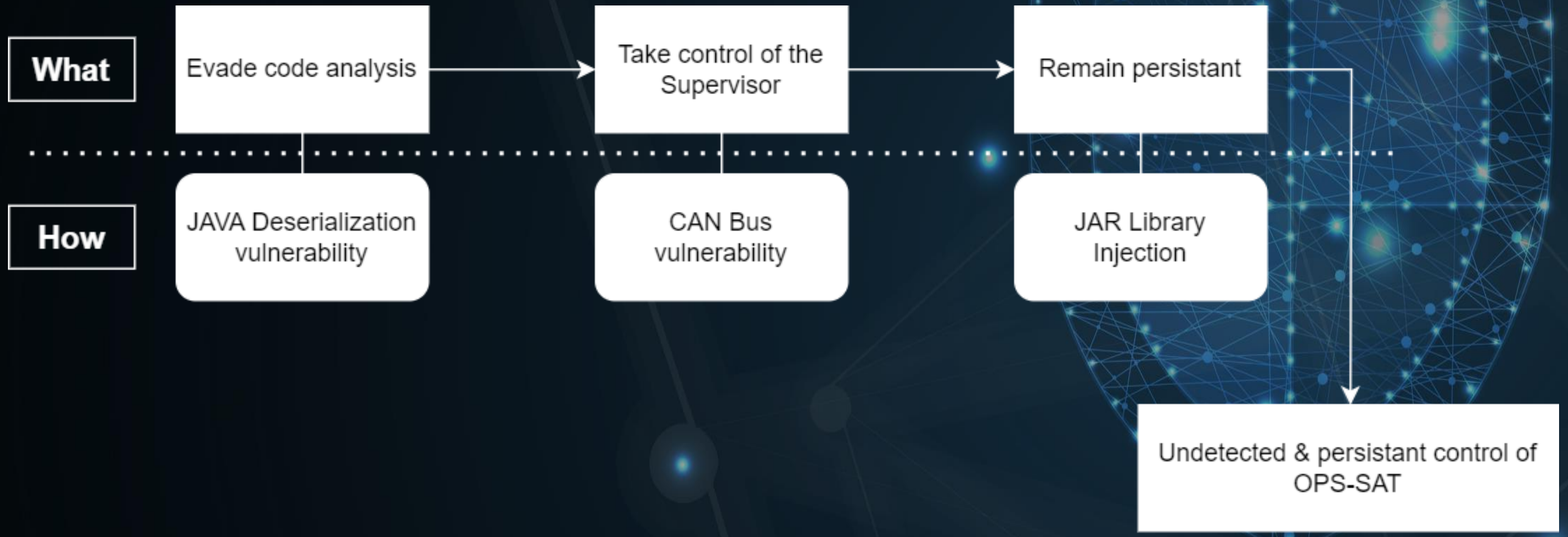
- Supervisor provides experiments with features they need: images, GPS
- It adapts standardized interfaces to low-level hardware
- **Perfect spot to control the information received by experiments**

- The jar library is writable by root user
- A jar is simply a zip file, with compiled Java bytecode inside
- We craft our bytecode based on the original one, and simply replace some files inside the jar
- **The supervisor now runs the jar containing our malicious bytecode**

TAKING CONTROL: INJECT INTO SUPERVISOR



SUMMARY: FULL ATTACK FLOW



THALES
Building a future we can all trust

Post Exploitation
starring: The Ugly

DEMO EFFECTS: TAMPERING WITH CAMERA & ADCS

- Root privileges allow us to take control on the **supervisor:**
  - Alter/delete all images captured by the camera
  - Override satellite attitude requested by other apps
  - This also provides **persistence** for our malicious code since the supervisor starts early and is almost always running

OTHER POTENTIAL EFFECTS

- **Non-demonstrated possible effects:**
  - Shutting down services used by other experiments
  - Draining the batteries by maintaining an unfavourable attitude
  - Tampering with GPS coordinates
  - Spying on other experiments data
  - …

THALES
Building a future we can all trust

Key takeaways
or Why it isn't all that bad… but it could well become so

NO SATELLITES WERE HARMED IN THE MAKING OF THIS PRESENTATION

- ESA supervised our tests and retained control throughout the demo
- The SEPP can only control most of OPS-SAT…
- … as long as the BUS* allows it
- ESA's design ensures they can always safely reset the SEPP and restore it to a known-good state through a simple TC
- The BUS also monitors the satellite's state to prevent it from becoming irrecoverable

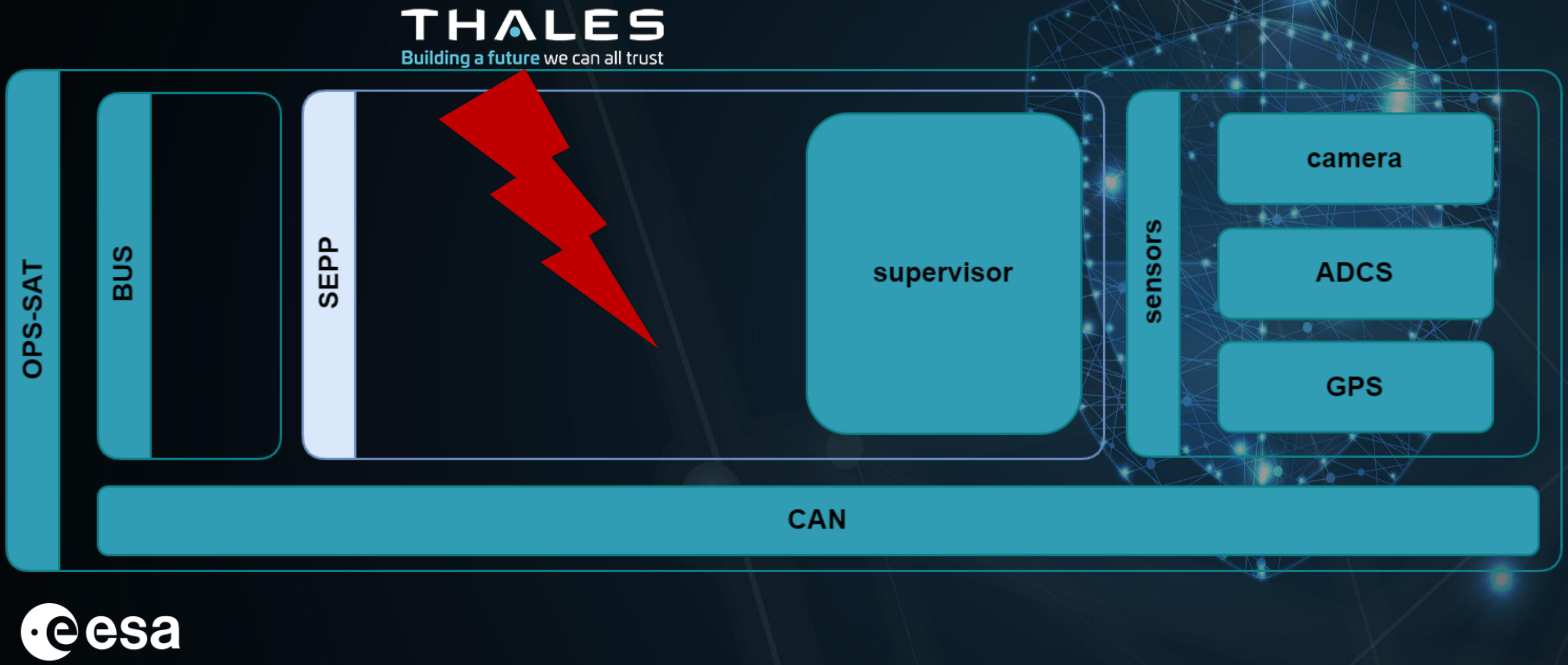
* *Core OPS-SAT component that can't be overridden by the SEPP*

NO SATELLITES WERE HARMED IN THE MAKING OF THIS PRESENTATION

- The attack scenario is built upon **non-trivial requirements**
- Code execution for random users is a **specific feature of OPS-SAT**!
- Probably less so on non-experimental spacecraft ☺
- We also had access to the SEPP system image:
  - Directly as it was provided to us by ESA as part of our cooperation
  - Indirectly during our tests on the FlatSat

- ESA is in the process of **fixing the vulnerabilities** we uncovered

NO SATELLITES WERE HARMED IN THE MAKING OF THIS PRESENTATION



IMPLICATIONS BEYOND OPS-SAT

- Satellites are key elements in numerous critical systems:
  - Telecommunication
  - Earth surveillance
  - Positioning (Galileo, GPS…)

- Satellite compromise can lead to:
  - Service disruption
  - Unreliable/tampered data transmission
  - Confidential data leaks
- Especially true if the compromise remains undetected!

Risk Mitigation
or *How to make sure this won't happen to you*

MITIGATING RISK - PREVENTION

- Design with security in mind:
  - Build threat model (e.g. MITRE ATT&CK)
  - Harden systems (e.g. CIS benchmark and RedHat STIG)
  - Isolate tasks (e.g. SELinux)
  - Grant least amount of privileges
- Code review
- Red-team designs & implementations

MITIGATING RISK – DETECTION

- Satellite status monitoring
- Filesystem integrity checks
- Log collection
- Network monitoring

THANKS! TIME FOR Q&A!

- Thank you for you attention!
- Heartfelt thanks to the whole OPS-SAT team at ESA for supporting us in this thrilling endeavour ☺
- Any questions?

THALES
Building a future we can all trust

THALIUM

ThalesAlenia Space
a Thales / Leonardo company