Steal This Drone

An Aerospace Village Cybersecurity Activity
DEFCON29

Introduction

The goal of DARPA's High Assurance Cyber Military Systems (HACMS) program was to use formal methods to build cybersecure embedded systems. Our Collins Aerospace HACMS team (including Galois, University of Minnesota, and the seL4 Trustworthy Systems group) focused on securing air vehicles. A quadcopter drone was used as a research vehicle for experimentation. Starting with a consumer-grade quadcopter, we extended and rebuilt the software using a variety of formal methods tools and formally verified software. The resulting high-assurance version of the quadcopter was dubbed the SMACCMCopter. The SMACCMCopter will be the unlucky subject of your attacks.

The SMACCMcopter architecture is shown in Figure 1 and features a number of enhancements to provide formally verified functionality and security. The Mission Software which interfaces with the outside world runs on the seL4 verified kernel. seL4 is the world's first formally verified, high-performance microkernel and is the foundation of SMACCMcopter security claims. Its functional specification has been verified using the Isabelle/HOL theorem prover and its high-level security properties have been formally verified to be correctly implemented all the way down to the binary. This means that the microkernel does precisely what it was designed to do and nothing more.

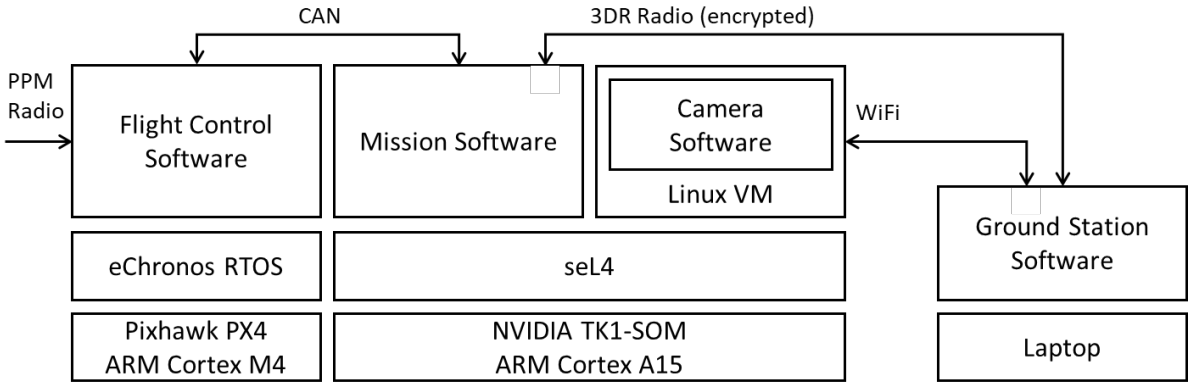

Figure 1. SMACCMcopter Architecture

The Flight Control Software runs on eChronos, a formally verified RTOS suitable for microcontrollers with limited computing resources. Both the Mission Software and Flight Control Software were written using a domain-specific language designed for writing memory-safe,

concurrent C code. For more information and papers about the HACMS technologies, visit http://loonwerks.com/projects/hacms.html

Software Description

The SMACCMCopter Mission Software architecture is shown in Figure 2. seL4 provides guaranteed separation between the software components to ensure that they only interact in ways that are allowed by the system design and limits the impact of any compromised component.

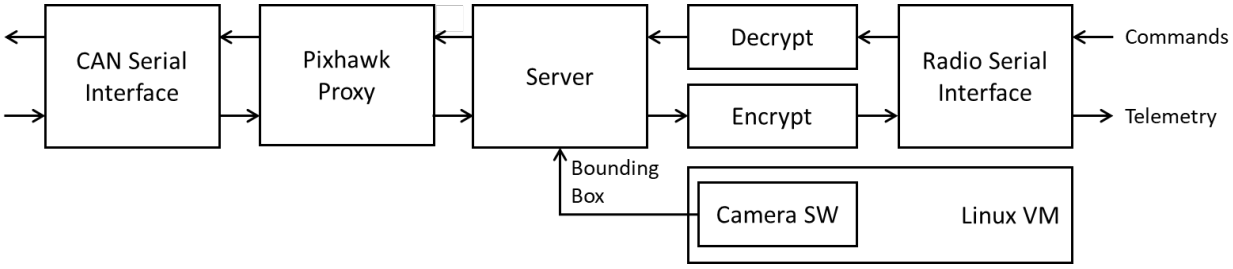

Figure 2. Mission Software Architecture (running on seL4 on TK1 mission computer).

The Mission Software communicates with the Pixhawk Flight Control Computer via a CAN bus. Telemetry data from the Pixhawk is made available in a Proxy component. Commands and data requests come from the Ground Station via a radio link encrypted with Advanced Encryption Standard using Galois/Counter Mode (AES-GCM). These requests are handled by the Server component and responses are sent back to the Ground Station on the encrypted radio link.

seL4 is able to act as a hypervisor hosting a guest OS running in a virtual machine (VM). This allows untrusted legacy software to be run with limited access to critical system resources. In SMACCMcopter, the VM is running commercially available camera software and streaming the images to the Ground Station via WiFi. This software is not mission-critical, and the drone can fly safely without it. The VM has been configured to allow the camera software to send bounding box data computed for objects of a specified color to the Server component so that it can be displayed on the Ground Station. The important aspect is that if this untrusted software misbehaves or becomes compromised, it cannot impact the safety-critical parts of the vehicle.

For more information on how the SMACCMcopter and its applications are set up, visit https://smaccmpilot.org/. The source code for the Mission Computer is located on GitHub at https://github.com/smaccm/smaccm/tree/master/models/smaccmpilot-tk1

Objectives

You will be given root SSH access via the WiFi connection to the Linux VM that runs the camera software. This will be your initial foothold into the vehicle software. For demonstration purposes, some simple attacks have been pre-staged in the Linux VM. Other tools may be available on the attacker laptop. The activity setup is shown in Figure 3.

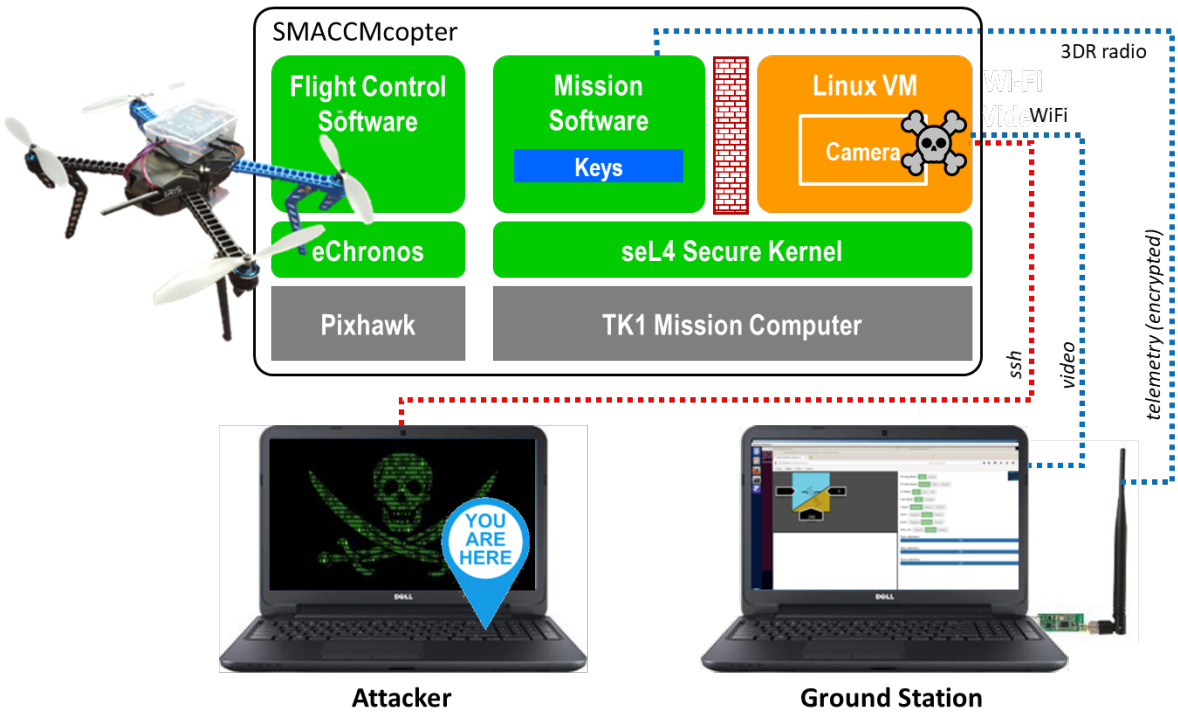

Figure 3. "Steal This Drone" Activity Setup.

Your goal is to affect the safety-critical software of the SMACCMcopter. This could mean interrupting the encrypted communication between the drone and the Ground Station, sending illegitimate telemetry or flight control commands, or otherwise affecting the Flight Control or Mission Software of the drone in any way. The data link encryption is sufficiently strong that you probably should not try to attack it directly. A more practical approach would be to attempt to read or write the encryption keys stored in memory using your foothold in the Linux VM. Modification of the encryption keys would result in complete loss of communication with the Ground Station and thus loss of flight control. Reading the keys could allow an attacker to send illegitimate telemetry or flight instructions, also affecting the flight of the drone. Our expectation is that you will be able to impact software running in the VM, but not the rest of the system. Happy drone hunting!

Activity

Start by logging into the Linux VM hosted on seL4 on the mission computer (`root@192.168.1.10`). You may wish to run some of the pre-staged attacks and observe the effects on quadcopter operation. For example:

- `camera_attack.py`: This attack will interrupt and replace the camera video sent to the Ground Station. What happens to the rest of the system if you attack the camera software?
- `forkbomb.py`: This attack launches new Linux processes until available computing resources are exhausted. Does crashing the camera software or crashing Linux impact telemetry?

- `rw_mem`: This application will allow you to attempt to read and write arbitrary locations in memory.

The real objective is to escape from the isolated VM. Can you bypass the seL4 separation to read or write the encryption keys? Or worse?

If you want to understand where the encryption keys are located in memory, it is necessary to replicate the build environment. A docker can be found here: https://github.com/smaccm/phase3/tree/test. The encryption data can be found in the binaries `Decrypt_inst_group_bin` and `Encrypt_inst_group_bin` located in `build/arm/tk1/smaccmpilot-tk1`. For example, the decryption data is stored in the C variable `ctx_dl_global_gec_sym_key_dec` which we can locate in the symbol table from objdump from a previous build:

```
00134478 l     O .bss    00001128 ctx_dl_global_gec_sym_key_dec
```

meaning that the variable will be at the virtual memory address 0x00134478 (offset 0x478 in page 0x134). Similarly, for the encryption data we find:

```
00136548 l     O .bss    00001128 ctx_dl_global_gec_sym_key_enc
```

Another possible attack vector is to crash or starve the other components running on the seL4 kernel. By starving or crashing the Proxy or Server that allows communication with the Flight Control software or the encrypt/decrypt components, an attacker can interrupt control of the drone. A possible approach is to use the existing communication path that allows the Camera software in the VM to send bounding boxes to the Server in the Mission Software. The code implementing this function is found here: https://github.com/smaccm/camera_demo/blob/master/src/bbox.c

Attack Scope

In general, the scope of your attack must be limited to attacks initiated within the foothold VM and you may not exploit physical access to the drone. For example, interrupting radio communications directly (jamming) or disabling the drone via external methods are obviously out-of-scope. Another stipulation is that the foothold VM you are given root access to is not considered part of the attack target. Our claim is that this VM can become fully compromised without impacting the other parts of the system. Side-channel attacks such as Spectre, Meltdown, and Rowhammer are all off-limits as they represent attacks that were intentionally not considered in creating the SMACCMcopter.

Limitations

The following are known issues in the system and using them to compromise the system will not be considered a successful attack against the security of the SMACCMcopter.

- Rowhammer, speculative execution, covert timing channels, using RAM as a transceiver (Side channel attacks are not included in the verification model.)
- DMA bus-mastering via compromised USB devices, controllers, or their firmware Separation is not guaranteed for DMA unless passing through the SMMU/IOMMU which is not currently verified.
- GCS 3DR radio attacks. This device and its firmware have not been formally verified.
- Compromising the Ground station. The software running on the Ground Station laptop was not developed using high-assurance techniques. It is conceivable that it could be compromised by sending malicious data via the WiFi video link or even the through bounding box telemetry data.
- Writing to some of the clock registers that are accessible from the Linux VM can cause the system to freeze. Linux requests are passed through by the seL4 VM Manager to a component which is supposed to filter those requests and limit access to only those capabilities which are safe and necessary. This is not a kernel error, but an application software issue that has not yet been fixed.