High-Assurance Cyber Space Systems (HACSS) for Small Satellite Mission Integrity

Daria C. Lane, Enrique S. Leon, Francisco C. Tacliad, Dexter H. Solio, Ian L. Rodney, Dmitriy I. Obukhov, Daniel E. Cunningham

SPAWAR Systems Center Pacific
53560 Hull Street, San Diego, CA 92152
daria.lane@navy.mil 619-553-8679
The demand for low cost, rapid fielding and fault tolerance can lead to compromising security for usability

- Furthermore, as systems become increasingly networked, the attack surface also grows

Small satellite cybersecurity challenges closely parallel Industrial Control Systems (ICS)

- “CIA” vs “AIC” conflicts
- Secure networking practices follow Confidentiality, Integrity, and Availability (CIA)
- ICS and Small Satellite Systems often reprioritize as Availability, Integrity, Confidentiality (AIC)
Hardware Threat Landscape

▼ COTS/GOTS hardware components and payloads may be vulnerable or compromised
  ▪ Critical safety resources may be exploited by an attacker

▼ Physical interfaces may be vulnerable to data extraction and code injection
  ▪ Common insecure interfaces include Ethernet, JTAG, serial, microUSB, UART and SPI

▼ Side channels may be vulnerable to exploitation
  ▪ Caches, power consumption, electromagnetic radiation and acoustics may be analyzed for leaked information
  ▪ In some cases an attacker may be able to cause bit flips

The findings discovered from Rowhammer suggest that an attacker may be able to cause targeted bit flips in order to gain kernel privileges.
Software Threat Landscape

- Software can be a heterogeneous mix of trusted and untrusted code
  - Including drivers, preloaded libraries/modules and mission payloads
  - Software heritage and chain of custody may be unknown

- Source code may contain exploitable bugs
  - Undefined behaviors and weak data type controls
  - Memory management safety and illegal inter-process communication

- Poor protocol implementations may allow Denial-of-Service (DoS) attacks
  - Protocol code must sanitize input to protect against invalid or malformed messages

- Poorly implemented encryption may allow an attacker to decrypt data or extract exploitable information
Full network connectivity between client nodes, combined with sloppy network implementations, can trivialize system exploitation.

- Air-gapped networks can be compromised by features that allow zone transferring and open port access.

Once in the network, an attacker may eavesdrop on traffic or inject commands or spoofed telemetry.

Small satellite vehicles may be sensitive to interference.

- Without capabilities to mitigate signal disruption, Quality of Service (QoS) and throughput can be degraded by an attacker equipped with commodity hardware.
Our Small Satellite Security Concept of Operations (CONOPS) starts with worst case assumptions:

1. The system consists of COTS/GOTS software and hardware.

2. An attacker has physical access to the space vehicle and payloads (insider threat, malicious hardware, untrusted supply chain, etc.).

3. An attacker has software access to flight software and/or mission payload.

4. An attacker has network access to space-to-ground and client-to-server node communication.
Red Team Assessment: Attack Methodology

▼ Hardware
- Firmware extraction via open interface (i.e. Ethernet, JTAG, serial, UART, microUSB, etc.)
- Firmware modification and injection (i.e. interrupt boot process, inject malicious code)
- Kernel privileges via targeted bit flips (i.e. repeated memory accesses in DRAM)

▼ Software
- Static and dynamic analysis of target source code to identify undefined behavior
- Analyze encrypted data at rest for weak encryption implementations

▼ Network
- Analyze system for vulnerable access points, misconfigured firewall policies, and permissive routing policies
- Fuzz target for undefined behavior and undocumented features in protocol implementation
- Analyze traffic for vulnerabilities in protocol and encryption implementation
High-Assurance Cyber Space Systems Approach

▼ Leverage DARPA High Assurance Cyber Military Systems (HACMS) tool stack

▼ Working from the hardware baseline up to ensure integrity

▼ Model-Based Engineering (Architecture Description Language)
  ▪ An accurate model allows for conclusions of High-Assurance
  ▪ Using code generation to ensure implementation matches the “interfaces” defined in the model

Approved for Public Release
High-Assurance Cyber Space Systems Approach

Formal Verification of Space System, including:

- Architecture
- Flight Software
- Ground Station Software
- Defined guarantees in system behavior such as hardware and software access, Inter-process communication (IPC)
HACSS Software Architecture

- Architecture Analysis & Design Language (AADL) is used to represent software and (computing) system architecture
  - Verified System Architecture
    - Reason about system component interactions
    - Ensure architecture guarantees adherence to system requirements and constraints
  - Code Generation
    - Generate component interfaces from verified architecture

Foundation of the HACSS architecture

- Verified microkernel
  - Formally verified via mathematical proof for correctness
  - Ensures memory isolation between components

- Domain specific language (DSL) for control components
  - Embedded in Haskell
  - Pure functional programming language
  - C code is synthesized from the DSL component implementation

High Assurance: Ensure Correctness, Safety and Security
HACSS Architecture Process

1. Component Implementation in DSL
   - Code Generator
   - Synthesized C Implementation
     - Secure Software Components
     - Secure Binary
     - Secure Separation Kernel

2. AADL Verified Architecture
   - Code Generator
   - Synthesized Component C Interfaces
High-Assurance Space Vehicle Approach

- Maintain interoperability, while designing and testing for functional correctness and high security
- Focus on system-level integrity, even if allowing certain subcomponents to function as “black boxes”
- Design flight processor interfaces to limit and/or mitigate side-channel monitoring and attack
- Design hardware interfaces with an arbiter to mediate access

Perfect Security—NO Functionality

Functional Correctness as key building block of Security
Isolated Software Components

Secure Separation Kernel

- Hardware Drivers
- Attitude Control
- Power
- Radio
- De/serialization
- Autonomy Rules
- Command
- Safe Mode
- Virtual Machine

Legacy Linux Applications
High Assurance Ground Station

- Ground system hardware and software will be included in the systems AADL model.
  - The functional correctness of the ground station behavior will be verified using theorem provers like Coq.

- Ground station software leverages programming languages that guarantee memory and type safety.

- A high-assurance stateful firewall implements packet forward and drop policies that are formally verified.

- Physical hardware interfaces shall be secured or closed. Access may only be granted by a trusted arbiter.
HACSS Nanosat Mission Objectives

Goal:

- Improve cyber resilience by constructing high-assurance cyber-physical systems aboard nanosatellite platforms for future DoD Space Missions

Approach:

- Identify, obtain, and evaluate the performance of available COTS & GOTS nanosatellite flight processors/software and ground station architectures to determine viability for use with HACMS tools
- Implement HACSS architecture aboard flight computers, bus subsystems, and ground segments
- Employ penetration testing as part of vulnerability assessment strategy

HACSS technologies and methodologies promote functionally correct high-assurance cyber-physical nanosat systems.
Summary

- Low-cost COTS/GOTS hardware and software used in small satellite systems introduces inherent cybersecurity risks

- Formal verification may be integrated into the system life-cycle to enhance assurance of mission critical functionality

- Architecture Analysis & Design Language (AADL) can increase confidence in system behavior

- Languages such as Rust and Haskell’s Ivory library can increase confidence in memory and type safety

- Additional assurance may be achieved by isolating mission critical components and incrementally verifying non-critical components
Questions?