Compartmentalizing Untrusted Code in Bare-Metal Embedded Devices

Liam Tyler and Ivan De Oliveira Nunes | Rochester Institute of Technology

RIT

Micro-Controller Units (MCUs)

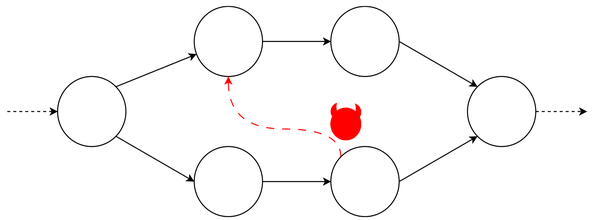
MCUs implement the de facto interface between the physical and digital worlds

MCUs often perform safety- and time-critical tasks but lack security features comparable with their overall importance

Runtime Attacks

Branching instructions (i.e. function calls, returns, loops, etc.) define the control flow of a program

Runtime attacks allow an adversary to remotely hijack these branching instructions and alter a program's intended behavior

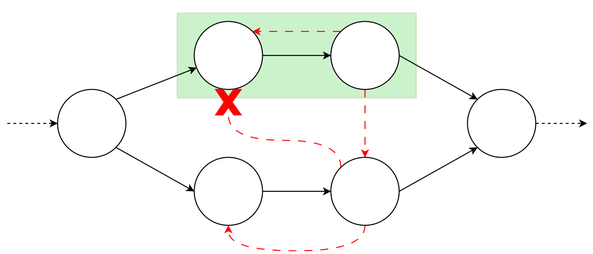


Existing Isolation Techniques

Some MCUs have support for isolation (Privilege levels and Memory Protection Units) to mitigate runtime attacks

These controls isolate security-critical functionality from the rest of the system

- Prevents malicious access to the isolated functionality
- Rest of the system remains vulnerable
- Vulnerabilities within the isolated functionality can still result in a full system compromise



Untrusted Code Isolation

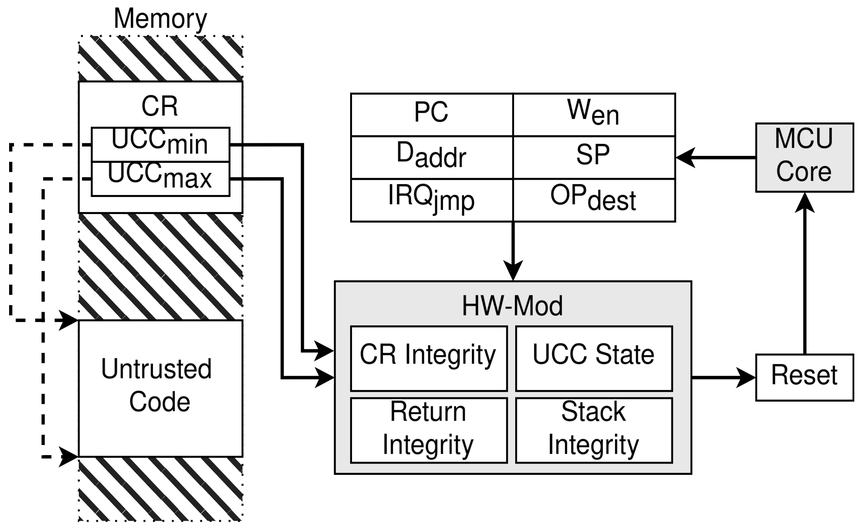
Runtime attacks typically originate from well-known code sections (e.g. third-party code and I/O manipulation)

There is a lack of untrusted code isolation for bare-metal devices due to resource restrictions

Untrusted Code Compartment Arch. (UCCA)

Hardware Monitor that allows for the definition and isolation of Untrusted Code Compartments (UCCs)

UCCs are independent, arbitrarily-sized, and immutable during runtime



UCCA Security Properties

Return Integrity prevents any malicious jumps from leaving UCCs

Stack Integrity prevents untrusted code from tampering with data in use by other functions on the device

- Prevents writes to Non-UCC data
- Ensures proper cleaning of the stack upon leaving a UCC

Both properties are formalized using Linear Temporal Logic and UCCA is verified to adhere to these specifications

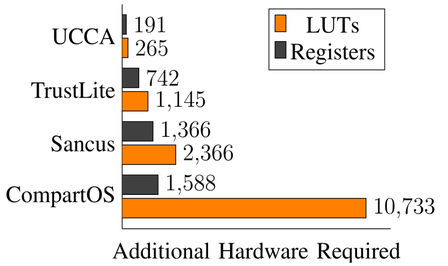
Early Results

Implemented on the OpenMSP430 core and deployed on a Basys-3 prototyping board

UCCA requires 85 Look-Up Tables (LUTs) and 86 registers to isolate a single UCC

Each additional UCC requires another 62 LUTs and 35 registers

UCCA security checks incur no runtime overhead



No. *UCCs*	Registers	LUTs
1	86	85
2	121	145
3	156	205
4	191	265
5	226	327
6	261	389
7	296	450
8	331	520

Next Steps

- Runtime configurability of UCCs
  - More UCC support without additional hardware overhead
- Shared UCC dependencies
  - Reduce code duplication across UCCs to reduce the potential memory impact of isolation

Resources

Contact: lgt2621@rit.edu

UCCA Preprint

UCCA Prototype

Acknowledgements

National Science Foundation
(Award SaTC-2245531)