Integrated Instruction Set Randomization and Control Reconfiguration for Securing Cyber-Physical Systems

Bradley Potteiger, Zhenkai Zhang, Xenofon Koutsoukos
Vanderbilt University
Nashville, TN
Motivation

• Zero day exploits increase the need for security in depth approaches

• Tightly coupled nature of CPS allow for traditional cyber domain attacks to inflict significant physical damage in the surrounding environment

• CPS not only have to maintain integrity while under cyber attacks, but also need to ensure safe behavior and operation.
Autonomous Vehicle Attack

- Buffer overflow vulnerability creates memory corruption opportunity
- Remote hijacking of vehicle from code injection attack
  - 11 seconds into video
- Adversarial instruction execution leads to unsafe vehicle behavior
Main Idea

• Protect against code injection attacks using Instruction Set Randomization
• Attack attempts with ISR enabled result in crashing controller
• Use reconfiguration to maintain safe operation
Problem Formulation

• **Hypothesis:** Using ISR in combination with control reconfiguration, we can detect and recover fast enough to ensure safety and stability in CPS physical behavior.

• **Attack Model:** Code injection attack on vehicle network using memory corruption, e.g. buffer overflow vulnerability.
System Architecture

- **Configuration Manager** – Oversees processes, configures randomization parameters, and implements attack detection, and recovery mechanisms.
- **Dynamic Binary Translator** – Randomization environment for underlying controller processes using MAMBO.
- **OS-Kernel** – Implements rate monotonic scheduling for running processes, and signal exception mechanisms for detecting attack instances.
Instruction Set Randomization Implementation

- Control Software ISR
- Machine code encoded with 32 bit key
- Runtime Derandomization
  - Code is executed through virtual binary translation layer (DBT)
  - Context Switch between DBT and host CPU
  - Derandomization during instruction decode stage of DBM pipeline
- Randomization Keys
  - Controllers are randomized with different keys

Randomization (Load Time)
  - Create 32 Bit Key
  - XOR Code Section Instructions with Key
  - Store randomized binary in DBT memory

Derandomization (Runtime)

DBT Environment

- Context Capture
- New PC
- Cached?
- New Basic Block
- Fetch
- Derandomize
- Decode
- Translate
- Next PC
- Host CPU (Execute Translated Code)
Control Reconfiguration

- Redundant controllers for failsafe execution during cyber-attack
- Attack monitoring conducted by Configuration Manager
- Configuration Manager decides when to execute the reconfiguration process
  - POSIX signal occurs from invalid instruction exception in default controller from attempted code injection attack.
  - Switches execution over to backup controller.

![Control Reconfiguration Diagram]
System Implementation

- **Design Time**
  - Training of neural network models offline
  - Execution time analyses
  - Scheduling

- **Runtime**
  - Configuration manager interacts with underlying controller processes through POSIX signals
  - **Attack Detection and Reconfiguration**
    - Configuration manager detects attack through kernel exception from DBT1
    - Configuration manager sends Resume POSIX signal to DBT 2 to take over execution
    - Configuration manager spawns DBT 3 to serve as new backup controller
Hardware in the Loop Testbed

• **Hardware Testbed**
  • ECU Cluster – 2 Beaglebone Black embedded boards
  • Controller Board – NVIDIA Jetson TK1 GPU Board
  • Two network interfaces
    • 100 Mbps Ethernet
    • 1 Mbps CAN Bus

• **Simulation Environment**
  • Udacity simulator – Open source autonomous vehicle simulator (Physical Domain)
  • Sensors and Actuators – Beaglebone Black
  • Vehicle Controllers – NVIDIA Jetson
  • MTD – Encapsulates controllers on NVIDIA Jetson
Software Implementation

- Operating System
  - Linux4Tegra 21.5
- Real Time
  - RT-Preempt Patch
  - Rate Monotonic Scheduling
- DBT Environment
  - MAMBO open source dynamic binary translator
- Communication
  - ZMQ – Ethernet
  - SocketCAN – Can Bus

Autonomous Vehicle Case Study

• **Sensors:**
  - Camera
  - GPS
  - Gyroscope

• **Actuators:**
  - Steering
  - Throttle

• **Controllers:**
  - 9 layer convolutional neural network using camera data as input
  - 2 Instances

Udacity Simulator - https://github.com/udacity/self-driving-car
Evaluation Metrics

• Safety Metrics
  • Distance of vehicle from center of road

• System Metrics
  • Performance Overhead
  • Recovery Time
  • Number of Deadlines Missed

• Security Metrics
  • Integrity
    • Protection against code injection attacks
  • Availability
    • Downtime after attack
Attack Scenario

• Vulnerability:
  • Buffer Overflow vulnerability in controller input processing of camera data

• Code Injection Attack:
  • Diverts control flow to malicious controller.
  • Drives vehicle straight off of the road at maximum speed
Scenario 1: Code Injection Attack Straight Road

Attack Process
- Use buffer overflow to inject payload on stack
- Payload opens remote root shell to attacker
- Attacker runs malicious controller (drive straight at full speed)

Defense Mechanism
- ISR – Results in invalid instruction exception.
- Recover to Neural Network Controller 2
Scenario 2: Code Injection Attack Curved Road

**Attack Process**
- Use buffer overflow to inject payload on stack
- Payload opens remote root shell to attacker
- Attacker runs malicious controller (drive straight at full speed)

**Defense Mechanism**
- ISR – Results in invalid instruction exception.
- Recover to Neural Network Controller 2
Controller Execution Times

Experiment Setup
- Logged 1000 iterations of controller operation under varying inputs

Measured Time Difference
- Sensor input received -> controller output is computed

Neural Network Controller
- Sampling Period: 100 ms
- Worst Case Execution Times
  - No ISR/Reconfiguration – 41.74 ms
  - ISR/Reconfiguration – 54.32 ms
- Averaged recorded execution times
  - No ISR/Reconfiguration – 38.5 ms
  - ISR/Reconfiguration – 42.9 ms
Recovery Time

- **Experimental Setup**
  - 100 experiments with attack with ISR/reconfiguration resulting in controller recovery

- **Measured Time Difference**
  - Attack detection -> Backup controller resumes execution
Conclusion

• Current Limitations
  • DOS susceptibility
  • Code reuse and data tampering attacks
  • Side channel attacks

• Future Directions
  • Diverse controllers and/or failsafe state
  • Integrating ASR and DSR
  • Dynamic Reconfiguration