Secure Autonomous CPS Through Verifiable Information Flow Control

Jed Liu  Joe Corbett-Davies  Andrew Ferraiuolo  Alexander Ivanov
Mulong Luo  G. Edward Suh  Andrew C. Myers  Mark Campbell

4th ACM Workshop on Cyber-Physical Systems Security and Privacy
19 October 2018
Networked CPSes are everywhere!
Networked CPSes are everywhere!
Networked CPSes are everywhere!
Networked CPSes are everywhere!
Networked CPSes are everywhere!
A new approach

- General architecture for secure CPS
- Co-develop hardware, software, control algorithms
- Security designed into all levels of system
- Leverage information-flow control
- Security-typed languages for software & hardware
System model (autonomous vehicle)

Vehicle hardware
System model (autonomous vehicle)
System model (autonomous vehicle)

- Makes control decisions
- e.g., planning, perception

Vehicle hardware

Safety-critical software

Untrusted software
System model (autonomous vehicle)

Vehicle hardware

- Safety-critical software
  - Makes control decisions
  - e.g., planning, perception

- Untrusted software
  - Everything else
  - e.g., entertainment
System model (autonomous vehicle)

Assumption: vehicle is a single monolithic hardware device

- Simplifies model
- Security more difficult
- Hardware isolation fails in practice
  - Jeep attack [MV’15]
System model (autonomous vehicle)
Adversary model

Security goal
Defend safety-critical software from remote adversary
Adversary model

Security goal
Defend safety-critical software from remote adversary

Adversary
- Can manipulate some sensors & network inputs

Vehicle hardware

Safety-critical software

Untrusted software

Sensors
GPS, Radar, Lidar, vision, etc.

Network
maps, traffic, music, etc.

Environment

Internet
Adversary model

Security goal
Defend safety-critical software from remote adversary

Adversary
- Can manipulate some sensors & network inputs
- Controls all untrusted software
Threats

- Manipulate sensors & network inputs
- Control untrusted software
Threats

- Manipulate sensors & network inputs
  - Attacks on control algorithms & implementation

- Control untrusted software
  - Attacks on underlying OS & hardware
Threats

- Manipulate sensors & network inputs
  - Provide bad maps, spoof sensors, tamper w/ env.

- Control untrusted software
Threats

• Manipulate sensors & network inputs
  – Provide bad maps, spoof sensors, tamper w/ env.
  – Exploit vulnerabilities in software implementation
    • memory safety bugs, inappropriate use of unverified inputs

• Control untrusted software
Threats

- Manipulate sensors & network inputs
  - Provide bad maps, spoof sensors, tamper w/ env.
  - Exploit vulnerabilities in software implementation
    - memory safety bugs, inappropriate use of unverified inputs

- Control untrusted software
  - Exploit OS bugs to break software isolation
  - Exploit hardware:
    - Bugs that break software isolation
    - Hardware-level *timing interference* slows down safety-critical software

Order of magnitude difference! [MM’07]
General architecture for secure autonomous CPS

- Security integrated into full system stack
  - Policies at language level, pushed into hardware
General architecture for secure autonomous CPS

- Security integrated into full system stack
  - Policies at language level, pushed into hardware

- Security-typed languages to design hardware & software
General architecture for secure autonomous CPS

Environment

raw sensors & inputs

Internet

Untrusted software
General architecture for secure autonomous CPS

Environment

raw sensors & inputs

Internet

Untrusted software

Verification

labelled sensors & inputs

- Design system w/ redundant inputs
- Verify each input against the others
- Highly consistent inputs → highly trusted
Threats

- Manipulate sensors & network inputs
  - Provide bad maps, spoof sensors, tamper w/ env.
  - Exploit vulnerabilities in software implementation
    - memory safety bugs, inappropriate use of unverified inputs

- Control untrusted software
  - Exploit OS bugs to break software isolation
  - Exploit hardware:
    - Bugs that break software isolation
    - Hardware-level *timing interference* slows down safety-critical software
Threats

• Manipulate sensors & network inputs
  – Provide bad maps, spoof sensors, tamper w/ env.
  – Exploit vulnerabilities in software implementation
    • memory safety bugs, inappropriate use of unverified inputs

• Control untrusted software
  – Exploit OS bugs to break software isolation
  – Exploit hardware:
    • Bugs that break software isolation
    • Hardware-level timing interference slows down safety-critical software
General architecture for secure autonomous CPS

Environment

raw sensors & inputs

Untrusted software

Verification

labelled sensors & inputs

Perception & Planning

Vehicle controls

Internet
General architecture for secure autonomous CPS

Environment

Internet

raw sensors & inputs

Untrusted software

Verification

labelled sensors & inputs

Perception & Planning

Programmed in Jif [POPL'99]

Vehicle controls

Jed Liu – Secure autonomous CPS through verifiable information flow control
Quick primer on Jif
- Java-based
  - Memory safety
Quick primer on Jif

- Java-based
- **Memory safety**
- Enforces **information-flow security**
  - Labels part of types
Quick primer on Jif

- Java-based
  - Memory safety
- Enforces information-flow security
  - Labels part of types

Jed Liu – Secure autonomous CPS through verifiable information flow control
Quick primer on Jif

- Java-based
  - Memory safety

- Enforces information-flow security
  - Labels part of types

Programmed in Jif [POPL'99]

“flows to”
Quick primer on Jif

- Java-based
  - Memory safety
- Enforces information-flow security
  - Labels part of types
  - Downgrading via endorse

Jed Liu – Secure autonomous CPS through verifiable information flow control
Threats

- **Manipulate sensors & network inputs**
  - Provide bad maps, spoof sensors, tamper w/ env.
  - Exploit vulnerabilities in software implementation
    - Memory safety bugs, inappropriate use of unverified inputs

- **Control untrusted software**
  - Exploit OS bugs to break software isolation
  - Exploit hardware:
    - Bugs that break software isolation
    - Hardware-level *timing interference* slows down safety-critical software
General architecture for secure autonomous CPS

Environment

Internet

raw sensors & inputs

Untrusted software

Verification

labelled sensors & inputs

Perception & Planning

Vehicle controls

Verified microkernel OS (e.g., seL4 [KEH’09])
General architecture for secure autonomous CPS

Environment

Internet

raw sensors & inputs

Untrusted software

Verification

labelled sensors & inputs

Perception & Planning

Vehicle controls

Verified microkernel OS (e.g., seL4 [KEH’09])

Processor with timing compartments

- Verified w/ ChiselFlow security-typed HDL [CCS’18]
  - timing-sensitive information-flow security

Jed Liu – Secure autonomous CPS through verifiable information flow control
Overview of HW timing isolation

SPECTRE

MELTDOWN
Overview of HW timing isolation

• Identify the security domain for each resource request
  – Timing compartment: security domain for timing isolation

• Allocate hardware resources to each timing compartment
  – Spatial partitioning for stateful resources
    • e.g., memory, caches, TLB, BHT, BTB
  – Temporal partitioning for stateless resources
    • e.g., I/O ports, interconnect, memory channels
Hardware security tags

Information-flow security enforced w/ explicit hardware tags

- Tag for each core, register, memory page, etc.
- Each cache/memory access tagged
- Similar to Jif labels
Spatial partitioning

- Removes timing interference through stateful elements
  - Caches, buffers, etc.
- Allocate state to each timing compartment
- Flush state to prevent vulnerabilities when allocation changes

L3 access comes with a TCID
Temporal partitioning

- Removes timing interference through resource contention
  - e.g., I/O ports, on-chip interconnects, DRAM channels

- Timing compartments take turns accessing the resource
  - Time-division multiplexing
General architecture for secure autonomous CPS

Programmed in Jif [POPL'99]:
- memory safety
- information-flow security

Untrusted software

Verification

labelled sensors & inputs

Perception & Planning

Vehicle controls

Verified microkernel OS (e.g., seL4 [KEH+09])

Processor with timing compartments
- Verified w/ ChiselFlow security-typed HDL [CCS'18]
  - timing-sensitive information-flow security
Two prototypes

1. Secure processor: HyperFlow [CCS’18]
   - Extends single-core RISC-V Rocket processor
   - Full timing-channel protection
   - Checked w/ security type system in ChiselFlow
Two prototypes

1. Secure processor: HyperFlow [CCS’18]
   - Extends single-core RISC-V Rocket processor
   - Full timing-channel protection
   - Checked w/ security type system in ChiselFlow

2. Segway robot software
   - Verifier & planner for lane following

Jif compiler for RISC-V under development
Software prototype

Map server (UNTRUSTED)
- Lane center
- Lane width
- Landmark locations

Camera (TRUSTED)
- Landmark range & bearing

Vicon (TRUSTED)
- x, y, heading, velocity

Waypoint (TRUSTED)
- goal (x, y)

LIDAR (TRUSTED)
- Local occupancy grid (obstacles)

Map verification

Path switch

Map-based planner

Sensor-only planner

ZedBoard

Verified?

Verified map

Paths

Safe path

Jed Liu – Secure autonomous CPS through verifiable information flow control
Map data

Expected landmark location
- Verified against landmarks in environment
- Used ArUco tags to simplify sensor processing

Ground truth lane centre (shown for reference)

Lane reward function
Software implementation

- Map verifier & A*-based planner—630 lines of Jif
  - 1,000 lines of Java code for network communication

```java
class Map[T,U] where T ⊑ U {
    Grid{U} unverif;
    Grid{T} verif;
}

void verify(map, sensor) {
    if (canVerify(map, sensor))
        map.verif = endorse(map.unverif);
    else map.verif = null;
}

Plan{T} plan(start, goal, map) {
    // If map unverified, use contingency.
    Grid grid = map.verif;
    if (grid == null)
        return contingency(start, goal);
    // Do A*.
    return astar(start, goal, grid);
}
```
Software implementation

- Map verifier & A*-based planner—630 lines of Jif
  - 1,000 lines of Java code for network communication

```java
class Map[T,U] where T ⊑ U {
    Grid{U} unverif;
    Grid{T} verif;
}

void verify(map, sensor) {
    if (canVerify(map, sensor))
        map.verif = endorse(map.unverif);
    else map.verif = null;
}

Plan{T} plan(start, goal, map) {
    // If map unverified, use contingency.
    Grid grid = map.verif;
    if (grid == null)
        return contingency(start, goal);
    // Do A*.
    return astar(start, goal, grid);
}
```
Software implementation

- Map verifier & A*-based planner—630 lines of Jif
  – 1,000 lines of Java code for network communication

```java
class Map[T,U] where T ⊑ U {
    Grid{U} unverif;
    Grid{T} verif;
}

void verify(map, sensor) {
    if (canVerify(map, sensor))
        map.verif = endorse(map.unverif);
    else map.verif = null;
}

Plan{T} plan(start, goal, map) {
    // If map unverified, use contingency.
    Grid grid = map.verif;
    if (grid == null)
        return contingency(start, goal);
    // Do A*.
    return astar(start, goal, grid);
}
```
Software implementation

- Map verifier & A*-based planner—630 lines of Jif
  – 1,000 lines of Java code for network communication

```java
class Map[T,U] where T ⊑ U {
    Grid[U] unverif;
    Grid[T] verif;
}

void verify(map, sensor) {
    if (canVerify(map, sensor))
        map.verif = endorse(map.unverif);
    else map.verif = null;
}

Plan{T} plan(start, goal, map) {
    // If map unverified, use contingency.
    Grid grid = map.verif;
    if (grid == null)
        return contingency(start, goal);

    // Do A*.
    return astar(start, goal, grid);
}
```
Software implementation

- Map verifier & A*-based planner—630 lines of Jif
  - 1,000 lines of Java code for network communication

```java
class Map[T,U] where T ⊑ U {
  Grid{U} unverif;
  Grid{T} verif;
}

void verify(map, sensor) {
  if (canVerify(map, sensor))
    map.verif =
      endorse(map.unverif);
  else map.verif = null;
}

Plan{T} plan(start, goal, map) {
  // If map unverified, use contingency.
  Grid grid = map.verif;
  if (grid == null)
    return contingency(start, goal);

  // Do A*.
  return astar(start, goal, grid);
}
```
Software implementation

- Map verifier & A*-based planner—630 lines of Jif
  – 1,000 lines of Java code for network communication

```java
class Map[T, U] where T ⊑ U {
    Grid{U} unverif;
    Grid{T} verif;
}

void verify(map, sensor) {
    if (canVerify(map, sensor))
        map.verif = endorse(map.unverif);
    else map.verif = null;
}

Plan{T} plan(start, goal, map) {
    // If map unverified, use contingency.
    Grid grid = map.verif;
    if (grid == null)
        return contingency(start, goal);
    // Do A*.
    return astar(start, goal, grid);
}
```
Evaluation: input validation

Robot position

Landmark measurement

Malicious map
Demo
Related work

Attack modalities
- Conventional vehicles (Checkoway\textsuperscript{+} 2011)
- Iran RQ-170 incident 2014

Control-algorithm security
- Signal cross-validation (Pajic\textsuperscript{+} 2017)
- Anomaly detection (Tian\textsuperscript{+} 2010, Xie\textsuperscript{+} 2011)

Formal methods
- Quant. info flow for CPS (Morris\textsuperscript{+} 2017)
- ROSCoq
- Timing verification w/ SpaceEx (Ziegenbein\textsuperscript{+} 2015)

Secure HDL

Secure processors
- Tiwari\textsuperscript{+} 2011, Ferraiuolo\textsuperscript{+} 2017

Secure CPS integration
- Veriphy (2018)
- Restart-based security (Abad\textsuperscript{+} 2016, Abdi\textsuperscript{+} 2017, Arroyo\textsuperscript{+} 2017)

Our contribution: a new system architecture
- Verified hardware
- Language-based information flow in software
- Cross-sensor input verification
Secure Autonomous CPS Through Verifiable Information Flow Control

**Jed Liu**  Joe Corbett-Davies  Andrew Ferraiuolo  Alexander Ivanov
Mulong Luo  G. Edward Suh  Andrew C. Myers  Mark Campbell

Programmed in Jif
- memory safety
- information-flow security

raw sensors & inputs

Untrusted software

Verification

labelled sensors & inputs

Perception & Planning

Vehicle controls

Verified microkernel OS

Verified processor with timing compartments