Formal Methods of Assurance for CPS

Dionisio de Niz

CPS Initiative Lead

Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213

Carnegie Mellon University Software Engineering Institute

Copyright 2019 Carnegie Mellon University. All Rights Reserved.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

Carnegie Mellon[®] is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM19-0289

Formal Assurance of DoD Systems

Assurance Automation for Safe-Critical Cyber-Physical Systems

• The DoD requires rapid fielding of critical capabilities to remain competitive with ongoing, urgent and emerging threats.

Challenge:

- Traditional Verification Does Not Scale
- Unpredictable Algorithms like machine learning (Autonomous CPS)
- Timely Interaction with Environment: correct actions at correct time

Our Solution:

- Add **simpler (verifiable)** runtime enforcer to make algorithms predictable
- Formally: specify, verify, and compose multiple enforcers:
 - Logic: Enforcer intercepts/replaces unsafe action
 - Timing: at right time
 - In sync with Physics (Control Verification)
- Protect enforcers against failures/attacks

Cers:

Logical Model

Statespace

- $S = \{s\}$
- $\phi \subseteq S$

Periodic actions

- Transition: $R_P(\alpha) \subseteq S \times S$
- Destination state: $R_P(\alpha, s) = \{s' | (s, s') \in R(\alpha)\}$

Identify states too close to safety border

- Inertia lead to unsafe state even if enforced
- Enforceable states:

 $C_{\phi} = \{s | \exists \alpha : R_P(\alpha, s) \in C_{\phi}\}$

Safe actions:

• $SafeAct(s) = \{\alpha | R_P(\alpha, s) \in C_{\phi}\}$



Logical Enforcer

Statespace & actions

- $S = \{s\}, \phi \subseteq S$
- $R_P(\alpha) \subseteq S \times S; R_P(\alpha, s) = \{s' | (s, s') \in R(\alpha)\}$

Enforceable states

• $C_{\phi} = \{s | \exists \alpha : R_P(\alpha, s) \in C_{\phi}\}$

Safe actions:

• SafeAct(s) = { $\alpha | R_P(\alpha, s) \in C_{\phi}$ }

Logical Enforcer: $E = (P, C_{\phi}, \mu)$

• Set of safe actions:

 $\mu(s) \subseteq SafeAct(s)$

• Monitor and enforce safe action:

$$\tilde{\alpha} = \begin{cases} \alpha, & \alpha \in \mu(s) \\ pick(\mu(s)), & otherwise \end{cases}$$



Drone Example

Statespace

- $S = \{s | s = (x, y, \theta)\}$
- $\phi = \{(x, y, \theta) \mid (x, y) \in Z\}$

Enforceable states

- δ_P : Max distance in one period P
- δ_B : Max distance in opposite direction of enforcement

•
$$C_{\phi} = \{(x, y, \theta) | (x + \delta_B, y + \delta_B) \in Z \land (x - \delta_B, y - \delta_B) \in Z\}$$

Action: constant speed at angle θ

Enforcement:
$$\tilde{\theta} = \begin{cases} \tilde{\theta} \in \tilde{\theta}_{1}, \text{ if } Y_{max} - y \leq \delta_{B} + \delta_{P} \\ \tilde{\theta} \in \tilde{\theta}_{2}, \text{ if } x - X_{min} \leq \delta_{B} + \delta_{P} \\ \tilde{\theta} \in \tilde{\theta}_{3}, \text{ if } y - Y_{min} \leq \delta_{B} + \delta_{P} \\ \tilde{\theta} \in \tilde{\theta}_{4}, \text{ if } X_{max} - x \leq \delta_{B}\theta + \delta_{P} \\ \theta, \text{ otherwise} \end{cases}$$

Carnegie Mellon University

Software Engineering Institute



Composing Enforcers

Enforcer Details: E: (P, C_{ϕ}, μ, U)

- $\forall s \in C_{\phi}: \mu(s) \subseteq SafeAct(s)$
- U: utility

Composition without conflict

- $E_1: (P_1, C_{\phi_1}, \mu_1, U_1)$
- $E_2: (P_2, C_{\phi_2}, \mu_2, U_2)$
- $\mu_{1,2}$: $\mu_1 \cap \mu_2$

Conflicting: Priority:

• $\mu_{1,2}$: $\mu_1 \cap \mu_2 \neq \emptyset$? $\mu_1 \cap \mu_2$: μ_1

Conflicting: Utility

• $\mu_{1,2}: \mu_1 \cap \mu_2 \neq \emptyset$? $argmax_{\alpha \in \mu_1 \cap \mu_2} \sum U_i(s, \alpha'): argmax_{\alpha \in \mu_1} \sum U_i(s, \alpha')$



Drone piloted by human Virtual Fence Marked by Black Posts -- No Enforcers Active --

Carnegie Mellon University Software Engineering Institute

-

Are We Done Yet?

Timing Assumption:

- Unverified software + enforcer finish before end of every *P* period.
 - Unverified software executes for less than its Worst-Case Execution Time (WCET)
 - Other software running executes for less than its WCET
 - Schedulability analysis successful

What can go wrong?

- Unbounded preemption
 - High priority software executes longer than WCET
 - Can make other software miss deadlines: late actions with old sensing
- Unbounded execution
 - Software executes longer than WCET
 - Misses its own deadline: Does **NOT** produce output on time: late action + old sensing
 - Inertia takes it to unsafe state

Fixed-Priority Scheduling + Rate Monotonic



Overload -> old sensed data + late actuation



Unbounded preemption Solution: Enforce timing budgets (timing enforcement)



Carnegie Mellon University Software Engineering Institute

Unbounded preemption Solution: Enforce timing budgets (timing enforcement) $e^{\alpha} e^{\alpha} e^{\alpha}$ STILL: Old sensing, late



Carnegie Mellon University Software Engineering Institute

Unbounded Execution: Solution: safe actuation on timing enforcement



Are we done yet?

Unverified software may corrupt Logical Enforcer

• It can even be malicious

Unverified software uses

- Unverified OS/kernel
- Unverified libraries

Temporal Enforcer relies on

• Unverified kernel / scheduler

Mixed-Trust Computing

System composed of trusted (verified) and untrusted (unverified) components

- Trusted : Verified Enforcers
- Untrusted: Unverified software

Untrusted should not corrupt trusted

Trusted should not depend on untrusted

• Cannot depend on unverified kernel / scheduler

Trusted components

• Preserve safety

Untrusted components

- Provide mission capability / performance
- Potential spurious failures

Uber XMHF: Verified Micro-Hypervisor Protection



Carnegie Mellon University Software Engineering Institute

Uber XMHF: Verified Micro-Hypervisor Protection

Only temporal enforcer can be protected if untrusted does not finish



Carnegie Mellon University Software Engineering Institute

Uber XMHF: Verified Micro-Hypervisor Protection



Carnegie Mellon University Software Engineering Institute

Two schedulers: VM scheduler + XHMF Scheduler Mixed-trust task: $\mu_i = (\tau_i, \kappa_i)$



Two schedulers: VM scheduler + XHMF Scheduler Mixed-trust task: $\mu_i = (\tau_i, \kappa_i)$



Carnegie Mellon University Software Engineering Institute

Simulation Demo Drone Protection (VM Crash) – Hardware in the loop



Application to Security Intrusion: Controller Rejuvenation ONR Project

Problem:

- Controller compromised by security attack
- Difficult to detect

Solution:

- Reboot (rollback to previous safe state)
- Re-establish stability of system
- Track mission progress

Software Rejuvenation Operating Modes

- 1. Tracking Control (TC)
- 2. Software Refresh (SR)
- 3. Secure Control (SC)
 - The switch from TC to SR is triggered by a timer (unsecure information)
 - From SR to TC or SC there is a condition to be satified (secure information)

Net



Carnegie Mellon University Software Engineering Institute t, time out

 $x \in \mathcal{E}_{TC}^s$

Secure Control Software

Refresh

 T_{SR}

± ∉ Etc

Tracking

Control

 T_{TC}

 $x \in \mathcal{E}_{TC}^{*}$

IN/OUT

Attack

Software Rejuvenation Secure Control

Recoverable Set •

 $\mathcal{E}_{SC^{j}}(1)$ Lyapunov Theory and Positively **Invariant Sets** • Safety Set $\mathcal{E}_{SC^{j}}(\epsilon_{s}) \triangleq \epsilon_{s} \mathcal{E}_{SC^{j}}(1)$ $\epsilon_s = T_{UC}$ $\mathcal{R}(T_{UC}; \mathcal{E}_{SC^j}(\epsilon_s), U) \subseteq \mathcal{E}_{SC^j}(1)$

R. Romagnoli, B.H. Krogh, and B. Sinopoli, Design of software rejuvenation for cps security using invariant sets, accepted to 2019 American Control Conference (ACC).

Attac Esci (Es $\mathcal{E}_{SCI}(1)$ t. time out Tracking IN/OUT Net Control $x \in \mathcal{E}_{TC}^s$ Attack Trc $x \in \mathcal{E}_{TC}^{*}$ Secure Control [Distribution Statement A] Approved for public release and unlimited distribution

Secure Control

Software

Refresh

 T_{SR}

I FETC

SW refresh

Software Rejuvenation Secure Control

Controlled System: $\dot{x} = f_{\varphi}(x) \triangleq f(x, \varphi(x))$ **Lyapunov Function**: $V_{\varphi} : \mathbb{R}^n \to \mathbb{R}$, $\mathcal{N}_{V_{\varphi}}(x_{eq}) \subseteq \mathcal{N}_{\varphi}(x_{eq})$, $V_{\phi}(x_{eq}) = 0$ and $\forall x \in \mathcal{N}_{V_{\varphi}}(x_{eq}) - \{x_{eq}\} : (i) \quad V_{\varphi}(x) > 0$,

$$\dot{V}_{\varphi}(x) = \frac{\partial V}{\partial x} \cdot f_{\varphi}(x) < 0$$

< 1

Lyapunov level set:For $\epsilon > 0$,

$$\mathcal{E}_{\varphi}(\epsilon) = \{ x \in \mathcal{N}_{V_{\varphi}}(x_{eq}) | V_{\varphi}(x) \le \epsilon \}. \qquad \epsilon$$

Positively Invariant Set. For any $0 < \epsilon \leq 1$, $\mathcal{E}_{\varphi}(\epsilon)$ is an *invariant set*.

 $\forall t > 0, \ \mathcal{R}(t; \mathcal{E}_{\varphi}(\epsilon), \varphi) \subseteq \mathcal{E}_{\varphi}(\epsilon)$



Software Rejuvenation Secure Control

Prop.1. Given $\dot{x} = f_{\varphi}(x) \triangleq f(x, \varphi(x))$ with stabilizing controller φ for equilibrium state $(x_{eq}, \varphi(x_{eq}))$ and Lyapunov function $V_{\varphi}(x)$ as defined above, given $\epsilon > 0$ for any $\epsilon < \epsilon' \le 1 \exists \gamma > 0 \ \exists \forall t \ge (\epsilon' - \epsilon)\gamma^{-1}$,

 $\mathcal{R}(t; \mathcal{E}_{\varphi}(\epsilon'), \varphi) \subseteq \mathcal{E}_{\varphi}(\epsilon).$

Prop.2. For any $U \subseteq \mathcal{U}$ and any $0 < \epsilon < \epsilon' \leq 1$, $\exists T_U > 0 \ni \mathcal{R}(t; \mathcal{E}_{\varphi}(\epsilon), U) \subseteq \mathcal{E}_{\varphi}(\epsilon') \forall t < T_U$.



- Prop1. We can always recover in a finite time
- Prop2. Given a reduced version of the Safety Set we can always find a period of time where is allowed uncertain control.

Software Rejuvenation Analysis of mission progress

Idea:

Provide a sequence of way points that represent a sequence of equilibrium points around which we define the Safe Set.



Goal:

- Safety transition from one way point to the next one.
- Liveness (in the case of no attack)

Software Rejuvenation Analysis of mission progress

- Safety
- Liveness

R. Romagnoli, B.H. Krogh, and B. Sinopoli. Safety and liveness of software rejuvenation for secure tracking control, accepted to 2019 European Control Conference (ECC).



Software Rejuvenation Drone experiment

6 DOF \Rightarrow 12 state variables

$$\begin{split} \vec{p}_x &= -\cos\phi\sin\theta\frac{F}{m} \\ \vec{p}_y &= \sin\phi\frac{F}{m} \\ \vec{p}_z &= g - \cos\phi\cos\theta\frac{F}{m} \\ \vec{\phi} &= \frac{1}{J_x}\tau_\phi \\ \vec{\phi} &= \frac{1}{J_y}\tau_\theta \\ \vec{\psi} &= \frac{1}{J_z}\tau_\psi. \end{split}$$

Linear design:

- linearize at equilibrium
- assume full state available
- LQ state feedback design
- reference points =
 equilibrium states



Software Rejuvenation: Drone experiment



Software Rejuvenation Analysis of mission progress

6 DOF \Rightarrow 12 state variables $\vec{p}_x = -\cos\phi\sin\theta\frac{F}{m}$ $\vec{p}_y = \sin\phi\frac{F}{m}$ $\vec{p}_z = g - \cos\phi\cos\theta\frac{F}{m}$ $\vec{\phi} = \frac{1}{J_x}\tau_{\phi}$ $\vec{\theta} = \frac{1}{J_y}\tau_{\theta}$ $\vec{\psi} = \frac{1}{J_z}\tau_{\phi}$.

Linear design:

- linearize at equilibrium
- assume full state available
- LQ state feedback design
- reference points =
 equilibrium states



Current Experiments Micro-reboot in indoor drone



[Distribution Statement A] Approved for public release and unlimited distribution.

Summary

Scalable formal verification

- Using enforcers
- Untrusted components guarded by trusted (verified) ones

Full verification of CPS

- Control
- Logical
- Time

Protected verification

- Enables building trusted system with untrusted components
- Protection verified down to the metal