Mixed-Trust Computing for Real-Time Systems

Dionisio de Niz*, Bjorn Andersson*, Mark Klein*, John Lehoczky*, Amit Vasudevan*, Hyoseung Kim†, and Gabriel Moreno*

* Carnegie Mellon University
† University of California, Riverside
“Trust” in Safety-Critical Systems

- Verification via formal methods
  - Critical components, OSs, libraries, etc.

- Verified properties can be easily compromised if the verified components are *not* protected from unverified ones

Our notion of “Trust”
- Both verification and protection should be jointly considered
Challenges

• The complexity of today’s OSs making them impractical to verify
• Alternative: minimize the trusted computing base (TCB) by developing small verified hypervisors (HVs) and microkernels
  – e.g., seL4, CertiKOS, and uberXMHF

**Trusted parts in TCB**

• Made small and simple due to verification difficulties
• Isolated from untrusted parts

**Untrusted parts in VM**

• Hosted in a virtual machine (VM)
• Implements rich functionalities on full-scale OSs, e.g., Linux

**Disjoint-trust computing**: trusted and untrusted parts co-exist but in a completely isolated and disjoint manner
Limitations of Disjoint-Trust Computing

- Does *not* allow the use of untrusted components in critical functionality where safety must be assured through verification
  - The verified components must be isolated from the untrusted ones if they are to be trusted

- Example: self-driving car
  - Prevents untrusted machine learning algorithms to drive a car if such functionality needs to be verified
    - Very difficult or practically impossible to verify the entire software/hardware stack, e.g., GPUs, drivers, ML libraries, frameworks, etc.
  - Instead, a separate trusted component would need to be in charge of the driving, isolating it from any untrusted component
Real-time Mixed-Trust Computing

- **Goal**: to give the flexibility to use untrusted components even for critical functionalities

  - Output checked by a verified component, called **Logical Enforcer (LE)**
  - LE is protected by HV (e.g., uberapps*, PAL†)

- **Temporal Enforcer (TE)**: performs a default safety action if the untrusted task has not produced a correct output by a specified time

Contributions

• Mixed-trust software architecture
  – Interplay of two schedulers
    1. Preemptive fixed-priority scheduler in the VM
    2. Non-preemptive fixed-priority scheduler in the HV
  – Mixed-trust task model & analysis

• Design of a mixed-trust coordination protocol
  – Preserves timing dependencies between trusted and untrusted parts
  – Prevents logical dependencies that can compromise the trusted part

• Implementation in the uberXMHF hypervisor* on Raspberry Pi

Outline

• Introduction
  – Motivation & Limitations
  – Overview

• Mixed-trust computing
  – Logical model and protection domains
  – Mixed-trust task scheduling and analysis
  – Fail-safe coordination protocol

• Case study results

• Conclusions
Logical Model of Mixed-Trust Computing

- LE-enforced action
  \[ \tilde{\alpha} = \begin{cases} 
  \alpha & \text{if } \alpha \in \mu(s) \\
  \text{pick}(\mu(s)) & \text{otherwise} 
  \end{cases} \]  
  Safe action for state \( s \)

- TE-enforced action
  \[ \hat{\alpha} = \begin{cases} 
  \alpha_T & \text{if } \tilde{\alpha} = \bot \\
  \tilde{\alpha} & \text{otherwise} 
  \end{cases} \]  
  Default safe action for any state if no output generated by a specific time \( E \)
Conditions Required by Logical Model

• To prevent an untrusted component from causing behaviors not present in the logical model

| C1. Each task must produce an output every period |
| C2. There is only one output per period |
| C3. The output produced by a task in a period is either from LE or TE |
| C4. An output produced by the task and validated by the LE must be the product of a computation that executes within a single period |
  □ i.e., sensing, computing, and output should be done within the same period |
| C5. The TE of a task must execute $E$ time units after the arrival of the job it guards and finish before the end of the period |
Behavior of Periodic Mixed-Trust Task

```
1  while true do
2      try:
3          \textcolor{orange}{s \leftarrow currentState()}
4             \textcolor{orange}{\alpha \leftarrow computeActuation(s)}
5          \textcolor{blue}{\tilde{\alpha} \leftarrow LE(s, \alpha)}
6          \textcolor{blue}{\text{actuate}(\tilde{\alpha})}
7          \textcolor{green}{\textbf{catch} \ timeout(E):} 
8             \textcolor{green}{\text{actuate}(\alpha_T)}
9      end
10  waitForNextPeriod()
11 end
```

Every period

Formally-verified LE and TE need to be protected against unintended modifications
Mixed-Trust Protection Domains

1. Untrusted Spatio-Temporal protection Domain (USTD)
   - Untrusted task code execution in the VM

2. Trusted Spatial protection Domain (TSD)
   - LE execution in secure enclaves (memory protection)

3. Trusted Spatio-Temporal protection Domain (TSTD)
   - TE execution in the verified HV (memory and spatial protection)
System Modeling

• Mixed-trust task \( \mu_i = (T_i, D_i, \tau_i, \kappa_i) \)
  - \( T_i \): period, \( D_i \): deadline
  - Two execution segments: Guest Task \( \tau_i \) and Hyper Task \( \kappa_i \)

1. Guest Task \( \tau_i = (T_i, E_i, C_i) \)
   • \( C_i \): worst-case execution time of \( \tau_i \)
   • \( E_i \): intermediate deadline for \( \tau_i \) \( \iff \) set by analysis such that \( \kappa_i \) can finish by \( D_i \)

2. Hyper Task \( \kappa_i = (T_i, D_i, \kappa C_i) \)
   • \( \kappa C_i \): worst-case execution time of \( \kappa_i \)
   If there is no hyper-task part, \( \kappa C_i = 0 \)

• Uniprocessor system

  **Preemptive scheduler in VM**
  • Full-scale guest OSs, e.g., Linux

  **Non-preemptive scheduler in HV**
  • Simplifies HV logical verification by removing task interleavings*

Mixed-Trust Task Scheduling

• How to determine $E_i$?
  – For hyper tasks to be schedulable, $\forall \mu_i \in \Gamma \quad E_i \leq D_i - R^k_i$
  – This work uses $E_i = D_i - R^k_i$
    • Optimal $E$ assignment presented in online appendix
Hyper Task Schedulability

• Based on non-preemptive fixed priority scheduling*
  – Maximum duration of a level-\(i\) active period
    \[ t^\kappa_i = \max_{j \in \kappa L_i} \kappa C_j + \left[ \frac{t^\kappa_i}{T_i} \right] \kappa C_i + \sum_{j \in \kappa H_i} \left[ \frac{t^\kappa_i}{T_j} \right] \kappa C_j \]
  – Latest starting time of \(\kappa_{i,q}\) in the level-\(i\) active period
    \[ \omega^\kappa_{i,q} = \max_{j \in \kappa L_i} \kappa C_j + (q - 1) \kappa C_i + \sum_{j \in \kappa H_i} (\left\lfloor \frac{\omega^\kappa_{i,q}}{T_j} \right\rfloor + 1) \kappa C_j \]
  – Worst-case response time of \(\kappa_i\)
    \[ R^\kappa_i = \max_{q \in \{1, \ldots, \left\lfloor \frac{t^\kappa_i}{T_i} \right\rfloor \}} (\omega^\kappa_{i,q} + \kappa C_i - (q - 1)T_i) \]

• The corresponding guest task \(\tau_i\)’s deadline \(E_i = D_i - R^\kappa_i\)

Guest Task Schedulability

- Level-\(i\) busy period (BP) for a guest task \(\tau_i\)
  - Processor is busy with \(\left\{\right.\) Hyper tasks (HTs)
    \(\left.\right\}\) Guest tasks (GTs) with higher priority than \(\tau_i\)

**Theorem IV.1.** The longest response time for all GT jobs of task \(\tau_i\) of a mixed-trust task \(\mu_i\) occurs in a level-\(i\) busy period initiated by the arrival of either \(\tau_i\) or \(\kappa_i\) and the arrival of higher-priority GTs or HTs of other mixed trust tasks, \(\mu_j\).

- Request-bound function

\[
rbf_{i}^{y}(t, b) = \begin{cases} 
\left[\frac{t-(T_i-E_i)}{T_i}\right]^+ C_i b + \left[\frac{t}{T_i}\right] \kappa C_i & \text{if } y = E, \\
\left[\frac{t}{T_i}\right] C_i b + \left[\frac{t-E_i}{T_i}\right]^+ \kappa C_i & \text{if } y = A,
\end{cases}
\]

BP starts with HT

BP starts with GT
Guest Task Schedulability (cont’d)

- Maximum level-$i$ busy period
  \[
  t_{i}^{g,x} = \left( \sum_{j \in L_{i}} rbf_{j}^{E}(t_{i}^{g,x}, 0) \right) + rbf_{i}^{E}(t_{i}^{g,x}, 1) \\
  + \sum_{j \in H_{i}} \max_{y \in \{E,A\}} rbf_{j}^{y}(t_{i}^{g,x}, 1).
  \]

- Maximum finishing time of $\tau_{i,q}$ in the level-$i$ busy period
  \[
  w_{i,q}^{g,x} = \left( \sum_{j \in L_{i}} rbf_{j}^{E}(w_{i,q}^{g,x}, 0) \right) + qC_{i} + (q - 1 + I_{(x=E)})\kappa C_{i} \\
  + \sum_{j \in H_{i}} \max_{y \in \{E,A\}} rbf_{j}^{y}(w_{i,q}^{g,x}, 1).
  \]

- Worst-case response time of $\tau_{i}$:
  \[
  R_{i,q}^{g,x} = w_{i,q}^{g,x} - ((q - 1)T_{i} + I_{(x=E)}(T_{i} - E_{i})) \\
  \times \max_{q \in \{1, \ldots, \left\lfloor \frac{t_{i}^{g,x} - I_{x=E}(T_{i} - E_{i})}{T_{i}} \right\rfloor \}} R_{i,q}^{g,x}
  \]
Experiments

- Impact of HTs
  - A GT can experience delay from the HTs of other tasks
  - A HT can experience delay from the HTs of other tasks
- But HTs are made small for verifiability

\[\text{double-accounting effect}\]
Fail-safe Coordination Protocols

• To prevent any dependency of trusted code from untrusted code

1. Secure HT Bootstrapping
   – Required for Trusted Spatio-Temporal protection Domain (TSTD)
   – Ensures that HTs can start and execute periodically even if the VM is unable to run GTs

2. Fail-Safe HT Triggering
   – Prevents a failure in the VM from disabling or corrupting the periodic arrival of HTs

3. Late-Output Prevention
   – Prevents the output of a GT $\tau_i$ after its deadline $E_i$
Case Study: Temporal Failure Scenarios

- Implemented in uberXMHF on Raspberry Pi 3
- Transient Failures

- Permanent Failures
Case Study: Drone Application

- Mission controller: sends velocity vectors (VV) to Flight controller
  - Guest task (VV Gen) generates velocity vectors
  - Hyper task (Hyp-Safe) generates the safe drone action

- Tested with hardware-in-the-loop simulation
Conclusions

• Both *protection* and *verification* are required for the safe use of untrusted components in critical functions

• Real-time mixed-trust computing
  – First framework that satisfies these two requirements
    1. Using trusted components to monitor and replace unsafe untrusted component outputs with safe ones
    2. Protecting the logical and temporal behavior of trusted components
      – Mixed-trust task \[\begin{array}{l}
        \text{guest task (untrusted component & logical enforcer)} \\
        \text{hyper task (temporal enforcer)}
      \end{array}\]

• Prototype implementation in the uberXHMF hypervisor

• Tested with transient and permanent failures
Thank You

Mixed-Trust Computing for Real-Time Systems

Dionisio de Niz*, Bjorn Andersson*, Mark Klein*, John Lehoczky*, Amit Vasudevan*, Hyoseung Kim†, and Gabriel Moreno*

* Carnegie Mellon University
† University of California, Riverside