Empirically Measuring Memory Safety Overheads for CPS

[Draft]

ABSTRACT

The abstract goes here ... Nowadays, cyber-physical systems (CPS), that integrate computations and communications with physical processes, are receiving high popularity and being widely adopted in various application areas such as healthcare, water treatment, power grid, transportation, military, robotics, and so forth. However, the increasing prevalence of cyber attacks targeting them poses a growing security concern. Besides deception attacks, that mostly target communication channel vulnerabilities, firmware-based attacks are another widely common cyber attacks in CPS. This class of attacks mainly exploit memory safety vulnerabilities, e.g., buffer overflows and dangling pointers. These vulnerabilities are common in CPS firmwares, e.g., PLC firmwares, because of the fact that those firmwares are implemented with C/C++ languages for performance and many other reasons.

As a countermeasure, wide range of memory safety tools have been developed that dynamically detect and mitigate memory safety violations. However, these tools incur high runtime overheads, which might not be tolerable in most delay-sensitive and real-time constrained cyber-physical systems. Delay could result severe damage in the system or open a security hole for delay-based attacks. This essentially makes performance as critical as security in most CPS.

On this research, we empirically measure memory safety overheads for CPS and quantify tolerability of CPS on such overheads. By doing so, we can able to 1) specify the minimum hardware requirements, e.g., processor speed, that a given cyber-physical system needs to have to work normally with a specific memory safety tool or 2) identify the memory safety tool whose runtime overhead does not affect real-time constraints of the cyber-physical system.

1. INTRODUCTION

Introduction of the paper will include the following points:

1. **Problem description**
   - Clear description of memory safety vulnerabilities in CPS (specifically in context of SWaT) - *preliminary problem - P0.*
   - Explaining (class of) attacks that target those vulnerabilities (firmware-based attacks, in general) - *initial problem - P1.*
   - High level explanation of memory safety solutions: specifically the MS tool we are using (its features, reason of choosing it, etc).
   - Elaborating memory safety overheads and how it affects a CPS plant, e.g., in SWaT - *main problem and focus of the paper - P2.*
2. **Our proposed solution**
   - Introduce a memory safety tool in CPS environment, which is
     - Effective in security (in detecting memory errors) - hence solving *P1.*
     - Efficient in performance (tolerable in the system) - hence solving *P2.*
3. **Contributions**: contributions (novelty claims) of our work.
   - Adopting memory safety solutions in the CPS environment - *referring the evidence on Section xx.*
   - Benchmarking and empirically measuring its runtime overhead (MSO) in CPS environment - *referring the evidence on Section xx.*
   - Quantifying and ensuring its tolerability (before it creates any disruption to the system) - *referring the evidence on Section xx.*
4. **Structure of the paper**: explaining how our paper is organized.
   - Section 2: xx
   - Section 3: xx
   - Section 4: xx
   - Etc.

2. BACKGROUND

[Optional section] [1]

- Characteristics of a CPS environment
  - Ever lasting process (could result buffer overflows for unbounded buffers)
  - Severe resource constraints, e.g., in PLCs: limited memory and CPU speed
  - Hard real-time constraints
- Overview of SWaT (details will be included in Section 3)
  - Overview
  - Architecture
  - etc
- Address sanitizer
  - How it works - a compile-time instrumentation tool.
  - Memory errors coverage
  - Performance
  - Reason to choose it
- OpenPLC
- ScadaBR

3. SYSTEM MODEL: SWAT

1. **General architecture of the plant**
   - Internal architecture of the system
     - Hardware architecture of the PLC, e.g., processor, registers, memory, etc.
     - The PLC scan cycle
   - Interacting components
     - Sensors, actuators
     - Other PLCs
     - SCADA systems
     - Communication topology
     - Interaction constraints, e.g., frequency, accessing shared resources (e.g., mutual exclusion and the implication of locking on performance), etc.
2. **Security concerns of the plant**:
   - Focusing on memory safety vulnerabilities.
   - Highlighting attack vectors (and exploits) targeting those vulnerabilities.
3. **Performance constraints**
   - Real-time constraints (hard-deadline) of the system
   - Elaborating how the runtime overhead, i.e, MSO, can affect the plant (e.g., violating the real-time constraint).

[1] Could be merged with Section 3

4. EMPIRICALLY MEASURING MEMORY SAFETY OVERHEADS

1. Benchmark design
2. Measurement details, e.g., implementation.
3. Measure MSO of each individual PLC operations: scan inputs, program execution, update Outputs, others (locking/unlocking threads).
4. Measure overall MSO
5. Show if there is exceptional deviation in performance
   - MSO is an average value, but a single exceptional scan time could disrupt the system.
   - Hence, needs to graphically show variance of each scan time

5. QUANTIFY TOLERABILITY

- Identifying and characterizing performance constraints of the system
  - Internal constraints, e.g., cycle time.
  - External constraints, e.g., frequency of communications with external components.
- Quantifying a tolerable MSO range.

6. EVALUATION

- Security evaluation
  - Coverage
  - Effectiveness
- Performance evaluation (referring Section 4 and 5)
  - Analyzing MSO and its tolerability

7. RELATED WORK

- Security and memory safety related works in CPS or embedded systems.
- Performance overhead related works in CPS environment.

8. CONCLUSION AND FUTURE WORKS

- Conclusion
- Future works

9. REFERENCES

[1] XX. Reference 1. 2015.
[2] YY. Reference 2, 2014.