TCIPG Vulnerability Assessment Tool Using Model Checking

P. Patel, F. Yuan, T. Kalbarczyk, and R. K. Iyer

Goals

- Ensure secure execution of software and protect critical data.
- Identify data dependencies of critical data and protect them against possible attacks.
- Develop a technique to formally reason about memory corruption attacks that violate the integrity of an application/system.

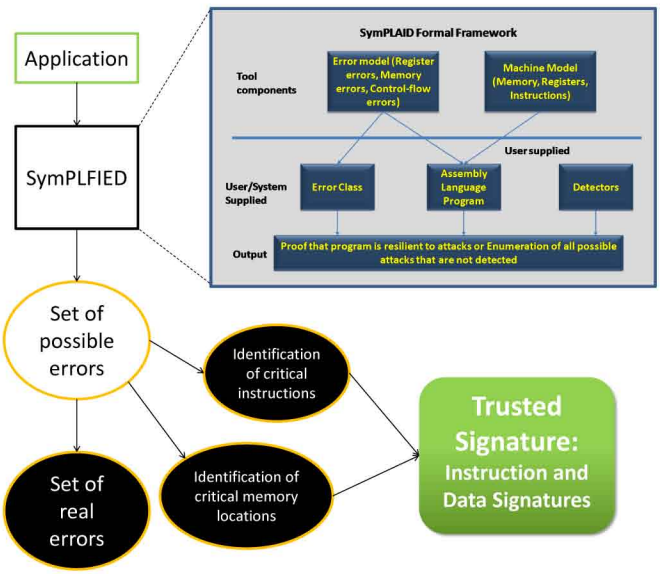
Fundamental Questions/Challenges

- Early identification of potential vulnerabilities in software.
- Use combination of symbolic execution and model checking to systematically analyze corruptions of the application data and identify cases that lead to a successful attack.
- Remove application-level vulnerabilities and guide design of defense mechanisms to protect application from attacks.

Research Plan

- Design a technique and a tool to discover vulnerabilities in an application using symbolic execution and model checking.
- Generate signatures for critical data to be checked at runtime.
  - Identify critical data, i.e., data that, if corrupted, lead to a successful attack.
  - Identify critical code sections, i.e., an instruction sequence during the execution of which corruption of critical data allows the attacker to achieve objectives (e.g., login with an invalid password).
- Demonstrate the tool on applications used in the context of power grid applications.

Formal Analysis Framework



Research Results

Symbolic Error Injection Using SymPLFIED

- Identifies conditions (e.g., what data and when at runtime to corrupt) under which the attacker (e.g., an insider) can penetrate the system.
  - Introduce memory errors and propagate the error's consequences using symbolic execution and model checking.
- Output produced:
  - *When to inject the fault:* at which instruction.
  - *Where to inject:* what memory address.
  - *What to inject:* what value (range of values).
- Results used to determine memory locations that need to be protected to prevent application failure or system compromise.

Trusted Signature Generation

- **Instruction Signature**
  - Stored unique identifiers (e.g., PC) for critical instructions.
  - Only critical instructions can write to critical data (prevents code injection).
- **Data Signature**
  - Stored addresses of critical data in the program.
  - Content of critical memory locations is tracked on all writes through maintenance of a copy of the data.
  - On all reads, the copy of the data fetched by the program is checked against the one stored as part of the data signature (ensure data integrity).
- **SymPLAID tool is available as a research prototype**
  - Can analyze applications compiled for SPARC or MIPS ISA.
  - Demonstrated on real-world applications, including embedded code and network applications, e.g., SSH.

Broader Impact

- Trusted signature derived using SymPLFIED can be used as a basis for implementing (using software or specialized hardware) runtime error-checking modules.

Interaction with Other Projects

- Collaborate with the activity on providing application-specific reliability and security support.

Future Efforts

- Apply to broader set of applications.
- Address issues of scalability.
- Build frontend to support x86 architecture.