Trustworthy Cyber Infrastructure for the Power Grid

Vulnerability Assessment Tool Using Model Checking

tcipg.org

Overview and Problem Statement

This activity focuses on identification of potential vulnerabilities in software. We use a combination of symbolic execution and model checking to systematically analyze corruptions of the application data and identify cases that lead to a successful attack. The results from the analysis can then be used to remove application-level vulnerabilities and guide design of defense mechanisms to protect applications from attacks.

Research Objectives

- 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., instruction sequences 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.

Technical Description and Solution Approach

The core component of our approach is SymPLFIED, a formal framework that uses symbolic execution and model checking to:

- Identify 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).
- Determine memory locations that need to be protected to prevent application failure or system compromise.

The proposed overall formal framework is shown in the below figure.

Results and Benefits

The analysis conducted using the SymPLFIED-based formal framework allows generation of trusted signatures, which can be used to implement runtime checking. They consist of two parts:

- *Instruction Signature:*
    - Stores unique identifiers (e.g., PC) for critical instructions.
    - Only critical instructions can write to critical data (prevents code injection).
- *Data Signature:*
    - Stores 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 (ensures data integrity).

The SymPLAID tool is available as a research prototype and can analyze applications compiled for SPARC or MIPS ISA (Instruction Set Architecture). The tool has been demonstrated on real-world applications, including embedded code and network applications, e.g., SSH.

Researchers

- Prateek Patel, patelprateek@gmail.com
- Zbigniew Kalbarczyk, kalbarcz@illinois.edu
- Ravishankar Iyer, rkiyer@illinois.edu

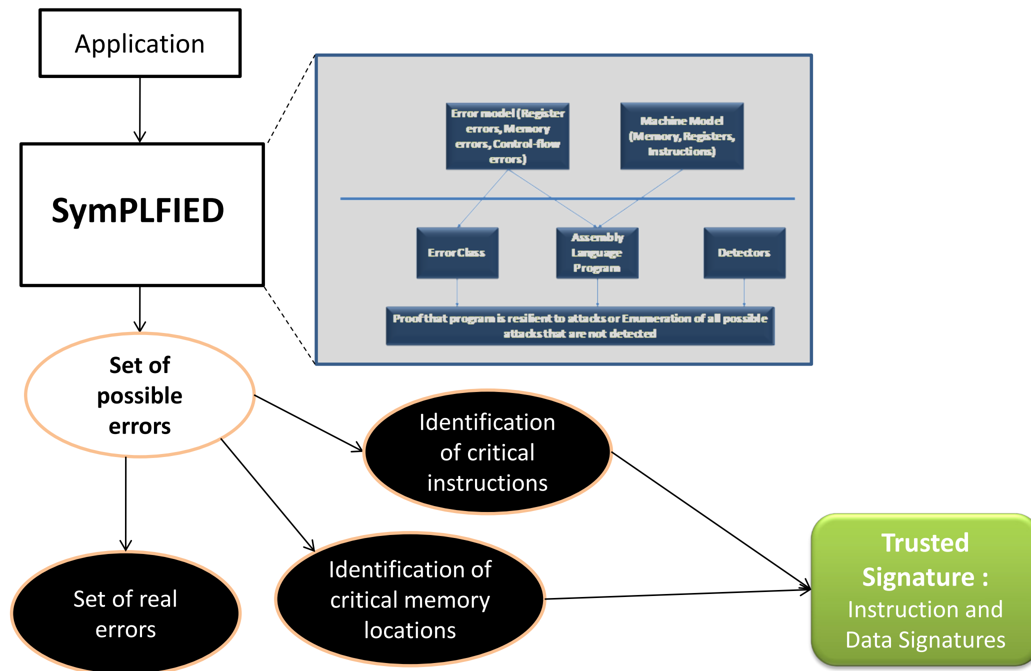

Figure 1: Formal Analysis Framework