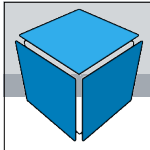
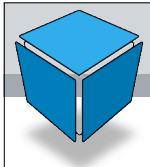


Experiment Based Validation of CIIP

Per Mellstrand and Rune Gustavsson

Blekinge Institute of Technology
School of Engineering

Societies of Computation Laboratory



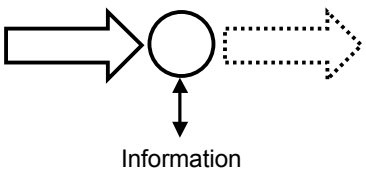
Setting the Scene

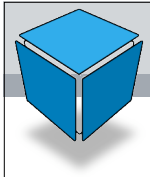
Critical Infrastructure Protection

Prevent Unwanted Behavior

Different System Levels

Basic Approach





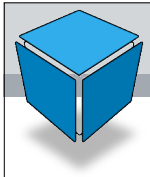
Critical Information

Different types of *Critical Information*:

I_1 Information Interpreted by Users
System Information Payload

I_2 Information Exchanged Between two Infrastructures
High-level Communication Information

I_3 Information that Enable Proper Execution
Low-level Execution Information

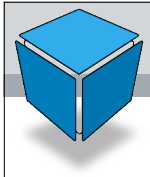


Experimental Approach

Experimental Approach - Why?
- Existing Principles - *Good!*
- Audited Components - *Good!*
- Skilled People Adopting the Systems - *Very Good!*

Avoiding Unwanted Execution
- What constitutes unwanted (wanted) Execution?

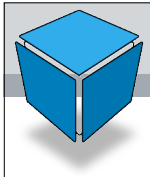
Is this sufficient? - *Probably* not.
- Probably - we don't know.



Experimental Approach

Experimental Approach - Why?
- Avoiding Unwanted Execution
  - We need information to tell unwanted from wanted!

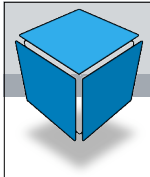
Experimental Approach - How?
- Topic of this presentation
  - Technical *Mechanisms* - to extract information
  - Iterative Method - to elicit relevant information
  - Example



1. Extraction Mechanisms

Extracting Information from Executing Software
- Identifying Information about the *actual execution.*

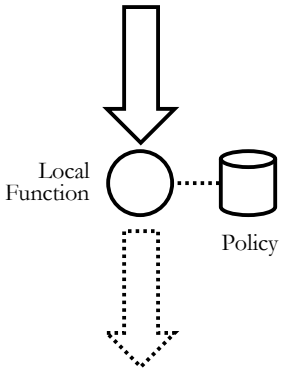
Example Mechanisms:
- Tracing Operating System Calls - *Almost in every operating system!*
- Analyzing Public Interfaces - *COM, CORBA,* …
- Function Calls Inside Program - *debuggers*, plibc library

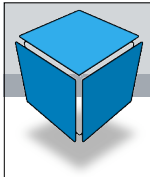


Example Mechanism

The *plibc* policy execution engine

- Execution policy on a local function-level
- Analyzes local state and assures function calls and parameters comply with policy
- Engine can take a number of actions on a non-cooperating program
- plibc was presented on CRIS2004 the conference

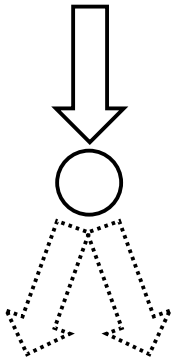


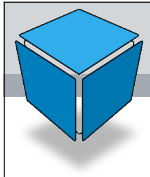


2. Modification Mechanisms

Where we can measure we can modify!

- Fault Injection
- Hardening / Fault Tolerance
- Logging / Tracepoints



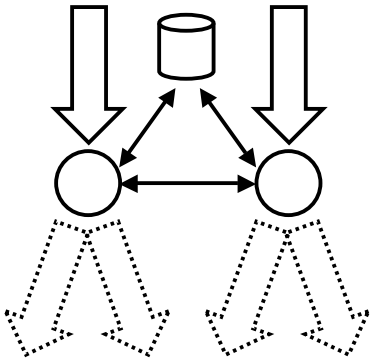


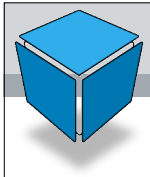
3. Environments

Environments modify the programs' view of the reality

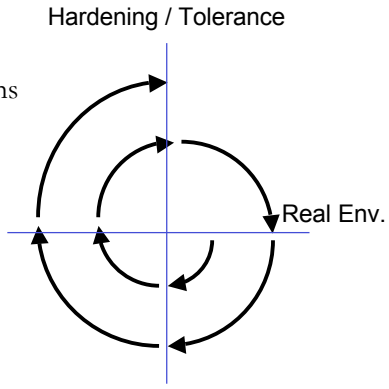
Connecting mechanisms at different levels

Applying External Stress





4. Iterative Approach

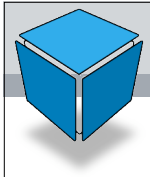


Knowing where to place mechanisms require good information

Placing good mechanisms produce good information

Mixing Hardening/Tolerance with Fault Injection and Probing

Where to start, and when/where to stop?



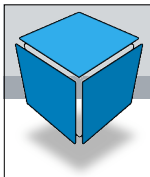
Outcomes

Extraction Mechanisms - *Extracts Relevant Information* (I_3)

Modification Mechanisms - *Modifies Execution/Behavior*

Environments - *Connected Mechanisms*

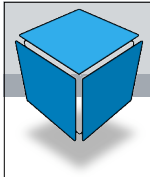
Method - *Iteratively Identifies Relevant Information and Behavior*



Example: fetch/OpenSSL

Example program/component:
fetch using OpenSSL 0.9.7e

Environment
Clean FreeBSD 6.1-STABLE/i386



Environment Modification

Remote Environment

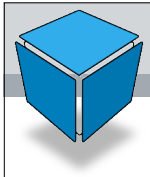
External Experiment Controller Set policies and collected data

Local Environment

plibc trigged various memory management functions

Low-probability stochastic faults in memory alloc (0.1 - 0.01%)

Hooking trace-points where faults were inserted



Feedback

Analysis

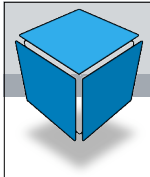
A number of typical crashes, minor issues in OpenSSL

Memory handling error over 23 different functions in 16 files

fetch would randomly 'hang' forever - underlying string functions

Feedback & Next Cycle

Existing Wrapper functions are ineffective/not used

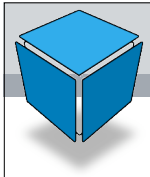


Experiment Results

Immediate results first iteration
- string formatting in fetch
- memory allocation in OpenSSL

Protecting Environment
- memory allocations - trace / trigger
- string formatting / hang - ?

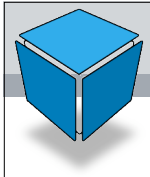
Further Experimentation
- high-level OpenSSL memory allocation
- probing string functions for error conditions



Relevance

Relevance of *Example* / *Experiment*:
- Why not just fix the program? - **Embedded / Modified use.**
- Is this Actually in Use? - **Yes.**

Relevance of *Method*:
- Can we use this to find real vulnerabilities - **Yes.**
- Is domain-specific testing relevant? - **Yes.**



Correctness

Correctness of *Example / Experiment*:

Does the mechanism/environment cause/hide faults on its own?

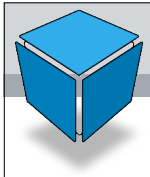
Performance issues with monitored execution

Correctness of *Method*:

Can we prove that all/more vulnerabilities are found?

How to know where to place traces/modifications/protection?

What are the stop-conditions?



Conclusions

Experiment Based Validation of C(I)I

Provides Useful Information about Program Execution

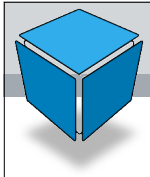
Information which can be used to Protect Execution

Mechanisms & Connections

Mechanisms to extract and modify Execution

Method

Iteratively Extracting More Specific Information

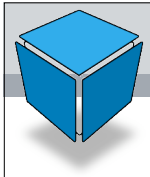


Future Work

Large Experiments

Enhanced Information bus Between Components

Industrial Cooperation



Questions & Comments