LISSOM, A Source Level *Proof Carrying Code* Platform for the Safe Execution of Mobile Code

Daniel Martins\(^1\) Simão Melo de Sousa\(^1\)

\(^1\)Computer Science Department  
University of Beira Interior, Portugal

SINO, 2005
Outline

1. Background
   - Security Issue in Mobile Code Execution
   - The Actual Picture

2. Proof Carrying Code Architectures
   - Underlying Principle
   - Machine Level PCC
   - Source Level PCC

3. LISSOM
   - Underlying Motivation
   - The Components
   - The Glue

4. Conclusion
Outline

1. Background
   - Security Issue in Mobile Code Execution
   - The Actual Picture

2. Proof Carrying Code Architectures
   - Underlying Principle
   - Machine Level PCC
   - Source Level PCC

3. LISSOM
   - Underlying Motivation
   - The Components
   - The Glue

4. Conclusion
Security Issue in Code Mobility

Mobile Code Paradigm

- Interconnected World = Reality $\implies$ Computer = One Execution Node;
- Mobile code paradigm $\implies$ security issues.

Problem: Mobile Code

- can come from unknown source;
- can be produced by unknown means;
- can be modified during its life cycle;
- or, simply, can be malicious.

Classical Solutions

- Code Signing;
- Sandboxing;
  - Runtime Checking;
  - Software Fault Isolation;
- Firewall;
- Static Analysis of Code.
The Actual Picture

Runtime Verification
- Performance penalty;
- Checks performed for every execution.

Static Verification
- Static Checking = Static Security Policies;
- Changing the Security Policy $\implies$ complete re-implementation.
Outline

1 Background
   - Security Issue in Mobile Code Execution
   - The Actual Picture

2 Proof Carrying Code Architectures
   - Underlying Principle
   - Machine Level PCC
   - Source Level PCC

3 LISSOM
   - Underlying Motivation
   - The Components
   - The Glue

4 Conclusion
The Actors

The Code Consumer
- Knows better what is safe for him;
- Require services from outside;
- Must verify if these services are compliant with its own safety requirements.

The Code Producer
- Knows better how its code is built and behaves;
- Must ensure that its code is safe.

The Game
- The Code Producer must provide to the Code Consumer a certificate that its code is safe;
- After successful verification the Code Consumer can safely run the code (with no runtime verification).
Certificates of Innocuousness

Certificates = Proof Objects

- Computer objects that represents proofs;
- As in real life: potentially difficult to build, easy to verify.

Proof Systems

- Several proof systems support proof objects;
- provide a formal language for the expression of security policies;
- provide means for the construction and the verification of formal proofs of security policy compliance.
Machine Level PCC in a Picture

**Producer Side**

- Source Code
- Compiler
- VCGEN_{machine}
- Proof System
- Compiled Code + Certificate

**Loading Stage**

- Proof Obligations

**Consumer Side**

- Security Policies
- Execution Platform
- Proof Checker
- VCGEN_{machine}

- Code
- Certificate

- NO

**Proof Carrying Code Architectures**

- LISSOM
  - Underlying Principle
  - Machine Level PCC
  - Source Level PCC

D. Martins, S. Melo de Sousa  LISSOM
Machine Level PCC

Architecture “à la” Necula

- Machine Level PCC architecture for Java;
- Automatic certificate generation;
- Trusted Computing Base (TCB) relatively important;
- Security policies are somewhat low level (by nature).
Machine Level PCC

Architecture “à la” Necula

- Machine Level PCC architecture for Java;
- Automatic certificate generation;
- Trusted Computing Base (TCB) relatively important;
- Security policies are somewhat low level (by nature).

Security Policy Examples:

There is no “write” operations on “read-only” register
or, more formally

\[ \forall r_i, r_j \in \text{Registers}, \text{store } r_i r_j \implies r_i \in \text{Writable \ Registers} \]
Source Level PCC in a Picture

**Producer Side**

- Source Code
  - VCGEN\text{Source}
    - Proof Obligations
    - Certificate
    - Compiler
  - Compiled Code + Certificate

**Consumer Side**

- Security Policies
  - Execution Platform
    - Proof Checker
      - Code \(\checkmark\)
      - NO
    - VCGEN\text{machine}
      - Proof Obligations
      - Certificate
      - Code
        - Loading Stage
Source Level PCC

Highlights

- The security policy compliance is ensured at source code level;
- Needs 2 VCGENs (proof obligations generators);
- The compiler must be able to translate certificates;
- At this time, there is no complete PCC architecture of this paradigm.
Highlights

- The security policy compliance is ensured at source code level;
- Needs 2 VCGENs (proof obligations generators);
- The compiler must be able to translate certificates;
- At this time, there is no complete PCC architecture of this paradigm.

Security Policy Example

- There is no information flow from the variable \( x \) to the variable \( y \);
- There is no transitive information flow from the applets \( A \), \( B \) and \( C \).

Achievements

The first compiler with certificate translation: Spring 2005 by the Everest team from INRIA-France
Outline

1. Background
   - Security Issue in Mobile Code Execution
   - The Actual Picture

2. Proof Carrying Code Architectures
   - Underlying Principle
   - Machine Level PCC
   - Source Level PCC

3. LISSOM
   - Underlying Motivation
   - The Components
   - The Glue

4. Conclusion
Our Focus

Claims:
System Designers (the code producers) care about high (source) level concept and source code. Source Level PCC is the way

Facts:
- There is no complete source level PCC (again);
- There are very good tools for the (annotated) source code formal verification (e.g. Java Modeling Language (JML) and friends, Spec#, etc.).

Status
LISSOM, A work in progress source level PCC platform.
LISSOM Architecture

Highlights

- Compiler = LISS language and compiler (available);
- $VCGEN_{source}$ = the WHY Tool (available);
- Proof System and Proof Checker = the COQ Proof Assistant (available);
- Execution platform = a stack based virtual machine (like the Java VM) (available);
- $VCGEN_{machine}$ for the VM language (to do);
LISSOM Architecture

Highlights

- JML-like Annotation Language for LISS upon the WHY Tool (in progress);
- Design of a Proof System for LISS upon the COQ Proof Assistant (in progress);
- Design of a Certificate Translator for the LISS compiler (in progress);
- Design of a Proof System for virtual machine upon the COQ Proof Assistant (to do).

Design implications

Trusted Computing Base: The COQ Proof Checker
Outline

1 Background
   - Security Issue in Mobile Code Execution
   - The Actual Picture

2 Proof Carrying Code Architectures
   - Underlying Principle
   - Machine Level PCC
   - Source Level PCC

3 LISSOM
   - Underlying Motivation
   - The Components
   - The Glue

4 Conclusion
Conclusion

About LISSOM

- LISSOM is an attempt to fill the gap between seducing concepts and real life needs;
- LISSOM is at an early development stage.

Actual Focus

The compiler with certificate translation + JML-like annotation system.