Memory Usage Estimation for Java Smart Cards

GERARDO SCHNEIDER

IRISA/INRIA - RENNES, FRANCE

CASTLES: *Conception d'Analyses Statiques et de Tests pour le Logiciel Embarqué Sécurisé*

NWPT'04 - Uppsala, 06 October 2004

IRISA

institut de recherche en informatique et systèmes aléatoires

Overview

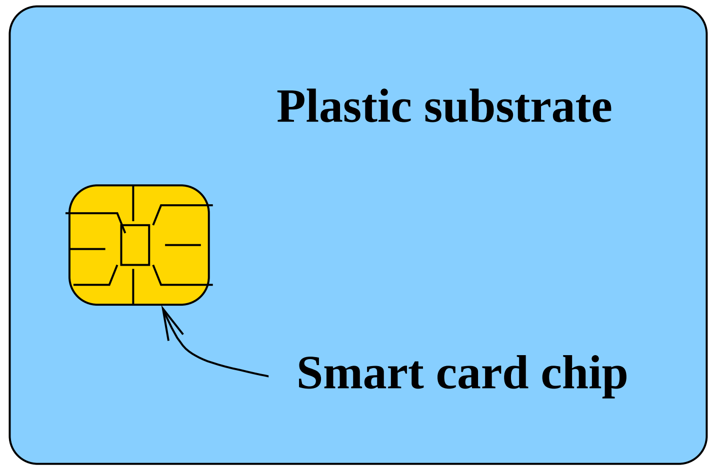
- Introduction and motivation
- Objective - Our approach
- Our solution
- Final discussion

IRISA
institut de recherche en informatique
et systèmes aléatoires

Introduction and Motivation

IRISA
institut de recherche en informatique
et systèmes aléatoires

Smart cards



- Small communicating devices with restricted resources
- Execute stand-alone applications specifically written for the hardware it runs on

IRISA
institut de recherche en informatique et systèmes aléatoires

New generation of Java smart cards

- High-level language for programming applets (JavaCard Language)
- Multi-application: various applets may be downloaded and interact in the same card
- Post-issuance: applets may be loaded on the card after issued by the manufacturer

Size (banking - high-tech cards): EEPROM (16K - 200K), ROM (16K - 64K), RAM (1K - 4K)

Applications: mobile phones, e-purse, e-identity, medical file management, etc

Security Issues

Downloaded applets may attack by leaking or modifying confidential information, causing malfunctioning, etc

IRISA
institut de recherche en informatique
et systèmes aléatoires

Security Issues

Downloaded applets may attack by leaking or modifying confidential information, causing malfunctioning, etc

The “Sandbox” model relies on that applets are:

- Compiled to bytecode for a virtual machine
- Not given direct address to hardware resources
- Subject to a static analysis: bytecode verification (check applets are well-typed)

Security Issues (cont.)

Extension of the bytecode verifier are needed to guarantee (among others)

- Information flow (i.e. an applet does not “leak” confidential information)
- Reactiveness (bounding the running time of the applet between two interactions with the environment)
- Availability of services

Security Issues (cont.)

Extension of the bytecode verifier are needed to guarantee (among others)

- Information flow (i.e. an applet does not “leak” confidential information)
- Reactiveness (bounding the running time of the applet between two interactions with the environment)
- Availability of services (resource-awareness analysis - Memory)

How to program in small devices?

Quoted from “Java Card Technology for Smart Cards - Sun Series” [Chen,2000; Chapter 13]

- “...neither persistent nor transient objects should be created willy-nilly.”
- “You should also limit nested method invocations...”
- “..applets should not use recursive calls.”
- “An applet should always check that an object is created only once.”

The problem

- Nothing in the standards prevents a(n) (intentionally) badly written applet to allocate all persistent memory on a card!
- State-of-the-art tools do not detect whether a given applet will make the card run out of memory

Example:

```
public class Example
    ...
    while(arg > 0)
        new Example();
    ...
```

IRISA
institut de recherche en informatique et systèmes aléatoires

Objectives - Our Approach

Objective

An analyser for estimating memory usage on Java smart cards, which

- Statically analyses the bytecode
- Does not assume any structure on the bytecode
- Comprises intra- and Inter-procedural analysis
- Is as precise as possible
- Is compositional
- Has low complexity (on-card analyser)

Objective (Cont.)

The technique used should allow us to:

- Develop a **certified analyser**
- Extract a correct analyser

Moreover, we want the formalism to be compatible with previous work (certified Data Flow Analyser developed at IRISA)

IRISA
institut de recherche en informatique
et systèmes aléatoires

How to obtain a certified analyser?

- Formalise the operational semantics of the language in a Proof Assistant (Coq)
- Define the abstract domains (lattices)
- Prove well-foundedness of the lattices
- Code the algorithm into Coq (as a constraint-based algorithm)
- Prove the correctness of the algorithm w.r.t. (an abstraction of) the operational semantics
- Extract a program (proof-as-program paradigm) using Coq's extraction mechanism

IRISA
institut de recherche en informatique et systèmes aléatoires

How to obtain a certified analyser?

- Formalise the operational semantics of the language in a Proof Assistant (Coq)
- Define the abstract domains (lattices)
- Prove well-foundedness of the lattices
- Code the algorithm into Coq (as a constraint-based algorithm)
- Prove the correctness of the algorithm w.r.t. (an abstraction of) the operational semantics
- Extract a program (proof-as-program paradigm) using Coq's extraction mechanism

IRISA
institut de recherche en informatique et systèmes aléatoires

Our Solution

The JavaCard bytecode language

- Stack manipulation: `push`, `pop`, `dup`, `dup2`, `swap`, `numop`;
- Local variables manipulation: `load`, `store`;
- Jump instructions: `if`, `goto`;
- Heap manipulation: `new`, `putfield`, `getfield`;
- Array instructions: `arraystore`, `arrayload`;
- Method calls and return: `invokevirtual`, `invokedefinite`, `invokeinterface`, `return`

Algorithm - Outline

- Detection of potential intra-method loops ($Loop$)
- Propagation of $Loop$ inter-procedurally
- Detection of (mutually) recursive methods and methods reachable from those (Rec)
- Identification of dynamic instantiation of classes (Γ)

What is new about it?

Audience: But we know how to detect cycles in (assembly-like) programs!! (Compiler...)

What is new about it?

Audience: But we know how to detect cycles in (assembly-like) programs!! (Compiler...)

Answer: Yes.

What is new about it?

Audience: But we know how to detect cycles in (assembly-like) programs!! (Compiler...)

Answer: Yes.

Audience: What is the challenge, then?

What is new about it?

Audience: But we know how to detect cycles in (assembly-like) programs!! (Compiler...)

Answer: Yes.

Audience: What is the challenge, then?

Answer: To write a constraint-based algorithm suitable to be formalised in Coq and to extract a certified analyser

IRISA
institut de recherche en informatique
et systèmes aléatoires

What is new about it?

Audience: But we know how to detect cycles in (assembly-like) programs!! (Compiler...)

Answer: Yes.

Audience: What is the challenge, then?

Answer: To write a constraint-based algorithm suitable to be formalised in Coq and to extract a certified analyser

Presented as a set of rules defining one (or more) constraint(s) for each bytecode instruction

Algorithm - Constraints

The constraints are of the form:

$$\frac{(m, pc) : \texttt{Instr} \qquad \texttt{Cond}}{F(\Delta(m, pc)) \sqsubseteq \Delta(m', pc')}$$

- $\texttt{Instr}$ is the current instruction
- $\texttt{Cond}$ is a set of conditions (predicate)
- F is a monotonic function
- Δ is the *context* being generated
- (m', pc') is the *next* instruction

IRISA
institut de recherche en informatique et systèmes aléatoires

Detecting loops ($Loop$)

$$\frac{(m, pc) : \texttt{goto}\ pc'}{F_1(Loop(m, pc)) \sqsubseteq Loop(m, pc')}$$

$$\frac{(m, pc) : \texttt{if}\ t\ op\ \texttt{goto}\ pc'}{\begin{array}{c} F_1(Loop(m, pc)) \sqsubseteq Loop(m, pc') \\ F_3(Loop(m, pc)) \sqsubseteq Loop(m, pc+1) \end{array}}$$

$$\frac{(m, pc) : \texttt{invokevirtual}\ m'}{Loop(m, pc) \sqsubseteq Loop(m, pc+1)}$$

$$\frac{(m, pc) : \texttt{return}}{\bot \sqsubseteq Loop(m, \mathrm{END}_m)} \qquad \frac{(m, pc) : \texttt{Instr}}{Loop(m, pc) \sqsubseteq Loop(m, pc+1)}$$

IRISA
institut de recherche en informatique et systèmes aléatoires

Detecting recursive methods (Rec)

$$\frac{(m, pc) : \texttt{invokevirtual}\ m'}{\begin{array}{l} F(Rec(m, pc), m') \sqsubseteq Rec(m', 1) \\ Rec(m, pc) \sqsubseteq Rec(m, pc + 1) \end{array}}$$

$$\frac{(m, pc) : \texttt{return}}{Rec(m, pc) \sqsubseteq Rec(m, \mathrm{END}_m)}$$

$$\frac{(m, pc) : \texttt{Instr}}{Rec(m, pc) \sqsubseteq Rec(m, pc + 1)}$$

IRISA
institut de recherche en informatique
et systèmes aléatoires

The algorithm - Γ

$$\frac{(m,pc) : \mathtt{new}(cl) \qquad Cycle(m,pc)}{\Gamma(m,pc) \cup \{<!>_{(m,pc)}\} \sqsubseteq \Gamma(m,pc+1)}$$

$$\frac{(m,pc) : \mathtt{new}(cl) \qquad \neg Cycle(m,pc)}{\Gamma(m,pc) \cup \{(m,pc)\} \sqsubseteq \Gamma(m,pc+1)}$$

$$\frac{(m,pc) : \mathtt{Instr}}{\Gamma(m,pc) \sqsubseteq \Gamma(m,pc+1)}$$

IRISA
institut de recherche en informatique
et systèmes aléatoires

Algorithm - How does it work?

- The abstract domains (lattices) chosen and the "form" of the constraints guarantees the existence of a *least fix-point*
- The well-foundedness of the lattices guarantees termination
- A constraint solver computes the least fix-point

IRISA
institut de recherche en informatique et systèmes aléatoires

Final Discussion

IRISA
institut de recherche en informatique
et systèmes aléatoires

Achievements

- We have written a constraint-based algorithm for detecting possible memory overflow due to dynamic instantiation of classes inside cycles

Already done:

- Handwritten proof of
  - Termination
  - Soundness and completeness w.r.t. to an abstraction of the operational semantics

IRISA
institut de recherche en informatique et systèmes aléatoires

Features of our algorithm

+ Written in a “good” way to be fed into Coq (certification)
+ Modular; $Loop$ and Rec reusable
+ Compositional
+ Static analysis

? Low computational complexity

– Over-approximation:
  - It detects (all the) syntactic cycles
  - An instruction in a method (not in a cycle) called more than once is counted once

Current Work

Currently adapting the algorithm slightly in order to reuse (in Coq):

- Lattice library
- Auxiliary lemmas
- Fix-point and constraint solver
- Proof strategies

Current Work

Currently adapting the algorithm slightly in order to reuse (in Coq):

- Lattice library
- Auxiliary lemmas
- Fix-point and constraint solver
- Proof strategies

Current approach: We considered a maximal semantics (total runs of the program)

New approach: We have to consider a partial semantics (prefixes of runs of the program)

IRISA
institut de recherche en informatique et systèmes aléatoires

Future Work

Still to be done:

- A more precise analysis: Exact amount of memory used if no `new` occurs in a cycle
- “Implement” the algorithm we have presented in Coq and extract the analyser
- Compare performance of both approaches: complexity Vs simplicity of proofs

Besides this work:

- Other techniques for resource-bounded analysis and other security properties

IRISA
institut de recherche en informatique et systèmes aléatoires

Thank you very much!

IRISA
institut de recherche en informatique et systèmes aléatoires

Rules for *Loop*

$$\frac{(m, pc) : \texttt{goto}\ pc' \qquad pc' \leq pc}{F_1(Loop(m, pc)) \sqsubseteq Loop(m, pc')}$$

$$\frac{(m, pc) : \texttt{goto}\ pc' \qquad pc' > pc}{F_2(Loop(m, pc)) \sqsubseteq Loop(m, pc')}$$

$$\frac{(m, pc) : \texttt{if}\ t\ op\ \texttt{goto}\ pc' \qquad pc' \leq pc}{\begin{array}{c} F_1(Loop(m, pc)) \sqsubseteq Loop(m, pc') \\ F_3(Loop(m, pc)) \sqsubseteq Loop(m, pc+1) \end{array}}$$

$$\frac{(m, pc) : \texttt{if}\ t\ op\ \texttt{goto}\ pc' \qquad pc' > pc}{\begin{array}{c} F_2(Loop(m, pc)) \sqsubseteq Loop(m, pc') \\ F_3(Loop(m, pc)) \sqsubseteq Loop(m, pc+1) \end{array}}$$

IRISA
institut de recherche en informatique et systèmes aléatoires

Rules for *Loop* (cont.)

$$\frac{(m, pc) : \texttt{invokevirtual}\ m'}{Loop(m, pc) \sqsubseteq Loop(m, pc+1)}$$

$$\frac{(m, pc) : \texttt{return}}{\bot \sqsubseteq Loop(m, \mathrm{END}_m)}$$

$$\frac{(m, pc) : \texttt{Instr}}{Loop(m, pc) \sqsubseteq Loop(m, pc+1)}$$

Definition of the functions

$$F_1(L_{m,pc}) = \begin{cases} L_{m,pc} \cup \{Yes_{pc}\} & \text{if } \{pc, pc'\} \subseteq L_{m,pc} \\ L_{m,pc} \cup \{pc, pc'\} & \text{otherwise} \end{cases}$$

$$F_2(L_{m,pc}) = \begin{cases} L_{m,pc} \setminus \mathbb{Y}_{<pc'} & \text{if } \{pc, pc'\} \subseteq L_{m,pc} \\ (L_{m,pc} \setminus \mathbb{Y}) \cup \{pc, pc'\} & \text{otherwise} \end{cases}$$

$$F_3(L_{m,pc}) = \begin{cases} L_{m,pc} \setminus \mathbb{Y}_{<pc+1} & \text{if } \{pc, pc+1\} \subseteq L_{m,pc} \\ (L_{m,pc} \setminus \mathbb{Y}) \cup \{pc, pc+1\} & \text{otherwise} \end{cases}$$

Where $\mathbb{Y}_{<pc'} \stackrel{\text{def}}{=} \{Yes_{pc} \mid pc < pc'\}$

IRISA
institut de recherche en informatique et systèmes aléatoires

Rules for Rec

$$\frac{(m, pc) : \texttt{invokevirtual}\ m' \quad m = m'}{\begin{array}{l} Rec(m, pc) \cup \{m, Yes\} \sqsubseteq Rec(m', 1) \\ Rec(m, pc) \sqsubseteq Rec(m, pc+1) \end{array}}$$

$$\frac{(m, pc) : \texttt{invokevirtual}\ m' \quad m \neq m'}{\begin{array}{l} F(Rec(m, pc), m') \sqsubseteq Rec(m', 1) \\ Rec(m, pc) \sqsubseteq Rec(m, pc+1) \end{array}}$$

$$\frac{(m, pc) : \texttt{return}}{Rec(m, pc) \sqsubseteq Rec(m, \mathrm{END}_m)}$$

$$\frac{(m, pc) : \texttt{Instr}}{Rec(m, pc) \sqsubseteq Rec(m, pc+1)}$$

IRISA
institut de recherche en informatique et systèmes aléatoires

Definition of F

$$F(R_{m,pc}, m') = \begin{cases} R_{m,pc} \cup \{m, Yes\} & \text{if } \{m'\} \in R_{m,pc} \\ R_{m,pc} \cup \{m\} & \text{if } \{m'\} \notin R_{m,pc} \end{cases}$$

IRISA
institut de recherche en informatique et systèmes aléatoires

Example of *Loop*

```
20 ...             {30,50,31,41,40,70,20,Y70}
30 if goto 50      {30,50,31,41,40,70,20,Y70}
   ...             {30,31,50,41,40,51,70,20,Y70}
40 if goto 90      {30,31,50,41,40,51,70,20,Y70}
   ...             {30,31,41,40,50,51,70,20,Y70}
50 if goto 90      {30,31,41,40,50,51,70,20,Y70}
   ...             {30,31,41,40,50,51,70,20,Y70}
70 goto 20         {30,31,41,40,50,51,70,20,Y70}
   ...
90 ...             {30,31,40,90,41,50,51,70,20}
```

(b)

IRISA
institut de recherche en informatique et systèmes aléatoires

Example of *Loop*

```
30 if goto 50
31 goto 49      {30,31}
   ...
40 goto 60      {30,50,31,49,40}
   ...
49 if goto 60   {30,31,49}
50 goto 40      {30,50,31,49}
   ...
60 ...          {30,31,49,60,40}
```

(a)

IRISA
institut de recherche en informatique et systèmes aléatoires