Inline-Reference Monitor Optimization using Automata Modulo Theory (AMT)

Fabio Massacci
Ida Siahaan
Inlined Reference Monitors

- Policy Enforcement Toolkit (PoET)
  - Implementing IRMs for Java Virtual Machine Language (JVML) applications
  - Primary concern: trusted computing base (TCB) 17,500 loc Java source code
Optimizing Security Policy or Rewriter

- Security Automata SFI Implementation (SASI)
  - Implementing IRMs for x86 and JVML
  - Minimizing TCB by working at the level of object code
Optimizing Security Policy or Rewriter

- **Trade off between moving more processes out of trusted part and the complexity of the whole process**

- **Efficient IRM Enforcement**
  - a constrained representation of history-based access control policies
  - exploit the structure of this policy representation
  - extended into a distributed optimization protocol
• **SxC device view**
  
Given an (un)trusted code and a policy that a platform specifies to be inlined, how can we obtain an optimized IRM?
Components of IRM Optimization

- **Contract Extractor**
  - extract *security relevant behaviors* from code
Components of IRM Optimization

- **Claim Checker**
  - *verify that the claimed contract complies to the code*
  - *digitally signed by a trusted code provider*
Components of IRM Optimization

- **Simulation Checker**
  - *check a policy simulates a contract*
Components of IRM Optimization

- **Optimizer**
  - *discharge behaviors which are already enforced by code*
Components of IRM Optimization

- Rewriter
  - inject policy to the code
IRM Optimization Models
Rewriter on Trusted part

Model 1: Contract Extractor on Trusted part

- Contract Extractor
- Security Policy Application Contract
- Simulation Checker
- Optimizer
- Optimized Security Policy
- Rewriter

Untrusted Code

Execute

 Inline Application Code
Optimizer and Rewriter on Untrusted part

Model 6: Contract Extractor on Untrusted part
Automata Modulo Theory (AMT)
Security Automata

- A class of Büchi automata that accept safety properties (recognizers)
  - a countable set $Q$ of automaton states,
  - a countable set $Q_0 \subseteq Q$ of initial automaton states,
  - a countable set $I$ of input symbols, and
  - a transition function $\delta : (Q \times Q) \rightarrow 2^Q$

Edit Automata

• **Truncation automaton (recognizer)**
  – terminate application

• **Suppression automaton (transducer)**
  – truncation automaton + suppress undesired or dangerous actions without necessarily terminating the program

• **Insertion automaton (transducer)**
  – truncation automaton + insert additional actions into the event stream

• **Edit automata = Suppression automaton + Insertion automaton**
Automata Modulo Theory (AMT)

- **AMT = Büchi automata + Satisability Modulo Theories (SMT)**
  - a set $E$ of formulae in the language of the theory $T$ as *input symbols*
  - a finite set $Q$ of *automaton states*,
  - an *initial state* $q_0 \in Q$,
  - a set $F \subseteq Q$ of *accepting states*, and
  - a *labeled transition function* $\delta : (Q \times E) \rightarrow 2^Q$

- F. Massacci, I. Siahaan, “Matching midlet’s security claims with a platform security policy using automata modulo theory.”, NordSec’07
Satisfiability Modulo Theories (SMT)

- The problem of deciding the satisability of a first-order formula with respect to some decidable first-order theory $T$ (SMT(T))
  - A $\Sigma$-theory is a set of first-order sentences with signature $\Sigma$
- Examples of theories of interest:
  - Equality and Uninterpreted Functions (EUF),
  - Linear Arithmetic (LA): both over the reals (LA(Q)) and the integers (LA(Z))
- Examples of SMT tools:
  - Z3
  - MathSAT
- Primary interest for SMT(T) when $T$ is a combination of two or more theories $T_1, \ldots, T_n$.
  - Example of an atom: $f(x + 4y) = g(2x - y)$
Example of AMT

(a) Infinite Transitions Security Policies

(b) Abbreviations for Java APIs
IRM Optimization using AMT
Searching an Optimized Policy

• **Given two automata C and P representing resp. the formal specification of a contract and of a policy, we have an efficient IRM OptP derived from P with respect to C when:**
  – every APIs invoked by the intersection of OptP and C can also be invoked by P [sound]
  – OptP is smaller than P with respect to C [optimal]
Contract-Policy Example

Contract

Policy
Removes non existing actions

Contract

Policy

Optimize 1

Policy
Removes already promised actions

Contract

Optimize 1 Policy

Optimize 2 Policy
Future Work

• Implementation and study of IRM with or without optimization

Assumption: \( t_{use} \gg t_{pre} \)
\( t_{us} \gg t_{dep} \)

\[
c \cdot t_{use} \gg c_{pre} \cdot (t_{pre} + t_{dep}) + c_{opt} \cdot (t_{use} - (t_{pre} + t_{dep}))
\]
Future Work

- Effect of changes both in frequency (how often a code modified) and size (how much a code modified).

\[ c \cdot t_{use} = c_{pre} \cdot (t_{pre} + t_{dep}) + c_{opt} \cdot (t_{use} - (t_{pre} + t_{dep})) \]
Thank you