S3MS Security of Software and Services for Mobile Systems

UNIVERSITÀ DEGLI STUDI DI TRENTO

Matching Policies with Security Claims of Mobile Applications

N. Bielova, M. Dalla Torre, N. Dragoni and I. Siahaan
Department of Information Engineering and Computer Science
University of Trento, ITALY
bielova@disi.unitn.it

ARES'08 - Barcelona , Spain

Research partly supported by the project EU-FP6-IST-STREP-S3MS
(http://www.s3ms.org)

Talk outline

- Security-by-contract
  - Introduction
  - Key concepts
  - Workflow
- Contract/Policy Matching
  - Prototype overview
  - Specifications language
  - Automata Modulo Theory
  - On-the-Fly Model Checking with Decision Procedure
- Conclusions

Motivation

- Mobile devices are increasingly popular and powerful
- Lack of applications for mobile devices
- Problems of current model based on trust relationship: *mobile code is accepted if it is digitally signed by a trusted party*
  - Signature can be either rejected or accepted
  - There is no semantic attached to the signature

Security by Contract
Key Concepts

- The key idea: (Dragoni et al., EuroPKI'07)
  - *the digital signature* should not just certify the origin of the code but rather bind together the code with a contract
  - Model-Carrying Code(Sekar et al.)
    - captures the *security-relevant behavior* of code
    - BUT finite-state automata
  - Design-by-contract (Meyer)

- Contract carried by application:
  - Claimed Security behavior of application;
  - (Security) interactions with its host platform

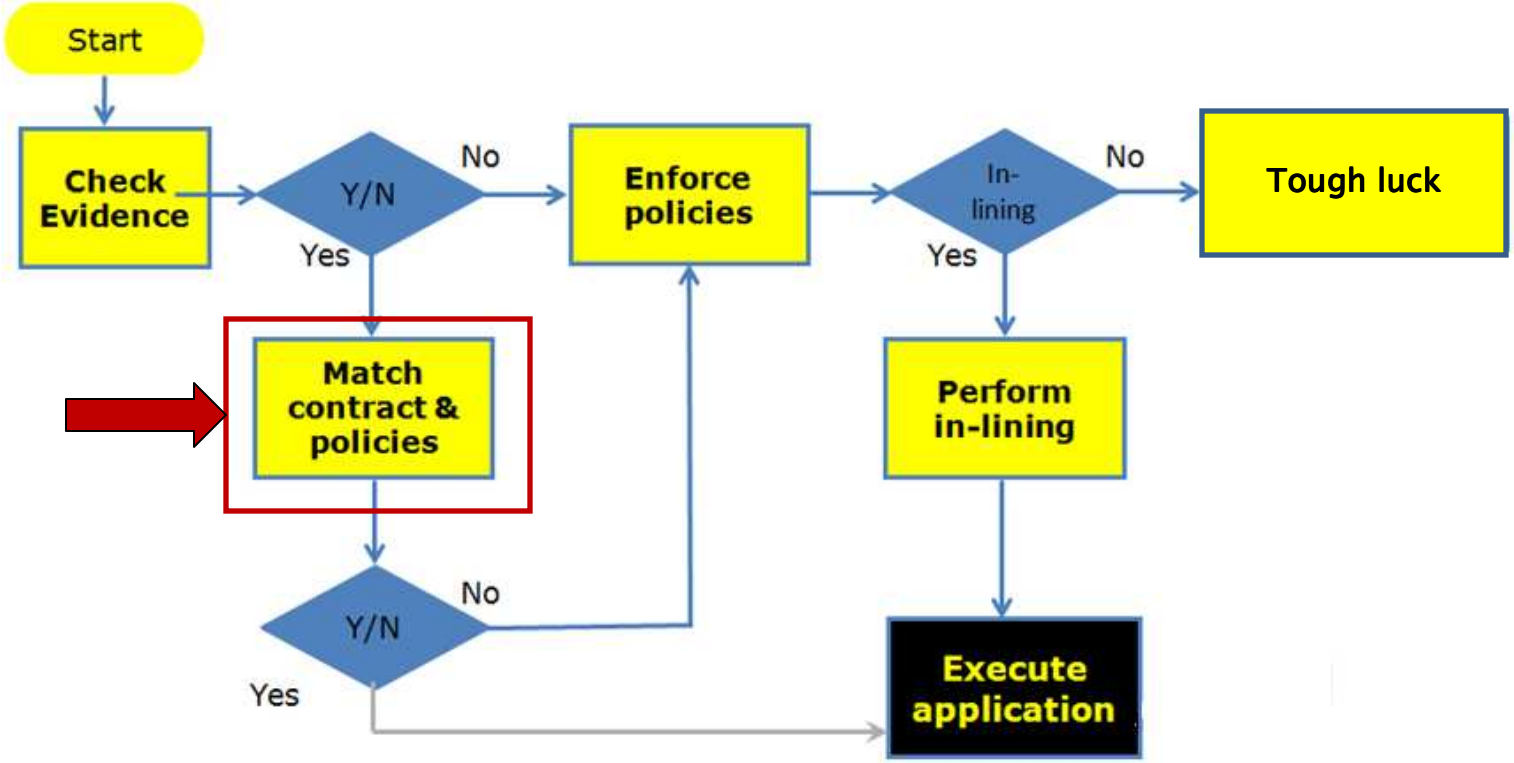
  Example: The application only uses HTTPS network connections

- Policy specified by a platform:
  - Desired Security behavior of application

  Example: The application should use only high-level (HTTP, HTTPS) network connections

Security-by-Contract workflow

- One of the key problems in the overall security-by-contract workflow is the contract-policy matching issue.



Contribution

- The algorithms presented:
  - meta-level algorithm (Dragoni et al. EuroPKI'07)
  - mathematical structure for algorithm to do the matching (Massacci & Siahaan, NordSec'07)
- Does it work in practice?
- Our main contribution of this paper is a proof of concept that shows that contract/policy matching is practical.

Language of contract/policy

- ConSpec – automata-based language
- The specifications in ConSpec is suitable for all phases of Security-By-Contract lifecycle
  - Contract / Policy Matching
  - Monitor In-lining
- Contract and Policy are mapped to the specific automata representation
- Matching = Language inclusion
  - all possible traces claimed by mobile code (contract automaton) all traces allowed by platform (policy automaton)

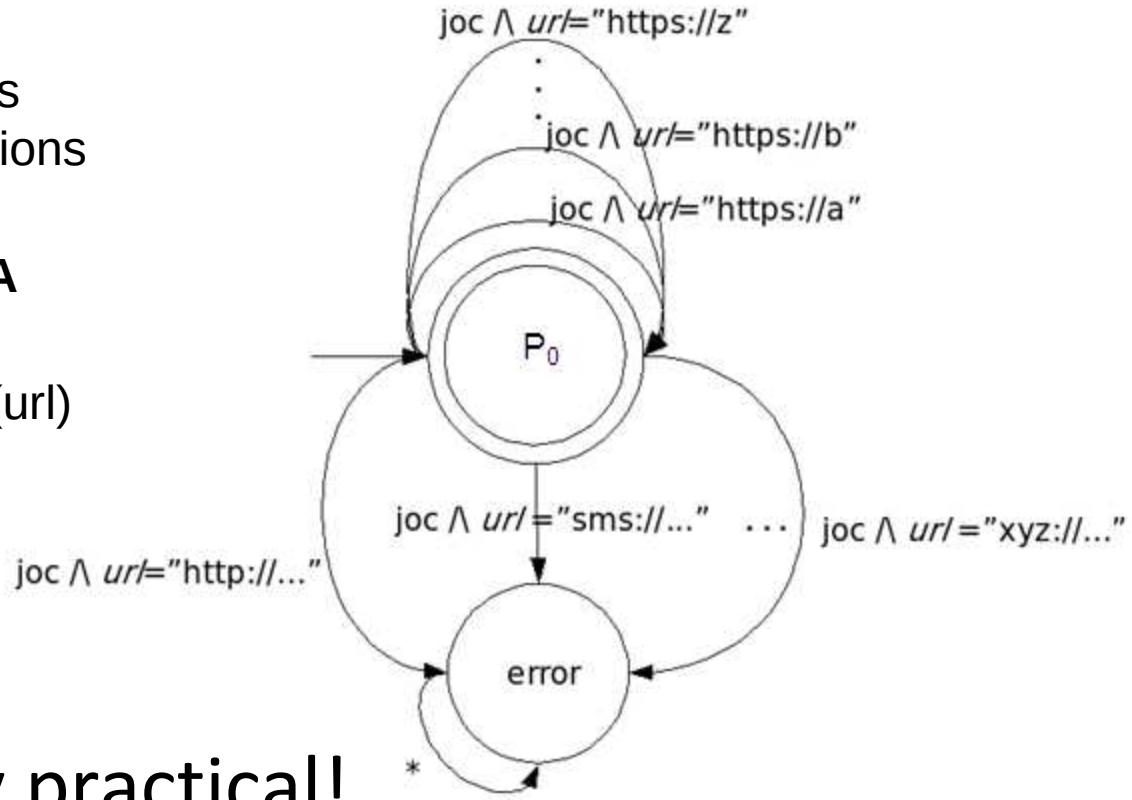
What kind of automaton?

- We need “infinite” edges to describe policies

CONTRACT:
The application only uses HTTPS network connections

Abbreviations for JAVA APIs:

joc = io.Connector.open(url)



- This is not very practical!

Automata Modulo Theory (*AMT)*

- AMT
  - Finite state automata with “infinite” edges
  - BUT Finitely represented with Expressions:

    p = io.Connector.open(url) &&
    (url.startsWith(”http://”) || url.startsWith(”https://”))

- Matching = Language inclusion can be reduced to an emptiness test:
  - $L_{AutC} \subseteq L_{AutP} \Leftrightarrow L_{AutC} \cap L_{NEG\ AutP} = \varnothing$

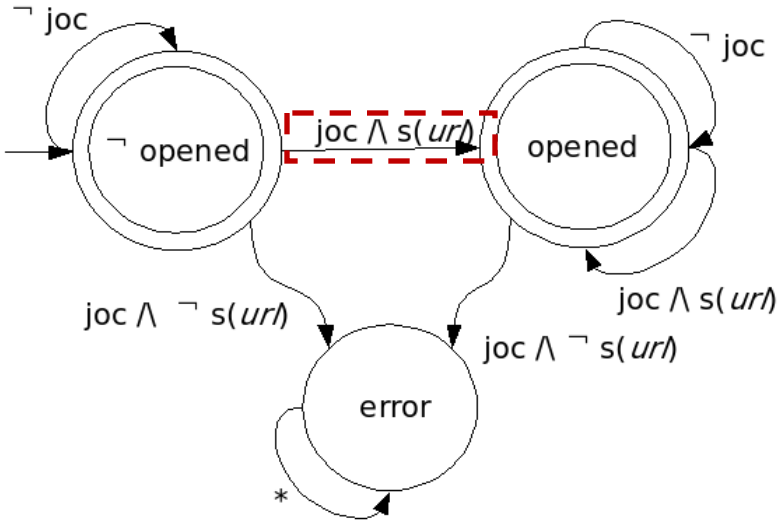
- *Search for counterexamples:*
  - Path allowed by contract but NOT allowed by policy

Automata Modulo Theory (*AMT*) examples

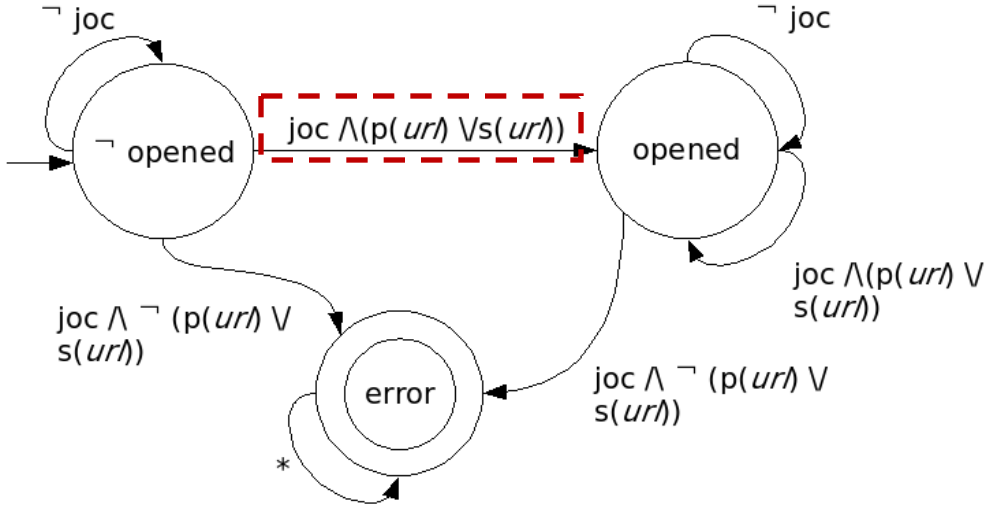
CONTRACT:
The application only uses HTTPS network connections

POLICY:
The application uses only high-level (HTTP, HTTPS) network connections

Automaton:

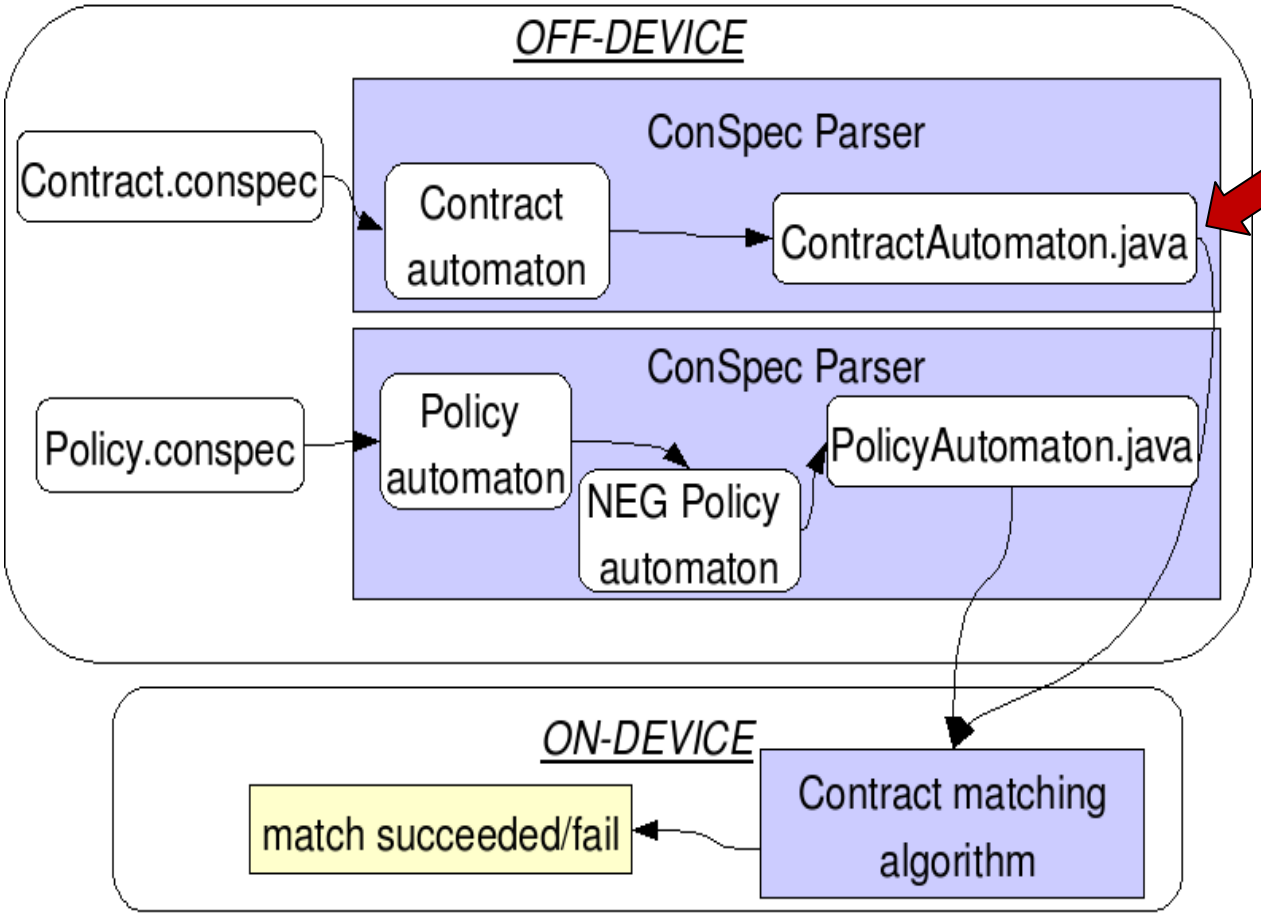


Negated automaton:



- **Abbreviations for JAVA APIs:** *joc* = io.Connector.open(url)
  p(url) = url.startsWith(”http://”)
  s(url) = url.startsWith(”https://”)

Architecture of Matching Prototype



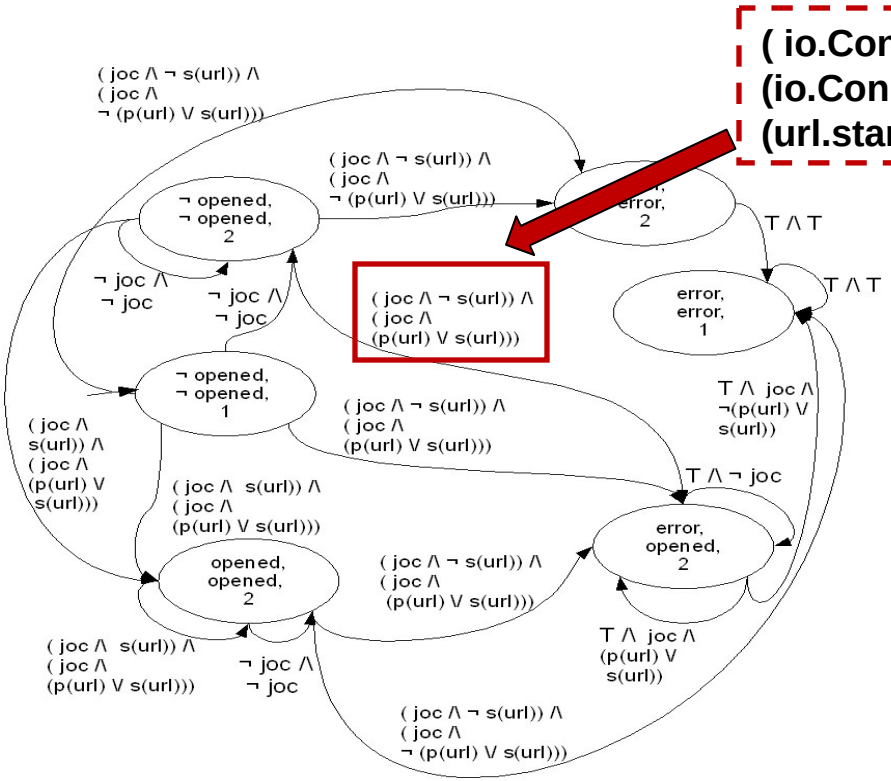
Java source file

```
import org.s3ms.copv.automaton.*;
import org.s3ms.copv.expressions.*;
import org.s3ms.copv.matcher.*;

public class PolicyRules{
PolicyRules(){

public static AutomatonMTT Create(){
//=========******* AUTOMATA FOR Policy ********=======
//--------- Initialization of the states --------
HashSet<State> states_1 = new HashSet<State>(2,1);
//--------- State errorState_1 initialization --------
HashMap<String, Object> fieldserrorState_1_1 =
ew HashMap<String, Object>(0,1);
//--------- Add the fields to the state --------
State errorState_1 = new State(fieldserrorState_1_1);
states_1.add(errorState_1);
// ================******* FROM STATE ERRORSTATE_1********==
HashSet<Transition> from_errorState_1 =
new HashSet<Transition>(1,1);
SpecificBoolExp e_0_1 =  new SpecificBoolExp(false);
from_errorState_1.add(new Transition(e_0_1, errorState_1));
transitionsMap_1.put(errorState_1,
new StateDef(false, true, from_errorState_1));
```

Off-line: mapping to automaton – expensive operation complementation – for optimization
On-line: On-The-Fly algorithm

On-the-Fly Model Checking

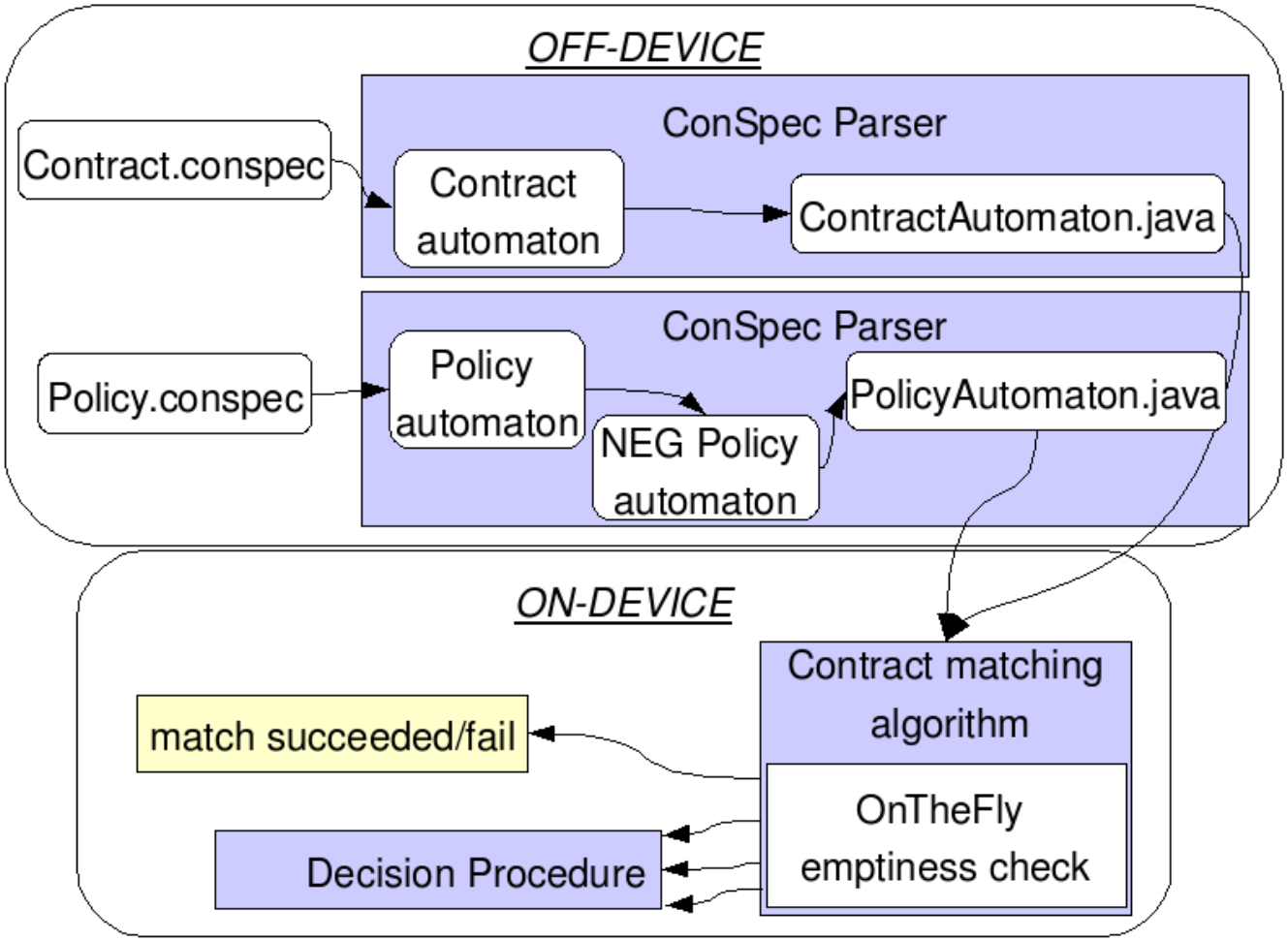
- The search space for counterexample (a trace that satisfies the Contract and violates the Policy)



(io.Connector.open(url) /\ !url.startsWith(”https://”))/
(io.Connector.open(url) /
(url.startsWith(”http://”) \/ url.startsWith(”https://”)))

On-the-Fly Algorithm with decision procedure for SMT Interaction with the solver of math expression NuSMV for satisfiability checks.

On-the-Fly Model Checking with Decision Procedure



Conclusions

- The main goal is to provide a concrete answer:
  - *given a contract that an application carries with itself and a policy that a platform specifies, how can we check whether or not the contract is compliant with the policy?*
- A prototype implementing a matching algorithm based on a well-defined automata theory was proposed.
- Both the theory and the Desktop prototype as well as several illustrative examples were presented.
- Future work:
  - Device version
  - Richer policy mechanisms

Thank you!

Questions?..

Related work

- *Sandboxes* limit the instructions available for use
- *Code signing* ensures that code originates from a trusted source
- *Security automata* proscribes execution of mobile code containing violations of the security policy
- *Proof-carrying code (PCC)* carries explicit proof of its safety
- *Model-carrying code (MCC)* carries security-relevant behavior of the producer mobile code

Benchmark Contract and Policies

- USE of Costly functionalities
- NETwork connectivity
- PRIvate information management
- INTeraction with other applets

Example ID	Natural Language description	Coverage
httpHttps	The application only uses high-level network connections.	NET
https	The application only uses HTTPS network connections.	NET, PRI
maxKB512	The data received by application is bounded by 512Kb	USE, NET
maxKB1024	The data received by application is bounded by 1024Kb	USE, NET
noPushRegistry	The application does not use the push registry mechanism	USE
oneConnPushRegistry	Only one connection registered to the Push registry at a time	USE, NET
notCreateRSt	The policy allows to open record stores, but it is not allowed to create new record stores.	INT
notCreateSharedRS	The application does not create shared record stores.	INT, PRI
noSMS	No messages are sent by the application	USE
100SMS	Maximum 100 text messages can be sent by the application	USE
pimNoConn	After PIM was opened no connections are allowed	USE, PRI, NET
pimSecConn	After PIM was accessed only secure connections (HTTPS) can be opened	USE, PRI, NET

Problems suit

- – SC: Number of States Contract
- – SP: Number of States Policy
- – TC: Number of Transitions Contract
- – TP: Number of Transitions Policy

Problem	Contract	Policy	SC	TC	SP	TP
P1	size_100_512_contract.pol	size_10_1024_policy.pol	2	4	2	4
P2	maxKB512_contract.pol	maxKB1024_policy.pol	2	4	2	4
P3	noPushRegistry_contract.pol	oneConnRegistry_policy.pol	2	3	3	9
P4	notCreateRS_contract.pol	notCreateSharedRS_policy.pol	2	4	2	4
P5	pimNoConn_contract.pol	pimSecConn_policy.pol	3	7	3	9
P6	2hard_contract.pol	2hard_policy.pol	3	7	3	7
P7	httpI_contract.pol	httpsI_policy.pol	3	7	3	7
P8	3hard_contract.pol	3hard_policy.pol	3	7	3	7
P100	noSMS_contract.pol	100SMS_policy.pol	2	4	102	304

Running Problem Suit

Problem	ART (s)	SV	TV	Result
P1	2.4	2	6	Match
P2	2.4	2	6	Match
P3	2.4	3	11	Match
P4	2.4	2	6	Match
P5	4.7	3	11	Match
P6	2.9	4	4	Not Match
P7	2.8	5	7	Not Match
P8	2.9	5	7	Not Match
P100	9.3	102	307	Match

- ART: Average Runtime for 10 runs
- SV: Number of Visited States
- TV: Number of Visited Transitions