Software is everywhere!

**producers**
- licensed applications
- right-reserved content
- advertisement

**users**
- personal data
- credentials
- sensitive information
- financial data

**software protection is needed!**
Environment cannot be trusted!
...and hardware is unknown

user = attacker
full control
multiple executions
tools
methodologies

Man-At-The-End
(MATE)

remote
hardware-independent
software protection is needed!
(Software) Remote Attestation

- Environment
  - Target
  - Attester
  - INTEGRITY
  - Root of trust

- Manager
- Verifier

Network
Software Attestation (SA)
Thesis Contributions

high-level contribution
- propose a comprehensive model
  - unify literature specifications
- generalise existing approaches
  - define procedures and protocols

low-level contribution
- static & dynamic model instances
  - assets nature
- validating model applicability
- limitations and security issues
  - of model
  - of instances
General Architecture

1. prepare_request()

0.1 connect()

2. request(nonce)

3. collect_evidence()

4. response(evidence)

5. evaluate_evidence()

6. store(result)

TargetX-Session_1

ATT1  VALID
ATT2  VALID
ATT3  VALID
ATT4  INVALID
ATT6  VALID
ATT7  INVALID
..  
ATTn  VALID

network

manager
Model Instances

root of trust

assets

integrity evidence
extraction procedures

general model instantiations

structural assets

STATIC SOFTWARE ATTESTATION

behavioural assets

DYNAMIC SOFTWARE ATTESTATION
Static Software Attestation

f1()

check_license();

f2()

f3()

asset selection

extraction procedure

nonce

asset representation

hash function

nonce generation

nonce interpretation

hash()
Security Analysis

cloning attacks

- non-structural modifications
  - run-time data
  - debugging attacks

attestation operations prevention

- attester’s start
- stop requests
- inhibit data collection
- stop responses

OPEN ISSUE

DYNAMIC TECHNIQUES

BY DEFINITION

non-structural modifications

ENGINEERING

SERVER-SIDE REACTIONS
Tamper Reaction for Static RA

• software attestation issues:
  • only detection
  • local attacks vulnerabilities

• idea: combine with another technique
  • Client Server Code Splitting
  • achieve tamper reaction mechanism

• result: Reactive Attestation

validate
software attestation
composability
Reactive Attestation

Server-side

static SA components

CS-CS server

emulate_client()

...
Validation: test bench

• application
  • license checking Android app
  • critical native (C) code parts → protected with Reactive Attestation

• tampering – 7 tampered versions
  • modification in protected application binary
  • NOP instructions replacements
  • hard-coded constant values modification
Validation: results

- **effectiveness**
  - all the tampered copies have been blocked 100% True Positives
  - untampered copy execution is not affected 100% True Negatives

- **overhead**
  - memory usage: one order of magnitude increase Low impact
  - execution time: two orders of magnitude increase Possible issue, Engineering
  - network: constant amount of network usage Acceptable

software attestation composability validated!
Dynamic Software Attestation

Invariants Monitoring

invalid = 1;
...
check_license()
{
  if (invalid == 1){
    exit(1);
  }
}

$> invalid = 0

d debugger

collect_variables

attester

test invariants

verifier

data integrity
evaluation correctness

likely invariant

invalid == 1
Invariants Monitoring

Problem Statement

Investigation:

- practical experience
- theoretical study
- experimental evaluation
- empirical assessment

... to check “likely-invariants expressiveness to the purpose of security-grade software behavioural modelling”

Assumption from the literature:

data constraints \( \leftrightarrow \) program behaviour

TO BE VALIDATED
## Practical Evidences

<table>
<thead>
<tr>
<th>$\mathcal{L}$</th>
<th>limitation</th>
<th>low</th>
<th>mid</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>false positives</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>concurrent extraction</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>only pre- and post-conditions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>inter-function variables' types</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>data types</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>inferred invariants types</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>invariants' research scope</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>sends all the variables</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>external attester</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>constructor option</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>false negatives</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>12</td>
<td>variables info in protected app</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>manual effort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>compiler optimization</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>variables unrelated attacks</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>16</td>
<td>availability of invariants</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Notes:
- **attack detected even if the application is sound**
- **attack in place, NOT detected**
- **attack that do not alter variables values**
IM Theoretical Evaluation

an example

```c
void main(int argc, char * argv[])
{
    int param1 = parse_a(argv);
    foo(param1);
}
void foo(int p){
    /* do stuff */
}

$> prog -a 5
    inference execution

$> prog -a 10
    normal execution

$> prog -a 1
    inference phase
$> prog -a 2
    ...
$> prog -a n
    inference phase

Inferred
p == 5

Evaluated
p != 5

TOO RESTRICTIVE

Inferred
p > 0

TOO BROAD

single execution trace collection

multiple execution trace collection

17/19
IM Experimental Evaluation

automatically protected use cases:

- Bzip2 – compression tool
- MOC – music on console
- Lynx – console web browser
- mpg123 – CLI music player
- oathtool – HOTP generator and checker
- gamespace – CLI game

attacks: attach a debugger and alter variables' values
aim: verify false positives and false negatives

modifying variable attack 100% detected
trapping and redirecting control 0% detected
IM Empirical Evaluation

- **Use case:** gamespace
- **Experiment:** students were asked to port an attack
  - **Change the behaviour by modifying code:**
    - make pieces move of 2 steps instead of 1
  - **Two attack strategies**:
    - **Modify code**
      - doubling the move function call
    - **Modify data**
      - alter coordinates variables

**NEVER detected:**
- it does not alter variables, thus invariants

- **Invariants**
  - point->col ∈ [0, 80]
  - point->row ∈ [0, 25]

- **NEVER detected:**
  - it alters variable in a way that do not make invariants invalid

- **Point values**
  - point->col = 67; point->row = 11;
  - point->col = 69; point->row = 13;
Conclusions

• software attestation model
  • abstract architecture
  • high-level procedures

• reactive attestation
  • invariants monitoring
  • non-secure protection
  • unavoidable issues

• static software attestation
  • intrinsic limitations
  • valid building blocks for complex protections
  • Reactive Attestation

Software attestation method is valid,
“limitations are introduced by particular instantiations of the abstract model”
Questions?