framework for code transformations

Tobiáš Smolka

FI MUNI

April 12, 2010

BUSLab
Brno University Security Laboratory

Background

What we had?

- A lot of ideas how to secure smart card applications
- List of possibly vulnerable constructions and their semantically equivalent more secure alternatives

What we needed?

- An easy way how to use the results of research in real projects
- A way to perform automatic code transformations

Solution should be

- simple enough to be easily used and extended
- robust enough to handle all kinds of transformations

BUSLab
Brno University Security Laboratory

Background

What we had?

- A lot of ideas how to secure smart card applications
- List of possibly vulnerable constructions and their semantically equivalent more secure alternatives

What we needed?

- An easy way how to use the results of research in real projects
- A way to perform automatic code transformations

Solution should be

- simple enough to be easily used and extended
- robust enough to handle all kinds of transformations

BUSLab
Brno University Security Laboratory

Background

What we had?

- A lot of ideas how to secure smart card applications
- List of possibly vulnerable constructions and their semantically equivalent more secure alternatives

What we needed?

- An easy way how to use the results of research in real projects
- A way to perform automatic code transformations

Solution should be

- simple enough to be easily used and extended
- robust enough to handle all kinds of transformations

BUSLab
Brno University Security Laboratory

Outline

BUSLab
Brno University Security Laboratory

Proposed solution

We decided to develop

1. general framework for misc code transformations
   - no restriction on target language or type of transformation
   - powered by Ant
2. sample bundle of transformations
   - focused on improving resiliency of JavaCard code against power analysis
   - powered by ANTLR and StringTemplate engine

BUSLab
Brno University Security Laboratory

Proposed solution

We decided to develop

1. general framework for misc code transformations
   - no restriction on target language or type of transformation
   - powered by Ant
2. sample bundle of transformations
   - focused on improving resiliency of JavaCard code against power analysis
   - powered by ANTLR and StringTemplate engine

BUSLab
Brno University Security Laboratory

Framework properties

Most important requirements:

- Easy integration into development
- Extensibility and variability

Instead of inventing the wheel we used well-known build tool Ant

- Transformations are executed from general build scripts via custom task

BUSLab
Brno University Security Laboratory

Framework properties

Most important requirements:

- Easy integration into development
- Extensibility and variability

Instead of inventing the wheel we used well-known build tool Ant

- Transformations are executed from general build scripts via custom task

BUSLab
Brno University Security Laboratory

Example

Executing transformation

```
<transform classname="IfSwitchTransformation">
  <classpath location="${run.classpath}" />
  <fileset dir="../sample_project" >
    <include name="**/*.java" />
    <modified/>
  </fileset>
  <mapper type="glob" from="*" to="*.transformed" />
  <param name="generateComments" value="true" />
</transform>
```

Snippet from build script, that executes specified transformation on concrete files.

BUSLab
Brno University Security Laboratory

Framework properties

We got (thanks to Ant's developers):

- Powerful scripting environment for executing transformations
  - Properties, conditions, many other useful tasks
- Ant's concepts for customizing execution of transformations
  - Resource Collections (Fileset, Filelist, Dirset) for specifying what to transform
  - File selectors (filename, contains, size, modified, ...) for fine-grained file selection
  - Mappers for specifying how to map file names
  - Classpath parameters for specifying where to find transformations

Transformations can be fully adapted to concrete environment or project.

BUSLab
Brno University Security Laboratory

Framework properties

We got (thanks to Ant's developers):

- Powerful scripting environment for executing transformations
  - Properties, conditions, many other useful tasks
- Ant's concepts for customizing execution of transformations
  - Resource Collections (Fileset, Filelist, Dirset) for specifying what to transform
  - File selectors (filename, contains, size, modified, ...) for fine-grained file selection
  - Mappers for specifying how to map file names
  - Classpath parameters for specifying where to find transformations

Transformations can be fully adapted to concrete environment or project.

BUSLab
Brno University Security Laboratory

Framework properties

We got (thanks to Ant's developers):

- Powerful scripting environment for executing transformations
  - Properties, conditions, many other useful tasks
- Ant's concepts for customizing execution of transformations
  - Resource Collections (Fileset, Filelist, Dirset) for specifying what to transform
  - File selectors (filename, contains, size, modified, ...) for fine-grained file selection
  - Mappers for specifying how to map file names
  - Classpath parameters for specifying where to find transformations

Transformations can be fully adapted to concrete environment or project.

BUSLab
Brno University Security Laboratory

Properties of transformations

- Transformations are general dynamically loaded Java classes
- Developer is not restricted at all
  - only simple interface: setFiles(), setParams(), execute(), ...
- Target language independent (Java, JavaCard, .NET, Java bytecode, ...)

Our build-in transformations

- Target language is JavaCard
- Focused on improving resiliency against power analysis
- Replacement of vulnerable constructions with semantically equivalent alternatives

BUSLab
Brno University Security Laboratory

Properties of transformations

- Transformations are general dynamically loaded Java classes
- Developer is not restricted at all
  - only simple interface: setFiles(), setParams(), execute(), ...
- Target language independent (Java, JavaCard, .NET, Java bytecode, ...)

Our build-in transformations

- Target language is JavaCard
- Focused on improving resiliency against power analysis
- Replacement of vulnerable constructions with semantically equivalent alternatives

BUSLab
Brno University Security Laboratory

Build-in transformation: Shadow variables

- General protection mechanism against error induction
- Each variable has own shadow copy, which maintains inversion of the value
- Value is checked against shadow variable on each read access
- Shadow variable is updated on each modification

BUSLab
Brno University Security Laboratory

Example: Shadow variables

```
short i=1,j=2;
if (i==j) i+=j;
```

```
1   /* help array, that holds shadow variables */
2  private short fault_resistant_short[] = new short[2];
3  ...
4  /* first argument is value, second variable ID */
5  short i=__set_short(1,0),j=__set_short(2,1);
6  if (__get_short(i,0)==__get_short(j,1))
7    i=__set_short(
8      __get_short(i, 0)
9        +
10     __get_short(j,1),
11   0);
```

BUSLab
Brno University Security Laboratory

Build-in transformation: If-switch

- Some cards leak information about argument in IF statement
- Can be determined whether THEN or ELSE branch will be executed
  - An additional jump is performed before ELSE execution
  - Time is noticeable longer, power trace is different

Proposed protection mechanism

- All IF-THEN-ELSE constructions are replaced with equivalent SWITCH constructions
- SWITCH constructions are randomized and power trace doesn't leak the information

BUSLab
Brno University Security Laboratory

Build-in transformation: If-switch

- Some cards leak information about argument in IF statement
- Can be determined whether THEN or ELSE branch will be executed
  - An additional jump is performed before ELSE execution
  - Time is noticeable longer, power trace is different

Proposed protection mechanism

- All IF-THEN-ELSE constructions are replaced with equivalent SWITCH constructions
- SWITCH constructions are randomized and power trace doesn't leak the information

BUSLab
Brno University Security Laboratory

Example: If-switch

```
if (i==1) j=10;
else j=20;
```

```
1  // result of original expression
2  boolean expr_res1 = (i==1);
3  // negation of the result
4  boolean expr_res_neg1 = !(i==1);
5
6  switch (__getRandomBit()){ // 0 or 1
7    case -1:
8      /* any code, that will not be
9       left out by compiler */ break;
10   case 0:
11     if (expr_res1) { j=10; }
12     else { j=20; }
13     break;
14   case 1:
15     if (expr_res_neg1) { j=20; }
16     else { j=10; }
17     break;
18 }
```

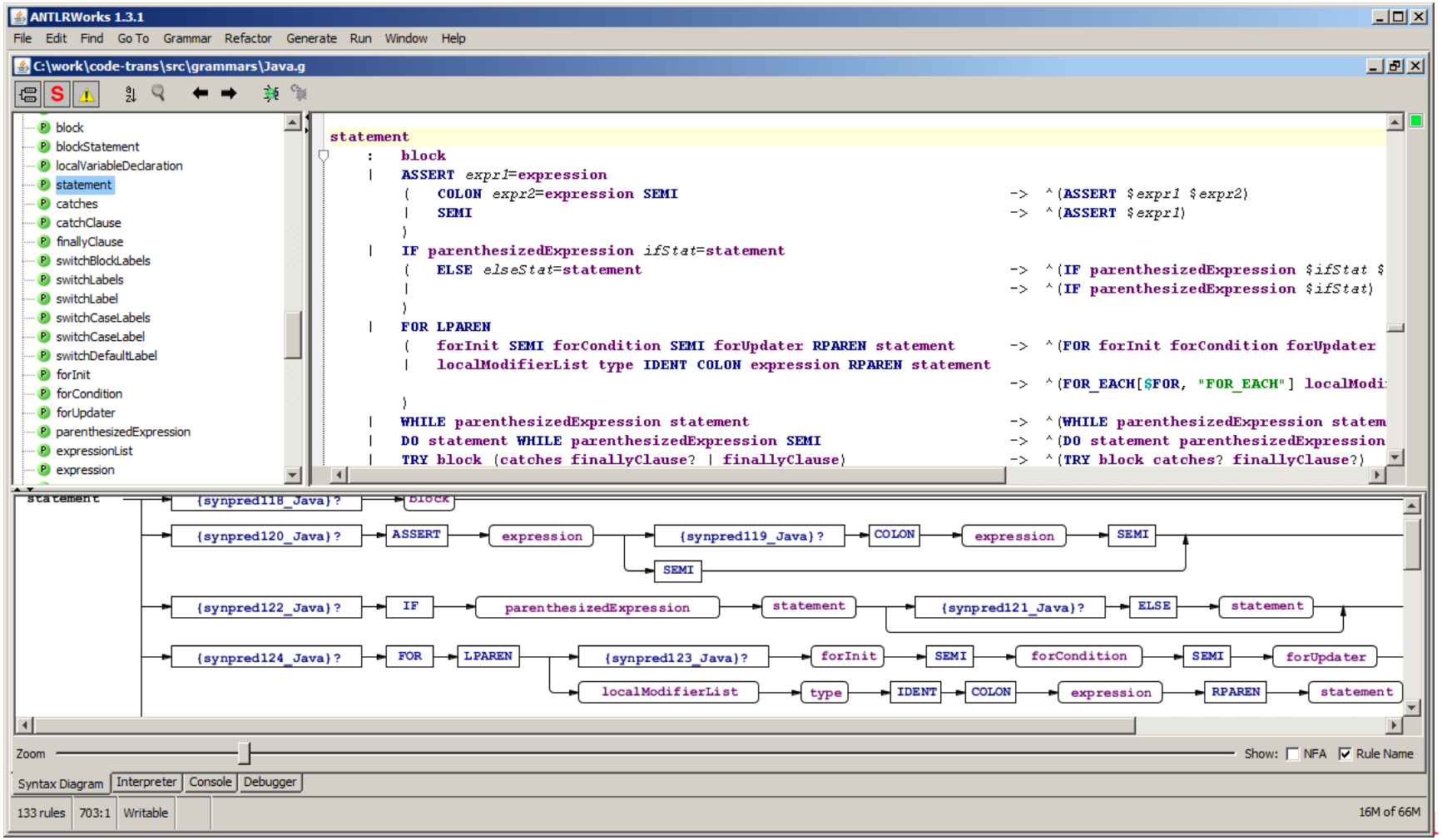
BUSLab
Brno University Security Laboratory

Implementation details

- Transformations have to be robust, but easy to maintain
- We use external tools (ANTLR, StringTemplate)
- Parsers and lexers are generated from nice-looking grammar files
- Grammar development environment ANTLRWorks
- Actual code replacement is performed via templates
  - Logic of transformation and generated code are separated
  - Code replacements are written naturally (see example)
  - Audit and modification of the templates is easy
  - Templates are loaded from classpath, user can specify own versions

BUSLab
Brno University Security Laboratory

Grammar development environment ANTLRWorks



Brno University Security Laboratory

Example: code replacement template

```
1  // result of original expression
2  boolean expr_res<ruleParams.id> = <ruleParams.expr>;
3  // negation of the result
4  boolean expr_res_neg<ruleParams.id> = !<ruleParams.expr>;
5
6  switch (__getRandomBit()){   // 0 or 1
7    case -1:
8      /* any code, that will not be
9       left out by compiler */ break;
10   case 0:
11     if (expr_res<ruleParams.id>) <ruleParams.ifTrue>
12     else <ruleParams.ifFalse>
13     break;
14   case 1:
15     if (expr_res_neg<ruleParams.id>) <ruleParams.ifFalse>
16     else <ruleParams.ifTrue>
17     break;
18 }
```

BUSLab
Brno University Security Laboratory

Implementation details

Each from our build-in transformations has 3 steps:

1. Lexical analysis
   - Lexical analyzer (lexer) creates stream of vocabulary symbols (tokens)
2. Parsing
   - Parser creates abstract syntax tree (AST) from tokens
3. Tree parsing and rewriting token stream
   - Tree parser reads AST and if needed, rewrites underlying token stream
   - Another approach can be code generation, where new token stream would be created from modified AST

Modified token stream is then written back to file

BUSLab
Brno University Security Laboratory

Example: IF-SWITCH tree parser

```
1  statement :
2    block
3    | ^(FOR forInit forCondition forUpdater statement)
4    | ^(WHILE parenthesizedExpression statement)
5    | ^(TRY block catches? block?)
6    | ifSwitchTransformation
7    ...
8
9  ifSwitchTransformation :
10 ^(IF parenthesizedExpression
11   ifTrue=statement
12   ifFalse=statement?){
13     performRewrite(new STAttrMap().
14       put("expr", $parenthesizedExpression.text).
15       put("ifTrue", $ifTrue.text).
16       put("ifFalse", $ifFalse.text).
17       put("rule", "IF_SWITCH")
18     );
19 };
```

BUSLab
Brno University Security Laboratory

Testing

- Transformations can be pretty tricky, testing is needed
- Our Ant task can be used
- How?
  1. Prepare standard JUnit test classes with all special cases
  2. Automatically transform test classes via Ant (from build script)
  3. Check whether transformed tests pass or not

BUSLab
Brno University Security Laboratory

Example: JUnit test

```
int j=0;
for (int i=0;i<10;i++){
  if (i==5) break;
  j+=i;
}
assertEquals(10, j);
```

```
1  int j=0;
2  BLOCK_1: {
3    for (int i=0;i<10;i++){
4      boolean expr_res1 = (i==5);
5      boolean expr_res_neg1 = !(i==5);
6
7      switch (__getRandomBit()){
8        case -1:
9          /* any code, that will not be
10         left out by compiler */ break;
11       case 0:
12         if (expr_res1) {
13           break BLOCK_1;
14         }
15         break;
16       case 1:
17         if (expr_res_neg1) {
18         } else {
19           break BLOCK_1;
20         }
21         break;
22     }
23     j+=i;
24   }
25 }
26 assertEquals(10, j);
```

BUSLab
Brno University Security Laboratory

Summary

Code replacement framework

- Easy integration into development process (Ant)
- Can handle any transformation (general classes + dynamic loading)

Build-in transformations

- Focused on improving resiliency of JavaCard code against power analysis
- Robust, but easy to maintain (ANTLR, StringTemplate engine)

BUSLab
Brno University Security Laboratory

Thank you for your attention.

Any questions or comments?

BUSLab
Brno University Security Laboratory