Tokeneer: Beyond Formal Program Verification

Yannick Moy and Angela Wallenburg

June 21, 2010
The Tokeneer Project

The Tokeneer Challenge

The Tokeneer Lessons
The Tokeneer Project

The Tokeneer Challenge

The Tokeneer Lessons
The Tokeneer Project

**Project goals** to demonstrate:

- Common Criteria **EAL5** achievable in a **cost effective** manner
- Praxis **“Correctness by Construction”** approach
  - sound, formal notations
  - keep it simple
  - remove errors as soon as possible
  - generate evidence as you go

Tokeneer originally developed by NSA
What is Tokeneer?

- Protect secure information in physically secure enclave
- Demonstrate use of Smart Cards and Biometrics
### Metrics – Development Statistics

<table>
<thead>
<tr>
<th></th>
<th>Ada LOC</th>
<th>SPARK LOC</th>
<th>LOC/day (coding)</th>
<th>LOC/day (overall)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>core</strong></td>
<td>9,939</td>
<td>16,564</td>
<td>203</td>
<td>38</td>
</tr>
<tr>
<td><strong>support</strong></td>
<td>3,697</td>
<td>2,240</td>
<td>182</td>
<td>88</td>
</tr>
</tbody>
</table>

- **Total effort:** 260 man days
- **Team:** 3 people part-time
- **Total schedule:** 9 months elapsed
Metrics – Development Defects

- Requirements: 0
- Z Specification: 1
- Security Specification: 2
- Z Specification Proof: 0
- Z Design: 1
- Z Design Proof: 0
- INFORMED Design: 11
- Code: 6
- Code Proof: 1
- Integration/Interfaces: 0
- System Test: 0
Metrics – Post Delivery Defects

- software defects found in independent test: zero
- by NSA prior to public release (2004-2008): zero


- NSA “Technology Transfer Agreement”
- essentially open-source, entire project archive
- Google for “Tokeneer” or
- http://www.adacore.com tokeneer
Outline

The Tokeneer Project

The Tokeneer Challenge

The Tokeneer Lessons
## Finding Bugs in Tokeneer!

<table>
<thead>
<tr>
<th>Who</th>
<th>How</th>
<th>Where</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rod Chapman</td>
<td>SPARK tools</td>
<td>code</td>
<td>1</td>
</tr>
<tr>
<td>Diomidis Spinellis</td>
<td>code review</td>
<td>code</td>
<td>1</td>
</tr>
<tr>
<td>Woodcock &amp; Aydal</td>
<td>Alloy analyzer</td>
<td>specification</td>
<td>12</td>
</tr>
<tr>
<td>Moy &amp; Wallenburg</td>
<td>code review</td>
<td>code</td>
<td>11</td>
</tr>
<tr>
<td>Moy &amp; Wallenburg</td>
<td>static analysis</td>
<td>code</td>
<td>9</td>
</tr>
</tbody>
</table>
Absence of run-time errors, by applying the SPARK tools

| 4 | start | rtc check @ 234 | Undischarged |

```plaintext
if Success and then
   (RawDuration * 10 <= Integer(DurationT'Last) and
    RawDuration * 10 >= Integer(DurationT'First))
then
   Value := DurationT(RawDuration * 10);
else
```

- RawDuration is very large ⇒ overflow during multiplication
- Solution is to divide instead
Spinellis’s Bug

Security code review, focusing on error handling

```plaintext
534 File.Delete(TheFile => TheFile,
535    Success => OK);
536 AuditSystemFault := AuditSystemFault and not OK;
```

- File successfully deleted ⇒ AuditSystemFault is cleared
- Deletion fails ⇒ AuditSystemFault not always set
### Static Analysis Setup

<table>
<thead>
<tr>
<th>Tool</th>
<th>Category</th>
<th>Who</th>
<th>Run-Time</th>
<th>Review Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNAT</td>
<td>compiler</td>
<td>AdaCore</td>
<td>3s</td>
<td>10mn</td>
</tr>
<tr>
<td>CodePeer</td>
<td>analyzer</td>
<td>AdaCore &amp; SofCheck</td>
<td>&gt; 1mn</td>
<td>1 day</td>
</tr>
<tr>
<td>Examiner</td>
<td>analyzer</td>
<td>Praxis</td>
<td>10s</td>
<td>5s</td>
</tr>
<tr>
<td>Simplifier</td>
<td>prover</td>
<td>Praxis</td>
<td>30s</td>
<td>5s</td>
</tr>
<tr>
<td>Dead Path Analyzer</td>
<td>prover</td>
<td>Praxis</td>
<td>4mn</td>
<td>1 day</td>
</tr>
</tbody>
</table>
## Code Review Setup

<table>
<thead>
<tr>
<th>Blind Spot</th>
<th># in SPARK</th>
<th># in Ada</th>
</tr>
</thead>
<tbody>
<tr>
<td>hide annotations</td>
<td>29</td>
<td>-</td>
</tr>
<tr>
<td>loop termination</td>
<td>6</td>
<td>17</td>
</tr>
<tr>
<td>exception handling</td>
<td>-</td>
<td>101</td>
</tr>
</tbody>
</table>

Review time: 1 day
Categories of Problems

**Defect:**

*may* lead to an observable failure of the system

Example: sorting procedure does not sort

**Code quality issue:**

*cannot* lead to an observable failure of the system

*could* lead to an observable system failure during maintenance

Example: sorting procedure is called Remove_Duplicates
Disclaimer: Experimental Bias

Not a quantitative comparison of tools and methods

- During development, code was cleared by GNAT, Examiner and Simplifier
- During development, various reviews were performed
- First application of CodePeer, Dead Path Analyzer and focused reviews

Not a quantitative comparison between SPARK and Ada code

- SPARK tools only applicable to SPARK code
- Review of static analysis results focused on SPARK code
- Code review focused on SPARK code or its interface to Ada code
## Defects and Issues Found in Tokeneer

<table>
<thead>
<tr>
<th>Category</th>
<th># SPARK defects</th>
<th># SPARK issues</th>
<th># Ada defects</th>
<th># Ada issues</th>
<th># total (# exclusive)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNAT</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3 (0)</td>
</tr>
<tr>
<td>CodePeer</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>9 (6)</td>
</tr>
<tr>
<td>SPARK tools</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2 (1)</td>
</tr>
<tr>
<td>Static analysis</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>Hide annotations</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Loops</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Exceptions</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Other</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Code review</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>Total</td>
<td>4</td>
<td>7</td>
<td>7</td>
<td>4</td>
<td>22</td>
</tr>
</tbody>
</table>
Outline

The Tokeneer Project

The Tokeneer Challenge

The Tokeneer Lessons
Defects Found in SPARK Code

Only one run-time error

- Found by a new version of the SPARK toolset

One possible cause of non-termination

- Blind spot of the SPARK verifications
- Best handled by a methodology for code review for SPARK

Two defects related to error handling

- Exceptions are a better way to signal errors
- Best handled by a methodology for code review for SPARK

One functional error

- Side-effect of detecting logic errors in code
Defects Found in Ada Code

Many serious errors

- Three serious run-time errors (by static analysis)
- Four serious functional errors (by code review)

Most errors cannot occur in SPARK

- Prevented by the language: ignoring exceptions
- Examiner: uninitialized variable, ignoring error status
- Simplifier: access out of array bounds, accessing the null string

Remaining errors

- Low-level manipulations (string) are best verified
- Exceptions are a better way to signal errors
High-Level Lessons Learned

Need for dynamic code coverage

- Most issues in SPARK are due to lack of coverage
- Static detection of dead code is typically incomplete
- Need for executable annotations

Tool setup and configuration

- One issue in SPARK was missed by not using appropriate compiler switch
- Need to be reviewed even more than code

Tools should facilitate review

- Most time spent in static analysis: review of results
One language of executable annotations

Combine tests, static analysis and proofs

Sound static analysis applicable to DO-178C

Separate verification for scaling and early adoption

Free software structured as toolchains for Ada and C
Defect in SPARK: Wrong Variable Used

GNAT warns that a condition in a test is known to be true

keystore.adb:349:23: warning:
condition is always True (see test at line 344)

```
344 if RetValIni = Interface.Ok then
    Interface.FindObjects
        (HandleCount  => HandleCount,
         ObjectHandles => Handles,
         ReturnValue   => RetValDo);
349 if RetValIni = Interface.Ok then
```

- Functional error detected as a side-effect of logic error in code
- Could be detected by dynamic branch coverage
- Detected by three tools: GNAT compiler, CodePeer analyzer, SPARK Dead Path Analyzer
CodePeer warns that a condition in an annotation is always true

enclave.adb:1107:13: warning: condition is always true

```
1107  # (Admin.prf_rolePresent(TheAdmin) = Typ.Guard
1108  # ~> 
1109  # ((Admin.IsDoingOp(TheAdmin) and
1110  # Admin.TheCurrentOp(TheAdmin) = OverrideLock)
1111  # or not Admin.IsDoingOp(TheAdmin)));
```

- Benefit of sharing annotations between tools
- Could be detected by dynamic condition coverage on executable annotations
- Need for abstraction in contracts
Defect in Ada: Access Out of Array Bounds

CodePeer warns that an array access might be out of bounds

tcip.adb:205:16: medium:
array index check might fail: requires i >= 2

204 if Msg.Data(i) = ASCII.Lf and then
205   Msg.Data(i − 1) = ASCII.Cr then

- Class of errors that are automatically detected in SPARK code
- Hard to detect by testing (except concolic testing or fuzzing)
GNAT warns that a local variable is assigned but never used

tcip.adb:446:07: warning:
variable "Address" is assigned but never read

- Class of errors that are automatically detected in SPARK code
- Could not be detected by dynamic coverage
A loop may not terminate if a function called terminates in error.

```
149 while Stop=0 and not File.EndOfFile(TheFile) loop
150     -- Read the next (non-empty) line of the file
151     --# assert PrivateKeyPresent(KeyStore.State) =
152     --#     PrivateKeyPresent(KeyStore.State~);
153     File.GetLine(TheFile, TheCert, Stop);
154 end loop;
```

- Exceptions are a better way to signal errors
- Need adequate methodology for integrating proof and code review for exception handling
Issue in SPARK: Useless Postcondition

A postcondition is trivially true

```plaintext
174  --- post ( (Op = OverrideLock <->
175  --- prf_rolePresent(TheAdmin~) = Typ.Guard) )
176  ---
177  --- (Op = OverrideLock <->
178  --- prf_rolePresent(TheAdmin) = Typ.Guard))
```

- Logic errors best detected by static analysis tools (GNAT, CodePeer)
- Benefit of sharing annotations between tools
- Need for executable annotations
A procedure can return in error while setting a Success flag to True

```
283 Success := True;
284 ...
285 -- some code that can raise an exception
286 ...
287 exception
288 when E: others =>
289   DebugOutput("Send Error.");
```

▶ Exceptions are a better way to signal errors
An access can raise an exception which is caught by a local handler.