Pyrrhus Software
Enduring Solutions

Software Vulnerabilities Precluded by SPARK

Paul E. Black, PhD	NIST
Chris E. Dupilka	US DoD
F. David Jones	Pyrrhus Software
Joyce L. Tokar, PhD	Pyrrhus Software

Overview

- Background
  - *SPARK Programming Language and Toolset*
  - *Weaknesses Considered*
- Classes of Weaknesses
  - Weaknesses that Cannot Occur in SPARK Programs
  - Weaknesses that Can Be Certainly Excluded
  - Weaknesses that May Occur
  - Concurrent Programming Weaknesses
- Summary Classification
- So What?

© 2011 by Pyrrhus Software

SPARK Language and Toolset

- Designed for developing high assurance software.
- A subset of Ada augmented with annotations resulting in a programming language which is
  - unambiguous,
  - free from implementation dependencies, and
  - formally defined.
- RavenSPARK Option
  - Allows the development of concurrent software with the same level of verification and assurance available with SPARK.
  - Static analysis can find the worst-case execution time, schedulability, and memory use of multi-threaded applications.

© 2011 by Pyrrhus Software

SPARK Toolset

- Examiner an automated static-semantic checker of code characteristics such as type alignment and visibility. Supplies control, data, and information flow analyses which enable the detection of erroneous behaviors in a program such as dead code, undefined variables, redundant tests, and ineffective statements.
- Simplifier is an automated theorem prover that supplies proof of particular program properties such as the absences of run-time errors, proof of invariant properties, and formal code verification of partial correctness with respect to the formal specification.
- Proof Checker is an interactive proof tool that offers the same capabilities as the Simplifier.

© 2011 by Pyrrhus Software

Pyrrhus Software
Enduring Solutions

Development of a SPARK Program

```
-- get input fro
Read_Input (User

-- construct SQL
Form_Query (User_

-- Check validity
-- NOTE: unless
-- before Query,

if DB.Valid_Query
   -- Execute t
   DB.Query (SQ
else
   Error_Ha
end if;
```

1 write code

2 run Examiner

SPARK Examiner

VCs

3 run Simplifier

SPARK Simplifier

undis-charged VCs

4a write rules

4b proof reviews

4c run Proof Checker

SPARK Proof Checker

5 compile

Ada compiler

© 2011 by Pyrrhus Software

Pyrrhus Software
Enduring Solutions

Weaknesses Considered

- Common Weakness Enumeration (CWE)
  - A collection of well-defined software weaknesses.
  - Uses: document vulnerabilities, assess software tools, and improve communication between those working on assurance.

- 78 OS Command Injection
- 80 Basic XSS
- 89 SQL Injection
- 99 Resource Injection
- 121 Stack-based Buffer Overflow
- 122 Heap-based Buffer Overflow
- 134 Uncontrolled Format String
- 170 Improper Null Termination
- 244 Failure to Clear Heap Memory Before Release ('Heap Inspection')
- 251 Often Misused: String Management
- 259 Hard-Coded Password
- 367 Time-of-check Time-of-use (TOCTOU) Race Condition
- 391 Unchecked Error Condition
- 401 Failure to Release Memory Before Removing Last Reference ('Memory Leak')
- 412 Unrestricted Externally Accessible Lock
- 415 Double Free
- 416 Use After Free
- 457 Use of Uninitialized Variable
- 468 Incorrect Pointer Scaling
- 476 NULL Pointer Dereference
- 489 Leftover Debug Code

© 2011 by Pyrrhus Software

Classification of Weaknesses

- **Class 1: Weaknesses that Cannot Occur**

  These cannot even be expressed in syntactically correct SPARK code.

- **Class 2: Weaknesses that Can Be Certainly Excluded**

  These may exist in a SPARK program, but are detected by the SPARK toolset, which generates verification conditions (VCs). If VCs are ignored, then such weaknesses may be present.

- **Class 3: Weaknesses that May Occur**

  These are higher level and may occur in a SPARK program.

- **Class 4: Concurrent Programming Weaknesses**

  These are locks and races, which may be certainly excluded, except for files.

© 2011 by Pyrrhus Software

Pyrrhus Software
Enduring Solutions

Class 1

Weaknesses that Cannot Occur in SPARK

- Pointer types and programmer-managed heap storage do not exist in SPARK.
- Thus, the following CWEs are eliminated:
  - **Heap overflow (CWE 122)**
  - **Memory leak (CWE 401)**
  - **Double Free (CWE 415)**
  - **Use After Free (CWE 416)**
  - **Unintentional pointer scaling (CWE 468)**
  - **Heap Inspection (CWE 244)**
    - sensitive information left in heap – SPARK initializes all variables
  - **Null Pointer Dereference (CWE 476)**
    - Java can have "no object" – SPARK defines all return values

© 2011 by Pyrrhus Software

Class 1

Heap Inspection (CWE 244)

- This may occur when buffers with sensitive information are freed without being cleared. For instance, `realloc()` leaving information in memory. SPARK prevents this because the Examiner flags uninitialized variables. A SPARK program cannot, say, declare an array and then read that array for information from an earlier execution.

- CWE 244 is an instance of the more general CWE 226: Sensitive Information Uncleared Before Release. Leaks may occur in main memory or temporary files.
- Leaks via temporary files are not directly addressed in SPARK. However, Ada has temporary files which are not accessible after the main program completes. An appropriate SPARK package could be written giving access to the full Ada File_IO package.

© 2011 by Pyrrhus Software

Pyrrhus Software
Enduring Solutions

Class 1

Null Pointer Dereference (CWE 476)

Consider the Java code fragment:

```
String cmd = System.getProperty("cmd");
cmd = cmd.trim();    ← no object: NullPointerException is raised
```

The equivalent SPARK is:

```
subtype Property_Ix is Integer range 1 .. 128;
subtype Property is String( Property_Ix );

Cmd : Property :=
Property'( others => ' ' ); -- initialize to blanks

Cmd := Get_Property( "cmd" );  ← must return a blank string, or some other
                                 well-defined string, for a non-existant property
Cmd := Trim( Cmd );  ← no exception is raised
```

© 2011 by Pyrrhus Software

Pyrrhus Software
Enduring Solutions

Class 1

Weaknesses that Cannot Occur in SPARK

- ## Uncontrolled Format String (CWE 134)
  - Unfiltered input is used as a string to format data in the printf() style of C/C++ functions.
  - In SPARK there is no 'format string' as there is no printf-style subprogram. Note that SPARK has formatted I/O; it is built upon Ada's text I/O package (Ada.Text_IO).
- ## Improper Null Termination (CWE 170)
  - The software does not properly terminate a string with a null character.
  - In SPARK, when a string object is declared, its size must be known. Hence, we are always dealing with fixed length strings. Thus CWE 170 cannot occur because strings in SPARK are of fixed length; terminators are not necessary.
- ## Use of Error-Prone String Functions (CWE 251)
  - Many string manipulation functions, like strcpy() and subcat(), require special checking or reasoning to be used safely.
  - CWE 251 cannot occur because the only operations on string objects are assignment and concatenation.

© 2011 by Pyrrhus Software

Pyrrhus Software
Enduring Solutions

Class 2

Weaknesses that Can Be Excluded From Programs

- **Stack-based Buffer Overflow (CWE 121)**
  - Write outside the allocated memory of a buffer. “Stack-based” means the buffer is allocated on the stack, for instance, a local variable or a function parameter.
  - The storage requirements of a SPARK program can be determined statically. This is achieved by forbidding recursion and dynamic arrays. Note also that pointer types and heap storage are not allowed.

To implement a buffer in SPARK an array type must be used. Consider the following:

```
subtype Index is Integer range 1 .. 10;
subtype Sensor_value is Integer range -20 .. 60;
type Samples is array( Index ) of Sensor_value;
Buffer : Samples;
I : Index;
```

A statement such as:

```
Buffer(11) := -10;
```

is immediately detected by the Examiner. Consider:

```
I := 11;
...
Buffer(I) := -10;
```

Here the Examiner flags the assignment to `I`.

If the value of the index is determined by calling a function, such as:

```
I := Some_Complex_Function;
```

full analysis and proof for absence of buffer overflows requires the Simplifier to be run on the VCs.

© 2011 by Pyrrhus Software

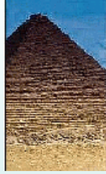
Pyrrhus Software
Enduring Solutions

Class 2

Weaknesses that Can Be Excluded From Programs

- **Unchecked Error Condition (CWE 391)**
  - No code exists to handle an error or exception condition which may occur. Ignoring exceptions and other error conditions may allow an attacker to induce unexpected behavior which goes unnoticed.
  - CWE 391 cannot occur in SPARK for two reasons:
    1. The SPARK eliminates all predefined exceptions.
    2. The Examiner insists that the code handle all return/status values and flags any (error) condition that is ignored.
- **Uninitialized Variable (CWE 457)**
  - A variable is created without assigning it a value. It is subsequently referenced in the program, leading to unpredictable or unintended results.
  - CWE 457 is detected by the Examiner in SPARK. All variables must be initialized.

© 2011 by Pyrrhus Software



Pyrrhus Software
Enduring Solutions

Class 3

Weaknesses that May Occur in SPARK

- **User Input Weaknesses**
  - **OS Command Injection (CWE 78)**
  - **Basic XSS (CWE 80)**
  - **SQL Injection (CWE 89)**
  - **Resource Injection (CWE 99)**
    - These CWEs involve inadequately validating or sanitizing user input or 'embedded' user input in the case of 80.
    - These CWEs correspond to the general problem of 'foreign code'. The problem is passing unconstrained strings to functions where they can create havoc.
    - SPARK can help reduce, but cannot eliminate, these code weaknesses.

© 2011 by Pyrrhus Software

Pyrrhus Software
Enduring Solutions

Class 3

SPARK mitigation in the context of CWE 89

- The first “caution” is that to make an SQL query one calls a non-SPARK subprogram. Thus we may write a SPARK package along the following lines to encapsulate this access:

```
package DB is
function Valid_Query( SQL_String : in String ) return Boolean;

procedure Query ( SQL_String : in     String;
                  Result     : out    String );
--# derives Result from SQL_String;
--# pre Valid_Query( SQL_String );

pragma Import (C, Query);
end DB;
```

This package contains two subprograms: one to validate the caller’s SQL string and the other is a reference to the actual query foreign subprogram (via the 'pragma import').

© 2011 by Pyrrhus Software

Pyrrhus Software
Enduring Solutions

Class 3

Other Weaknesses that may occur in SPARK

- **Hard-Coded Password (CWE 259)**
  - A password embedded in the code used for inbound authentication or for outbound access to external components.
- **Leftover Debug Code (CWE 489)**
  - Debug code can create unintended entry points in an application.

© 2011 by Pyrrhus Software

Pyrrhus Software
Enduring Solutions

Class 4

Concurrent Programming Weaknesses

- RavenSPARK provides protected objects (PO), which encapsulates data shared by cooperating tasks. The runtime system enforces mutual exclusion. An unprotected but shared variable is not permitted, so a programmer cannot forget to properly control access to a shared variable. Files cannot be protected objects.

- **Time-of-check Time-of-use race condition (CWE 367)**
  - Software checks some property of a resource, but the property may change before the resource is used.
  - Using protected objects in RavenSPARK can eliminate this problem, except for file access.
- **Unrestricted Critical Resource Lock (CWE 412)**
  - Software properly uses a lock, but the lock can be externally influenced.
  - There are no external locks; thus CWE 412 is prevented if protected objects are used.

© 2011 by Pyrrhus Software

Pyrrhus Software

Enduring Solutions

Summary Classification

CWE	Does Not Occur	Can Be Certainly Excluded	May Occur
78 OS Command Injection			x
80 Basic XSS			x
89 SQL injection			x
99 Resource Injection			x
121 Stack-based buffer overflow	x		
122 Heap-based Buffer Overflow	x		
134 Uncontrolled Format String	x		
170 Improper Null Termination	x		
244 Failure to Clear Heap Memory Before Release ('Heap Inspection')	x		
251 Often Misused: String Management	x		
259 Hard-Coded Password			x
367 Time-of-check Time-of-use (TOCTOU) Race Condition		x	
391 Unchecked Error Condition		x	
401 Failure to Release Memory Before Removing Last Reference ('Memory Leak')	x		
412 Unrestricted Externally Accessible Lock		x	
415 Double Free	x		
416 Use After Free	x		
457 Use of Uninitialized Variable		x	
468 Incorrect Pointer Scaling	x		
476 NULL Pointer Dereference	x		
489 Leftover Debug Code			x

© 2011 by Pyrrhus Software

So What?

- Programs can have far fewer bugs.
- Use better programming languages.
- Notions related to “bug” are very subtle.
- Formally define weaknesses and formally prove that they cannot occur or are certainly excluded.
- Talk to me: paul.black@nist.gov

© 2011 by Pyrrhus Software