

DEFENCE



DÉFENSE

Software Vulnerabilities & Verification Tools for C/C++ and Java

Frédéric Michaud & Frédéric Painchaud

Defence Scientists

Trusted C2IS Group



R et D pour la défense
Canada

Defence R&D
Canada

Canada



Introduction

- Building reliable and secure software is a difficult task
 - Unmanageable complexity is the main problem
- Flaws have many origins
 - **Design** (ex: backdoor)
 - **Implementation** (ex: buffer overrun)
 - ...



Context

- Sensitive but not safety-critical applications
- Built with familiar technologies that users want
 - Windows, Linux, C++, Java
- Our goal:
 - Get rid of common security problems using automated source code verification tools
- Design flaws:
 - *C2 Secure Design Patterns Study (04-05)*
- Implementation flaws:
 - *Verification tools study (05-06)*



Goals of this Project

- Identify common software defects related to C/C++ and Java usage
 - Non application-specific
- Investigate errors and vulnerabilities created by these defects
- Evaluate best of breed automatic verification tools for C/C++ and Java
 - Defect & error detection performance
 - Usability
- Infer best practices



Plan of the Presentation

1. **Terminology**
2. Errors
3. Vulnerabilities
4. Pitfalls & Shortcomings of C/C++
5. Defects
6. Tools Overview
7. Evaluation
8. Conclusion

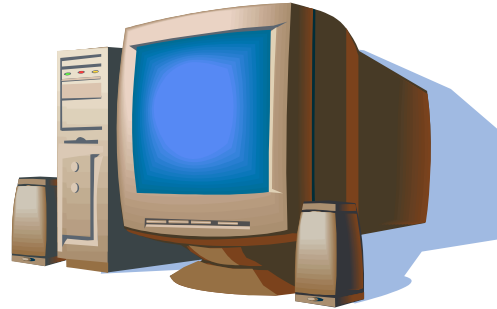


Program Sanity vs. Security

- **Program Sanity**
 - Low level rules/conventions
 - E.g.: C calling convention & parameters placement on the stack
 - Mostly related to programming
- **Security**
 - High level control mechanisms
 - For confidentiality, integrity and availability
 - Mostly related to design



Program Sanity vs. Security



Program Sanity

- Protected memory
- Valid control flow
- Valid data flow
- Correct management of resources

Security

- Access Control
- Anti-virus
- Intrusion Prevention Systems
- Firewall



Program Sanity vs. Security

- Automatic detection of security problems
 - Too much variability
 - Too much complexity
- Automatic detection of program sanity problems
 - More or less always the same thing
 - Especially interesting for C/C++
- Security begins with program sanity
 - **Program sanity problems are the main cause of software security problems**

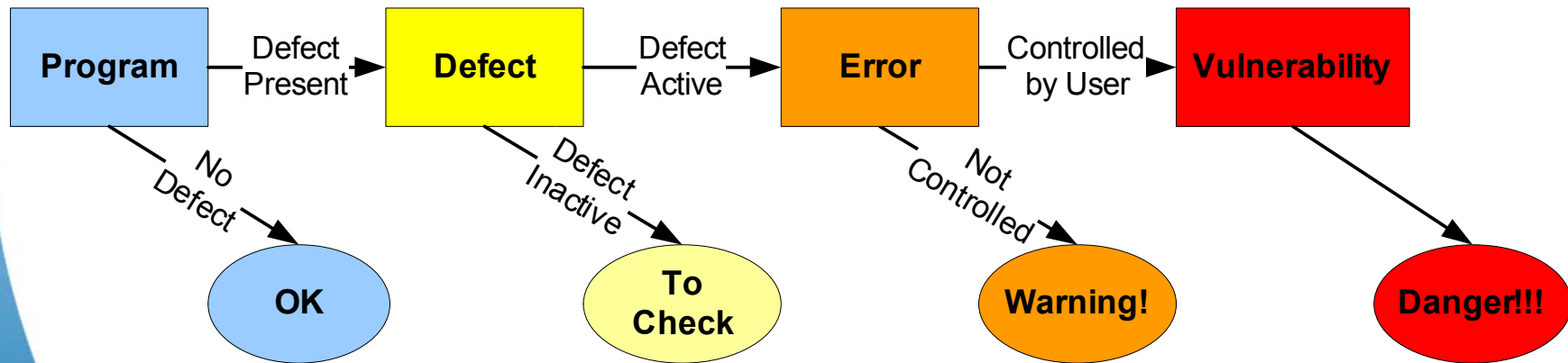


Some Terminology

- Error → Execution
 - Event that occurs when the behavior of a program diverges from “what it should be”
- Defect → Code
 - Cause of an error, a set of program instructions
 - Can be the lack of something
- Vulnerability → Exploitation
 - Defect allowing a user to control the program execution when it should not

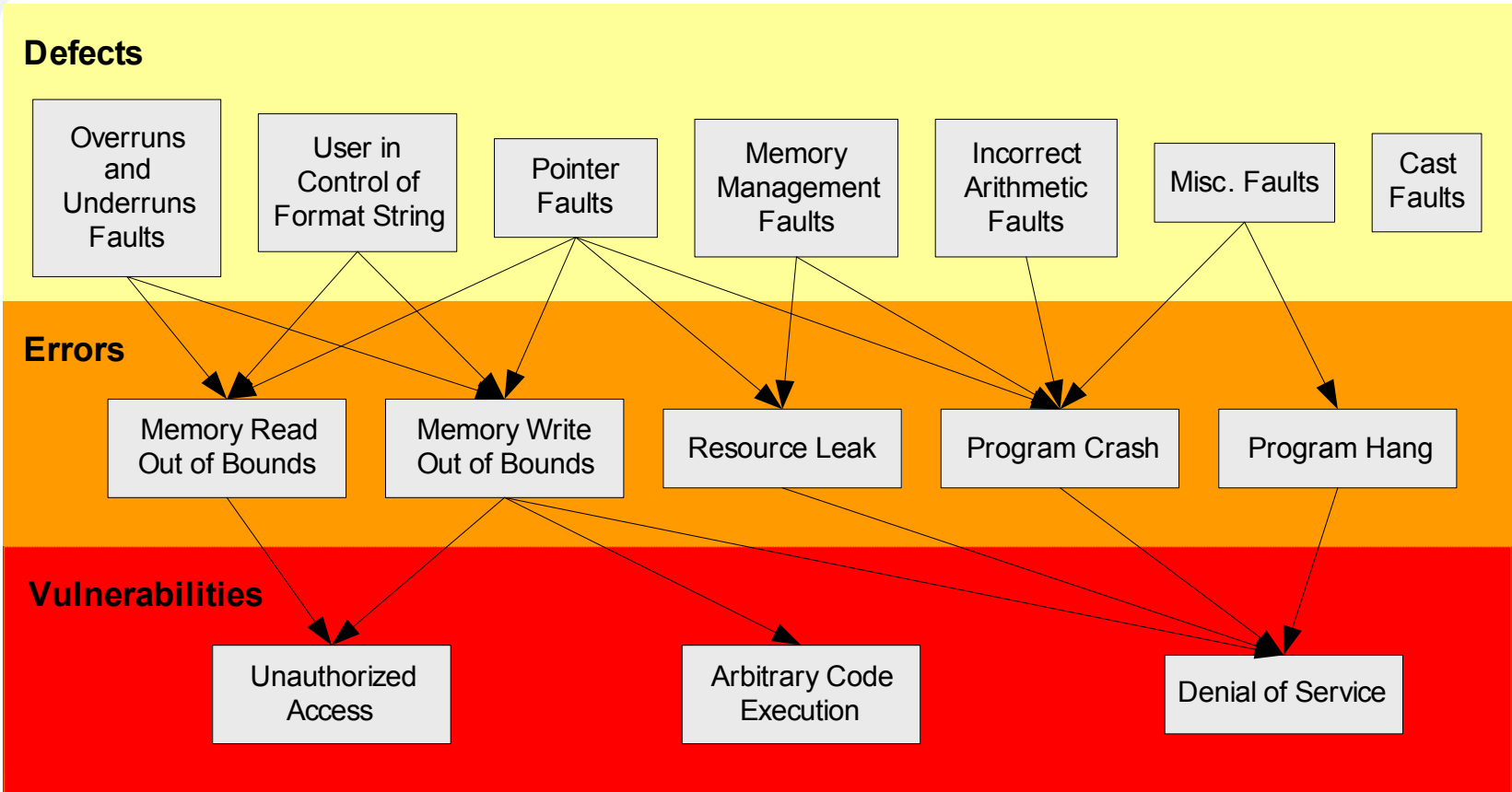


Defects, Errors and Vulnerabilities





Defects, Errors and Vulnerabilities





Plan of the Presentation

1. Terminology
- 2. Errors**
3. Vulnerabilities
4. Pitfalls & Shortcomings of C/C++
5. Defects
6. Tools Overview
7. Evaluation
8. Conclusion



Errors

- The list of possible low-level problems is almost endless
 - No interest in the correctness of computations with respect to specifications
- Correct low-level program execution
 - Memory access
 - Control flow
 - Resource allocation
- Java is immune to most program sanity problems



Memory Write Out of Bounds

- A region of valid memory is overwritten
- Impacts
 - Depends on what is overwritten
 - **Can lead to many serious vulnerabilities**
- Causes
 - Bad pointer arithmetic
 - Array walking with bad index value
- Java: cannot happen (will throw an exception)



Memory Read Out of Bounds

- A region of invalid memory is read instead of a valid one
- Impacts
 - Errors in computations
 - **Sensitive values could be read**
- Causes
 - Reading of a string not terminated by a null
 - Array walking with bad index
- Java: cannot happen (will throw an exception)



Resource Leak

- A no longer needed resource is not returned to the available pool
 - Memory, file handle, network connection, ...
- Impacts
 - Depends on the resource and its usage
 - **Can lead to slowdown and crash**
- Causes
 - Reference lost because of pointer reuse
 - Programmer forgot to free the resource
- Java: the garbage collector helps a lot



Program Hang

- Program is in an infinite loop or wait state
- Impacts
 - Denial of service
- Causes
 - Threads in deadlock state
 - Conditions to exit a loop never reached



Program Crash

- An unrecoverable error happens and the execution of the program is stopped
- Impacts
 - Denial of service
- Causes
 - Dereference of an invalid pointer (page fault)
 - Uncaught exception
 - Division by zero



Plan of the Presentation

1. Terminology
2. Errors
- 3. Vulnerabilities**
4. Pitfalls & Shortcomings of C/C++
5. Defects
6. Tools Overview
7. Evaluation
8. Conclusion



Vulnerabilities

- Errors in general are undesirable
- But the real problem is vulnerabilities
 - Especially the remotely-exploitable ones
- A vulnerability allows an attacker to have some form of control over the program
 - Influence the flow of control
 - Influence the flow of data
- Memory read or written out of bounds
 - Cause of most dangerous vulnerabilities



Denial of Service

- Allows an attacker to prevent users from getting correct service
- How it's usually done
 - Create an unrecoverable error condition
 - Exploit a resource leak
- Java
 - Most program sanity problems throw unchecked exceptions
 - Problems are “transformed” into denials of service if exceptions are not caught



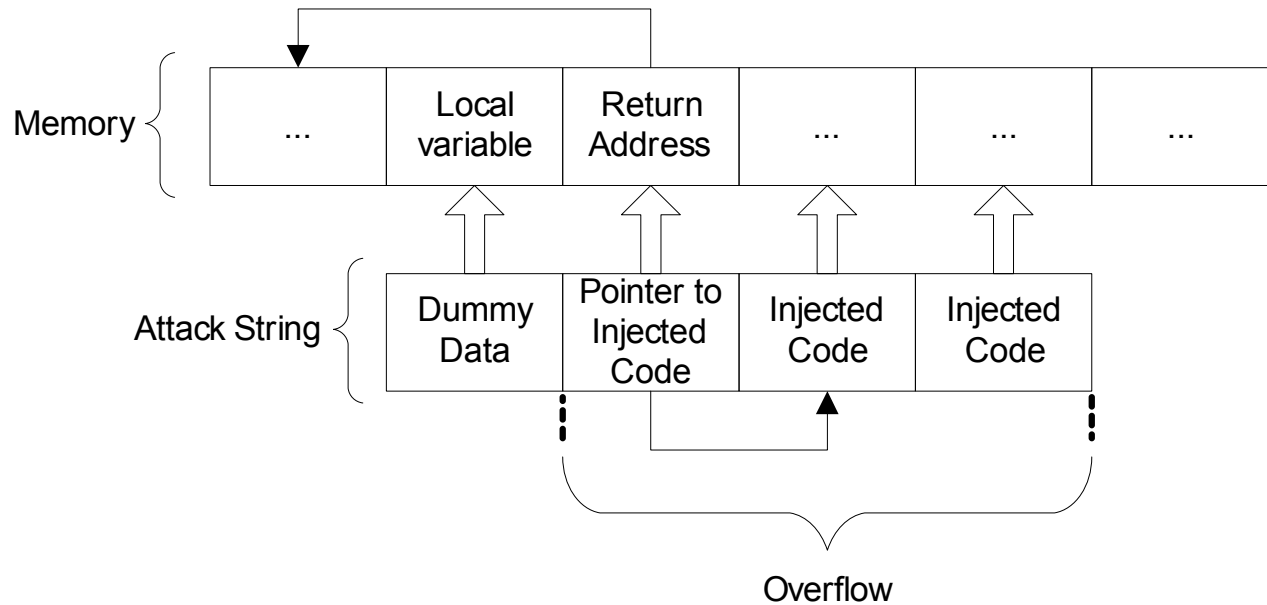
Unauthorized Access

- Allows an attacker to access functionalities without the required authorization
- How it's usually done
 - Bypass the control mechanism by modifying it in memory
 - Read sensitive values in memory and use them to get access



Code Injection

- Overwrite a function pointer that will be called
- Allows an attacker to take control of a process by redirecting its execution **to his own code**
- Also known as *buffer overflow* or *stack smashing vulnerability*





Plan of the Presentation

1. Terminology
2. Errors
3. Vulnerabilities
- 4. Pitfalls & Shortcomings of C/C++**
5. Defects
6. Tools Overview
7. Evaluation
8. Conclusion



Pitfalls & Shortcomings of C/C++

- Many errors are possible because of choices made when C/C++ were created
- These choices
 - Require too much “micro-management” of the program’s behavior
 - “Encourage” mistakes
 - Give serious consequences to seemingly benign errors
- Java creators had these problems in mind and got rid of the majority of them



C/C++ Lack of Type-safety

- Type-safety ensures values assigned to variables are correct
 - Type-safety helps enforce the execution model
 - **Type-safe programs are *fail-fast***
- Execution of erratic programs is not stopped
 - Many *exploits* are using this fact
- Java programs are type-safe
 - Verified at compile time and load time



C/C++ Pointer Arithmetic

- The ability to change the value of a pointer without restriction
 - Can read or write anywhere in memory
 - Control mechanisms can be *bypassed*
 - Easy to create very obscure bugs
 - **Much higher verification complexity**
- There is no pointer arithmetic in Java



C/C++ Buffers Have a Static Size

- Buffers cannot grow to accommodate data
 - Buffer accesses are not checked
 - An overflow will overwrite memory
 - Validation is cumbersome
 - Source of *buffer overflow* vulnerabilities
- Java will throw an exception when an overflow occurs



C Lack of Robust String Type

- C has no type for character strings
 - Static buffers with overflow problems are used instead
 - Size of string indicated by a *null* at the end
 - Strings are used a lot in programs
 - Very fragile: what if the *null* is not there?
 - Source of *buffer overflow* vulnerabilities
- C++ programs can use the string type in the STL
 - Not used enough
- Java only has a robust string type



C/C++ Vulnerabilities in Std Libraries

- String manipulation functions
 - `strcpy()`, `gets()` and friends
 - Lack bounds checks for destination buffer
 - Possible overflow if data size is not checked
 - Source of *buffer overflow* vulnerabilities
 - Use replacement functions: `strncpy()`



Plan of the Presentation

1. Terminology
2. Errors
3. Vulnerabilities
4. Pitfalls & Shortcomings of C/C++
- 5. Defects**
6. Tools Overview
7. Evaluation
8. Conclusion



Defects: Some Observations

- Many defects are not ‘always on’
 - They will not always generate errors
 - Complex conditions have to be met
 - Input values play an important role
- Most defects are composite
 - Cannot be attributed to a single program instruction
 - A defect can be the absence of something
 - Data validation
- Mostly C/C++ defects – (selection of 25)



Defects

1 – Memory Management Faults

- 1.1 – Reading of freed memory
- 1.2 – Under allocated memory for a given type
- 1.3 – Call of `free ()` with an invalid pointer
- 1.4 – Incorrect C++ array deletion
- 1.5 – Call of `memcpy ()` with overlapping memory regions
- 1.6 – Reading of an uninitialized variable
- 1.7 – Non-virtual destructor of derived class not called



Defects

2 – Overrun and Underrun Faults

- 2.1 – Overrun or underrun of an array
- 2.2 – Dereference of a past-the-end C++ iterator
- 2.3 – Dereference of an erased C++ iterator
- 2.4 – Incorrect size parameter to a buffer function
- 2.5 – Use of negative array index or size
- 2.6 – Reading of a string of arbitrary length without limit
- 2.7 – Reading of a non null-terminated string



Defects

3 – Pointer Faults

3.1 – Return of a pointer to a local variable

3.2 – Incorrect pointer arithmetic

3.3 – Dereference of a null pointer

3.4 – Resource reference lost

4 – Incorrect Arithmetic Faults

4.1 – Division by zero

4.2 – Integer overflow or underflow

4.3 – Bit shift bigger than integral type or negative



Defects

5 – Cast Faults

5.1 – Integer sign lost because of implicit unsigned cast

5.2 – Integer precision lost because of bad cast

6 – Miscellaneous Faults

6.1 – Unspecified format string

6.2 – Endless loop



Plan of the Presentation

1. Terminology
2. Errors
3. Vulnerabilities
4. Pitfalls & Shortcomings of C/C++
5. Defects
- 6. Tools Overview**
7. Evaluation
8. Conclusion



Tools Overview

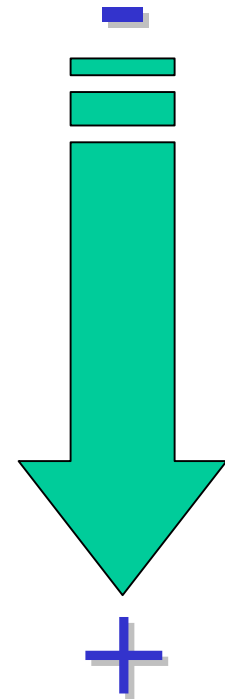
- Evaluated tools – most are multiplatform
 - C/C++: 27 tools
 - Java: 37 tools
- Free (open source) versus commercial tools
 - C/C++: best tools are commercial
 - Java: many good free tools
- Most academic tools are only proofs of concepts
- Evaluation criteria
 - Precision, scalability, coverage, diagnostic



Tools

1. Program Conformance Checkers
 - Detect defects
2. Runtime Testers
 - Detect errors
3. Advanced Static Analyzers
 - Detect defects

Required Investment*



* In money, time, training, resources, etc.



Program Conformance Checkers

- Check source code for common *bug patterns*
- Lightweight analysis based on syntax
 - Excellent scalability
 - Many false positives and negatives
 - Poor performance except for a few defects
 - E.g.: unspecified format string
- Many free tools are in this category



Program Conformance Checkers

C/C++

- **Secure Programming Lint**
 - C only
 - Many “parse errors”
 - Superficial analysis without annotations
- **FlawFinder**
 - Format strings
 - Vulnerable functions
 - A lot of false positives

Java

- **PMD**
 - Enforces coding conventions
 - Well integrated
 - Cut & paste detector
- **AppPerfect CodeAnalyzer**
 - Similar to PMD
 - Different rules
 - Affordable & effective



Runtime Testers

- Program behavior cannot always be deduced statically
 - Some values are not known before runtime
- Look for errors while the program is running
 - Code is instrumented with checks
 - Fine-grained analysis
 - Excellent scalability
 - Coverage can be poor without a good strategy
- Excellent for composite defects related to memory usage



Runtime Testers

C/C++

- **Parasoft Insure++**
 - Source instrumentation
 - Impressive performance
 - Easy to use (debugger)
 - Good diagnostic
- **Rational Purifier**
 - Similar to Insure++
 - Analysis not as thorough

Java

- **AppPerfect Java Profiler**
 - Heap, threads, objects, CPU usage, disk I/O, memory usage
 - Heap browser
 - Deadlock detection
- **JProfiler**
- **NetBeans Profiler**



Advanced Static Analyzers

- Work on program semantics instead of syntax
 - Use formal methods, like abstract interpretation or model-checking
 - Scalability is often problematic
- Code must be compiled into a model
 - A lot of code portability issues
- Generally much slower than other tools
- Very sophisticated tools: often expensive



Advanced Static Analyzers

C/C++

- **Coverity Prevent (SWAT)**
 - Good integration with makefiles
 - Excellent diagnostic with execution trace
 - Surprisingly scalable
- **PolySpace for C++**
 - Very thorough but slow and memory hungry
 - Can detect runtime exceptions statically

Java

- **ESC/Java 2**
 - Can prove properties on the behavior of programs
 - Have to add annotations
 - Very powerful
 - Hard to use
 - A must-have for critical Java software development



Plan of the Presentation

1. Terminology
2. Errors
3. Vulnerabilities
4. Pitfalls & Shortcomings of C/C++
5. Defects
6. Tools
- 7. Evaluation**
8. Conclusion



C/C++ Evaluation

- Preliminary tests showed only 3 tools could help us achieve our goal:
 - Coverity Prevent
 - Parasoft Insure++
 - PolySpace for C++
- 2 sets of tests
 - Synthetic tests for every kind of defect (25)
 - Buggy code in production (~10,000 lines)



Comparing Apples and Oranges

- Error detection vs. defect detection
 - A conversion is necessary
- Synthetic tests
 - Defects are known
 - The errors they will cause too
 - Easy to convert everything to defects
- Buggy code in production
 - Defects are not known in advance
 - Used best result as baseline (errors)

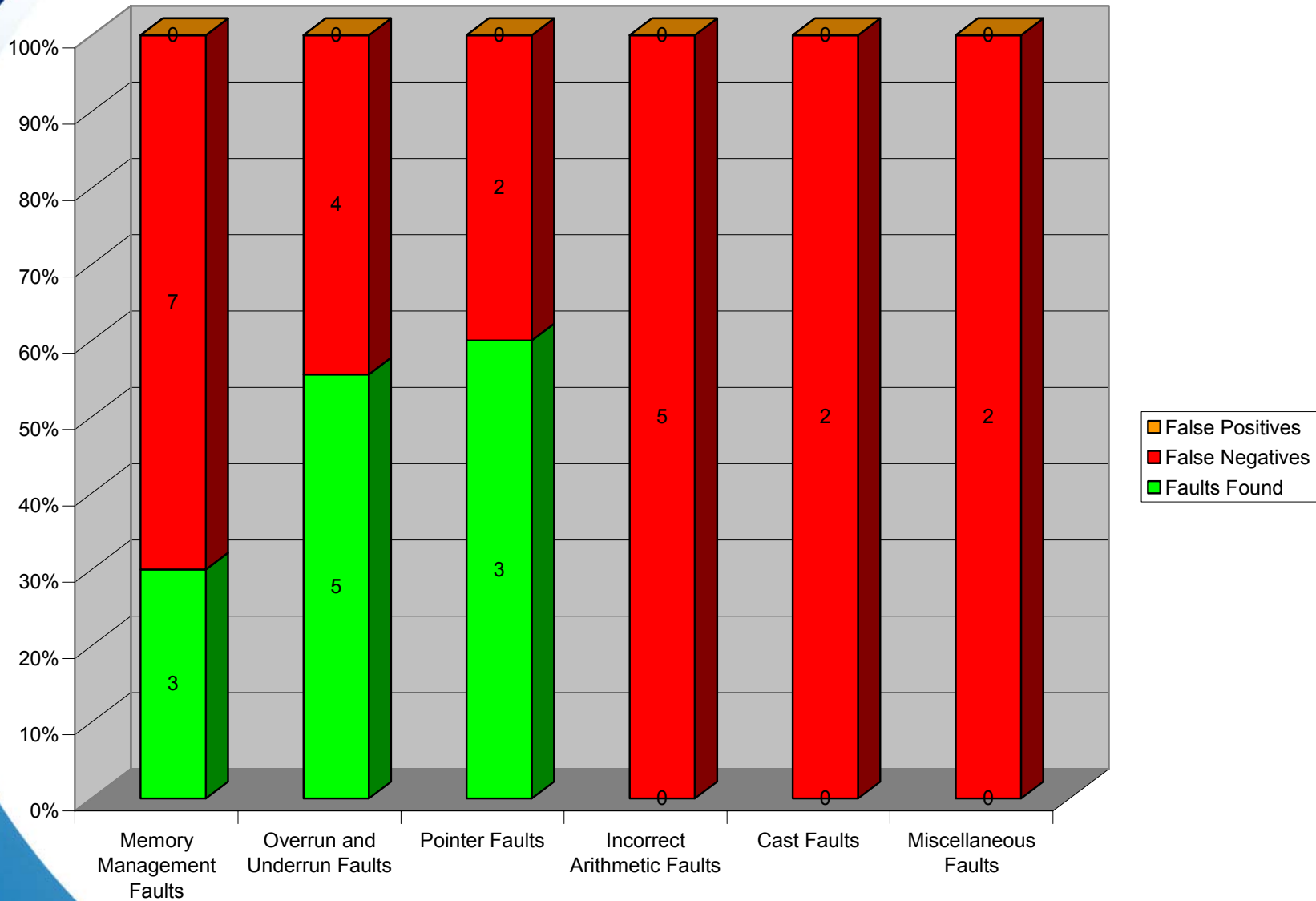


Results of Synthetic Tests

- A C++ class for every kind of defect (25)
- Integrated in a small high-quality open-source application (Windows MFC)
- Tests that would lead to a program crash or hang were deactivated for Insure++
- Tests are called from the `main()`
 - MFC applications have a “special” `main()`
 - PolySpace had to be used in a “class by class” analysis mode
- **No tool tries to detect every kind of defect or error**

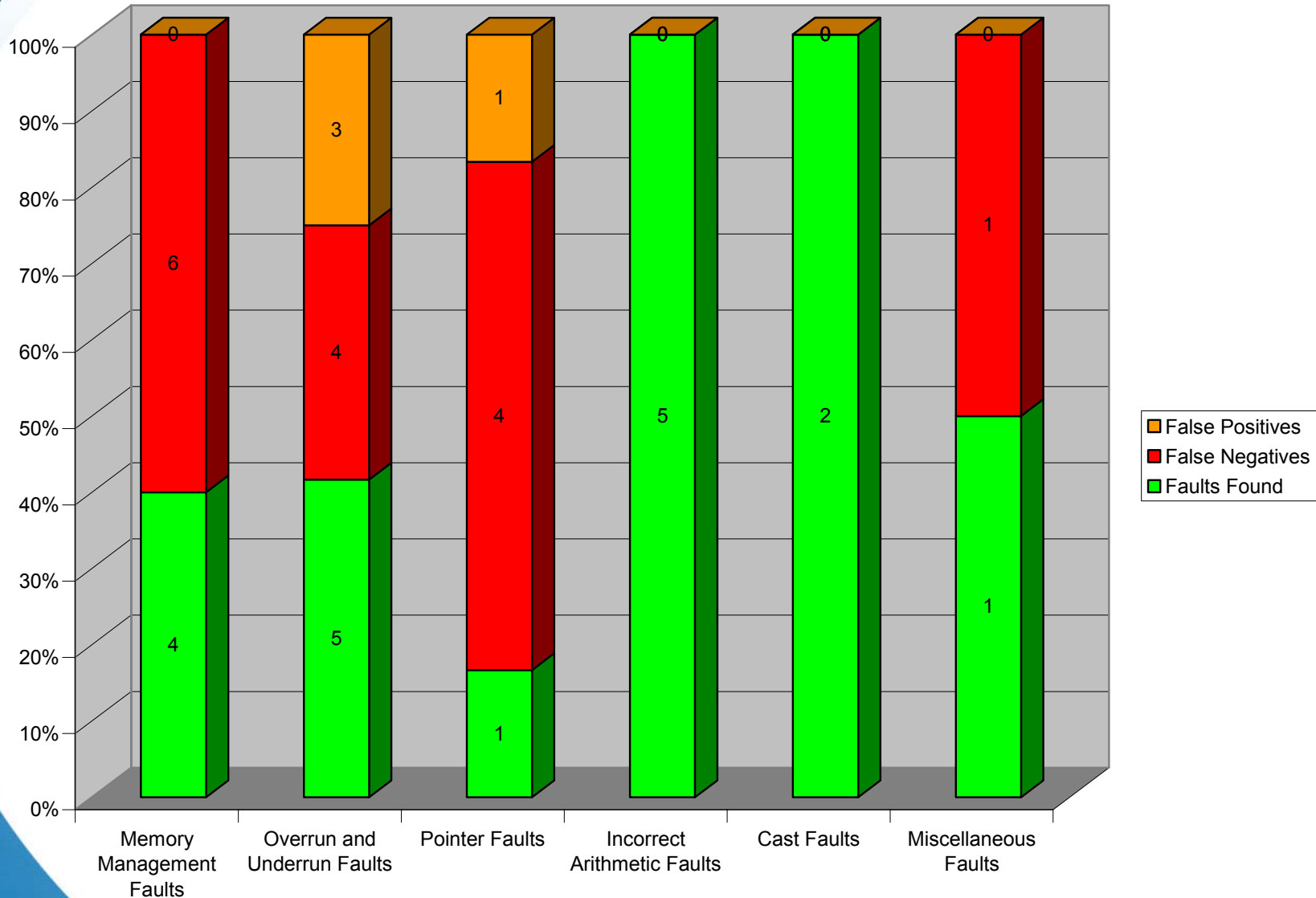


Results of Synthetic Tests - Coverity



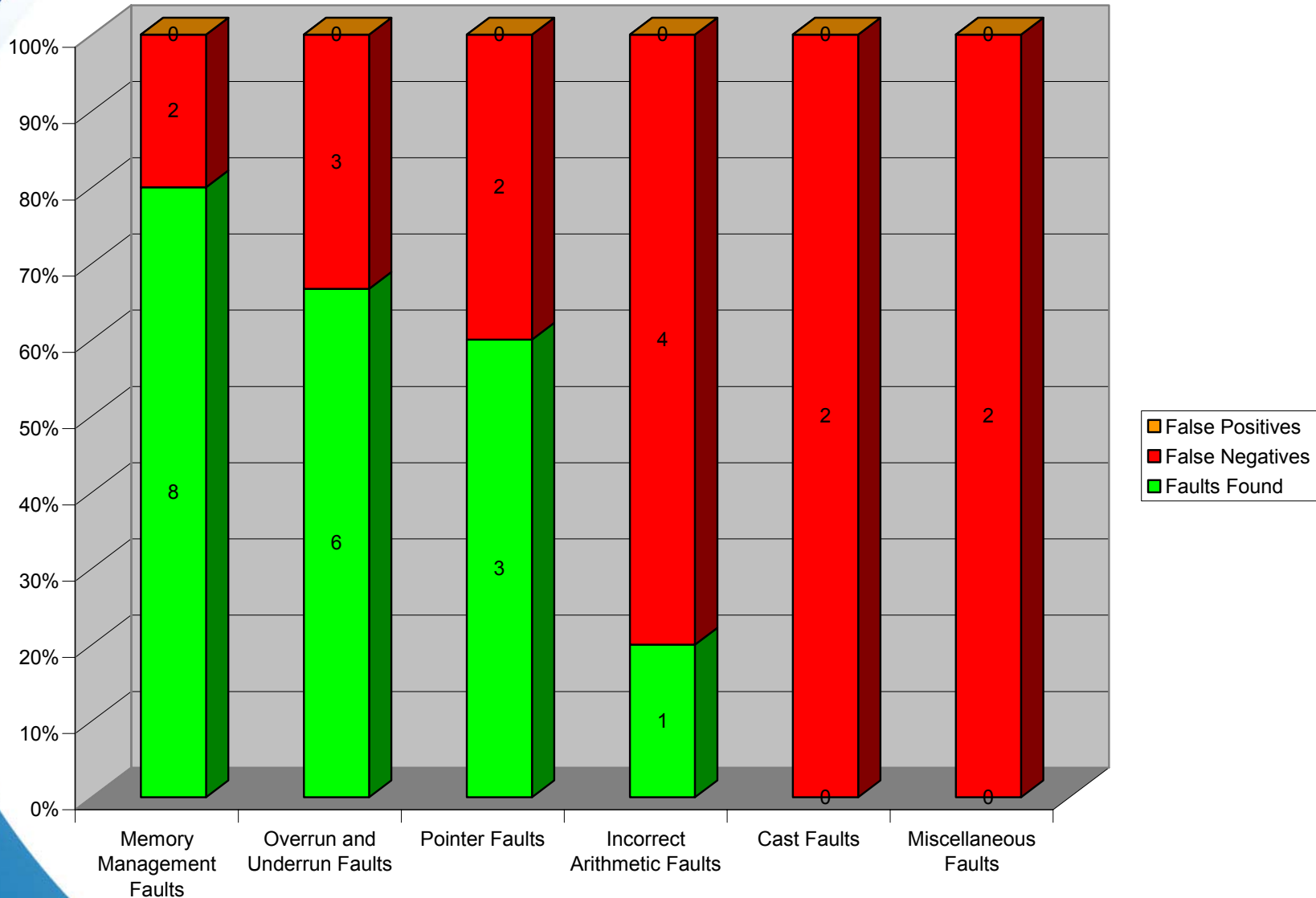


Results of Synthetic Tests – PolySpace





Results of Synthetic Tests – Insure++





Results of Buggy Code in Production

- Numerical analysis application
 - About 10,000 lines of code
 - In production for many years
 - Reads a file and displays the results
 - Not a reactive program like MFC Apps
 - Bad quality code
 - “C+” design
 - A lot of cut and pasted, “spaghetti” code
 - **Really a worst-case scenario**



Results of Buggy Code in Production

Errors	Cov	Ins	Pol*
Memory write out of bounds	0	42	2
Memory read out of bounds	1	114	0
Resource leak	2	10	0
Program crash	2	0	0

* Over 300 false positives, ~16 hours of computation



C/C++ Analysis

- Static analysis tools need good quality code to perform well
 - Pointer arithmetic and void pointers can also be problematic
 - PolySpace will stop the analysis of a branch when a critical error is found
- Code portability issues
 - Preprocessor definitions and conditional compilation
 - Compiler-specific extensions to C/C++



Java Evaluation

- Preliminary tests showed 11 tools could be useful
- The ones that stand out:
 - AppPerfect DevSuite
 - PMD
- 2 large, open-source applications tested



Java Analysis

- Java design is better: less low-level defects
 - Fewer problems to look for
 - **Tools for Java are great to assess software quality**
- No code portability issues
 - A lot easier than C/C++



Plan of the Presentation

1. Errors
2. Vulnerabilities
3. Pitfalls & Shortcomings of C/C++
4. Defects
5. Tools
6. Evaluation
- 7. Conclusion**



Conclusion

- Security problems generally don't come from the failure of security mechanisms
 - The failure occurs at a lower level
- C/C++ are especially problematic
 - Enforce almost no restriction on execution
 - Vulnerabilities with serious consequences
- Java is immune to most C/C++ problems
 - No serious vulnerabilities



Conclusion

- Best usage scenario for Coverity
 - Whole applications compiled with makefiles
- Best usage scenario for PolySpace
 - Small sections of critical code where runtime exceptions should never happen
- Best usage scenario for Insure++
 - Integrated to test cases
 - Test of hybrid systems based on many heterogeneous components
 - Values are always available at runtime



Conclusion

- Verifying C/C++ programs is a huge challenge
 - These languages are very hard to analyze
 - Undefined behaviors, pointers, compiler-specific extensions, etc.
 - No verification tool can reduce the risk significantly enough for this context
- For sensitive applications, we recommend the use of Java or any managed .Net language
 - Use C/C++ only if you really have to
 - Restricted language usage, test cases, and the use of verification tools are a must

Frederic.Michaud@drdc-rddc.gc.ca
Frederic.Painchaud@drdc-rddc.gc.ca

http://www.drdc-rddc.gc.ca/researchtech/malicots/home_e.asp





The Way Ahead

- Detection of higher level security problems
 - A model for the security behavior of programs is needed
- Automatic program hardening
 - Based on aspect oriented programming
- Current research project
 - Partnership with NSERC, Bell University Labs and Concordia University



PolySpace Viewer

The screenshot displays the PolySpace Viewer interface. The main window is titled "PolySpace Viewer - C:\Documents and Settings\fmichaud\My Documents\Rédaction\Practical Verification & Safeguard Tools for CC++\tests\Poly". The interface includes a menu bar (File, Edit, Windows, Help), a toolbar with various icons, and a toolbar with filters (Alpha, Beta, Gamma, Filter all). Below the toolbar is a list of procedural entities, including COR.27 through COR.75, and several IrblemP1ModuleClass methods. The status bar at the bottom indicates "COR.62 Details: unreachable failure of correctness condition [call never raises an exception (warning)]".

The "Variables View" window shows a list of variables with columns for "Variables", "Nb read", and "NI". The "IrblemP1ModuleClass.cxx" window displays the following code:

```
208 /* Initialization of application */
209 if (fRc)
210     fRc = MenuLib::InitApp(IrData.pszIniFile, eEcran);
211
212 /* Read meteo file METDATA */
213 if (fRc)
214     fRc = LireMeteoFile(eEcran, IrData.pszDefFile);
215
216 if (!IrData.fIrbx)
217     RunBatch(eEcran);
218 else
219     RunIrbx(eEcran);
220 }
221
222 if (IrData.fIrbx)
223 {
224     if (!ToScreen)
225         #ifdef unix
```




Coverity Prevent

Coverity Error Browser - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://localhost:5467/view-error.cgi?nid=18&runid=22&user=admin&magic=aae93c994bd9f81ea4f55ced07e494f3&prevpage=query-run-table.cgi%3Fmagic%3Daae93c994bd9f81ea4f55ced07e494f3%26prevpage%3D%252Fvie

Language Tools Merriam-Webster O... TERMIUM Plus® Google Advanced Gr... Computer and Infor... Google Scholar # The Metasploit Project

coverity

Return to [Search Results](#)

ERROR

CID: 15

Checker: RESOURCE_LEAK ([help](#))

File: c:/coverity/test/irblemp1/irblemp1moduleclass.cxx

Function: Irblemp1ModuleClass::FctMain(int, char **)

Description: Returned without freeing storage "fp"

Status:

- UNINSPECTED
- BUG
- FALSE
- RESOLVED
- IGNORE
- PENDING

Coverity 2005-Sep-13 08:12:44

New status row inserted by system.

Add Comment:

Change Owner:

Viewing file: c:/coverity/test/irblemp1/irblemp1moduleclass.cxx

Xrefs: On Off

Event list: [\[top\]](#) [\[alloc_fn\]](#) [\[var_assign\]](#) [\[leaked_storage\]](#)

Event alloc_fn: Called allocation function "fopen"

Event var_assign: Assigned variable "fp" to storage returned from "fopen"

Also see events: [\[var_assign\]](#)[\[leaked_storage\]](#)

```
153     fp = fopen(IrData.pszIniFile, "r");
```

At conditional (1): "fp == 0" taking false path

```
154     if (fp == NULL) {
155         Error.Level = ER_ERROR;
156         Error.Code = 110;
157         strcpy(Description, "Cannot open file \"");
158         strcat(Description, IrData.pszDefFile);
159         strcat(Description, "\".\n\"");
160         strncpy(Error.Description, Description, NBDESC);
161         strncpy(Error.FuncId, FuncId, NBFUNCID);
162         ErrorLib::CheckError(Error, ProgramId, ErrorMode, ER_FILE);
163     }
164
165
166     /* Execution in batch or irbxmode */
```

At conditional (2): "fRc != 0" taking false path

```
167     if (fRc)
168     {
169         /*
170         Creation d'un fichier pour indiquer que le programme a bien roule,
171         Si le programme se rend jusqu'au bout, ce fichier sera efface
172         */
173         /* le fichier n'est créé qu'en mode Irbx */
174         if(IrData.fIrbx)
175         {
176             fp = fopen("kz3f5dfail50.tmp", "w");
177             fprintf(fp, "If at the end this file is still in the directory, Irblemp1 didn't work properly.\n");
178             fclose(fp);
179         }
180
181         /* Initialization of screen parameters */
182         if (fRc)
183             fRc = MenuLib::InitPage(IrData.pszIniFile, &Ecran);
184
185         /* Initialization of system command and sub programs */
186         if (fRc)
187             fRc = MenuLib::InitCmdSys(IrData.pszIniFile, &Ecran);
188
189         /* Initialization of data structure IRBLEMP1 */
190         InitDataIrblemp1();
```

Transferring data from localhost...



Insure

```
Insure - irblemp1_case2.txt - Notepad
File Edit Format View Help
[[Irblemp1ModuleClass.cxx:843] **LEAK_ASSIGN**
>>
    IrData.pszDefFile = (char*) malloc(strlen(ppszParam[i+1])+1);

Memory leaked due to pointer reassignment: IrData.pszDefFile

Lost block : 0x01a81760 thru 0x01a8176a (11 bytes)
    IrData.pszDefFile, allocated at Irblemp1ModuleClass.cxx, 776
        malloc() (interface)
Irblemp1ModuleClass::InitVargGlob() Irblemp1ModuleClass.cxx, 776
    Irblemp1ModuleClass::FctMain() Irblemp1ModuleClass.cxx, 122
        start_module() mainirblemp1.cpp, 435
            main() mainirblemp1.cpp, 447

Stack trace where the error occurred:
Irblemp1ModuleClass::ParsParam() Irblemp1ModuleClass.cxx, 843
    Irblemp1ModuleClass::FctMain() Irblemp1ModuleClass.cxx, 125
        start_module() mainirblemp1.cpp, 435
            main() mainirblemp1.cpp, 447

[[Irblemp1ModuleClass.cxx:7299] **READ_OVERFLOW**
>>
    *((int*)IrData.lstOptUpper.pElem[0].pDefData) = atoi(pszvalue);

String is not null terminated within range: <argument 1>

Reading : 0x0012f87c
From block: 0x0012f87c thru 0x0012f97b (256 bytes)
    pszvalue, declared at Irblemp1ModuleClass.cxx, 7225

Stack trace where the error occurred:
        atoi() (interface)
Irblemp1ModuleClass::LireDefUpper() Irblemp1ModuleClass.cxx, 7299
Irblemp1ModuleClass::InitDefFile() Irblemp1ModuleClass.cxx, 1707
    Irblemp1ModuleClass::FctMain() Irblemp1ModuleClass.cxx, 193
        start_module() mainirblemp1.cpp, 435
            main() mainirblemp1.cpp, 447
```



AppPerfect DevSuite

AppPerfect Code Analyzer - Professional - [NewProject]

File View Project Tools Help

Packages

- net.percederberg.mibble.asn:
 - Asn1Analyzer.java
 - Asn1Constants.java
 - Asn1Parser.java
 - Asn1Tokenizer.java
- net.percederberg.mibble.brow:
 - AboutDialog.java
 - MibName.java
 - MibTreeBuilder.java
 - SnmpOperation.java
 - TreeListener.java
- net.percederberg.mibble:
 - CompoundContext.java
 - DefaultContext.java
 - FileLocation.java
 - Mib.java
 - MibAnalyzer.java
 - MibbleBrowser.java
 - MibblePrinter.java
 - MibbleTester.java
 - MibbleValidator.java
 - MibContext.java
 - MibException.java
 - MibLoader.java
 - MibLoaderException.java
 - MibLoaderLog.java
 - MibReference.java
 - MibSymbol.java
 - MibType.java
 - MibTypeSymbol.java
 - MibTypeTag.java

Rule Id	Seve...	Category	Line Number
String_concatenation	Critical	Optimization	532
String_concatenation	Critical	Optimization	547
Use_shift_operators	High	Optimization	120
Use_shift_operators	High	Optimization	120
Avoid_method_calls_in_loop	High	Optimization	449
Avoid_method_calls_in_loop	High	Optimization	529
Check_loop_counter_against_zero	High	Optimization	449
Check_loop_counter_against_zero	High	Optimization	516
Check_loop_counter_against_zero	High	Optimization	529
Declare_private_constant_fields_final	Medium	Optimization	603
Maximum_number_of_fields	Medium	Metrics	94
Define_initial_capacities	Medium	Optimization	109
Avoid_numeric_literals	Medium	CodeConvention	112
Avoid_numeric_literals	Medium	CodeConvention	116
Avoid_numeric_literals	Medium	CodeConvention	116
Avoid_numeric_literals	Medium	CodeConvention	120
Avoid_numeric_literals	Medium	CodeConvention	120
Avoid_numeric_literals	Medium	CodeConvention	217
Avoid_numeric_literals	Medium	CodeConvention	217
Avoid_numeric_literals	Medium	CodeConvention	217

Console Messages

```
Analyzing C:\Amex_Demo\mibble-2.3\src\net\percederberg\mibble\type\ValueConstraint.java
Analyzing C:\Amex_Demo\mibble-2.3\src\net\percederberg\mibble\type\ValueRangeConstraint.java
Analyzing C:\Amex_Demo\mibble-2.3\src\net\percederberg\mibble\value\BitSetValue.java
Analyzing C:\Amex_Demo\mibble-2.3\src\net\percederberg\mibble\value\BooleanValue.java
Analyzing C:\Amex_Demo\mibble-2.3\src\net\percederberg\mibble\value\NullValue.java
Analyzing C:\Amex_Demo\mibble-2.3\src\net\percederberg\mibble\value\NumberValue.java
Analyzing C:\Amex_Demo\mibble-2.3\src\net\percederberg\mibble\value\ObjectIdentifierValue.java
Analyzing C:\Amex_Demo\mibble-2.3\src\net\percederberg\mibble\value\StringValue.java
Analyzing C:\Amex_Demo\mibble-2.3\src\net\percederberg\mibble\value\ValueReference.java
```

30 day remaining... luation period. Finished Analyzing 00:00:09 Oct 13, 2004 4:41 PM