SAFE RIVER
Safety & Security Forge

Software security assessment based on static analysis

Christèle Faure

Séminaire SSI et méthodes formelles

Réalisé dans le projet Baccarat cofinancé par l'union européenne

l'Europe s'engage en Ile-de-France avec le FEDER

UNION EUROPEENNE

Context

- **> 200 static tools for security at source code level**
- **Many programming languages: C, C++, Java, Ada, Perl, Python, PHP, Javascript …**
- **A wide range of underlying technologies: from grep to abstract interpretation**
- **Main security issues:**
  - Identification of “dangerous” function calls
    - Textual analysis from data base
  - Identification of dangerous patterns
    - Pattern matching from data base
  - Detection of non conformances to design and coding rules
    - Data and control flows
  - Proof of absence of “errors” and “weaknesses”
    - Sound semantic analysis

Illustration of underlying techniques

- **Problem: Identify calls to fscanf and their impact on security**
- **Code example**

```
(123) fscanf (file,format,precious);
(124) If (cond){compute1(precious, result1);}
(125)           {compute2(precious, result2);}
```

- **Results**

Textual analysis	line 123
Pattern matching	INPUT(file), FORMAT(format)
Data and control flows	file $\rightarrow_{\text{impacts}}$ precious
Sound semantic analysis	precious $\rightarrow_{\text{impacts}}$ {result1, result2}

Identification of dangerous calls

- RATS, (ITS4 dead), Flawfinder, Pscan
- Flawfinder log extract

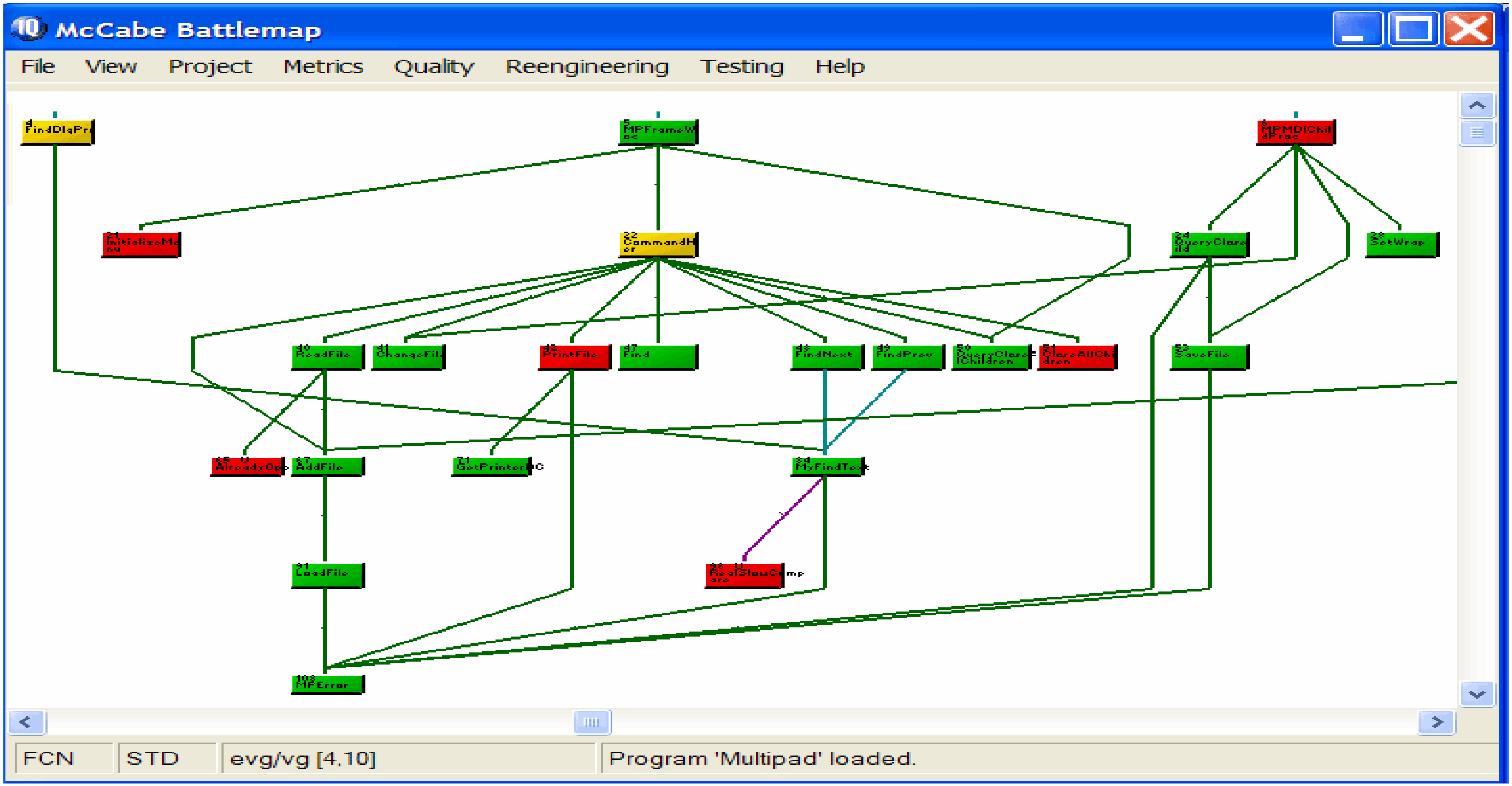
```
Flawfinder version 1.27, (C) 2001-2004 David A. Wheeler.
Number of dangerous functions in C/C++ ruleset: 160
Examining test.c
Examining test2.c
test.c:32:  [5] (buffer) gets:
  Does not check for buffer overflows. Use fgets() instead.
...
Not every hit is necessarily a security vulnerability.
There may be other security vulnerabilities; review your code!
```

Identification of dangerous patterns

- **McCabe IQ**
  - Microsoft SDL banned Function calls
  - Analysis kernel
    - Function call relationships connecting
      - Attack surface: input routines
      - Attack target: banned functions
    - Complexity measures
- **Klocwork**
  - CWE software weaknesses
  - Analysis kernel
    - Dataflow Analysis on the Control Flow Graph to monitor data objects creation, modification, use and deletion
    - Symbolic Logic to remove any path that cannot be executed at runtime
    - Accurate Bug Identification and Vulnerability Analysis

McCabe IQ

- **Complexity and presence of dangerous function calls**



Detection of non conformances

- **IBM Rational AppScan (Ounce Security analyst)**
  - Web flaws (OWAST Top 10)
  - PCI Data Security Standards, ISO 17799, ISO 27001, Basel II, SB 1386, PABP (Payment Application Best Practices).
  - Analysis kernel
    - String analyses
    - Hybrid analysis: automatic correlation of static and dynamic results
- **HP Fortify SCA**
  - Gary Mac Graw seven kingdoms
  - Analysis kernel
    - Semantic analyzer detects use of vulnerable functions
    - Data flow analyzer tracks tainted input
    - Control flow analyzer tracks improper sequencing of operations

Detection or proof of absence of errors

Class of errors	Example of errors	Tools
Concurrency	Deadlock, data race conditions	Fluid tools, RacerX, Warlock
Memory	Memory leak, buffer overrun …	Clousot, Sparrow, C Global Surveyor, BoundsChecker, Code Advisor, Coverity
Runtime	Non initialize variable, division by 0	Astrée, PolySpace, Frama-C, Inspector, Lintplus
Numerical	Cancellation	Fluctuat
Dead code	Dead code	PolySpace, STATIC

Detection or proof of absence of errors (cont.)

Constraints	Assertion failure	Astrée, PolySpace, Frama-C
Execution time	Worst case execution time	Ait WCET
Clones	Cut and paste not well used	Bauhaus, CCFinderX, Clone Doctor, AntiCutandPaste
64 –bit	Not portable constructs 32 to 64 bits	Viva64
Dangerous calls	Format string ...	Vulncheck (gcc option)
Protocol	Secrecy properties	CSur

Synthesis

- **Most security tools**
  - Combine several methods
    - Pattern matching and control flow
    - Static analysis and complexity
    - Static analysis and dynamic analysis (test)
- **Tools based on abstract interpretation**
  - Use one technique
  - Prove the absence of
    - Targeted errors (memory errors)
    - Roots of security flaws
  - Give results difficult to interpret

Approach

- **Development of a new tool**
- **Enable users to**
  - Define security objectives
  - Analyze exploitability of security flaws
  - Perform “depth on demand” analysis
- **Techniques**
  - Combine techniques from syntactic to abstract interpretation
    - Data and control flow analysis
    - Value analysis
    - Pointer analysis
  - Combine static and dynamic analysis

Features

- **Detection of common weaknesses**
  - CWE including complex (race conditions, ToCToU]
  - OWASP
- **Analysis of user defined filtering and protection means**
- **Verification of user defined security policies**
  - Access policies
  - Flow control policies
- **Detection of covert/side channels**
- **Exploitability of security flaws**

Carto-C tool

- **Software security audit**
- **Input:**
  - Security objective (security policy)
  - Piece of software (source code)
- **Output:**
  - Arguments to show
    - Insecurity w.r.t. the objective: problems found
    - Security w.r.t. the objective

Use cases *(1)*

- **Audit objective: Valid attack surface**
- **Tool knowledge**
  - Elements of the attack surface are IO functions, C library functions
- **User provided input:**
  - Source code
  - Specification of constraints on the attack surface
    - Overall size
    - Absence of some calls
- **Output:**
  - Actual attack surface
  - Verification of the constraints specified

Use case (2)

- **Objective: Absence of information leak**
- **User provided input:**
  - Source code
  - List of asset names
- **Output:**
  - Accesses to assets (read / write)
  - Impact of assets on output

SAFE RIVER
Safety & Security Forge

Use case (3)

- **Objective: Correctness of asset protections**
- **User provided input:**
  - Source code
  - List of asset names
  - List of asset protection functions
- **Output:**
  - Accesses to assets (read / write)
  - Location of call to protections on the source code
  - Presence/absence of protections on computational flows
  - List of unprotected assets

Use case (4)

- **Objective: Correctness of input/output filtering**
- **User provided input:**
  - Source code
  - List of IO filtering functions (sanitization)
- **Output:**
  - List of input/output channels
  - Location of filtering functions on the source code
  - Presence/absence of filtering on computational flows
    - Input: between input and use
    - Output: between definition and output
  - List of unprotected channels

Example Inputs

Source code :	User provided input :
0: x=getc(fic1); 1: gets(line); 2: system(line); 3: fscanf(fic2,format,precious); 4: y=compute(decrypt(precious),x); 5: z=makefullcomputation(x,y); 6: printf(z);	Asset (precious) Protection(precious,In, decrypt) Filter(system,In, filter_command)

SAFE RIVER
Safety & Security Forge

l'Europe s'engage en Ile-de-France avec le FEDER

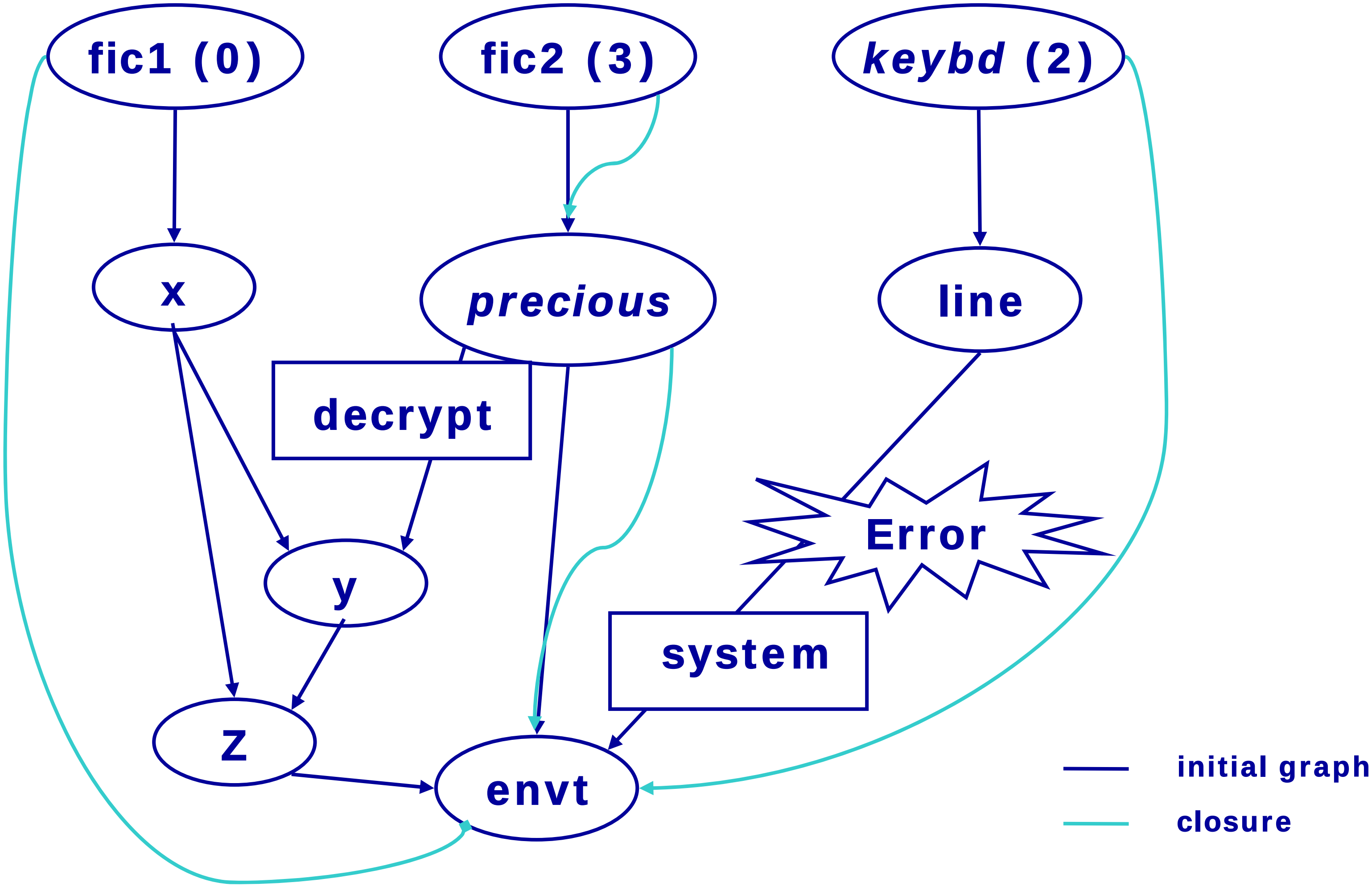
UNION EUROPEENNE

Exemple Output (1)

	Map	Channel	Asset
0	IN(getc)	fic1	
1	IN(gets)	*keyboard*	
2	OUT(system)		
3	IN(fscanf) Occurrence(precious)	fic2	precious
4	Occurrence(precious) Occurrence(decrypt)		precious decrypt
5			
6	OUT(printf)	*environment*	

Example output (2)

Protection(precious,In,decode)
Protection(system,In,filter_command)



Underlying technologies

- **Data-flow, control-flow, abstract interpretation**
- **Exploration of the source code**
  - Computation of IO interfaces
  - Search for occurrences of declared assets
  - Search for protections
- **Dependencies computation**
  - Inter-procedural
  - Aliasing
  - Projected on target paths
  - Closed by target input / output points

Conclusions

- **Carto-C: first implementation over Frama-C**
- **Need for C++, Java languages**
- **Coupling with**
  - Dynamic analysis (FLOID)
  - Binary/bytecode analysis (Binsec)
- **Open to new collaborations**
- **Need of access to user code for a proof of concept**