Secure coding training

Review of source code analyzers

Gerard Frankowski, Tomasz Nowak – PSNC

Poznań, 22-23 June 2010

Contents

- Scan it or not?
- Static source code analyzers
  - For Java:
    - *PMD*
    - *findbugs*
  - For C/C++
    - *RATS*
    - *cppcheck*
  - YASCA
  - For PHP:
    - *Pixy,*
    - *RIPS*

Secure coding training

Scan it or not?

connect • communicate • collaborate

Source code security scanners

- Tools especially designed for detecting security vulnerabilities.
  - May also detect code that does not follow conventions (especially in Java).
  - Will not cover lint etc. here (tools mainly for developers, although may contain security-related warnings).
- The questions for this short introduction:
  - By whom should the tools be used?
  - What are their advantages and drawbacks?
  - Is it easy to use them?

By whom should the tools be used

- In general, these are security, not development tools:
  - That is why we do not recommend you should use them for full security analyses (this is our job).
- Remember they say:
  - A fool with a tool is still a fool ;)
  - We have explicit claims in the team “do a code review, but do not (only) use automated scanners”

Advantages of automated scanners

- They may spare a lot of your time (quickly provide a list of points to look at).
  - Especially for large source code repositories.
- They usually present well structured results – a good starting point for a report.
- May be easily used periodically to detect new flaws
- Many tools are:
  - Free of charge.
  - Ready to use on multiple operating systems (especially Windows and Unix / Linux).

Disadvantages of automated scanners

- They are only tools, not intelligent beings.
  - May detect “well structured” errors (like using a “dangerous” function).
- Generate numerous false positives.
- Sometimes only find a subset of issues.
- May need full buildable sources
  - However, this will not be a problem for you!
- Free tools sometimes:
  - Have less potential.
  - Can be harder to configure.
  - Lack help and / or documentation.
  - Are no longer supported.

To summarize

- We do not recommend full usage of security code scanners by developers:
  - You learn secure programming principles.
  - Knowing them, you are able.
    - *To detect the most obvious errors.*
    - *To find apparent false positives.*
- Use the analyzers only to some extent: detect only the basic patterns:
  - Using dangerous functions (“advanced grep”).
  - Detecting the most apparent data sanitization vulnerabilities
    - *like* **`echo $_GET[‘param’];`**

Discussion?

- What do you think?
- Have you already used security-oriented code analyzers?
- Which ones?
- Do you like them? Or not? Why?
- …

Example resources

- OWASP section on source code analyzers:
  - http://www.owasp.org/index.php/Source_Code_Analysis_Tools
- List of source code security analyzers (both free and commercial tools):
  - http://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html
- Source code analysis tools – an overview :
  - https://buildsecurityin.us-cert.gov/bsi/articles/tools/code/263-BSI.html

Secure coding training
Java security tools – PMD

connect • communicate • collaborate

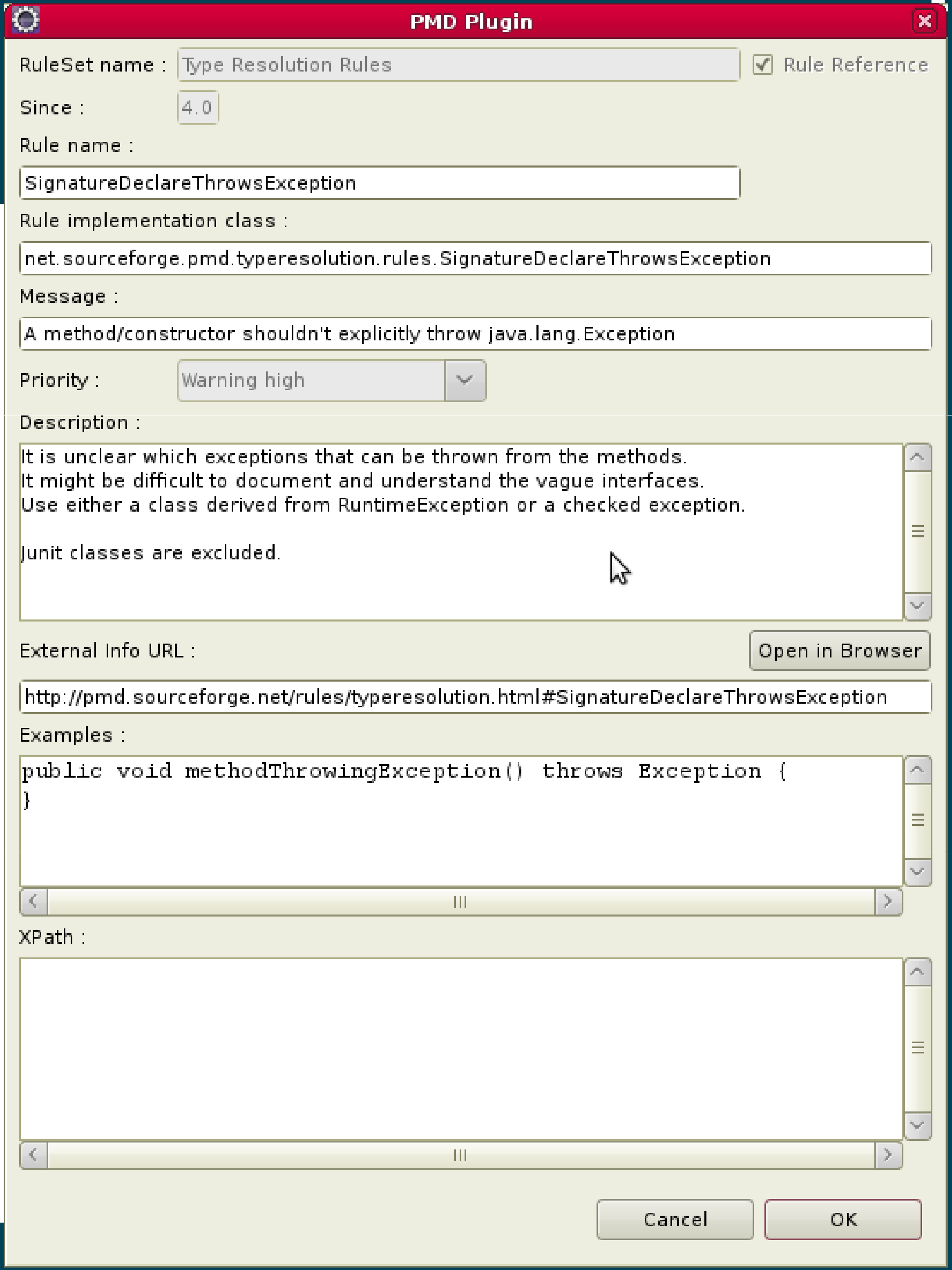
General information

- PMD – Java source code scanner
  - Last version: 4.2.5 (February 2009)
  - Copyright © 2002-2009 InfoEther, Inc.
  - *http://findbugs.sourceforge.net*
  - *BSD-style license*
  - Crossplatform (Java)
  - Contains special module *Copy/Paste Detector* (Java, JSP, C, C++, Fortran and PHP)

Usage

- Running PMD:
  - Download and unpack, or build from sources
  - Generate HTML report, or
  - Use a plugin – many available:
    “JDeveloper, Eclipse, JEdit, JBuilder, BlueJ, CodeGuide, NetBeans/Sun Java Studio Enterprise/Creator, IntelliJ IDEA, TextPad, Maven, Ant, Gel, JCreator, and Emacs”
  - Plugins for IDEs, e.g. Eclipse and NetBeans
  - Ant task or maven report plugin

PMD screenshot (DEMO)



RuleSet name : Type Resolution Rules ✓ Rule Reference

Since : 4.0

Rule name :

SignatureDeclareThrowsException

Rule implementation class :

net.sourceforge.pmd.typeresolution.rules.SignatureDeclareThrowsException

Message :

A method/constructor shouldn't explicitly throw java.lang.Exception

Priority : Warning high

Description :

It is unclear which exceptions that can be thrown from the methods.
It might be difficult to document and understand the vague interfaces.
Use either a class derived from RuntimeException or a checked exception.

Junit classes are excluded.

External Info URL : Open in Browser

http://pmd.sourceforge.net/rules/typeresolution.html#SignatureDeclareThrowsException

Examples :

```
public void methodThrowingException() throws Exception {
}
```

XPath :

Problems looked for

- Tens of sets at http://pmd.sourceforge.net/rules
  - You may want to disable or suppress warnings
- Most findings are not serious bugs
- Eclipse can fix many warnings – use „Clean up”

- Special module – Copy/Paste Detector
  - – Support for many languages
  - – Two times rewritten (algorithm changed)
  - – Sample results http://pmd.sourceforge.net/cpdresults.txt

Our opinion

- Good rulesets for „style” problems
- Numerous unimportant warnings
- Effort needed to filter out unnecessary alarms
- Integration with huge number of editors and building mechanisms
- Unique CPD functionality allows fixing maintainability problems
- Undergoing heavy refactorization – version 5

Hints for the developers

- Choose right rulesets – sorting through a thousand line report to find the few violations you're really interested in takes all the fun out of things
- Start with some of the obvious rulesets (unusedcode, basic) and then take the more controversial ones
- PMD rules are not set in stone – pick the ones you need, ignore or suppress others
- Use PMD IDE plugins to easily jump through the code

http://pmd.sourceforge.net/bestpractices.html

Example resources

- Tool documentation:
  - http://pmd.sourceforge.net/
  - http://pmd.sourceforge.net/cpd.html

Secure coding training

Java security tools – findbugs

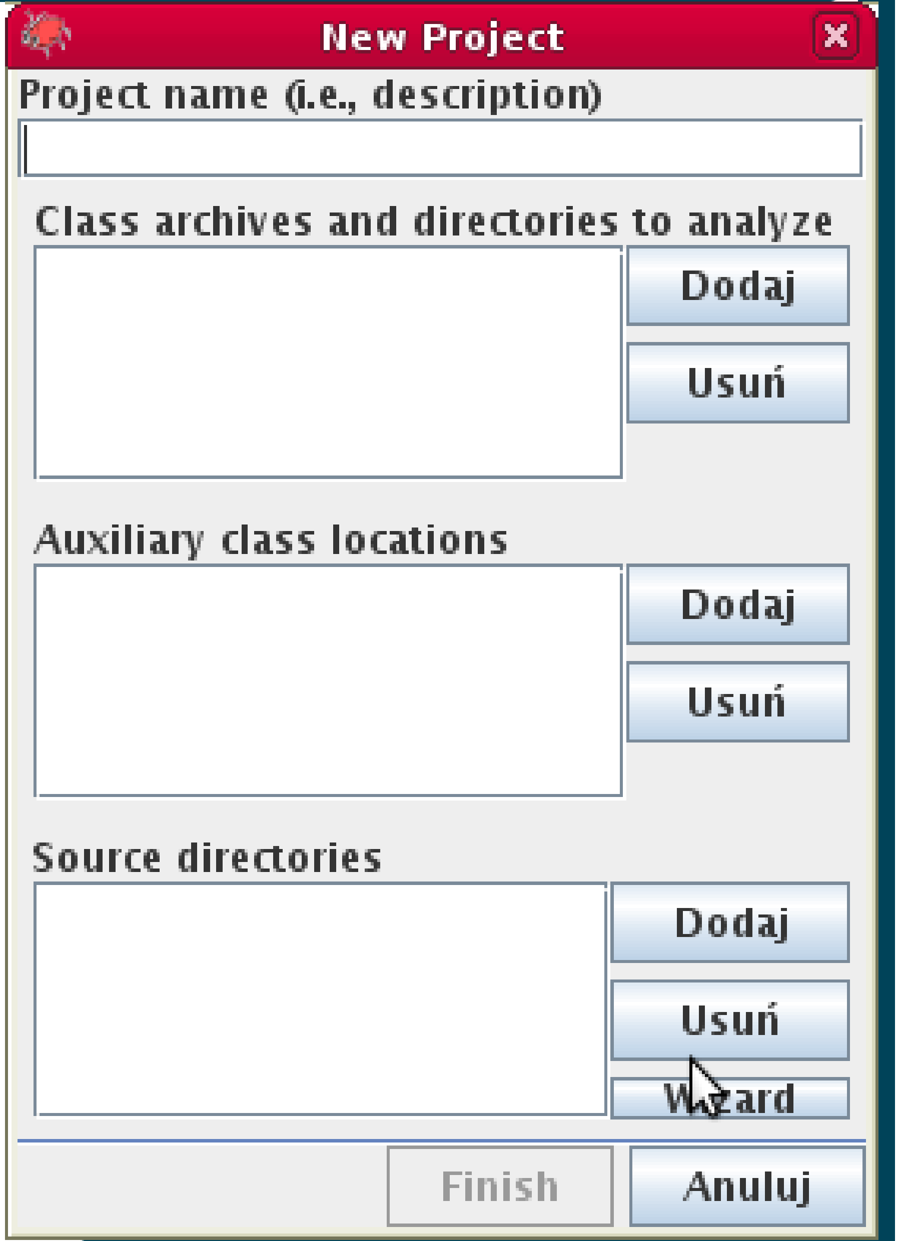
General information

- FindBugs – Java bytecode scanner
  - Last version: 1.3.9 (August 2009).
  - Trademarked by The University of Maryland
  - http://findbugs.sourceforge.net/
  - *GNU Lesser General Public License*
  - Crossplatform (Java)
  - Starts also from browser (WebStart)
  - 369 bug patterns
    - *Categories: CORRECTNESS, MT_CORRECTNESS, BAD_PRACTICICE, PERFORMANCE, STYLE*
    - *Priorities: 1 (high) to 3 (low)*
  - XML output and data mining

Usage

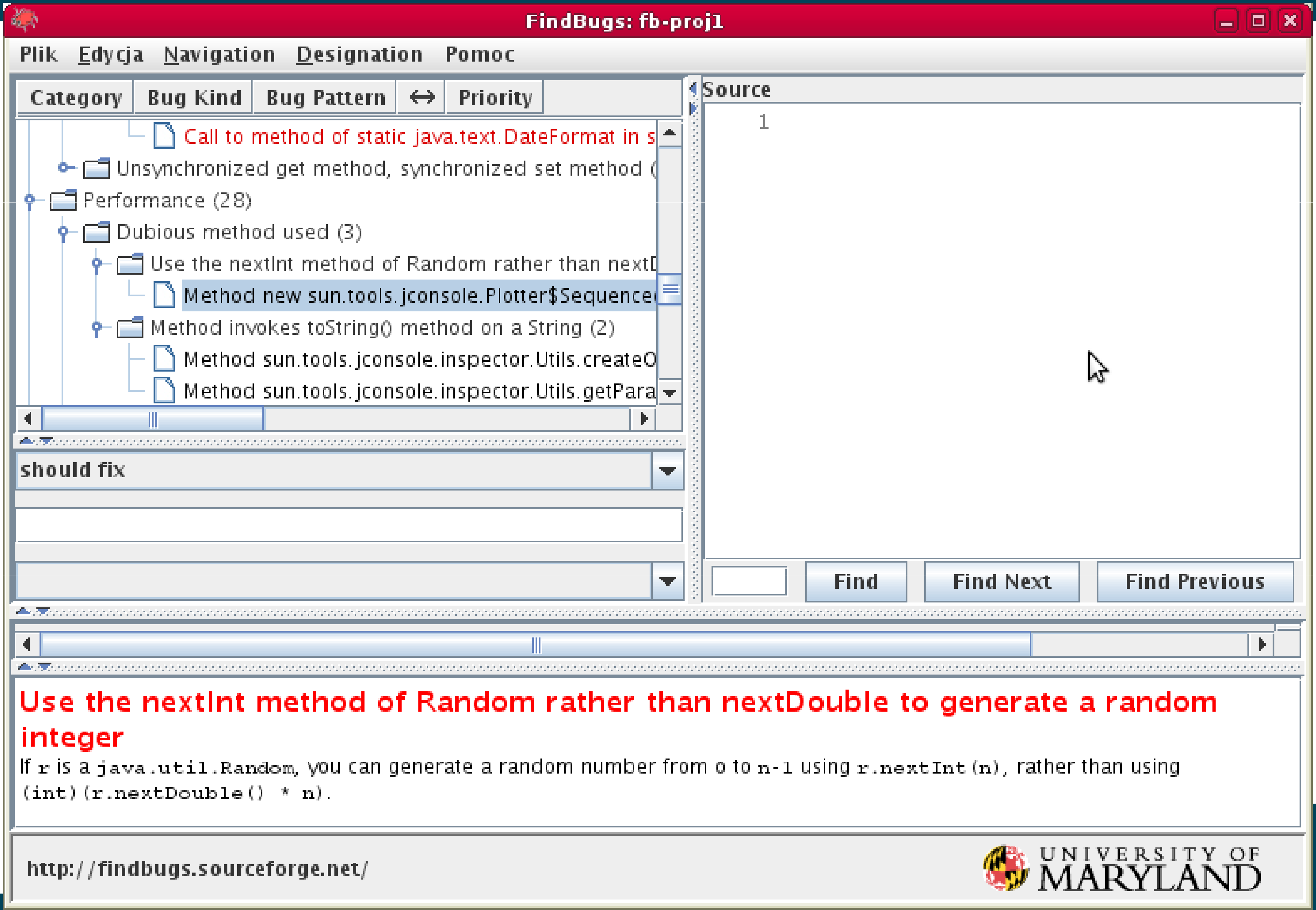
- Running FindBugs:
  - Download & run with **`java -jar`**
    - *binary (zip)*
    - *build from sources with Ant*
  - Plugins for IDEs, e.g. Eclipse and NetBeans
  - Ant task or maven report plugin
  - WebStart from findbugs.cs.umd.edu/demo/

- Additional rules in http://fb-contrib.sourceforge.net/


New Project
Project name (i.e., description)
Class archives and directories to analyze
Dodaj
Usuń
Auxiliary class locations
Dodaj
Usuń
Source directories
Dodaj
Usuń
Wizard
Finish
Anuluj

Quick Polish course: Dodaj = Add, Usuń = Remove

FindBugs screenshot


FindBugs: fb-proj1
Plik Edycja Navigation Designation Pomoc
Category Bug Kind Bug Pattern ↔ Priority
Source
should fix
Find Find Next Find Previous

Call to method of static java.text.DateFormat in s
Unsynchronized get method, synchronized set method (
Performance (28)
Dubious method used (3)
Use the nextInt method of Random rather than nextD
Method new sun.tools.jconsole.Plotter$Sequence
Method invokes toString() method on a String (2)
Method sun.tools.jconsole.inspector.Utils.createO
Method sun.tools.jconsole.inspector.Utils.getPara

Use the nextInt method of Random rather than nextDouble to generate a random integer

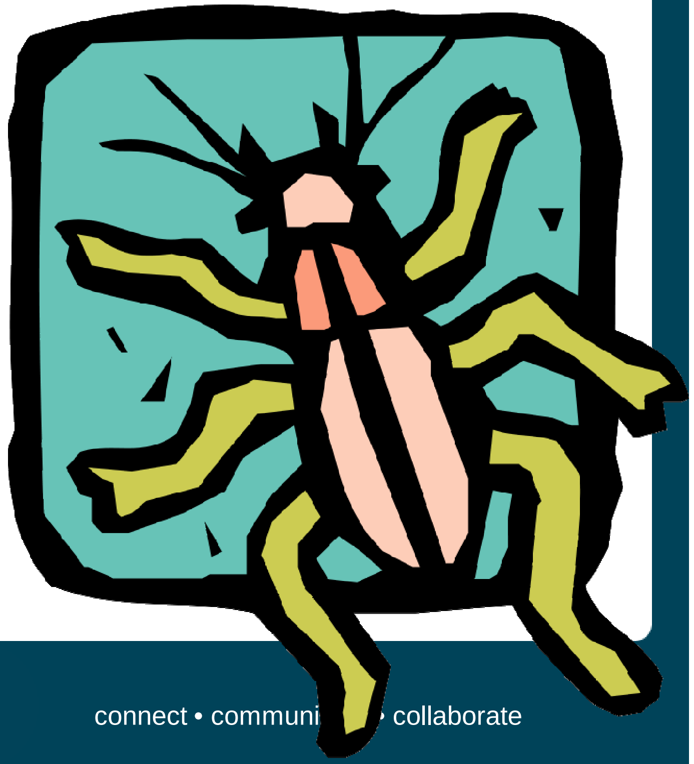
If `r` is a `java.util.Random`, you can generate a random number from `0` to `n-1` using `r.nextInt(n)`, rather than using `(int)(r.nextDouble() * n)`.

http://findbugs.sourceforge.net/

UNIVERSITY OF MARYLAND

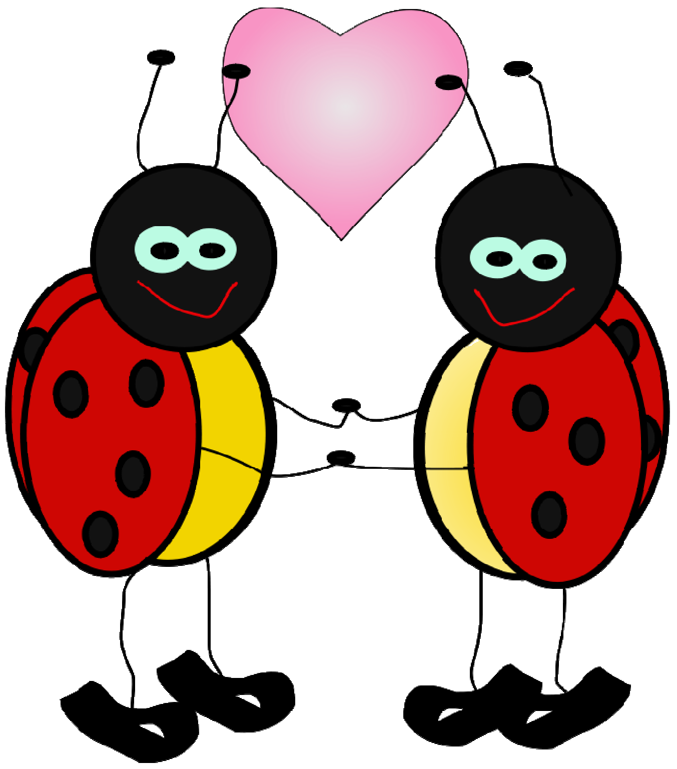
Findings

- Some findings are brilliant – so much to learn!
  - Returning mutable fields
  - Inner classes which could be static
  - Not fulfilled contracts, e.g. clone returning null
- A lot of performance remarks
  - Concatenation of strings with + in a loop
  - toString( ) called on strings
  - Unnecessary or missing null checks
- Inconsistent design
  - Protected fields in final classes
  - Missing synchronization
- Security bad practices
  - Hardcoded or empty db password
  - XSS and SQL injection discovery

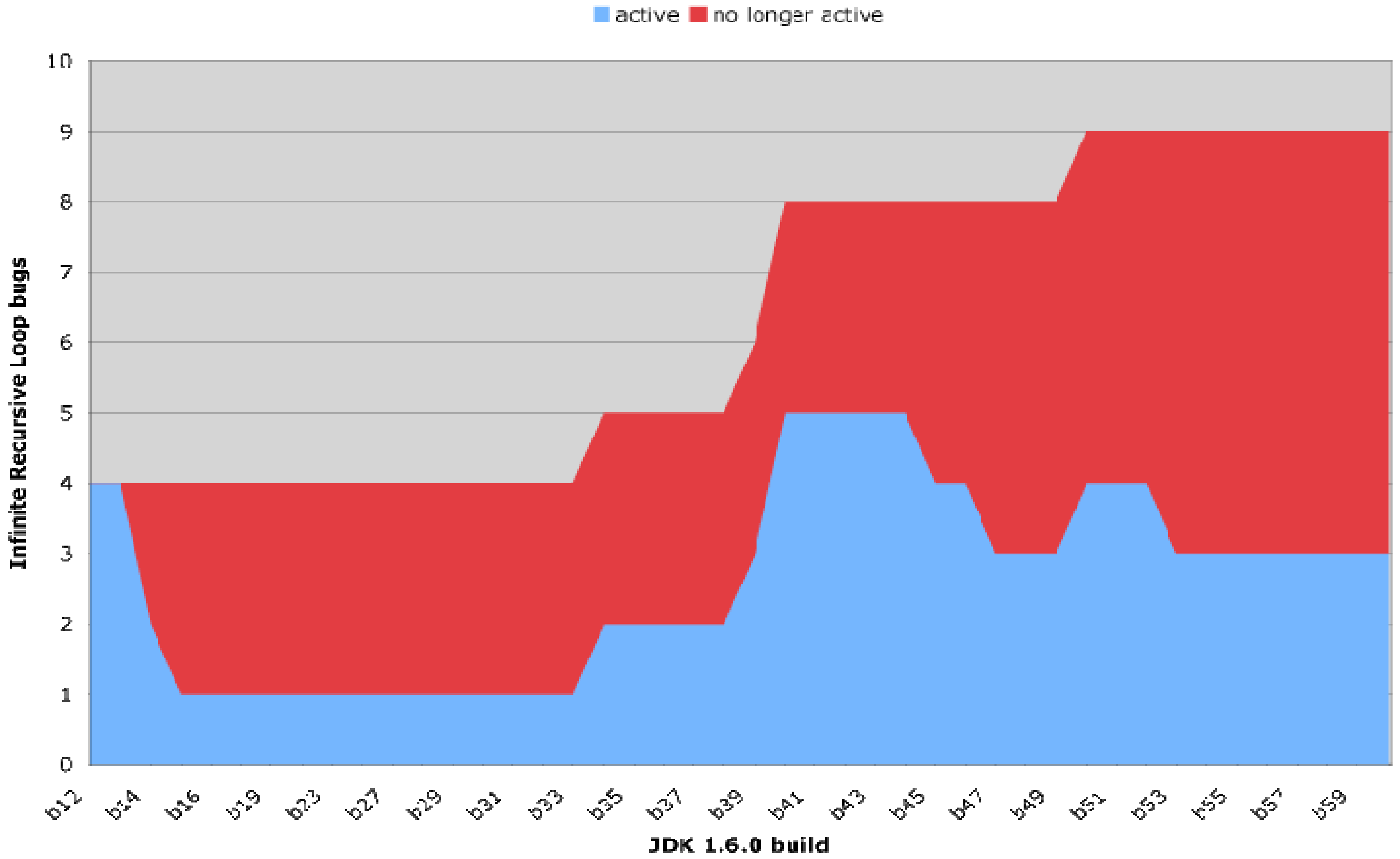


Results

- Bugs are grouped by various criteria
  - Categories
  - Priorities
  - Packages
- Source fragment displayed (if available)
- Detailed description of the finding
- Can set “designation” for every bug:
  - Needs further study
  - Not a bug
  - Mostly harmless
  - Should fix
  - Must fix
  - Bad analysis
  - Unclassified

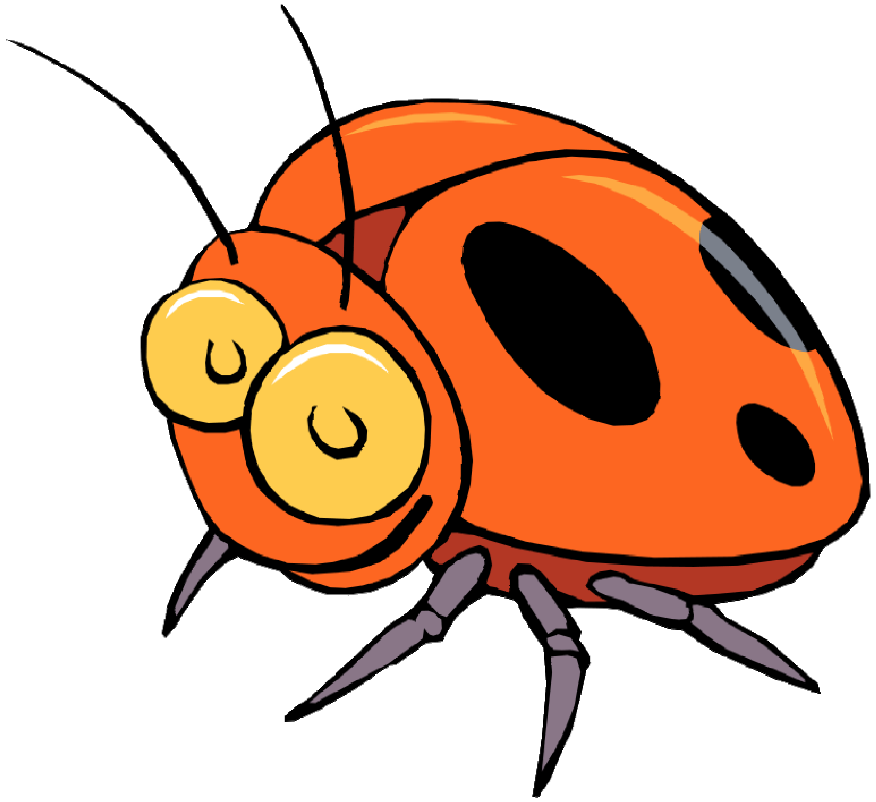


XML export & data mining capabilities
Visualization of bug databases



Our opinion about FindBugs

- Very easy to run and use.
- Rare false positives.
- Good explanation of found issues.
- Unique quality monitoring mechanisms.
- Up to date – still developed and extra rules available.



Hints for the developers

- FindBugs operates on bytecode, but attaching sources too comes very handy
- Provide dependent classes to get more detailed analysis
- If a description is not clear, google for its bug code

Example resources

- Bug descriptions:
  - http://findbugs.sourceforge.net/bugDescriptions.html
  - http://fb-contrib.sourceforge.net/bugdescriptions.html
- FindBugs on Google Code
  - http://code.google.com/p/findbugs

Secure coding training
Code scanners for C/C++: RATS

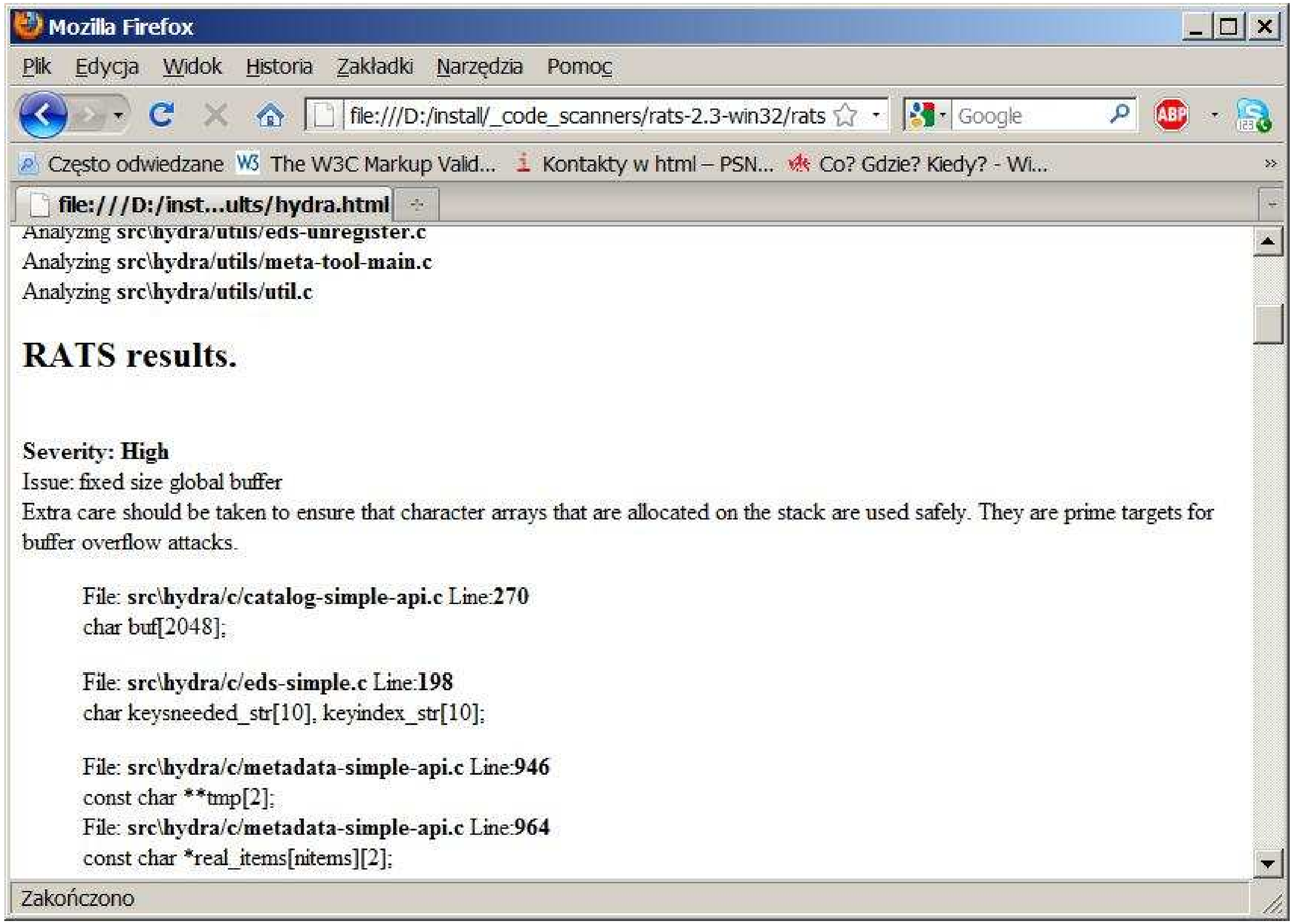
RATS - introducion

- RATS: Rough Auditing Tool for Security
  - Latest release: 2.3
    - *Seems not to be developed actively*
  - Made by Fortify Software
  - http://www.fortifysoftware.com/security-resources/rats.jsp
  - GNU Public License
  - Systems: Unix/Linux, Windows
  - Requires Expat parser (http://expat.sourceforge.net)
  - Languages: C, C++, Perl, PHP, Python
  - Vulnerabilities: including buffer overflows, TOCTOU (race conditions), Remote Code Execution, shows dangerous functions)

Usage

- Invoke the tool from the command line
  - rats [-d] [-h] [-r] [-w <1,2,3>] [-x] [file1 file2 ... fileN]
  - rats –h (or –help) gives more information
- We use RATS usually as follows:
  - All source files are copied to *src* subdirectory
    - *RATS uses recursion in source directories by default*
  - rats -w3 --html --context src > results\rats3.html
    - *w3 – maximum warning level*
    - *--html – output in HTML format*
    - *--context – display the problematic line*
    - *Redirection of the results to a file*
  - We do not use language specification, RATS is clever enough to detect it itself

Sample result for a C application


Mozilla Firefox
Plik Edycja Widok Historia Zakładki Narzędzia Pomoc
file:///D:/install/_code_scanners/rats-2.3-win32/rats
Google
Często odwiedzane The W3C Markup Valid... Kontakty w html – PSN... Co? Gdzie? Kiedy? - Wi...
file:///D:/inst...ults/hydra.html
Zakończono

Analyzing **src\hydra/utils/eds-unregister.c**
Analyzing **src\hydra/utils/meta-tool-main.c**
Analyzing **src\hydra/utils/util.c**

RATS results.

Severity: High
Issue: fixed size global buffer
Extra care should be taken to ensure that character arrays that are allocated on the stack are used safely. They are prime targets for buffer overflow attacks.

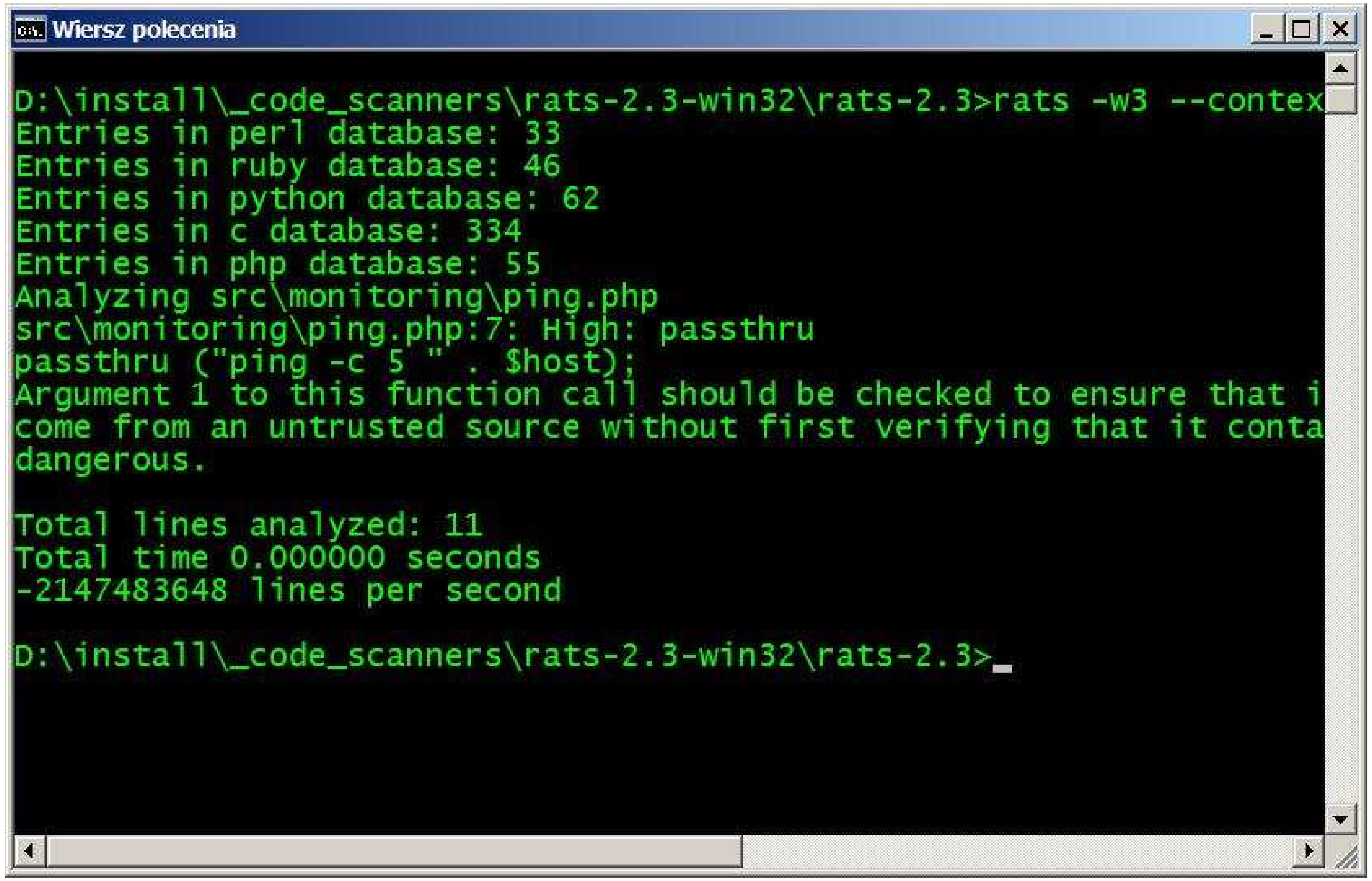
File: **src\hydra/c/catalog-simple-api.c** Line:**270**
char buf[2048];

File: **src\hydra/c/eds-simple.c** Line:**198**
char keysneeded_str[10], keyindex_str[10];

File: **src\hydra/c/metadata-simple-api.c** Line:**946**
const char **tmp[2];
File: **src\hydra/c/metadata-simple-api.c** Line:**964**
const char *real_items[nitems][2];

Sample result for a PHP application

- A short PHP file containing *passthru()* call



```
D:\install\_code_scanners\rats-2.3-win32\rats-2.3>rats -w3 --contex
Entries in perl database: 33
Entries in ruby database: 46
Entries in python database: 62
Entries in c database: 334
Entries in php database: 55
Analyzing src\monitoring\ping.php
src\monitoring\ping.php:7: High: passthru
passthru ("ping -c 5 " . $host);
Argument 1 to this function call should be checked to ensure that i
come from an untrusted source without first verifying that it conta
dangerous.

Total lines analyzed: 11
Total time 0.000000 seconds
-2147483648 lines per second

D:\install\_code_scanners\rats-2.3-win32\rats-2.3>_
```

Our opinion and advices

- RATS would be good for you at emphasizing:
  - Dangerous functions
  - TOCTOU issues
  - Fixed size buffers
- Many false positives (like other tools)
- Sufficient reporting facilities
- Works fast, but sometimes crashes…
  - Try to change e.g. warning level or output format then, may help
- You could run it on the 1 or 2 warning level to avoid being informed about complicated stuff

Secure coding training

Code scanners for C/C++: cppcheck

connect • communicate • collaborate

cppcheck – introduction

- C/C++ source code scanner
  - Latest version: 1.43 (May 2010)
  - http://cppcheck.wiki.sourceforge.net
  - GNU GPL license
  - Command line mode + GUI mode
  - Systems: at least cmd line mode should work on all
    - *Available as .msi for Windows*
  - Languages: C/C++
  - Vulnerabilities: bounds checking, variable range, memory leaks, NULL pointer dereference, many others
- The community goal: no false positives

Usage

- Command line usage:

  cppcheck [--all] [--auto-dealloc file.lst] [--error-exitcode=[n]] [--force][--help] [-Idir] [-j [jobs]] [--quiet] [--style] [--unused-functions][--verbose] [--version] [--xml] [file or path1] [file or path] ...

- The result is sent to the standard output by default, so we recommend to redirect it to a file
  - The output may be customized through XSLT

Our favourite cppcheck options:

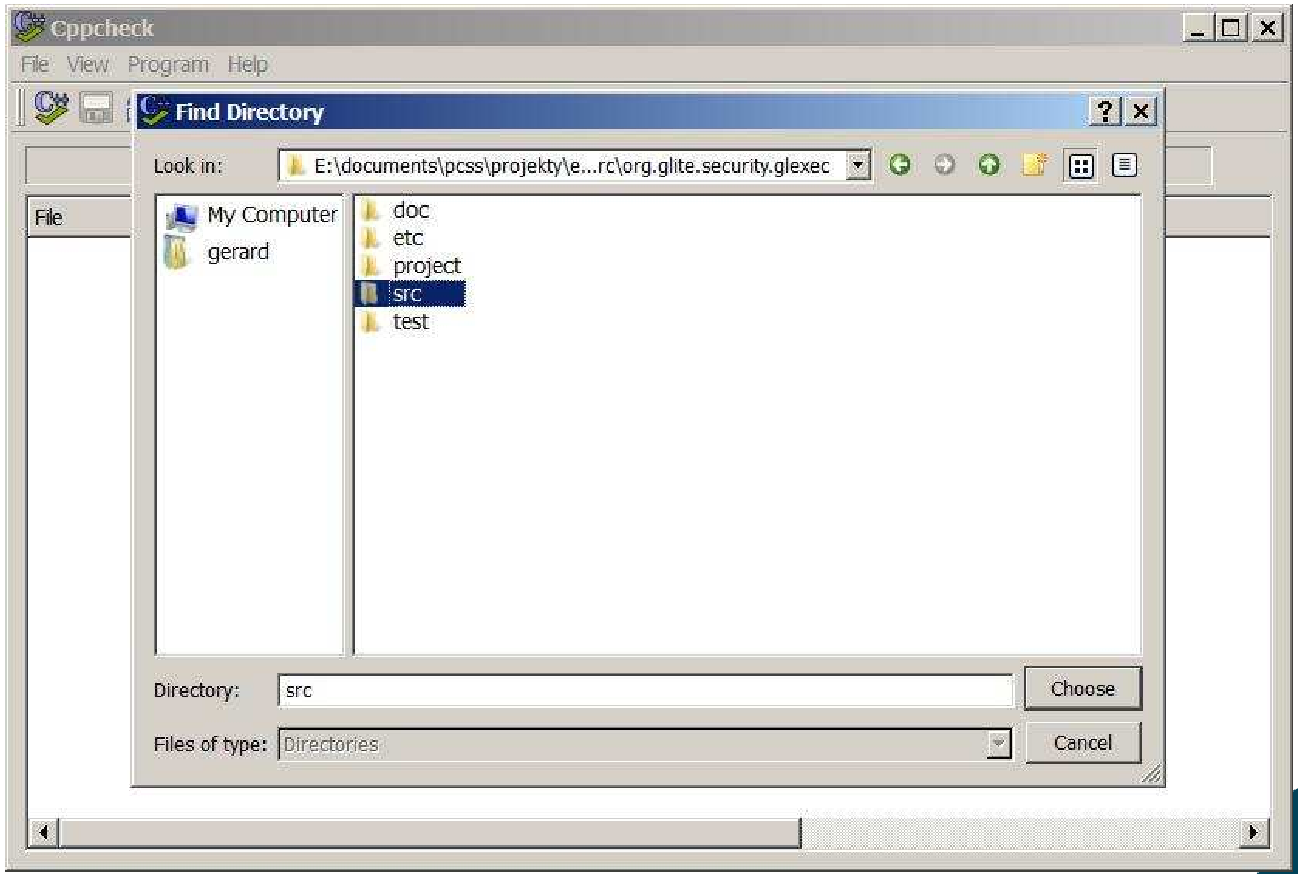
- We use it usually in the following way:

  cppcheck -a -s -v --unused-functions [src_path] > result.txt

  - a (= --all) – more checks, but also more false positives
  - s (= --style) – check coding style
  - v (= --verbose) – more detailed error reports
  - --unused-functions – detect functions that are unused
- You may omit –a switch to avoid sophisticated analysis
- Adjust report verbosity as you wish
  - If too much seems to be out of your area of interest, switch it off

GUI usage

- Select directory with source code:
  - File | Check directory | Choose
  - Please note that cppcheck starts to work at once!

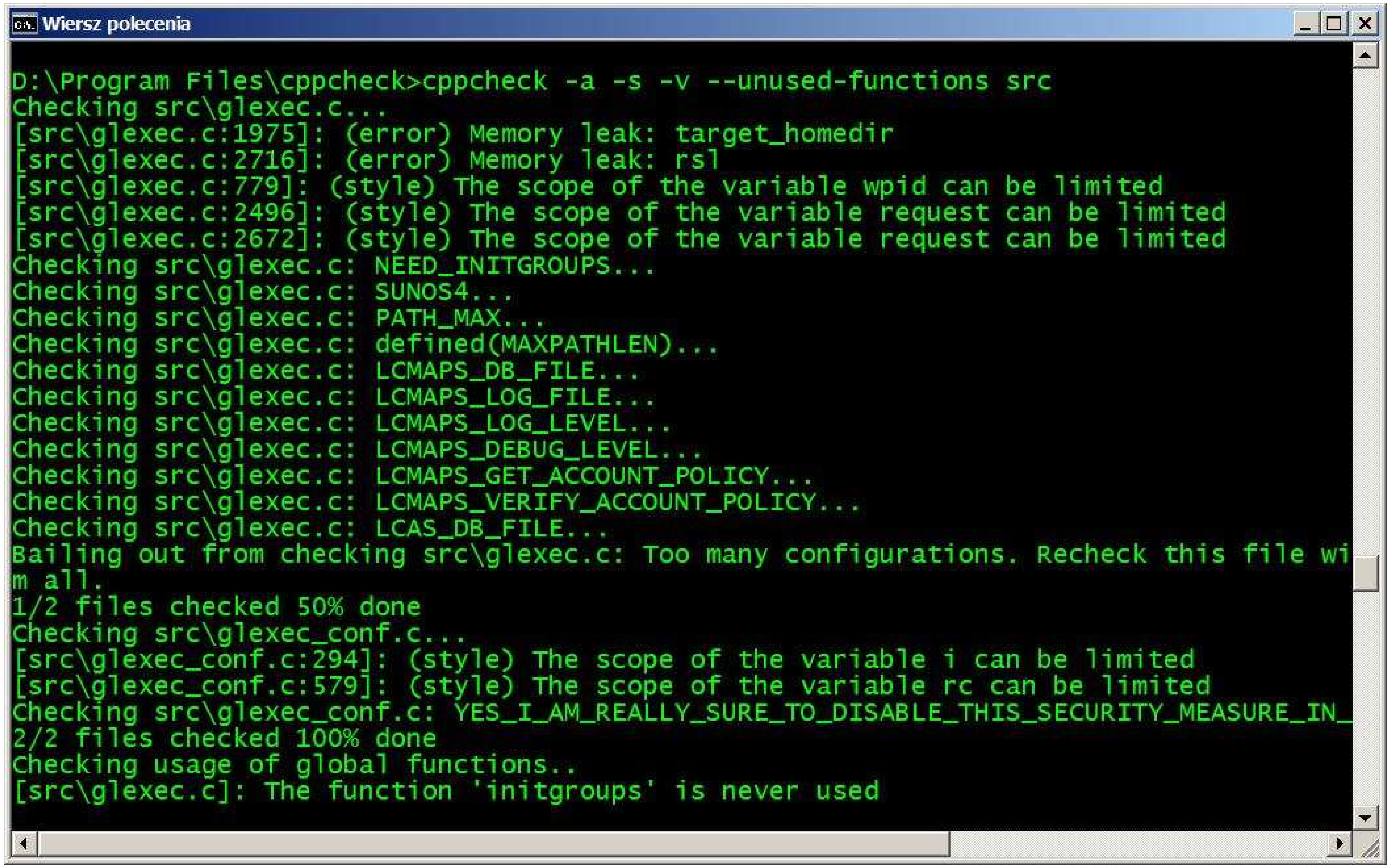


Look in: E:\documents\pcss\projekty\e...rc\org.glite.security.glexec

Directory: src

Files of type: Directories

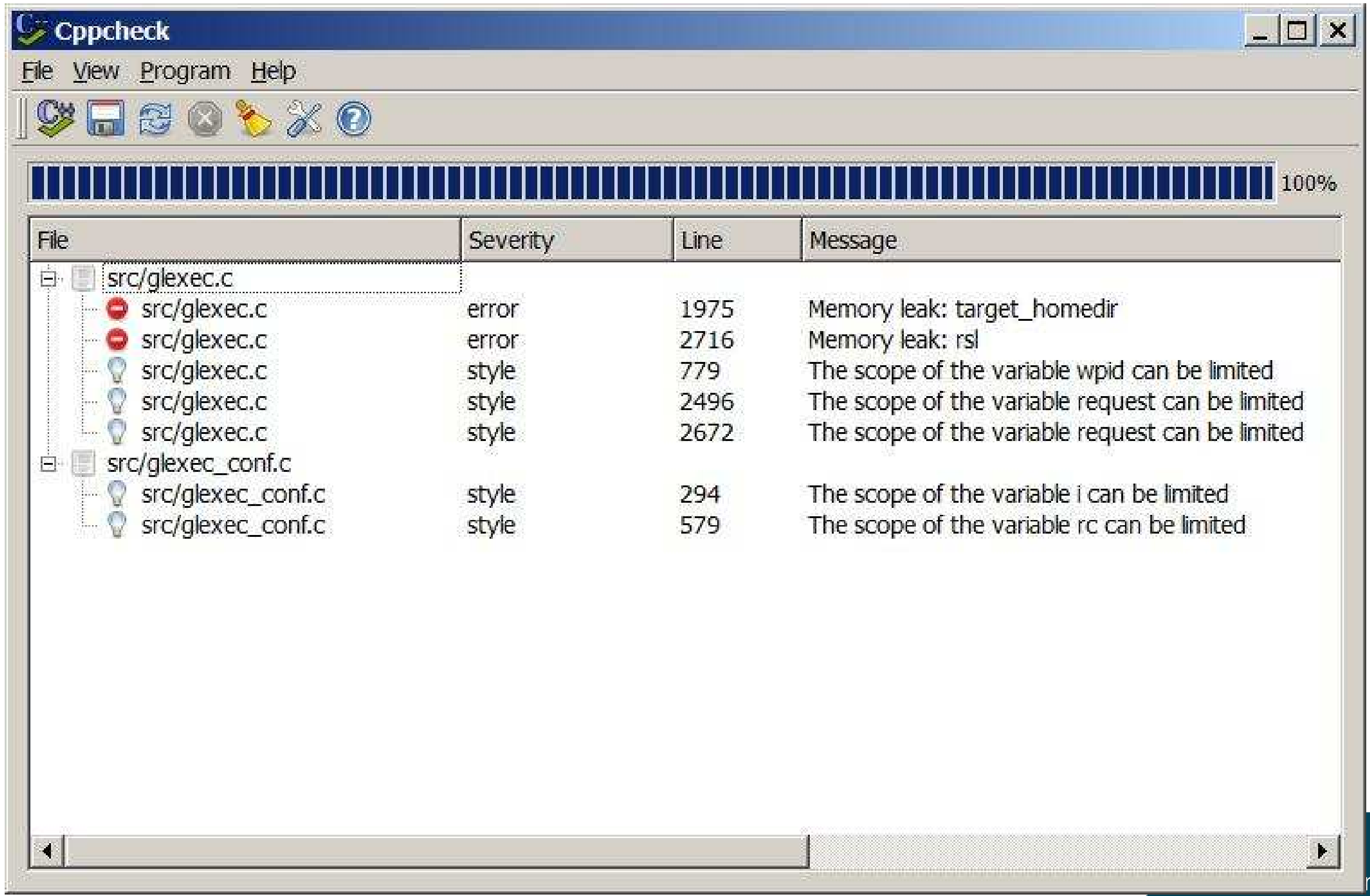
Command line results



```
D:\Program Files\cppcheck>cppcheck -a -s -v --unused-functions src
Checking src\glexec.c...
[src\glexec.c:1975]: (error) Memory leak: target_homedir
[src\glexec.c:2716]: (error) Memory leak: rsl
[src\glexec.c:779]: (style) The scope of the variable wpid can be limited
[src\glexec.c:2496]: (style) The scope of the variable request can be limited
[src\glexec.c:2672]: (style) The scope of the variable request can be limited
Checking src\glexec.c: NEED_INITGROUPS...
Checking src\glexec.c: SUNOS4...
Checking src\glexec.c: PATH_MAX...
Checking src\glexec.c: defined(MAXPATHLEN)...
Checking src\glexec.c: LCMAPS_DB_FILE...
Checking src\glexec.c: LCMAPS_LOG_FILE...
Checking src\glexec.c: LCMAPS_LOG_LEVEL...
Checking src\glexec.c: LCMAPS_DEBUG_LEVEL...
Checking src\glexec.c: LCMAPS_GET_ACCOUNT_POLICY...
Checking src\glexec.c: LCMAPS_VERIFY_ACCOUNT_POLICY...
Checking src\glexec.c: LCAS_DB_FILE...
Bailing out from checking src\glexec.c: Too many configurations. Recheck this file wi
m all.
1/2 files checked 50% done
Checking src\glexec_conf.c...
[src\glexec_conf.c:294]: (style) The scope of the variable i can be limited
[src\glexec_conf.c:579]: (style) The scope of the variable rc can be limited
Checking src\glexec_conf.c: YES_I_AM_REALLY_SURE_TO_DISABLE_THIS_SECURITY_MEASURE_IN_
2/2 files checked 100% done
Checking usage of global functions..
[src\glexec.c]: The function 'initgroups' is never used
```

GUI results

- May be saved to an XML or a TXT file
- External application may be configured as code viewer



File	Severity	Line	Message
src/glexec.c			
src/glexec.c	error	1975	Memory leak: target_homedir
src/glexec.c	error	2716	Memory leak: rsl
src/glexec.c	style	779	The scope of the variable wpid can be limited
src/glexec.c	style	2496	The scope of the variable request can be limited
src/glexec.c	style	2672	The scope of the variable request can be limited
src/glexec_conf.c			
src/glexec_conf.c	style	294	The scope of the variable i can be limited
src/glexec_conf.c	style	579	The scope of the variable rc can be limited

Our opinion or advice

- Although GUI mode has got Settings page, the command line mode is much easier to customize
- Very little false positives indeed
  - However the tool seems not to detect everything it should
  - Good for you (no complicated stuff reported)
- The tests take relatively much time
- Fine reporting facilities, although customizing the reports requires your own effort
  - But fine that this is possible at all!
- Our advice to the developers
  - Rescan your code as a complement to other measures, it is possible that several bugs will be easily found

More resources

- Flawfinder – another famous tool not described here
  - Made by David A. Wheeler
  - http://www.dwheeler.com/flawfinder, documentation at http://www.dwheeler.com/flawfinder/flawfinder.pdf
  - The page contains also a list of other scanners with links and short descriptions
  - Latest release: 1.27 (January 2007)
  - Released under GPL license
  - Designed for Unix/Linux, but you may use with Cygwin for Windows
  - Requires Python 1.5
  - Programming languages: C/C++

Secure coding training

YASCA – Yet Another Source Code Analyzer

YASCA

- *Stands for "Yet Another Source Code Analyzer."*
- *An open source program which looks for* ***security vulnerabilities****,* ***code-quality****,* ***performance****, and* ***conformance to best practices*** *in program source code, integrating with other open-source tools as needed.*
- *http://www.yasca.org/*
- Main functionality:
  - Aggregating results from other analyzers.
  - Automated grepping.
- Reports in HTML, CSV, XML, MySQL, SQLite, and other formats

Supported languages

- Java
- C/C++
- .NET(VB.NET, C#, ASP.NET)
- PHP
- ColdFusion
- COBOL
- HTML
- JavaScript
- CSS
- Visual Basic
- ASP
- Python
- Perl
- Raw HTTP Traffic

Architecture

- Written in PHP.
- Plugins contain external utilities:
  - FindBugs, PMD, JLint, JavaScript Lint, PHPLint, Cppcheck, ClamAV, Pixy, and RATS.
- Appropriate utilities are executed and results consolidated.
- YASCA itself contains many interesting rules.

YASCA
Grep rules

- yasca
  - doc
  - etc
  - lib
  - plugins
    - *ClamAV.php*
    - *CppCheck.php*
    - ***default***
      - ***grep***
        - ***C***
        - ***PHP***
        - ***Java***
      - ***pmd***
      - ***...***
    - ...
  - resources

YASCA
Sample grep rule

- yasca-core/plugins/default/grep/Injection.XSS.PHP.grep

```
name = Cross Site Scripting via concatenation from source in PHP
file_type = PHP
grep =  /(print|echo).*?\.*\s*\$_(POST|GET)\[.*?\]\.*\s*/
category = Cross-Site Scripting
severity = 1
category_link = http://www.owasp.org/index.php/Cross_Site_Scripting
description = (...)
```

Secure coding training

PHP security tools – Pixy

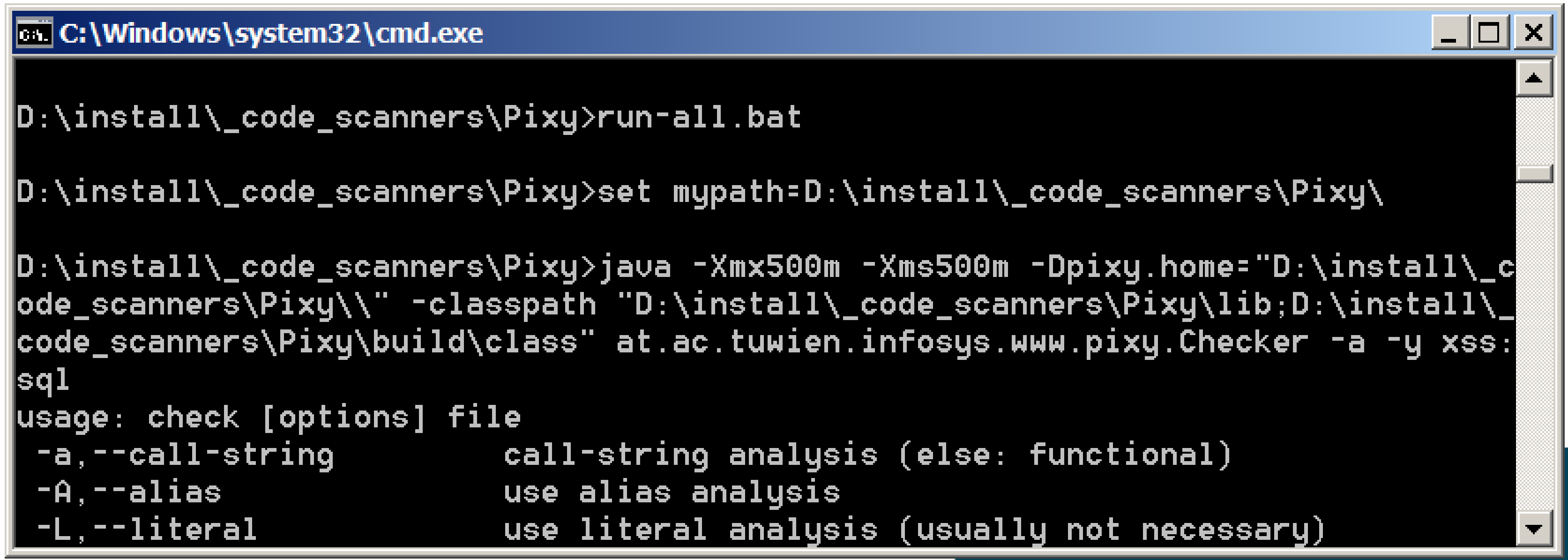
connect • communicate • collaborate

General information

- Pixy – PHP source code scanner :
  - Last version: 3.03 (July 2007).
  - Made by Secure Systems Lab, Vienna University of Technology.
  - http://pixybox.seclab.tuwien.ac.at/pixy.
  - *Freeware.*
  - Systems: Unix/Linux, Windows.
  - Requires Sun Java Runtime Environment.
  - Requires dotty tool for result analysis (Graphviz package – http://www.graphwiz.org).
  - Languages: PHP 4 (more general: not class-oriented).
  - Vulnerabilities: XSS, SQL Injection.

Usage

- Pixy takes a single PHP file as input.
  - For scanning real applications, we encourage to prepare appropriate scripts .
  - Run the following command in the installation directory **`run_all [options] [file]`.**
- Running with no parameters will show help.



C:\Windows\system32\cmd.exe

```
D:\install\_code_scanners\Pixy>run-all.bat

D:\install\_code_scanners\Pixy>set mypath=D:\install\_code_scanners\Pixy\

D:\install\_code_scanners\Pixy>java -Xmx500m -Xms500m -Dpixy.home="D:\install\_c
ode_scanners\Pixy\\" -classpath "D:\install\_code_scanners\Pixy\lib;D:\install\_
code_scanners\Pixy\build\class" at.ac.tuwien.infosys.www.pixy.Checker -a -y xss:
sql
usage: check [options] file
 -a,--call-string              call-string analysis (else: functional)
 -A,--alias                    use alias analysis
 -L,--literal                  use literal analysis (usually not necessary)
```

The results

- Status information is sent to stdout.
  - You may want to redirect.
- Vulnerability information is sent to *graphs* subdirectory.
- The vulnerability graphs should be reviewed by *dotty* tool.
- The Documentation page contains a tutorial about how to understand the results:
  - http://pixybox.seclab.tuwien.ac.at/pixy/documentation.php

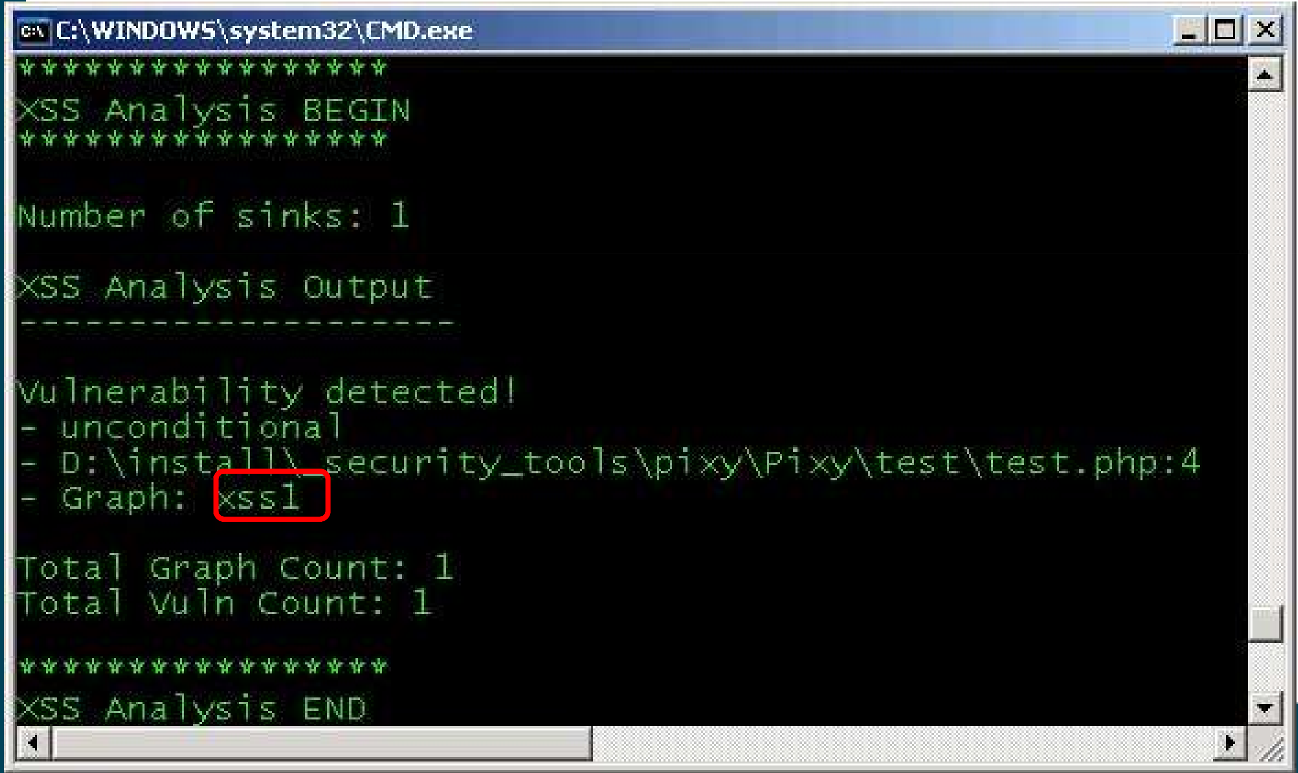
The results – vulnerability information

- List of files that refer to the file
  - calledby_[filename].txt
- List of includes for the file
  - includes_[filename].txt
- Data flow graphs for found XSS vulnerabilities
  - xss_[filename]_[n]_dep.dot
  - **xss_[filename]_[n]_min.dot**
- Data flow graphs for found SQL Injection vulns
  - sql_[filename]_[n]_dep.dot
  - **sql_[filename]_[n]_min.dot**
- The files marked with bold font should be analyzed (contain simplified graphs)

Example – a short demo

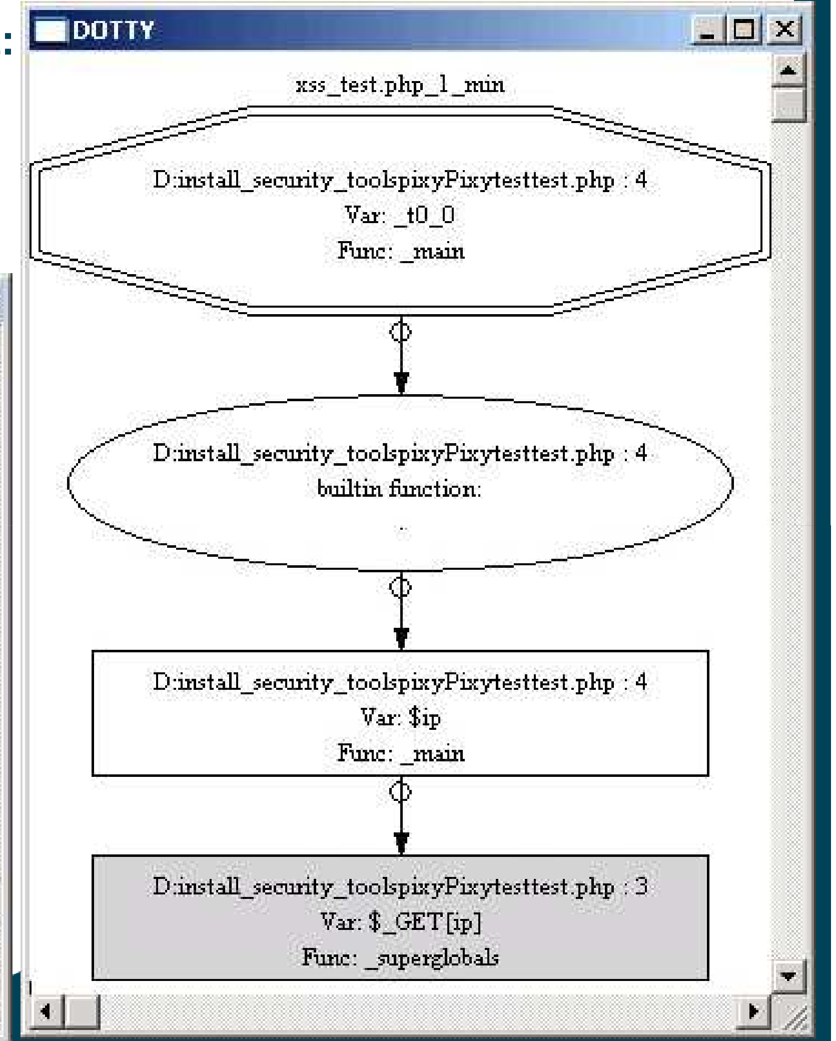
- Vulnerable ping.php file
  - Remembered from Remote Code Execution talk

dotty: xss_test.php_1_min.dot:



C:\WINDOWS\system32\CMD.exe

```
******************
XSS Analysis BEGIN
******************

Number of sinks: 1

XSS Analysis Output
--------------------

Vulnerability detected!
- unconditional
- D:\install\_security_tools\pixy\Pixy\test\test.php:4
- Graph: xss1

Total Graph Count: 1
Total Vuln Count: 1

******************
XSS Analysis END
```


DOTTY
xss_test.php_1_min
D:install_security_toolspixyPixytesttest.php : 4
Var: _t0_0
Func: _main
D:install_security_toolspixyPixytesttest.php : 4
builtin function:
.
D:install_security_toolspixyPixytesttest.php : 4
Var: $ip
Func: _main
D:install_security_toolspixyPixytesttest.php : 3
Var: $_GET[ip]
Func: _superglobals

Our opinion

- An interesting approach.
- Numerous false positives.
- Effort needed to filter out unnecessary alarms, but those remaining spare a lot of work – especially for large sites.
- Relatively complicated result analysis .
- Inability to work with object-oriented PHP 5.x is a significant disadvantage.
- Seems that development has ceased.

Hints for the developers

- Find the simplest graphs (.dot files are actually simple text files, so appropriate tools may be easily developed (look for files with only a few items).
- Look at the last item (where the malicious data may be introduced?) and the top one (where it is displayed?)

Example resources

- Tool documentation:
  - http://pixybox.seclab.tuwien.ac.at/pixy/documentation.php
- Conference papers & reports:
  - http://www.seclab.tuwien.ac.at/papers/pixy.pdf
  - http://www.seclab.tuwien.ac.at/papers/pixy_techreport.pdf
  - http://www.seclab.tuwien.ac.at/papers/pixy2.pdf

Secure coding training

PHP security tools – RIPS

General information

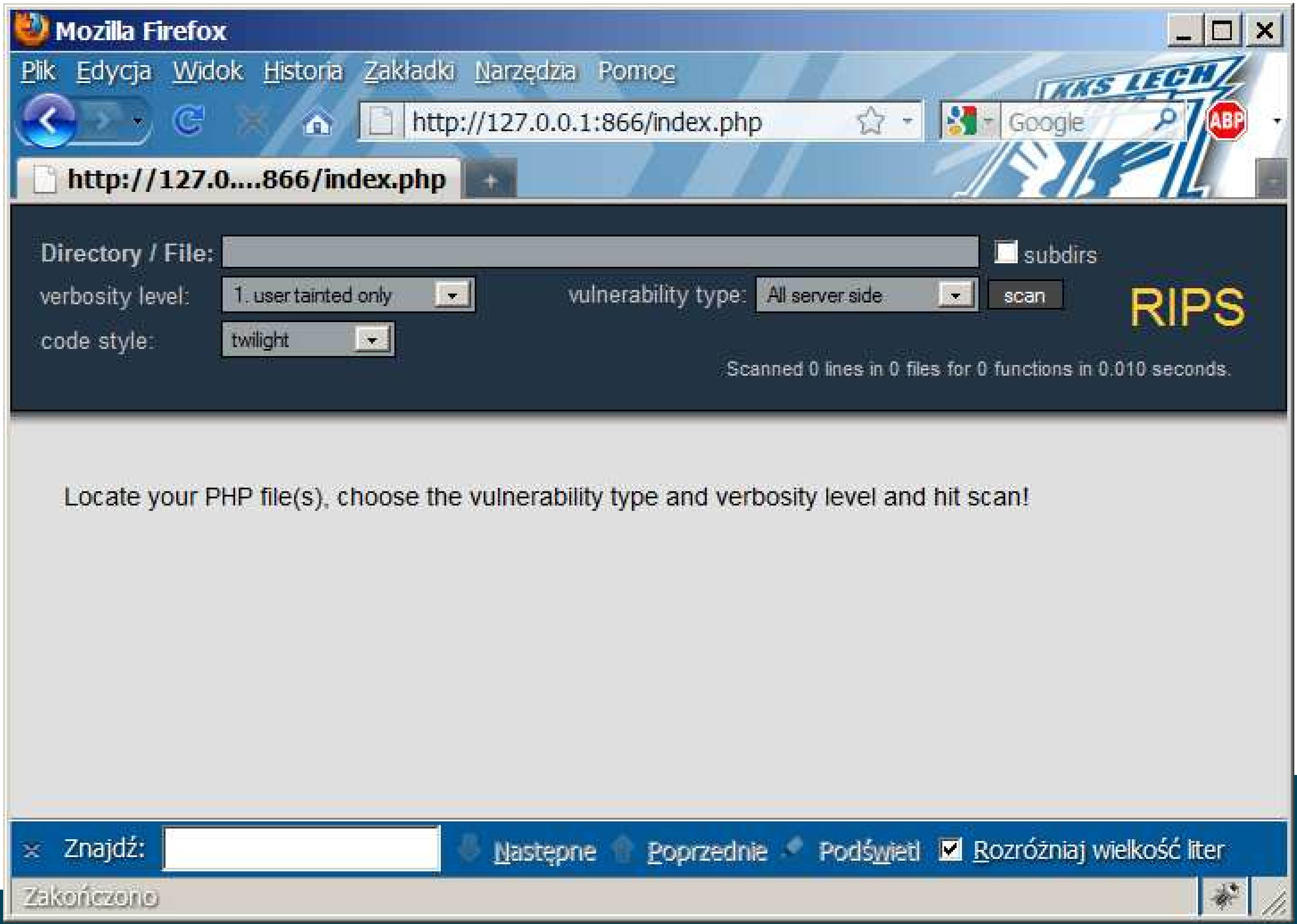
- RIPS – PHP source code scanner:
  - Last version: 0.3 (24 May 2010).
  - http://rips-scanner.sourceforge.net
  - BSD Licence.
  - Systems: Wherever PHP can be run.
  - Reguires a Web server and a browser (Opera, Firefox).
  - Languages: PHP (partial support for object-oriented).
  - Vulnerabilities:
    - *XSS.*
    - *SQL Injection.*
    - *Local/Remote File Inclusion.*
    - *Remote Code Execution*
    - *And more…*

Technical details (those that you need)

- Tokens:
  - The code is split into tokens which are analyzed.
  - Exemplary tokens are: opening tag, variable, whitespace, string.
- PVF = Potentially Vulnerable Functions.
  - Functions where vulnerabilities may be introduced, e.g.:
    - *system() for Remote Code Executions*
    - *echo() for XSS*
    - *readfile() for Information Disclosure*
  - Currently 139 functions, you may add your own.
- RIPS traces back, whether the suitable parameters of the PVFs could be tainted by the user.

Usage

- Simple Web interface:
  - Just prepare a local website and run in a Web server.



Mozilla Firefox

Plik Edycja Widok Historia Zakładki Narzędzia Pomoc

http://127.0.0.1:866/index.php

Directory / File: [ ] subdirs

verbosity level: 1. user tainted only

vulnerability type: All server side

scan

code style: twilight

RIPS

Scanned 0 lines in 0 files for 0 functions in 0.010 seconds.

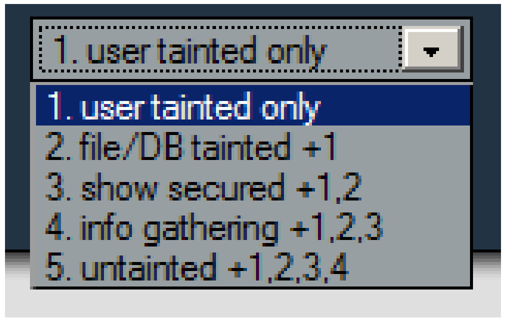
Locate your PHP file(s), choose the vulnerability type and verbosity level and hit scan!

Znajdź: Następne Poprzednie Podświetl ☑ Rozróżniaj wielkość liter

Zakończono

Verbosity levels

- 5 levels (the default is 1):
  - 1: traces tainted PVFs without any securing actions applied.
  - 2: files and local DBs treated as potentially malicious.
  - 3: shows PFVs even if securing actions have been applied.
  - 4: displays additional information about code structure.
  - 5: shows all PFVs calls and associated traces.
- Level X includes all levels from 1 to X-1
- The higher level,
  - The higher chance of detecting a vulnerability.
  - The more false postivies.
  - The longer takes the scan.


1. user tainted only

1. user tainted only
2. file/DB tainted +1
3. show secured +1,2
4. info gathering +1,2,3
5. untainted +1,2,3,4

Our opinion

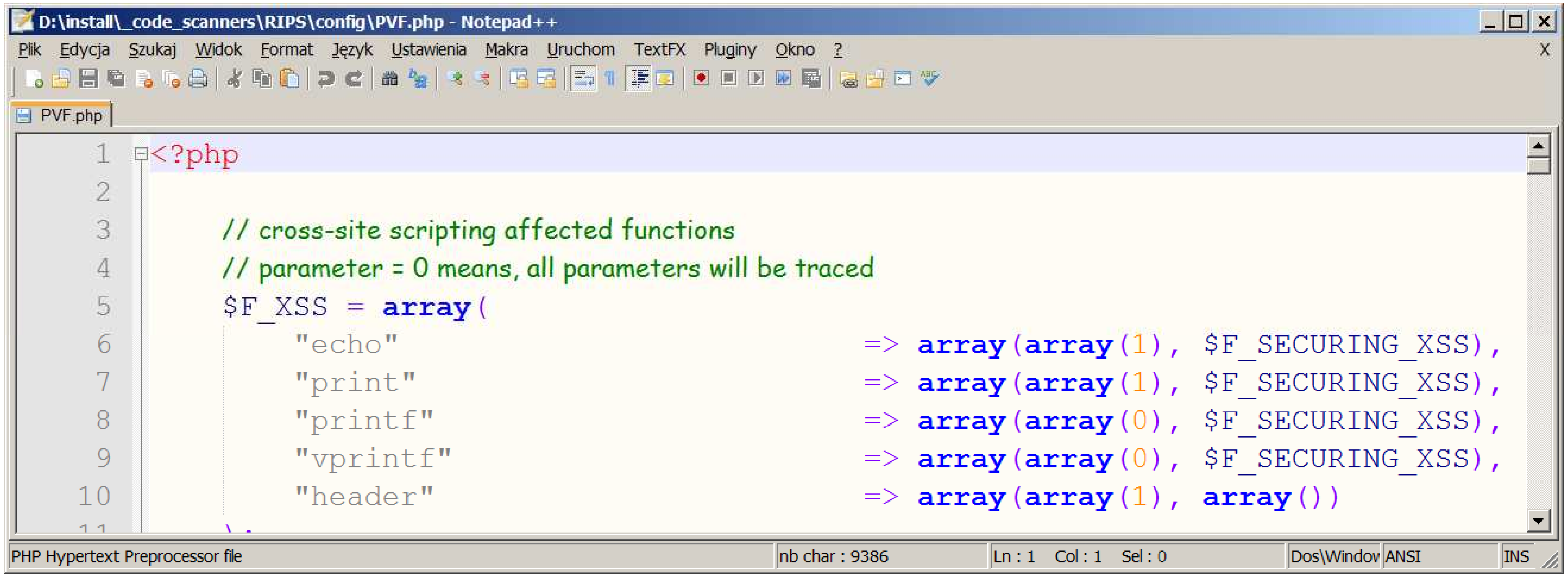
- A promising approach:
  - 2nd best submission during PHP Security Month.
  - **But beware that this is a very initial release!**
    - ***Sometimes I am not sure why and how it does something.***
- Easy installation and use:
  - However, some GUI improvements would be useful (e.g. “Scan for all vulnerabilities” setting, or GUI-based selection of source code to be scanned).

Hints for the developers

- We suggest two ways of using by the developers.
- Scan your code with verbosity level 1:
  - Will stop the most obvious cases of lack data sanitization and the apparent-XSS-who-to-hell-wrote-this-code remarks in your internal security reports ;).
- Define your subset of PVFs and set verbosity level 5:
  - You will get a more advanced *grep* for dangerous functions.
    - *Added value: shows the data flow through the functions.*
  - Look at the reported code snippets if everything is OK.
  - Good e.g. for detecting functions calling OS commands.

Custom definition of PVFs

- Look into <install_dir>\config\PVF.php:
  - Comment or delete lines with functions you are not interested in:


D:\install_code_scanners\RIPS\config\PVF.php - Notepad++
Plik Edycja Szukaj Widok Format Język Ustawienia Makra Uruchom TextFX Pluginy Okno ?
PVF.php
PHP Hypertext Preprocessor file
nb char : 9386
Ln : 1 Col : 1 Sel : 0
Dos\Window ANSI
INS

```
<?php

	// cross-site scripting affected functions
	// parameter = 0 means, all parameters will be traced
	$F_XSS = array(
		"echo"					=> array(array(1), $F_SECURING_XSS),
		"print"					=> array(array(1), $F_SECURING_XSS),
		"printf"				=> array(array(0), $F_SECURING_XSS),
		"vprintf"				=> array(array(0), $F_SECURING_XSS),
		"header"				=> array(array(1), array())
```

More resources

- Unfortunately, not much for now :(
- Sourceforge site:
  - http://rips-scanner.sourceforge.net
- Paper by Johannes Dahse:
  - http://php-security.org/downloads/rips.pdf
- And the source code (especially *config* subdirectory).

- PHPLint – another tool: validator and documentator for PHP 4 and PHP 5 programs
  - http://www.icosaedro.it/phplint