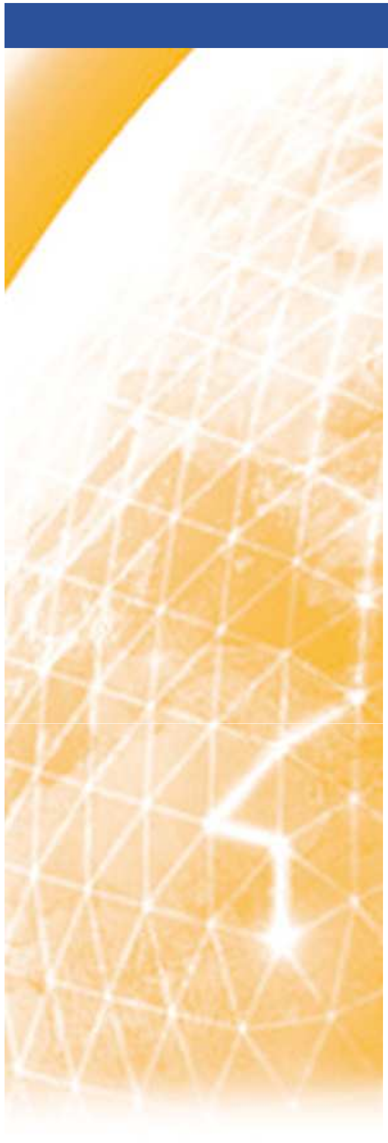


Systems and Software Security Session – A developer’s toolset

Gerard Frankowski

Poznań Supercomputing and Networking Center

www.eu-egee.org

Agenda

- **Source code scanners**
  - Why for developers?
  - Advantages and disadvantages
- **A short review: usage, results, remarks**
  - RATS
  - PiXy
  - cppcheck
  - Yasca
- **How to run source code review?**
  - A look at our methodology
- **Questions**

Scanners – the introduction

Source code scanners

- **As usually, they have advantages and disadvantages**
- **Advantages**
  - They may spare a lot of your time (give you a list of “look at” points)
  - They are able to present the results well structured – a good start point for writing the report
- **Disadvantages**
  - They are only tools, not intelligent beings: may detect “well structured” errors (like using a “dangerous” function)
  - Generate numerous false positives
- **So do not rely only on them! But are helpful with e.g.:**
  - Detecting of dangerous functions usage
  - Finding the cases of lacking data sanitization
  - Looking for memory and resource leaks

Source code scanners

- **They say:**

  A fool with a tool is still a fool ;)

- **We see the thing in the following way:**
  - The developers learn how to produce secure code
  - Knowing the secure coding principles, they support themselves in detecting the most obvious errors
    - Educated developers are able to find false positives
  - Security specialists perform a thorough source code review
    - Concentrated on defending against sophisticated attacks

Tools review

An overview

- **Several tools used by PSNC Security Team during our EGEE security reviews will be shown**
  - You will be able to see the real scanning results for EGEE codes we were investigating
  - All the presented source code scanners are free
  - Installation and usage is trivial
  - Work both for Unix/Linux and Windows

- **Our “big four” are:**
  - RATS
  - Pixy
  - cppcheck
  - Yasca

RATS

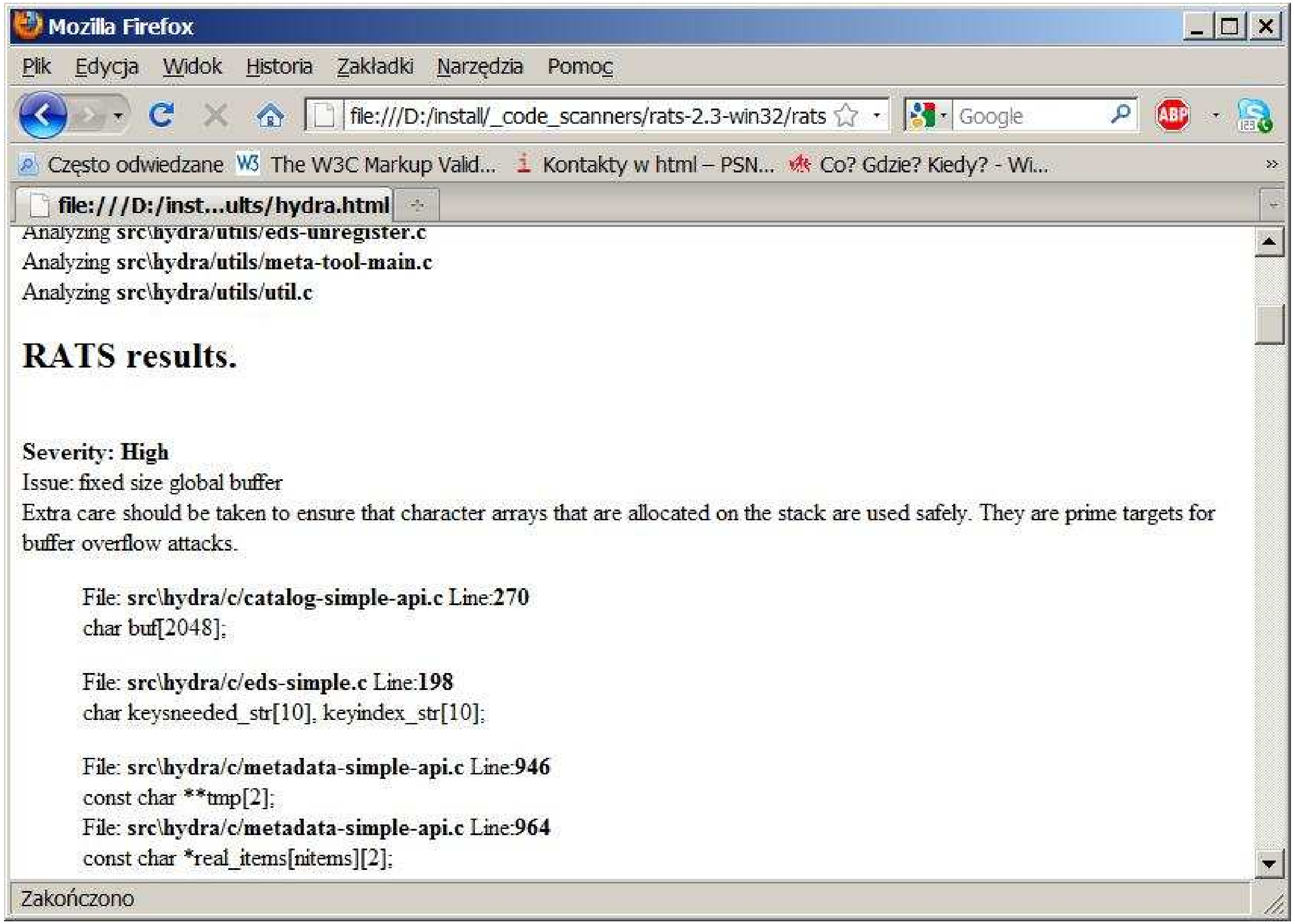
- **RATS: Rough Auditing Tool for Security**
  - Last version: 2.3
  - Made by Fortify Software
  - http://www.fortifysoftware.com/security-resources/rats.jsp
  - GNU Public License
  - Systems: Unix/Linux, Windows
    - Requires Expat parser (http://expat.sourceforge.net)
  - Languages: C, C++, Perl, PHP, Python
  - Vulnerabilities: including buffer overflows, TOCTOU (race conditions), Remote Code Execution, shows dangerous functions)

RATS (2)

- **Usage:**
  - 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 directory
    - RATS uses recursion in the 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

RATS (3)

- **Example results for Hydra client (written in C):**


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.

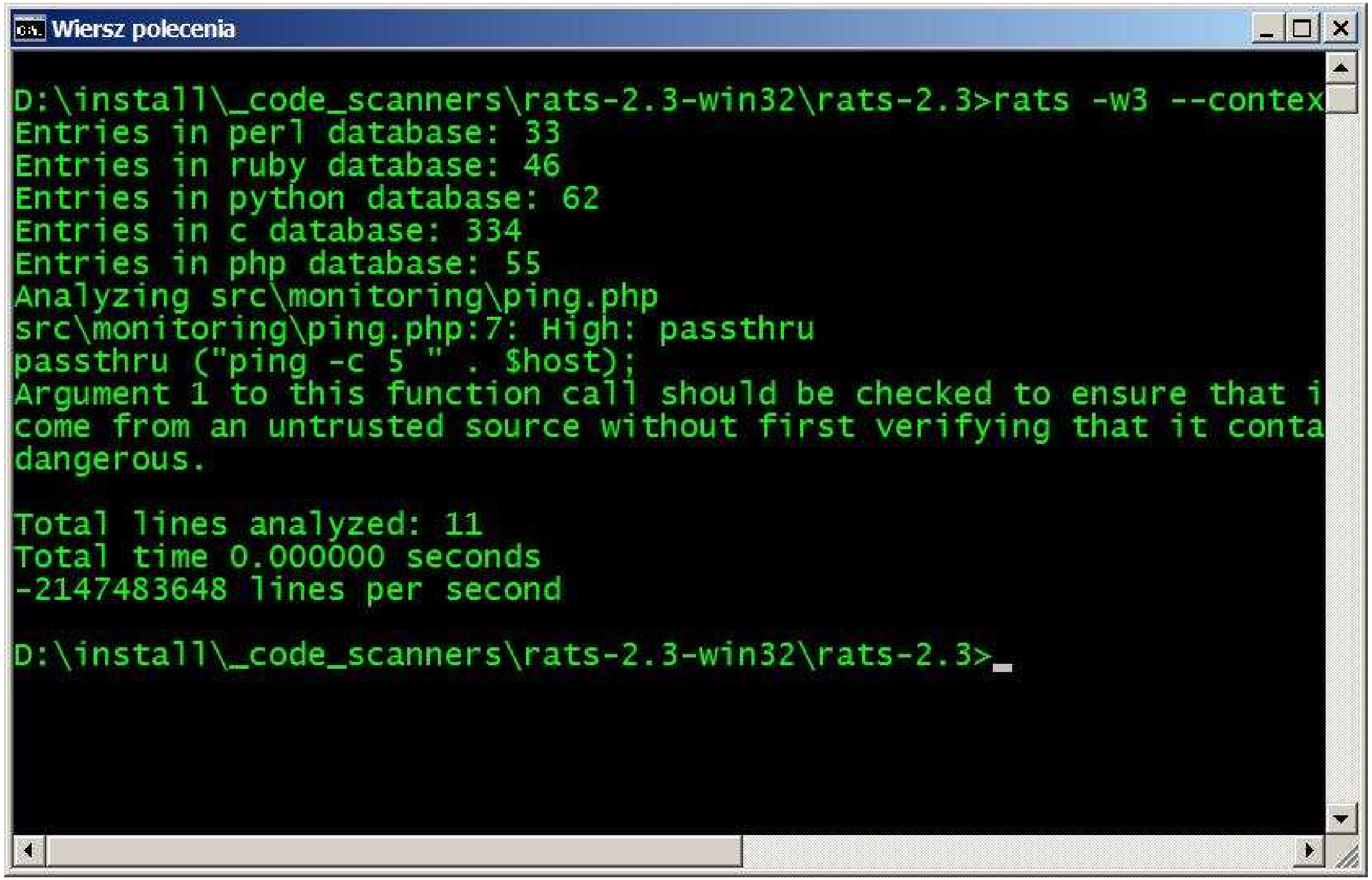
File: **src\hydra/c/catalog-simple-api.c** Line:**270**
char buf[2048];

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];

RATS (4)

- **Example results for a ping.php (written in PHP):**
  - The source code contained a 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>_
```

RATS (5)

- **Our opinion**
  - RATS is good at emphasizing:
    - Dangerous functions
    - TOCTOU
    - Fixed size buffers
  - Many false positives (like other tools)
  - Good reporting facilities
  - Works fast
  - Sometimes crashes…
    - Try to change e.g. warning level or output format then, may help

PiXy

- **Pixy – 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
  - Vulnerabilities: XSS, SQL Injection

PiXy (2)

- **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
- **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 how to understand the results

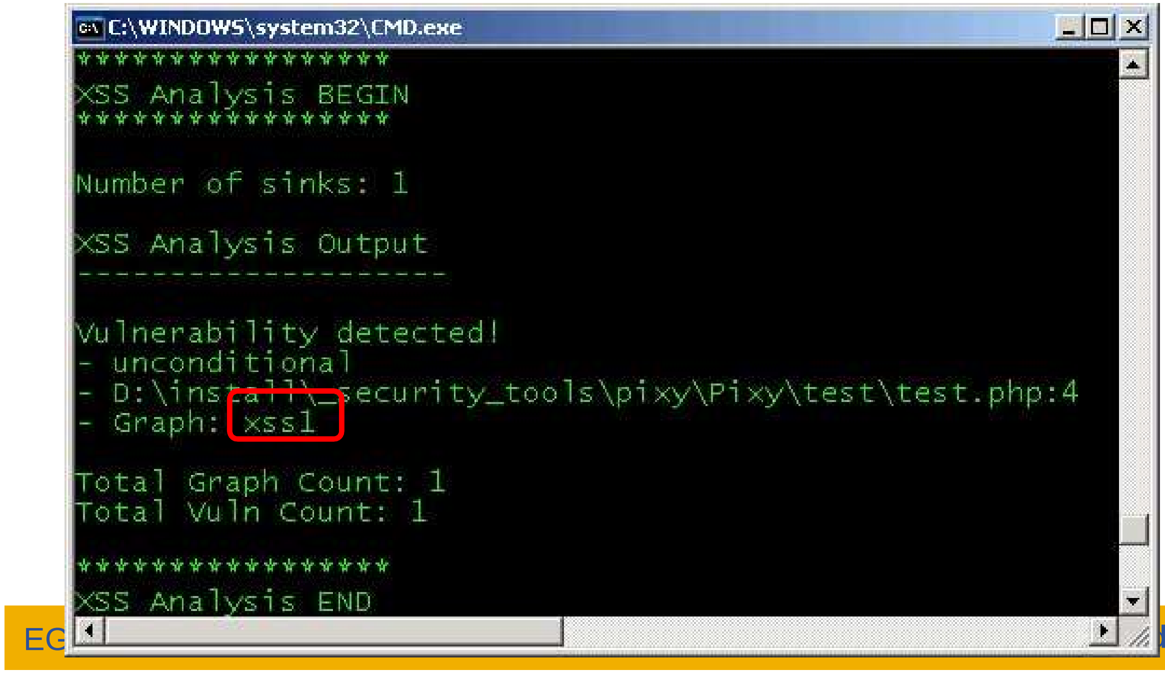
PiXy (3)

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

PiXy (4)

- **Example**
  - Vulnerable file: test.php (a simplified version of ping.php)

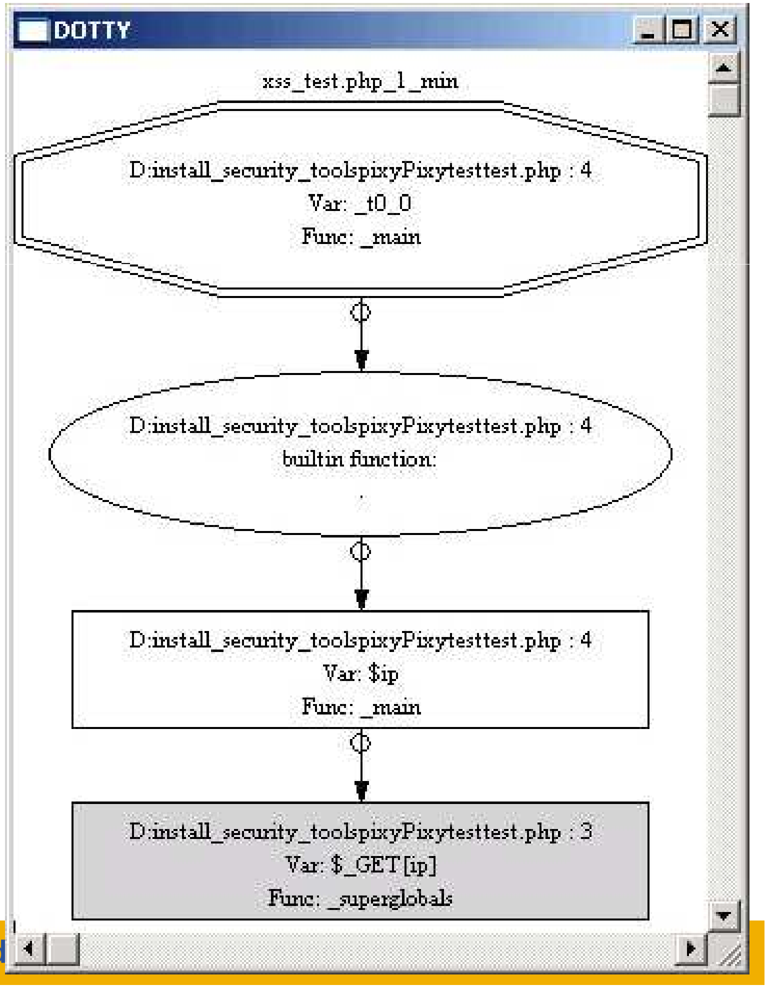
```
<html>
<?php
ip=$_GET['ip'];
echo "Pinging host $ip";
passthru("ping -c 5" . $ip); ?>
</html>
```



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.dot:



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

PiXy (5)

- **Our opinion**
  - An interesting approach
  - Numerous false positives
  - Effort needed to filter out unnecessary alarms, but the remaining spare a lot of work – especially for large sites
  - Relatively complicated result analysis
  - Not working with object-oriented PHP 5.x is a significant disadvantage
  - Seems not to be developed any more
- **Hint 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 bottommost item (where the malicious data may be introduced?) and the topmost one (where it is displayed?)

cppcheck

- **cppcheck – a C/C++ source code scanner**
  - Last version: 1.35
  - http://cppcheck.wiki.sourceforge.net
  - GNU GPL
  - Command line mode + GUI mode
  - Systems: at least cmd line mode should work on all
  - Languages: C/C++
  - Vulnerabilities: bounds checking, variable range, memory leaks, NULL pointer dereference, many others
- **The community goal: no false positives**

cppcheck (2)

- **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] ...
```

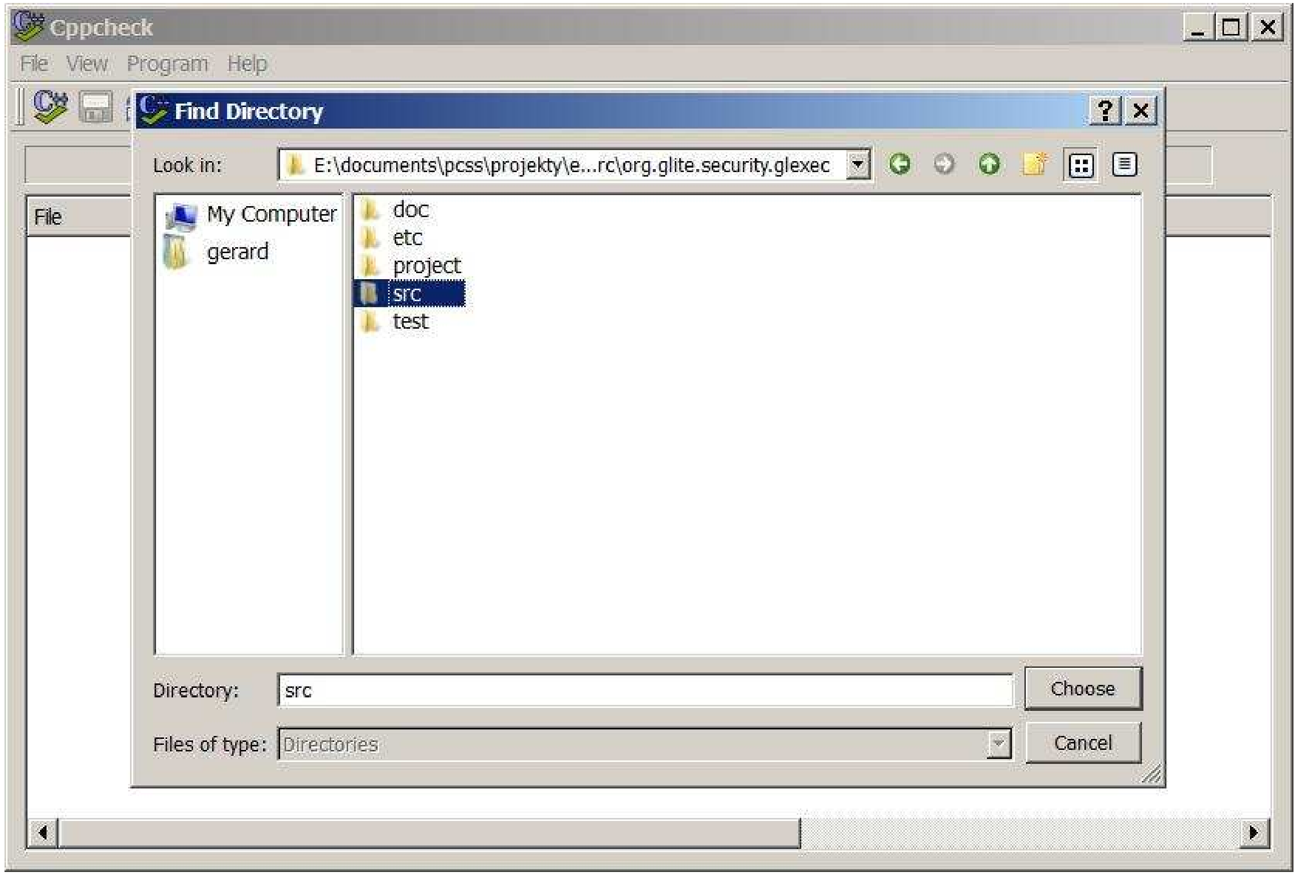
- **The result is sent to the standard output by default, so we recommend to redirect it to a file**
  - The output may be customized through XSLT
- **We use it usually in the following way:**

```
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

cppcheck (3)

- **GUI:**
  - File | Check directory | Choose
  - Please note that cppcheck starts to work at once!



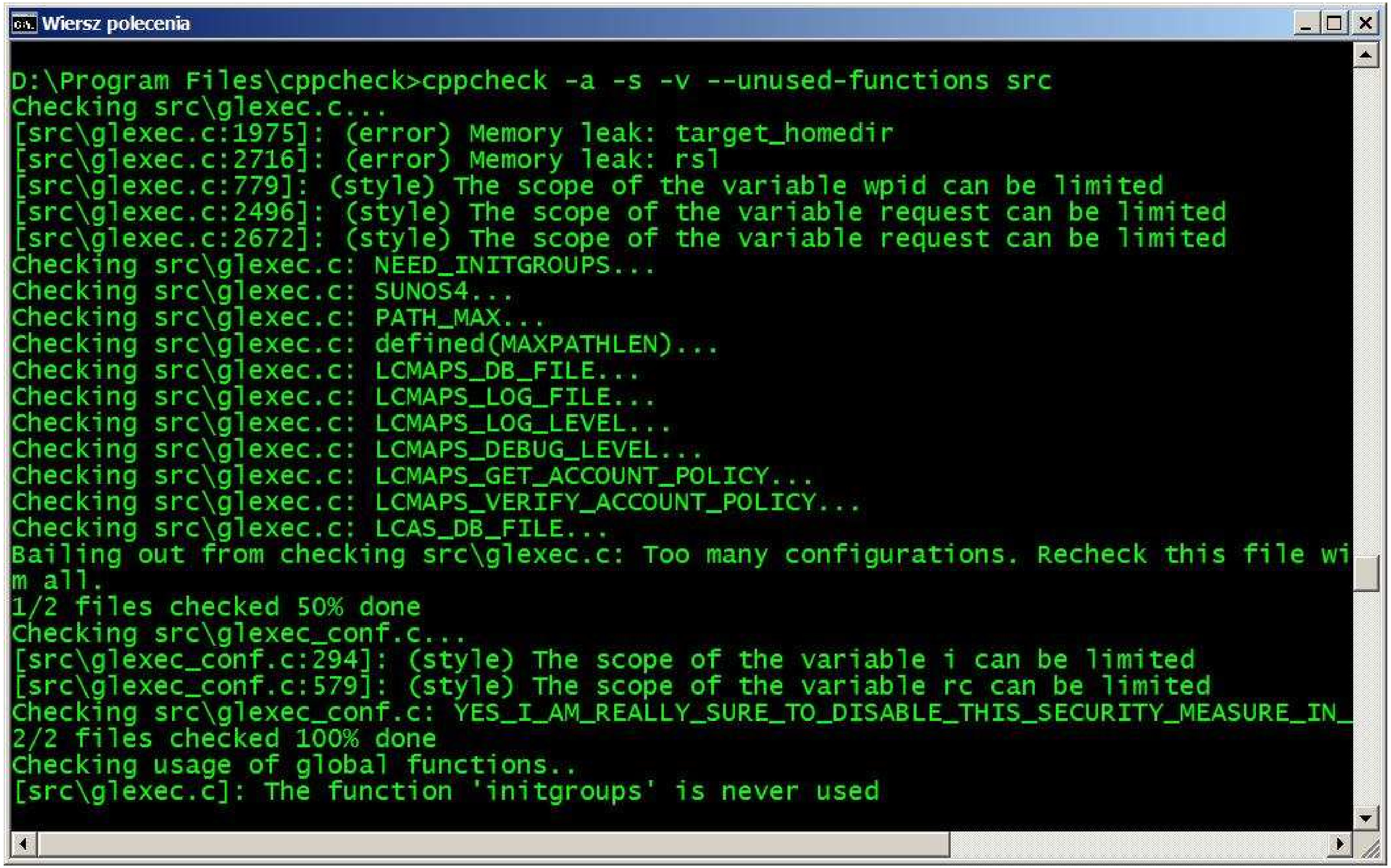
Look in: E:\documents\pcss\projekty\e...rc\org.glite.security.glexec

Directory: src

Files of type: Directories

cppcheck (4)

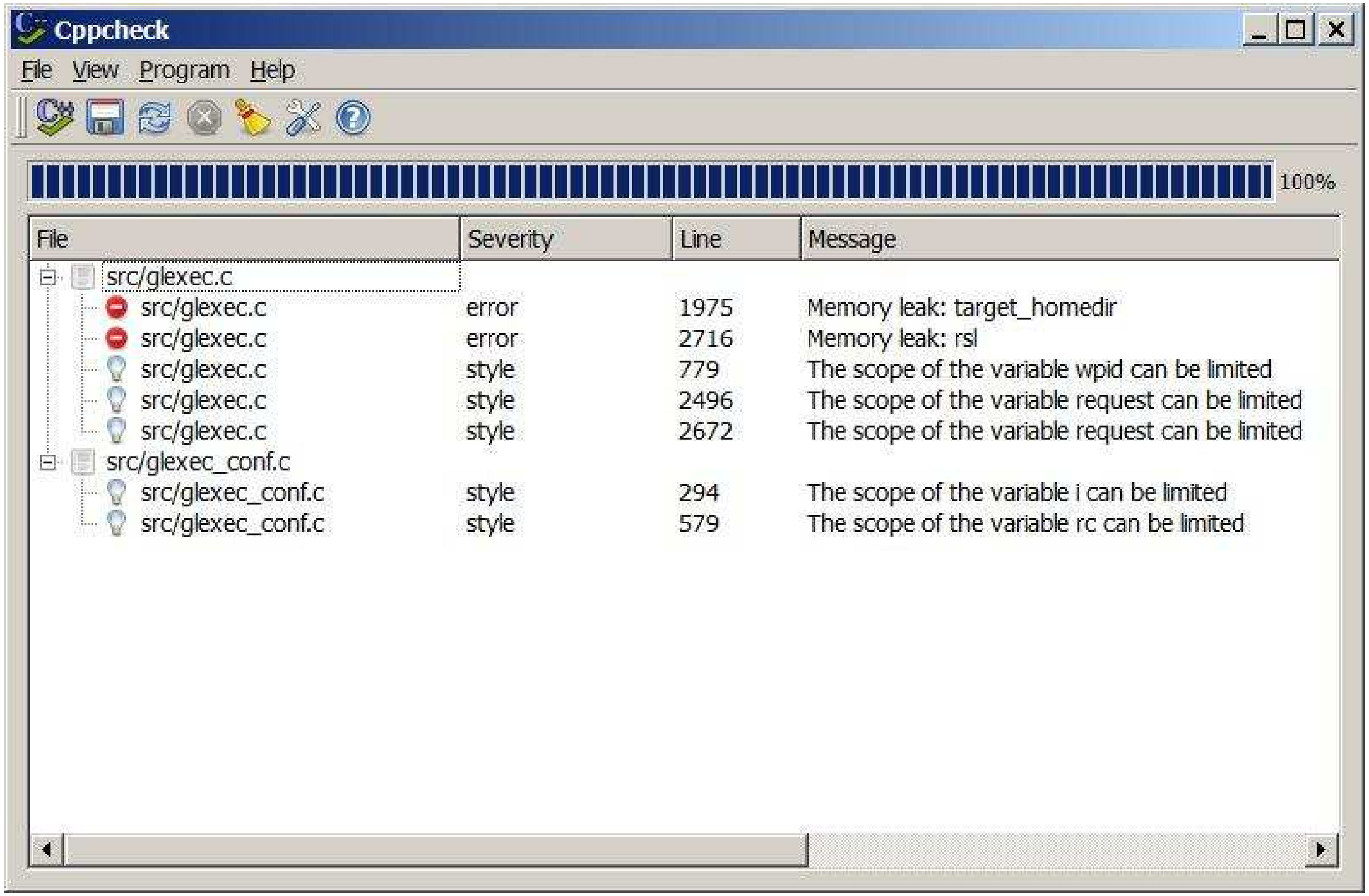
- **Example results (cmd line)**



```
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
```

cppcheck (5)

- **Example results (GUI mode)**
  - May be saved to a XML or TXT file



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

cppcheck (6)

- **Our opinion**
  - Although GUI mode has got Settings page, the command line mode is better to customize
  - Very little false positives indeed, however the tool seems not to detect everything it should
  - 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

YASCA

- **YASCA – Yet Another Source Code Analyzer**
  - Last version: 2.1
  - http://www.yasca.org, http://sourceforge.net/projects/yasca
  - BSD license
  - Command line tool
  - Two components:
    - A framework for source code analyzing
    - An implementation of the framework with plugins (including e.g. well known cppcheck and Pixy!)
    - Possibility of implementing own plugins
  - Systems: Widnows, Linux
    - Requires PHP and Java 1.5 (for plugins like PMD or FindBugs)
  - Languages: Many (C/C++, Java, PHP, COBOL, ASP, HTML, JavaScript, CSS – same as its plugins)
  - Vulnerabilities: Many (same as its plugins)

YASCA (2)

- **Usage**

  yasca [options] directory

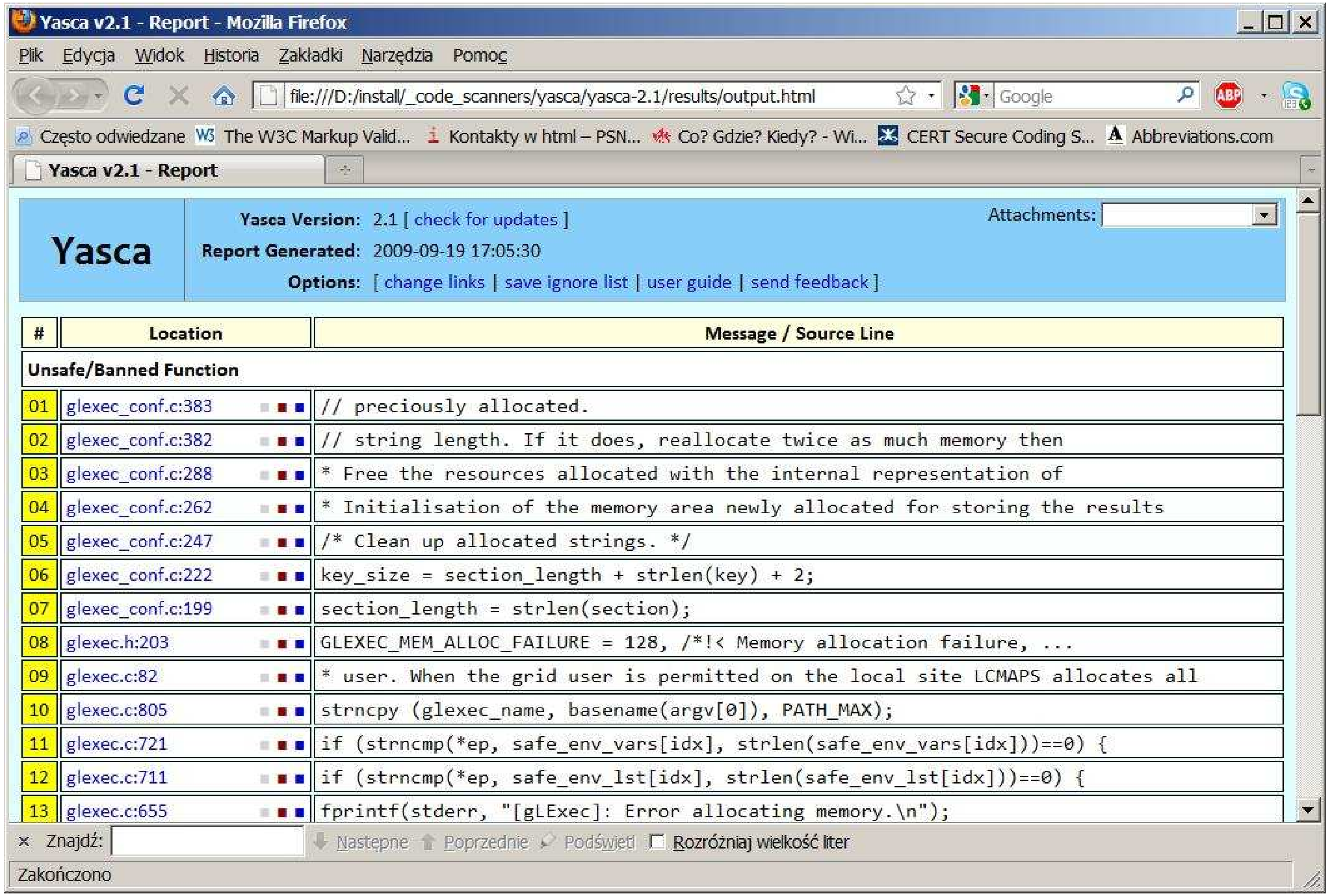
  – yasca without options (or yasca -h) will show help
- **Output**
  – YASCA generate HTML reports by default
  – Many other report templates may be selected
  – HTML reports are actually a small Web application, with results, source code preview, additional explanations, fix suggestions
  – Status information are directed to the standard output – you may want to redirect it to a file
- **We use it usually like:**

  ```
  yasca --debug -o results\output.html src
  ```

  – --debug for more information
  – sometimes we run with individual plugin(s)

YASCA (3)

- **Results (generated HTML page)**


Yasca v2.1 - Report - Mozilla Firefox
Plik Edycja Widok Historia Zakładki Narzędzia Pomoc
file:///D:/install/_code_scanners/yasca/yasca-2.1/results/output.html
Google
Często odwiedzane
The W3C Markup Valid...
Kontakty w html – PSN...
Co? Gdzie? Kiedy? - Wi...
CERT Secure Coding S...
Abbreviations.com
Yasca v2.1 - Report
Znajdź:
Następne
Poprzednie
Podświetl
□ Rozróżniaj wielkość liter
Zakończono

Yasca

Yasca Version: 2.1 [check for updates]

Report Generated: 2009-09-19 17:05:30

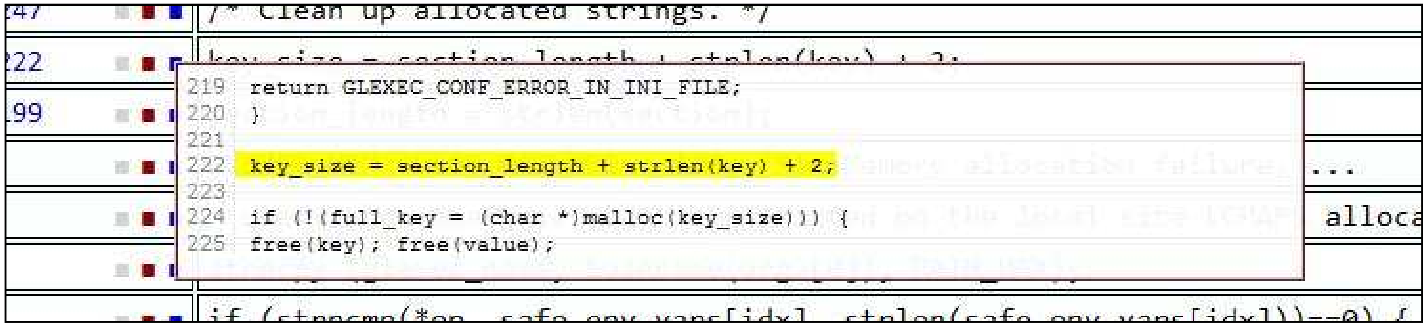
Options: [change links | save ignore list | user guide | send feedback]

Attachments:

#	Location	Message / Source Line
Unsafe/Banned Function		
01	glexec_conf.c:383	`// preciously allocated.`
02	glexec_conf.c:382	`// string length. If it does, reallocate twice as much memory then`
03	glexec_conf.c:288	`* Free the resources allocated with the internal representation of`
04	glexec_conf.c:262	`* Initialisation of the memory area newly allocated for storing the results`
05	glexec_conf.c:247	`/* Clean up allocated strings. */`
06	glexec_conf.c:222	`key_size = section_length + strlen(key) + 2;`
07	glexec_conf.c:199	`section_length = strlen(section);`
08	glexec.h:203	`GLEXEC_MEM_ALLOC_FAILURE = 128, /*!< Memory allocation failure, ...`
09	glexec.c:82	`* user. When the grid user is permitted on the local site LCMAPS allocates all`
10	glexec.c:805	`strncpy (glexec_name, basename(argv[0]), PATH_MAX);`
11	glexec.c:721	`if (strncmp(*ep, safe_env_vars[idx], strlen(safe_env_vars[idx]))==0) {`
12	glexec.c:711	`if (strncmp(*ep, safe_env_lst[idx], strlen(safe_env_lst[idx]))==0) {`
13	glexec.c:655	`fprintf(stderr, "[gLExec]: Error allocating memory.\n");`

YASCA (4)

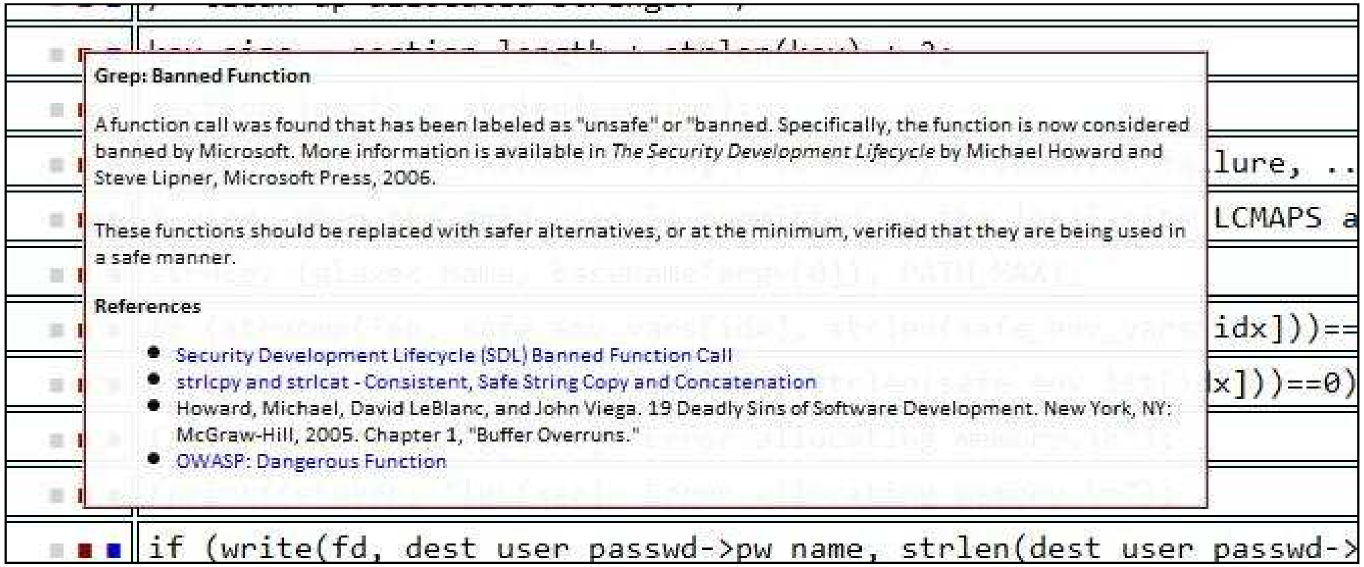
- Click 🟦 to view the source code



```
/* Clean up allocated strings. */
key_size = section_length + strlen(key) + 2;
```

```
return GLEXEC_CONF_ERROR_IN_INI_FILE;
}

key_size = section_length + strlen(key) + 2;

if (!(full_key = (char *)malloc(key_size))) {
free(key); free(value);
```

- Click 🟥 to view more explanations



Grep: Banned Function

A function call was found that has been labeled as "unsafe" or "banned. Specifically, the function is now considered banned by Microsoft. More information is available in *The Security Development Lifecycle* by Michael Howard and Steve Lipner, Microsoft Press, 2006.

These functions should be replaced with safer alternatives, or at the minimum, verified that they are being used in a safe manner.

References

- Security Development Lifecycle (SDL) Banned Function Call
- strlcpy and strlcat - Consistent, Safe String Copy and Concatenation
- Howard, Michael, David LeBlanc, and John Viega. 19 Deadly Sins of Software Development. New York, NY: McGraw-Hill, 2005. Chapter 1, "Buffer Overruns."
- OWASP: Dangerous Function

```
if (write(fd, dest user passwd->pw name, strlen(dest user passwd->
```

YASCA (5)

- **Our opinion**
  - We actually start with YASCA, therefore would not like to issue autoritative opinions
  - On one hand we do not like frameworks that group other tools (usually it brings more fruitful results to run several customized tools)
  - On the other hand, the idea looks very fine, and running tools as YASCA plugins may spare time
  - YASCA inherits all advantages and disadvantages of individual tools that it runs as a plugin
  - Sometimes gives strange (but easy to identify) false positives
  - The whole scan takes time!
  - Very fine reporting approach
- **Therefore consider learning more about Yasca, especially if you do not like the tools described before**

A look at our methodology

Our methodology (1)

- **What is our methodology for source code reviews?**
  - At least 2 persons should be involved
  - If requested for penetration testing, the best is to have another one
- **Preparations**
  - We start with learning the module
    - What it is for? What it does? Where it will be installed?
    - What data travel within it? Where?
    - Are the data sensitive in any way?
    - What are the interfaces to other modules.
  - Writing a test plan

Example of a test plan

- **Test plan for glexec source code tests**
  - Person A, B: reading documentation (basically) – 4 hours
  - Person A: source code manual review – 24 hours
  - Person B: source code automated review – 8 hours
  - Person A, B: cross-check of the results – 8 hours (2 persons x 4 hours)
  - Person A: writing a detailed report – 16 hours
  - Person B: assessment of the report – 4 hours
  - Person A: the final changes of the report – 6 hours

  - TOTAL: 66 hours

Our methodology (2)

- **Static analysis**
  - A thorough manual code review (just reading)
  - Scanning the code with tools
    - Never the same person
    - We always use several tools (if available)
  - Cross-checks of the results
  - The code reader writes the full report and the scanning guy assesses it
- **Dynamic analysis (penetration testing)**
  - Additional work, but often requested
  - A test environment is highly desired
  - May be run earlier, in parallel or later than the review
  - A person who made the review is never the pentester
    - Unless he or she wants to confirm or check everything

Our methodology (3)

- **Reporting**
  - Usually we give first a summary of vulnerabilities and general recommendations
  - Then every single issue found is described
    - They are grouped in “Vulnerabilities” and “Remarks” sections
  - The final report is assumed to be a potential discussion point with the developers
  - We know security deeper, the developers are better oriented with the specifics of their software
    - Sometimes we assume e.g. using a dangerous function as a vulnerability, but is may be justified with conditions we don’t know
  - The interaction may be assumed as risk analysis
  - Had some troubles in the past with it, but now we trying to keep an eye on it

Our methodology (4)

- **How we can help the developers here?**
- **Some advices**
  - Never test your own code
    - It makes no sense, you are too directed
    - Make a test: write a text, correct it for typos and give to someone else
  - If possible, use several scanners for the given programming language

More information

- **Flawfinder – another famous tool not described here**
  - http://www.dwheeler.com/flawfinder
  - Contains also a list of other scanners with links and short descriptions
- **Another list of source code scanners**
  - http://www.tech-faq.com/source-code-security-vulnerabilities.shtml
- **OWASP Code Review Project**
  - http://www.owasp.org/index.php/Category:OWASP_Code_Review_Project
  - Combination of a book on secure code review and tools to support such an activity

Questions or comments

Thank you for your attention!