

# Secure Programming Experiment Note (1)

## Code Review Tools

The followings are some open source tools

1. flawfinder (running in Linux system, see <http://www.dwheeler.com/flawfinder/> )
2. Splint (see <http://www.splint.org/> )
3. CPPCheck  
([http://sourceforge.net/apps/mediawiki/cppcheck/index.php?title=Main\\_Page](http://sourceforge.net/apps/mediawiki/cppcheck/index.php?title=Main_Page) )
4. Findbugs (<http://findbugs.sourceforge.net/> ) for Java

See the website below

[http://samate.nist.gov/index.php/Source\\_Code\\_Security\\_Analyzers.html](http://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html)

### ● Flawfinder



## Flawfinder

This is the main web site for *flawfinder*, a program that examines source code and reports possible security weaknesses ("flaws") sorted by risk level. It's very useful for quickly potential security problems *before* a program is widely released to the public. See "[how does Flawfinder work?](#)", below, for more information on how it works.

Flawfinder is specifically designed to be easy to install and use. After installing it, at a command line just type:

```
flawfinder directory_with_source_code
```

Flawfinder works on Unix-like systems today (it's been tested on GNU/Linux), and it should be easy to port to Windows systems. It requires Python 1.5 or greater to run (Python

Please take a look at [other static analysis tools for security](#), too. One reason I wrote flawfinder was to encourage using static analysis tools to find security vulnerabilities.

## Sample Output

If you're curious what the results look like, here are some sample outputs:

1. [The actual text output \(when allowing all potential vulnerabilities to be displayed\)](#)

Test the following commands:

```
flawfinder bfsucc.c > bfsucc.ff
```

```
Flawfinder version 1.27, (C) 2001-2004 David A. Wheeler.
Number of dangerous functions in C/C++ ruleset: 160
Examining bfsucc.c
bfsucc.c:25: [5] (buffer) gets:
  Does not check for buffer overflows. Use fgets() instead.
bfsucc.c:17: [4] (shell) system:
  This causes a new program to execute and is difficult to use safely.
  try using a library call that implements the same functionality if
  available.
bfsucc.c:23: [2] (buffer) char:
  Statically-sized arrays can be overflowed. Perform bounds checking,
  use functions that limit length, or ensure that the size is larger than
  the maximum possible length.

Hits = 3
Lines analyzed = 29 in 0.51 seconds (2247 lines/second)
Physical Source Lines of Code (SLOC) = 29
Hits@level = [0]  0 [1]  0 [2]  1 [3]  0 [4]  1 [5]  1
Hits@level+ = [0+]  3 [1+]  3 [2+]  3 [3+]  2 [4+]  2 [5+]  1
Hits/KSLOC@level+ = [0+] 103.448 [1+] 103.448 [2+] 103.448 [3+] 68.9655 [4+] 68.
9655 [5+] 34.4828
Minimum risk level = 1
Not every hit is necessarily a security vulnerability.
--更多--(94%)
```

- Splint (Secure Programming Lint)

**Splint** - Secure Programming Lint

[Download](#) - [Documentation](#) - [Manual](#) - [Links](#)



## Splint

Annotation-Assisted Lightweight Static Checking  
Inexpensive Program Analysis Group  
[University of Virginia, Department of Computer Science](#)

Splint is a tool for statically checking C programs for security vulnerabilities and coding mistakes. With minimal effort, Splint adding annotations to programs, Splint can perform stronger checking than can be done by any standard lint.

### Download

[Splint Version 3.1.2](#)

[Source code](#) - [\[tgz distribution\]](#)  
[Windows Installer](#)

[SourceForge Project Page](#)  
[Current Development Code](#)  
[Browse Code CVS](#)

[Mailing Lists](#) ([splint-discuss archives](#))  
[Links](#)

### Documentation

[Splint Manual](#)

**Papers:** [Improving Security Using Extensible Lightweight Static Analysis](#), IEEE Software Jan/Feb 2002; [Statically Detecting Likely Buffer Overflow Vulnerabilities](#), USENIX Security 2001; [Static Detection of Dynamic Memory Errors](#), PLDI 1996; [More...](#)

**Talks:** [USENIX Security 2001](#) [PPT] [PDF]; [UW/MSR](#) [PPT] [PDF]; [More...](#)

[Examples](#)  
[FAQ](#) (updated 3 May 2004)  
[Press](#) - external articles  
[Release](#) - latest release notes

Test the following commands:

```
splint bfsucc.c > bfsucc.sp
```

documentation: <http://www.splint.org/manual/manual.pdf>

Practice: Please test `qwik-smtpd.c` by using the above two tools.

Connect to: `hacknew@120.113.173.21`

### On-line tool

Checkmarx: <http://www.checkmarx.com/>

Kiuwan - SaaS Software Quality & Security Analysis <https://www.kiuwan.com/>

### Others

1. Owasp source code analysis tools

[https://www.owasp.org/index.php/Source Code Analysis Tools](https://www.owasp.org/index.php/Source_Code_Analysis_Tools)

2. [VCG](#) - Scans C/C++, Java, C# and PL/SQL for security issues and for comments which may indicate defective code. The config files can be used to carry out additional checks for banned functions or functions which commonly cause security issues.

3. Fortify 與資訊安全

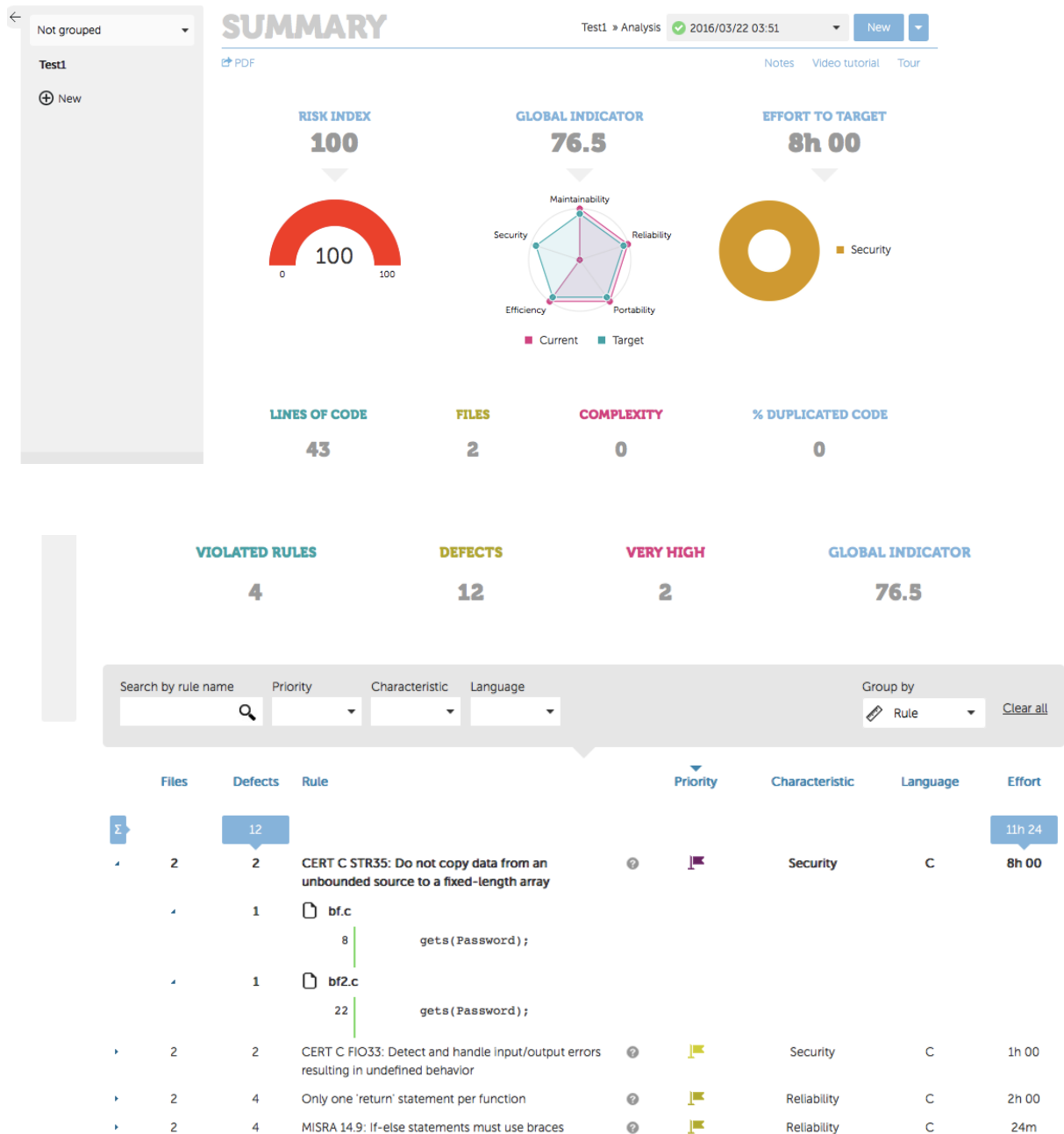
<http://www.gss.com.tw/index.php/focus/eis/44-eis59/122-fortify>

Kiuwan

<https://www.kiuwan.com/>

## Software Analytics in the Cloud

Know the risks, the technical debt, the security flaws, the performance issues...  
create automatic action plans and analyze their progress. Make better decisions,  
faster, with Kiuwan Software Analytics!



## Coverity - Code Advisor On Demand

### Free Trial (15 days)

Code Advisor On Demand works by intercepting builds through plugins that integrate with your build environment on Mac OS X, Linux, and Windows. We also provide a standalone command-line tool, Coverity CLI, that can help when a native plugin for your build environment is not available. Please consult our help pages for more information on How Code Advisor On Demand works.

Also can be used in eclipse (Java) and visual studio (C/C++/C#)



### Getting Started with Code Advisor On Demand

Code Advisor On Demand works by intercepting builds through plugins that integrate with your build environment on Mac OS X, Linux, and Windows. We also provide a standalone command-line tool, Coverity CLI, that can help when a native plugin for your build environment is not available. Please consult our help pages for more information on [How Code Advisor On Demand works](#).

- Maven
- Gradle
- Eclipse
- Visual Studio**
- Coverity CLI
- Other

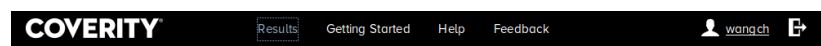
#### Code Advisor On Demand Visual Studio Extension (C/C++/C#)

Note: Code Advisor On Demand enables the analysis of C++ and C# code using Visual Studio 2012+ (including Visual Studio Community 2013 edition) with limited support for Visual Studio 2015 (Community and Enterprise). We are working on improving support for 2015. You are welcome to try Code Advisor On Demand in Visual Studio 2015, but cannot promise it will work. Express editions are not supported. If you need support for other versions of Visual Studio, please [let us know](#).

#### Installation

The Code Advisor On Demand extension is published to Microsoft's [Visual Studio Gallery](#). To install the extension, please open *Tools → Extensions and Updates* from the main menu and then search for "Code Advsiar On Demand" to locate the extension for installation.

### Results



#### Latest by Stream

##### wangch/Project1

coverity 1 high impact issue

Submitted Oct 21, 2015 9:52 PM

msvs.solution Project1  
build.tool msvc 12.0 Professional (1.1763)

Completed Oct 21, 2015 9:52 PM

Upload time 4 seconds  
Queue time 9 seconds  
Analysis time 12 seconds  
Analysis version 77.0.3  
Lines of code analyzed 24

Issues found: 1

High Impact 1  
Medium Impact 0  
Low Impact 0  
Suppressed 0

[View Results](#)

Results available until  
Nov 20, 2015 ([why?](#))

[more...](#)

##### wangch/jtest

coverity 1 medium impact issue

Submitted Oct 21, 2015 5:03 PM

eclipse.workspa... /home/wangch/workspace  
eclipse.projects jtest  
build.tool eclipse 3.8.1.dist (1.3.1142)

Completed Oct 21, 2015 5:10 PM

Upload time 4 minutes  
Queue time <1 second  
Analysis time 76 seconds  
Analysis version 77.0.3  
Lines of code analyzed 85

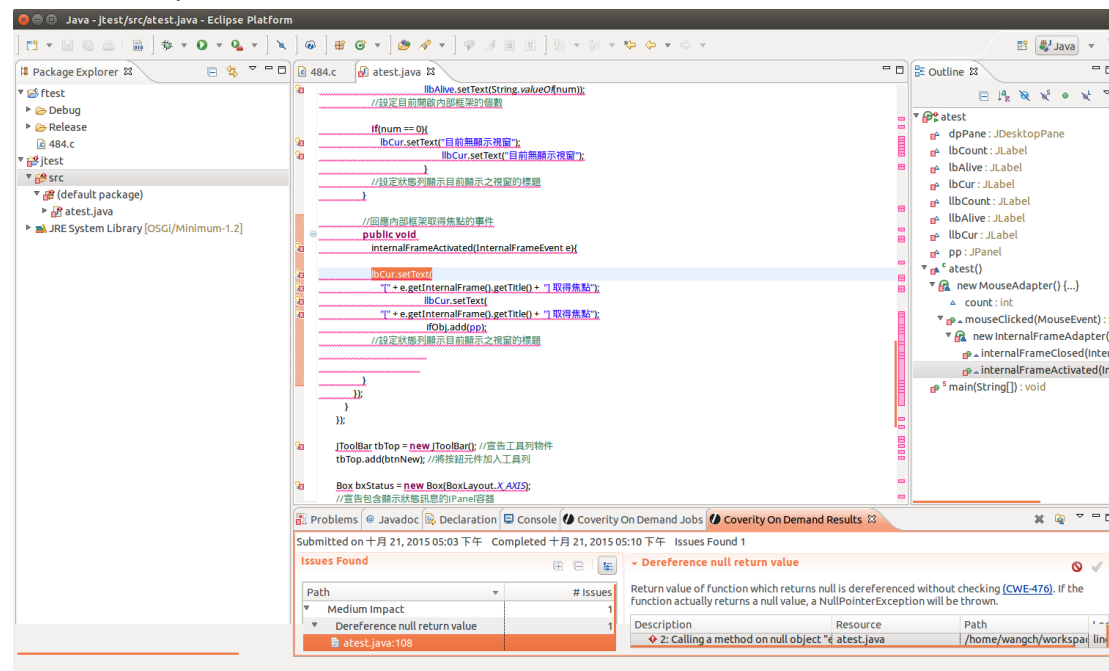
Issues found: 1

High Impact 0  
Medium Impact 1  
Low Impact 0  
Suppressed 0

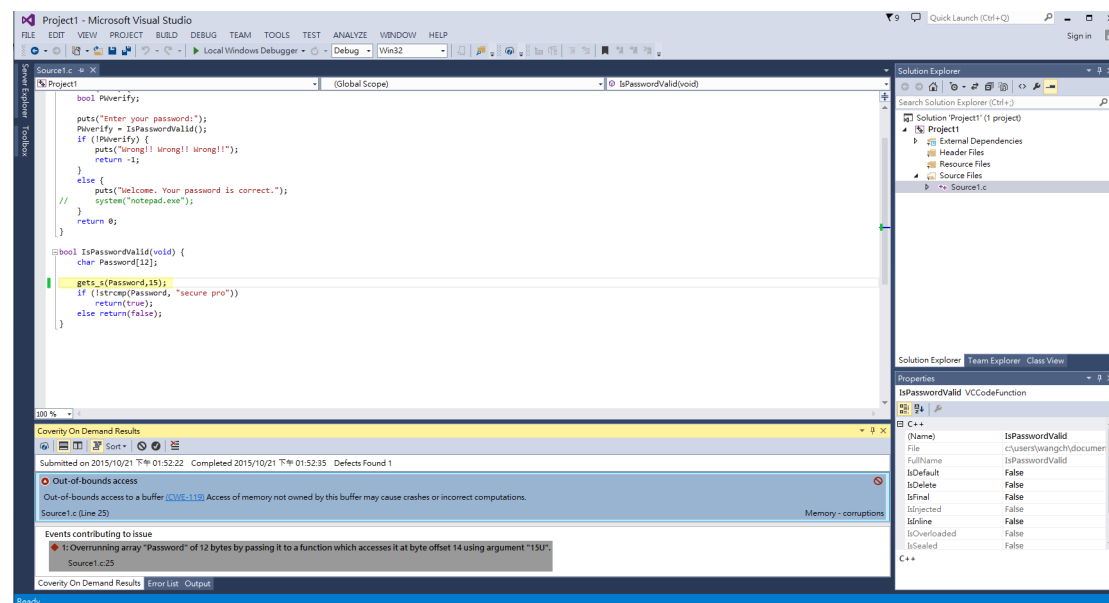
[View Results](#)

Results available until  
Nov 20, 2015 ([why?](#))

## Used for eclipse



## Used for visual studio 2013



## CWE (Common Weakness Enumeration) <https://cwe.mitre.org/>

CWE™ International in scope and free for public use, CWE provides a unified, measurable set of software weaknesses that is enabling more effective discussion, description, selection, and use of software security tools and services that can find these weaknesses in source code and operational systems as well as better understanding and management of software weaknesses related to architecture and design.

## PVS-Studio

PVS-Studio is a static analyzer that detects errors in source code of C/C++ applications. There are sets of rules included into PVS-Studio:

- General-purpose diagnosis
- Detection of possible optimizations
- Diagnosis of 64-bit errors (Viva64)

The screenshot shows the PVS-Studio website. At the top, there's a header with the title 'PVS-Studio' and subtitle 'Static Code Analyzer for C/C++'. Below the title are links for 'Product page', 'Documentation', and 'Troubleshooting FAQ'. There are two buttons: 'Download and try' and 'Buy'. On the left, a box displays statistics: 'Collected Errors: 9055', 'Checked Projects: 224', and 'Last Report: FreeSWITCH'. Below the header, there's a breadcrumb trail: 'Home > Blog > PVS-Studio for Visual C++'. The main content area features a blog post titled 'PVS-Studio for Visual C++' dated '09.02.2015' by 'Andrey Karpov'. The post content includes links to 'What static code analysis is and why do we need it', 'PVS-Studio static code analyzer', 'Initial settings', 'Project analysis', 'Working with the list of diagnostic messages', and 'Context menu'. On the right, there's a 'Blog:' section with a post dated '21.10.2015' titled 'Analyzing Wine: One Year Later', with a 'Read more' link. Below it, another date '11.10.2015' is visible.

<http://www.viva64.com/>

### Text-based HEX Editor

hexedit

F2: save

F3: load file

Ctrl-X: save and exit

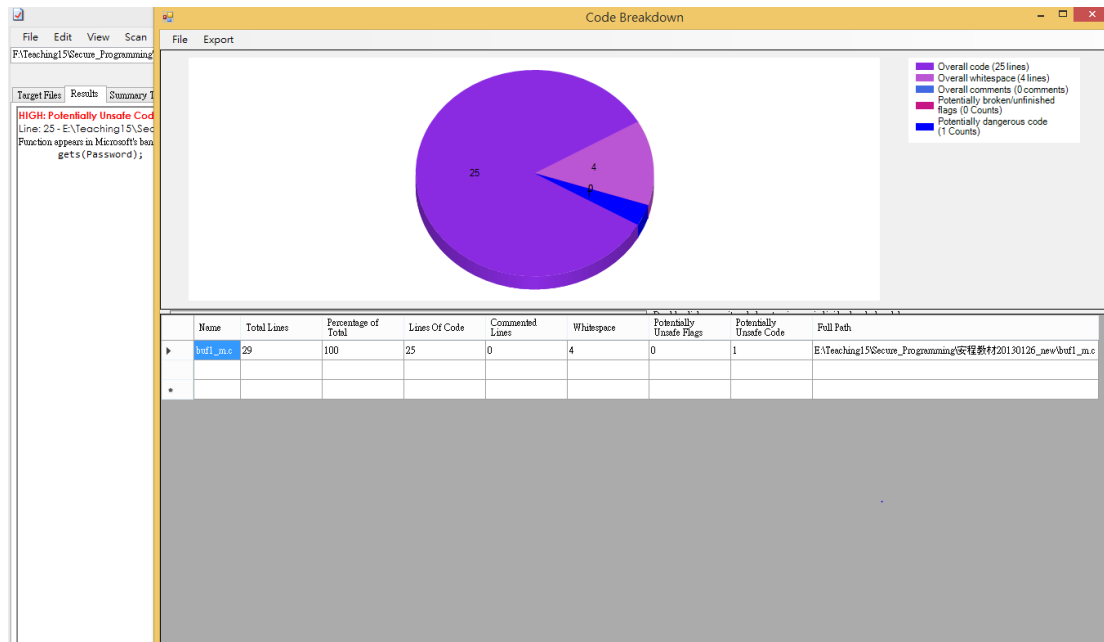
Ctrl-C: exit without saving

Esc+T - truncate the file at the current location

## VCG

Visual Codegrepp

<http://sourceforge.net/projects/visualcodegrepp/>



Other useful links

- NIST Source code security analyzers

[https://samate.nist.gov/index.php/Source\\_Code\\_Security\\_Analyzers.html](https://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html)

- SourceMeter <https://www.sourcemeter.com/>
- Static source code analysis tools for C <http://spinroot.com/static/>
- (WIKI) List of tools for static code analysis  
[https://en.wikipedia.org/wiki/List\\_of\\_tools\\_for\\_static\\_code\\_analysis](https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis)
- CPPCHECK <http://cppcheck.sourceforge.net/>
- PC-lint for C/C++ <http://www.gimpel.com/html/index.htm>

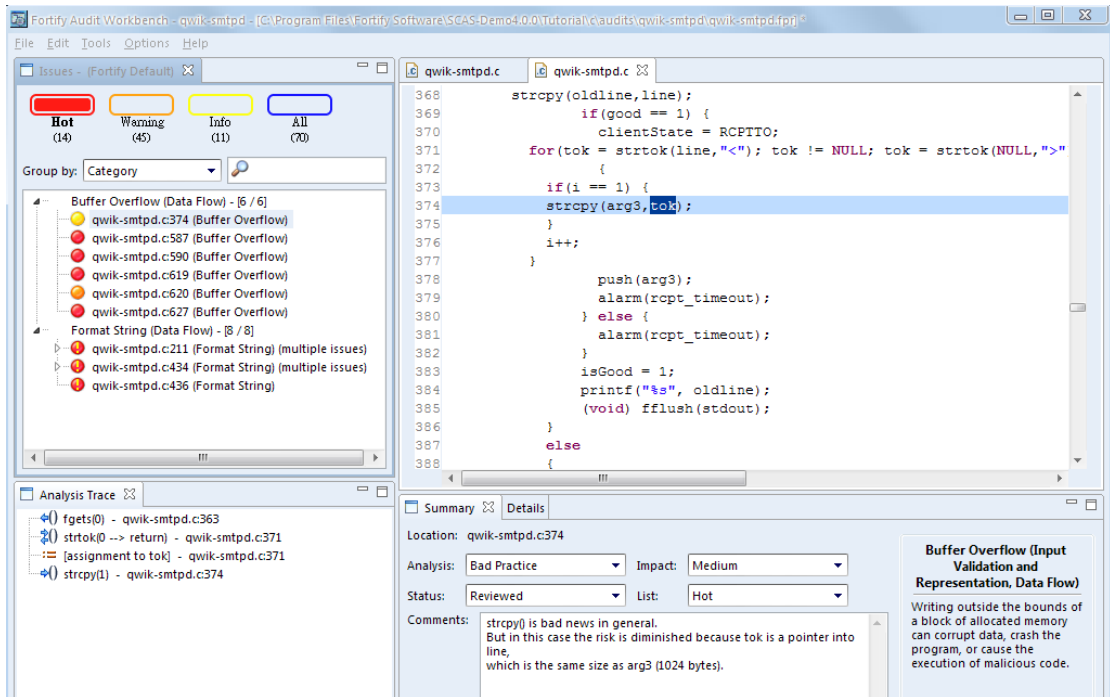
.....

## Fortify Software

### 1. Start Audit Workbench

Open the audit project:

<install\_dir>/Tutorial/c/audits/qwik-smtpd/qwik-smtpd.fpr



Read all Hot issues (buffer overflow and format string)

Generate Audit Report

### 2. Consider the source code for winner.c

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <string.h>
```

```
#define BUF_SIZE (1024)
```

```
int main(int argc, char* argv[]) {  
    char* inBuf;  
    char* outBuf;  
    char* fmt = "the winner is: %s";
```



```

inBuf = (char*) malloc(BUF_SIZE);
if (inBuf == NULL) {
    return -1;
}
read(0, inBuf, BUF_SIZE);
outBuf = (char*) malloc(BUF_SIZE);
if (outBuf == NULL) {
    return -1;
}
sprintf(outBuf, fmt, inBuf);
fprintf(stdout, "%s\n", outBuf);
fprintf(stderr, "%s\n", outBuf);
free(inBuf);
free(outBuf);
}

```

---

Answer the question: How is this code vulnerable to attack?

Open the file of winner.fpr

The screenshot displays the Fortify Static Code Analyzer (SCA) interface. On the left, the 'Issues' pane shows a list of detected problems, including a 'Memory Leak (Control Flow)' at line 12 of winner.c. The main editor shows the source code of winner.c, with line 12 highlighted: `inBuf = (char*) malloc(BUF_SIZE);`. Below the code editor, the 'Analysis Trace' pane shows the assignment to `inBuf` at line 12 and its scope ending at line 19. On the right, the 'Summary' and 'Details' tabs are visible, showing the location as `winner.c:12`, the analysis as 'Unknown', the impact as 'Unknown', the status as 'Not reviewed', and the list as 'Warning'. A 'Memory Leak (Code Control Flow)' warning is also visible on the far right, stating 'Memory is allocated but not freed.'

Read all security issues

Try to review and make comments!

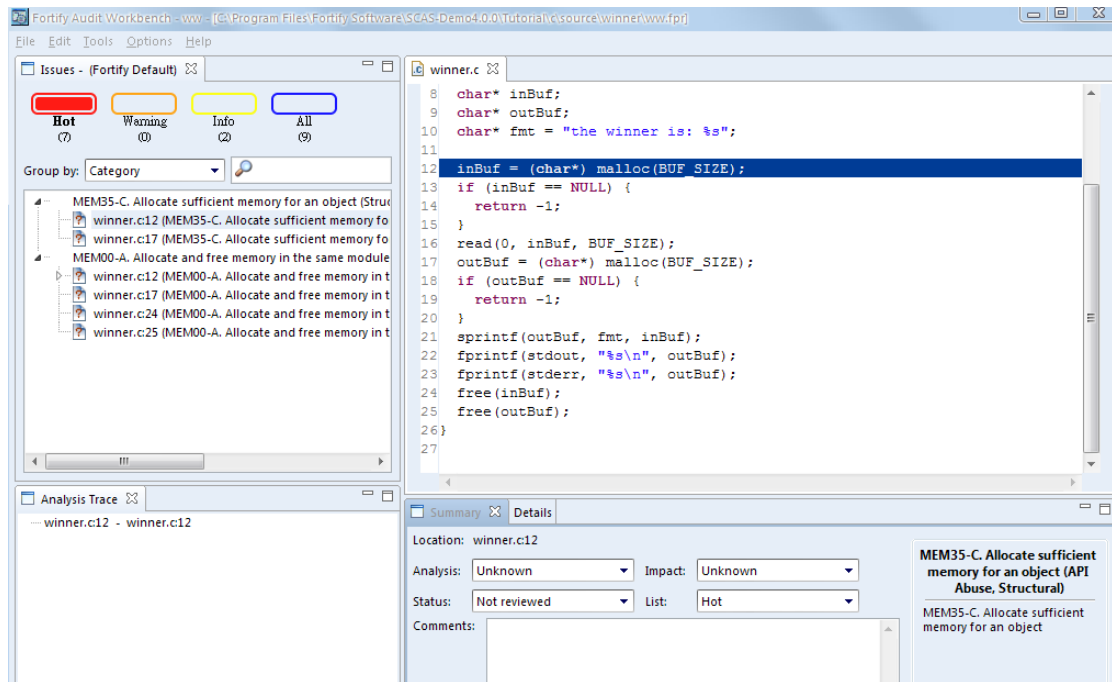
### 3. Running Fortify SCA

Using rules of CSCS-C.xml (CERT C Rule Pack)

sourceanalyzer -f ww.fpr -rules CSCS-C.xml gcc winner.c

Open ww.fpr with Audit Workbench

**Note:** You much install gcc (or cygwin)



#### 4. Test other codes

(1) Test.c from flawfinder homepage

Use flawfinder and splint

(2) Sort.c (a textbook example)

