Flawfinder – A Tool Analysis for Determining Security Weakness

Software Testing

for

Presentation ~~of~~ Exercise

Brandenburg
University of Technology
Cottbus - Senftenberg

Siddique Reza Khan

MSc in Cyber Security

Matrikel-Nr.:3846259

21.01.2019

Introduction

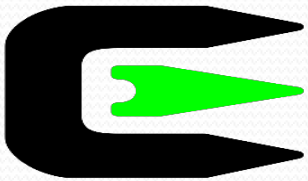
- A simple ~~program~~ tool
  - Examines C/C++ source code
  - Reports possible security weaknesses (“flaws”) sorted by risk level
- Very useful
  - Quickly finding and
  - Removing at least some potential security problems
- Open source software(OSS)
  - Free for anyone to use and
  - Is available publically
- Flawfinder works on
  - GNU/Linux systems and
  - Windows by using Cygwin.
  - It requires Python 2.7 or Python 3 to run.

Tool Comparison

- Flawfinder:
  - Scans C/C++ code
  - Database of 128 C/C++ vulnerabilities.
  - Reporting warnings of 6 levels, form level 0 to 5,
  - Level 5 only shows the most dangerous warnings.
  - Standard level in Flawfinder is level 1.
- ITS4: “It's The Software Stupid! Security Scanner”.
  - Scans C/C++ code
  - Database has 144 vulnerabilities.
  - 6 levels of reporting warnings, level 0 to 5
  - Level 5 only showed the most dangerous warnings.
  - Standard level in ITS4 is level 2.
- RATS: “Rough Automatic Tool for Security”.
  - Scans C/C++, Python, PHP and Perl source code.
  - Rats has 3 levels of reporting warnings, Low, Medium and High
  - High shows the most dangerous warnings.
  - Standard level is Medium.

Downloading and Installing

- The current version - 2.0.7
- Linux-based system
- First, download current released version flawfinder in .tar.gz (tarball) format
  - https://dwheeler.com/flawfinder/flawfinder-2.0.7.tar.gz
- Install.
  - First, uncompress the file
  - Become root to install:
    1. tar xvzf flawfinder-*.tar.gz
    2. cd flawfinder-*
  - Then install.
  - Typically you would do this (omit "sudo" if you are root):
    1. sudo make prefix=/usr install
- If you omit the "prefix=/usr" statement, it will store in the default directory /usr/local.
- If you use Windows, the recommended way is to install Cygwin first, and install it on top of Cygwin.



Source: https://en.wikipedia.org/wiki/Cygwin#/media/File:Cygwin_logo.svg

Speed

- Flawfinder is written in Python
- Python code is not as fast as C code
  1. Averaged ~~an~~ analysis speed of 45,126 lines/second
     - OS - Linux kernel
     - CPU - Intel Core 2 Duo CPU E8400
     - Speed - 3.00GHz (each CPU running at 2GHz)
  2. Averaged 24,475 lines/second
     - OS – Windows
     - Environment - 2.8GHz laptop and Cygwin
     - Cygwin on Windows tends to be much slower than Linux

How does Flawfinder Work?

- Not a sophisticated tool
- Simple tool but useful
- Built-in database of C/C++ functions
  - Buffer overflow risks (e.g., strcpy(), strcat(), gets(), sprintf(), and the scanf() family),
  - Format string problems (printf(), snprintf(), and syslog()),
  - Race conditions (such as access(), chown(), chgrp(), chmod(), tmpfile(), tmpnam(), tempnam(), and mktemp()),
  - Potential shell metacharacter dangers (most of the exec() family, system(), popen()), and
  - Poor random number acquisition (such as random()).
- ~~Needn't~~ create this database - it comes with the tool.

no need to

How does Flawfinder Work?

- Takes the source code text
- Matches the source code text against ~~those~~ names of library fucntions
- Ignoring text inside comments and strings
  - Except for flawfinder directives
- Knowledge about gettext
  - common library for internationalized programs
    - As example: _T() and _TEXT(), common Microsoft macros
  - reduces the number of false hits in internationalized programs.
- Produces a list of "hits" (potential security flaws)
  - Sorted by risk and by default the riskiest hits are shown first
  - Common Weakness Enumeration (CWE) - Compatible
  - This risk level depends
    - Both on the function and on the parameters values of the function
    - For example, constant strings are often less risky than fully variable strings in many contexts.
  - Reducing false positives
    - May be able to determine that the construct isn't risky at all

How does Flawfinder Work?

- Gives better information - and better prioritization
  - ignore comments and the insides of strings
  - examine parameters to estimate risk levels
- Fundamentally a naive program
  - Doesn't know about the data types of function parameters
  - Doesn't do control flow or data flow analysis
    - As example, other tools, like SPLINT, which do deeper analysis
- Analyze software
  - Can't build in some cases
  - Can't even locally compile

How does Flawfinder Work?

- During audited a program
  - Can mark source code lines that are actually fine but cause false warnings
  - To stop complaining about them
    - Put a specially-formatted comment either on the same line (after the source code)
    - All by itself in the previous line.
    - The comment must have one of the two following formats:
      - // Flawfinder: ignore
      - /* Flawfinder: ignore */

How does Flawfinder Work?

- Any file name can examined
  - Doesn't have a usual C/C++ filename extension
  - Can force to examine any specific files
- Examines regular files

font size too small

  - C/C++ filename extensions
  - Presumes that files are C/C++ files
    - Have the extensions ".c", ".h", ".ec", ".ecp", ".pgc", ".C", ".cpp", ".CPP", ".cxx", ".cc", ".CC", ".pcc", ".hpp", or ".H".
- Physical source lines of code (SLOC) analyzed
  - Non-blank, non-comment line
  - Number of hits at each level
  - Never be a hit at a level lower than min level (1 by default)
  - "[0] 0[1] 9" means, at level 0 total 0 hits reported, and at level 1 total 9 hits reported
  - Number of hits at a given level or larger
    - Example, level 3+ has the sum of the number of hits at level 3, 4, and 5.
  - KSLOC
    - each "level or higher" values multiplied by 1000 and divided by the physical SLOC
- The minimum risk level is displayed
  - By default this is 1

How does Flawfinder Work?

- Important reminders:
  - Not every hit is necessarily a security vulnerability
  - There may be other security vulnerabilities not reported by the tool.

try to find more specific titles ?

Options

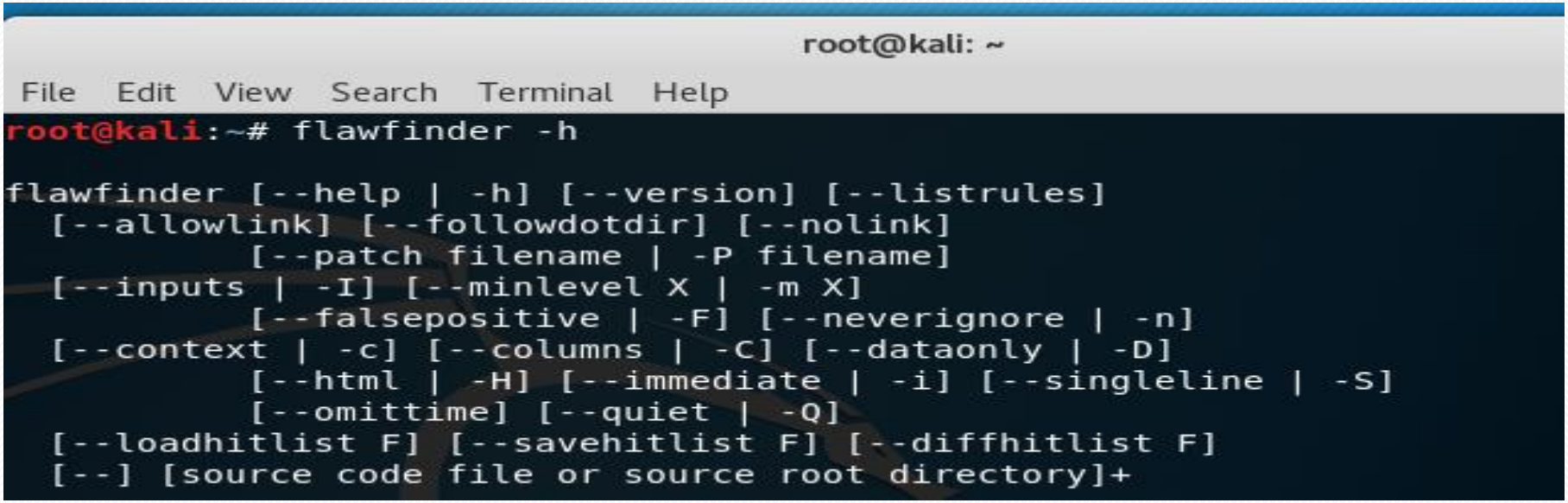
- Number of options and they are grouped into options
  - Control its own documentation
  - Select which hits to display
  - Select the output format and
  - Perform hit list management
- Long option arguments
  - Provided as “--name=value” or “-name value”.

Options : Documentation

- ––help or –h
  - Show usage (help) information.
- ––version
  - Shows (just) the version number and exits.
- ––listrules
  - List the terms (tokens) that trigger further examination
  - Default risk level, and
  - Default warning (including the CWE identifier(s), if applicable), all tab-separated.
  - Called potentially-dangerous functions.
  - Reported risk level and warning
    - Specific code may be different than the default
    - Combine with –D if you do not want the usual header

Options : Documentation

- ––help or –h
  - Show usage (help) information.



Options : Documentation

- ––version
  - Shows (just) the version number and exits.

Options : Documentation

- ––listrules
  - List the terms (tokens) that trigger further examination
    - Combine with –D if you do not want the usual header

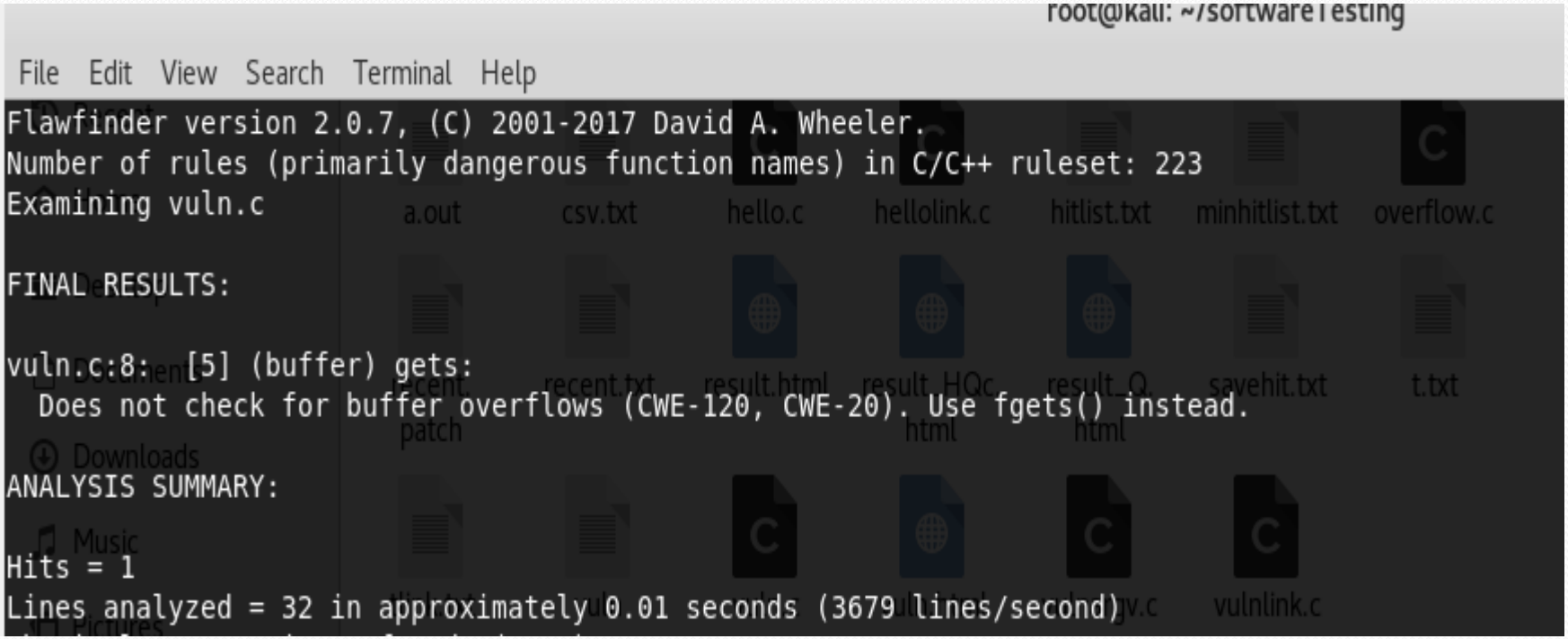
```
root@kali:~# flawfinder --listrules
Flawfinder version 2.0.7, (C) 2001-2017 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ ruleset: 223
```

Options : Selecting Hits to Display

- ––inputs or –I
  - Show only functions that obtain data from outside the program.
- ––minlevel=X or -m X
  - Set minimum risk level to X for inclusion in hit list. This can be from 0 (“no risk”) to 5 (“maximum risk”); the default is 1.
- ––falsepositive or –F
  - Do not include hits that are likely to be false positives.
  - Currently, this means that function names are ignored if they’re not followed by "(", and that declarations of character arrays aren’t noted.
- ––neverignore or -n
  - Never ignore security issues, even if they have an “ignore” directive in a comment.
- ––regexp=PATTERN or -e PATTERN
  - Only report hits with text that matches the regular expression pattern PATTERN.
    - For example, to only report hits containing the text "CWE-120", use “––regex CWE-120”.

Options : Selecting Hits to Display

- ––inputs or –I
  - Show only functions that obtain data from outside the program.



Options : Selecting Hits to Display

- ––minlevel=X or -m X
  - Set minimum risk level to X for inclusion in hit list.

```
Flawfinder version 2.0.7, (C) 2001-2017 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ ruleset: 223
Examining vuln.c
FINAL RESULTS:

vuln.c:8:  [5] (buffer) gets:
  Does not check for buffer overflows (CWE-120, CWE-20). Use fgets() instead.
vuln.c:18:  [4] (buffer) strcpy:
  Does not check for buffer overflows when copying to destination [MS-banned]
  (CWE-120). Consider using snprintf, strcpy_s, or strlcpy (warning: strncpy
  easily misused).

ANALYSIS SUMMARY:

Hits = 2
Lines analyzed = 32 in approximately 0.01 seconds (3412 lines/second)
Physical Source Lines of Code (SLOC) = 18
Hits@level = [0]   3 [1]   0 [2]   2 [3]   0 [4]   1 [5]   1
Hits@level+ = [0+]   7 [1+]   4 [2+]   4 [3+]   2 [4+]   2 [5+]   1
Hits/KSLOC@level+ = [0+] 388.889 [1+] 222.222 [2+] 222.222 [3+] 111.111 [4+] 111.111 [5+] 55.5556
Minimum risk level = 4
```

Options : Selecting Hits to Display

- ––falsepositive or –F
  - Do not include hits that are likely to be false positives.

```
Flawfinder version 2.0.7, (C) 2001-2017 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ ruleset: 223
Examining vuln.c

FINAL RESULTS:

vuln.c:8:  [5] (buffer) gets:
  Does not check for buffer overflows (CWE-120, CWE-20). Use fgets() instead.
vuln.c:18:  [4] (buffer) strcpy:
  Does not check for buffer overflows when copying to destination [MS-banned]
  (CWE-120). Consider using snprintf, strcpy_s, or strlcpy (warning: strncpy
  easily misused).

ANALYSIS SUMMARY:

Hits = 2
Lines analyzed = 32 in approximately 0.01 seconds (4690 lines/second)
Physical Source Lines of Code (SLOC) = 18
Hits@level = [0]   3 [1]   0 [2]   0 [3]   0 [4]   1 [5]   1
Hits@level+ = [0+]   5 [1+]   2 [2+]   2 [3+]   2 [4+]   2 [5+]   1
Hits/KSLOC@level+ = [0+] 277.778 [1+] 111.111 [2+] 111.111 [3+] 111.111 [4+] 111.111 [5+] 55.5556
Minimum risk level = 1
```

Options : Selecting Hits to Display

- ––neverignore or -n
  - Never ignore security issues, even if they have an "ignore" directive in a comment.

```
FINAL RESULTS:

vuln.c:8:  [5] (buffer) gets:
  Does not check for buffer overflows (CWE-120, CWE-20). Use fgets() instead.
vuln.c:18:  [4] (buffer) strcpy:
  Does not check for buffer overflows when copying to destination [MS-banned]
  (CWE-120). Consider using snprintf, strcpy_s, or strlcpy (warning: strncpy
  easily misused).
vuln.c:6:  [2] (buffer) char:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.
vuln.c:17:  [2] (buffer) char:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.

ANALYSIS SUMMARY:

Hits = 4
Lines analyzed = 32 in approximately 0.01 seconds (5109 lines/second)
Physical Source Lines of Code (SLOC) = 18
Hits@level = [0]   3 [1]   0 [2]   2 [3]   0 [4]   1 [5]   1
Hits@level+ = [0+]   7 [1+]   4 [2+]   4 [3+]   2 [4+]   2 [5+]   1
Hits/KSLOC@level+ = [0+] 388.889 [1+] 222.222 [2+] 222.222 [3+] 111.111 [4+] 111.111 [5+] 55.5556
Minimum risk level = 1
```

Options : Selecting Hits to Display

- ––regexp=PATTERN or -e PATTERN
  - Only report hits with text that matches the regular expression pattern PATTERN.

```
FINAL RESULTS:

vuln.c:8:  [5] (buffer) gets:
  Does not check for buffer overflows (CWE-120, CWE-20). Use fgets() instead.

ANALYSIS SUMMARY:

Hits = 1
Hits limited to regular expression CWE-20
Lines analyzed = 33 in approximately 0.01 seconds (4901 lines/second)
Physical Source Lines of Code (SLOC) = 18
Hits@level = [0]   0 [1]   0 [2]   0 [3]   0 [4]   0 [5]   1
Hits@level+ = [0+]   1 [1+]   1 [2+]   1 [3+]   1 [4+]   1 [5+]   1
Hits/KSLOC@level+ = [0+] 55.5556 [1+] 55.5556 [2+] 55.5556 [3+] 55.5556 [4+] 55.5556 [5+] 55.5556
Minimum risk level = 1
```

Options : Selecting Output Format

- ––columns or –C
  - Show the column number (as well as the file name and line number) of each hit;
- ––context or –c
  - Show context, i.e., the line having the "hit"/potential flaw. By default the line is shown immediately after the warning.
- ––csv
  - Generate output in comma-separated-value (CSV) format.
  - Selecting this option automatically enables ––quiet and ––dataonly.
  - The headers are mostly self-explanatory.
    - "File" is the filename,
    - "Line" is the line number,
    - "Column" is the column (starting from 1),
    - "Level" is the risk level (0-5, 5 is riskiest),
    - "Category" is the general flawfinder category,
    - "Name" is the name of the triggering rule,
    - "Warning" is text explaining why it is a hit (finding),
    - "Suggestion" is text suggesting how it might be fixed,
    - "Note" is other explanatory notes,
    - "CWEs" is the list of one or more CWEs,
    - "Context" is the source code line triggering the hit, and
    - "Fingerprint" is the SHA-256 hash of the context once its leading and trailing whitespace have been removed.

Options : Selecting Output Format

- ––columns or –C
  - Show the column number (as well as the file name and line number) of each hit;

```
Flawfinder version 2.0.7, (C) 2001-2017 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ ruleset: 223
Examining vuln.c

FINAL RESULTS:

vuln.c:8:2:  [5] (buffer) gets:
  Does not check for buffer overflows (CWE-120, CWE-20). Use fgets() instead.
vuln.c:17:2:  [4] (buffer) strcpy:
  Does not check for buffer overflows when copying to destination [MS-banned]
  (CWE-120). Consider using snprintf, strcpy_s, or strlcpy (warning: strncpy
  easily misused).
vuln.c:6:2:  [2] (buffer) char:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.
vuln.c:16:2:  [2] (buffer) char:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.
```

Options : Selecting Output Format

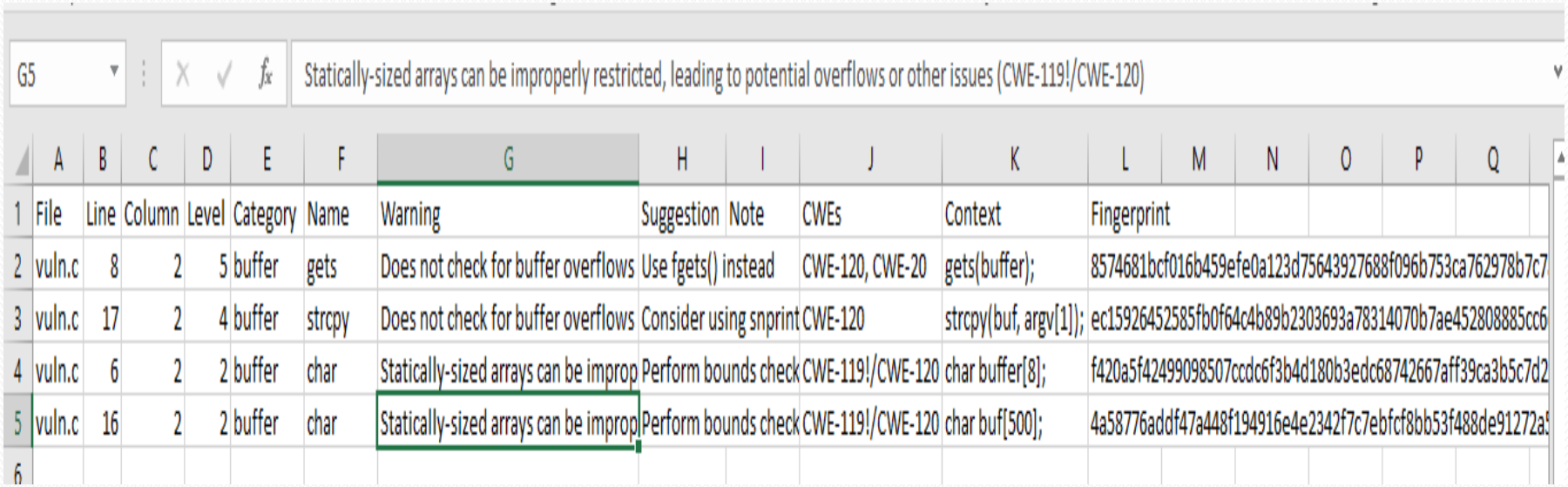
- ––context or –c
  - Show context, i.e., the line having the "hit"/potential flaw. By default the line is shown immediately after the warning.

```
FINAL RESULTS:

vuln.c:8:  [5] (buffer) gets:
  Does not check for buffer overflows (CWE-120, CWE-20). Use fgets() instead.
        gets(buffer);
vuln.c:17:  [4] (buffer) strcpy:
  Does not check for buffer overflows when copying to destination [MS-banned]
  (CWE-120). Consider using snprintf, strcpy_s, or strlcpy (warning: strncpy
  easily misused).
        strcpy(buf, argv[1]);
vuln.c:6:  [2] (buffer) char:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.
        char buffer[8];
vuln.c:16:  [2] (buffer) char:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.
        char buf[500];
.
```

Options : Selecting Output Format

- ––csv
  - Generate output in comma-separated-value (CSV) format.



G5 | Statically-sized arrays can be improperly restricted, leading to potential overflows or other issues (CWE-119!/CWE-120)

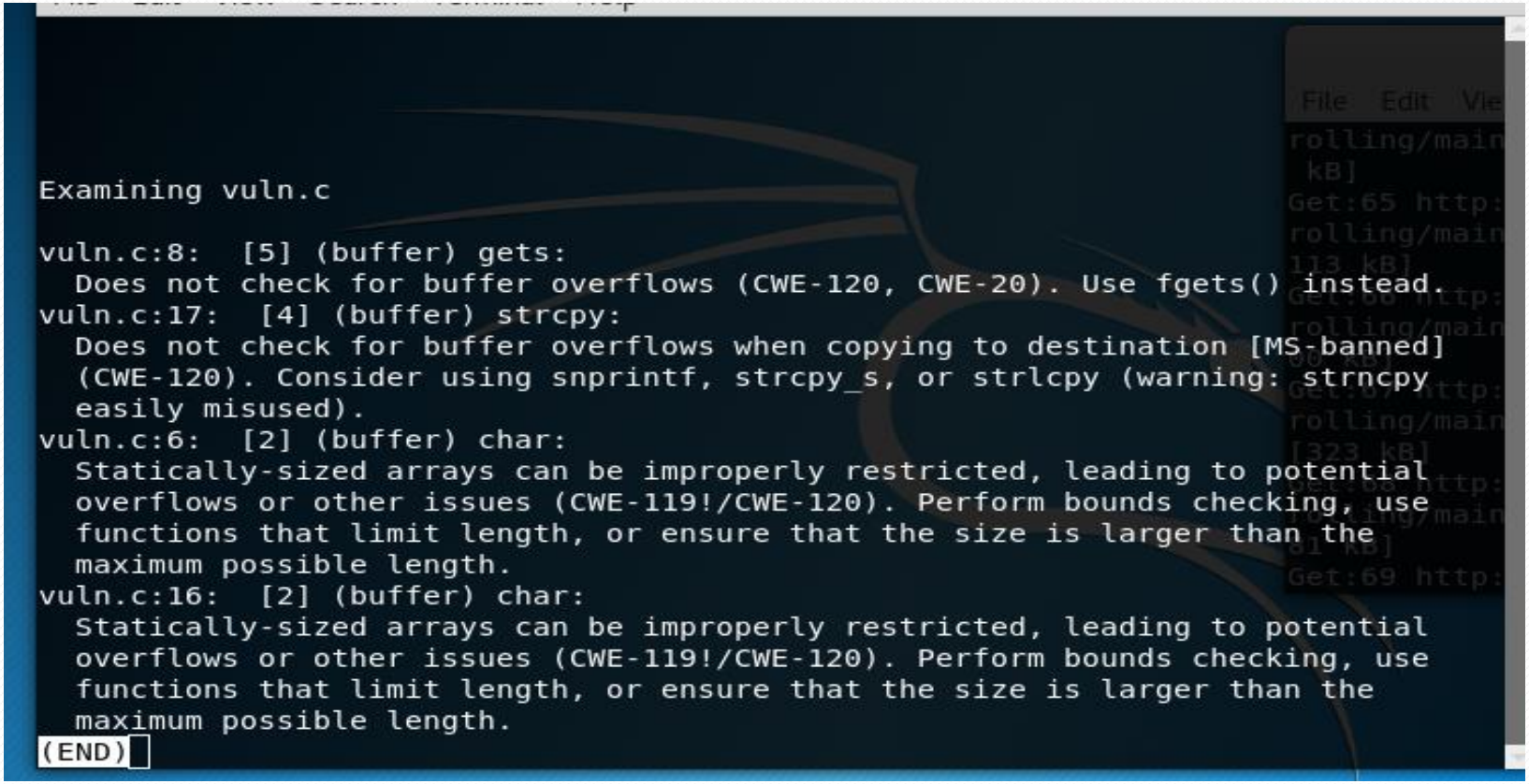
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	File	Line	Column	Level	Category	Name	Warning	Suggestion	Note	CWEs	Context	Fingerprint					
2	vuln.c	8	2	5	buffer	gets	Does not check for buffer overflows	Use fgets() instead		CWE-120, CWE-20	gets(buffer);	8574681bcf016b459efe0a123d75643927688f096b753ca762978b7c7					
3	vuln.c	17	2	4	buffer	strcpy	Does not check for buffer overflows	Consider using snprint		CWE-120	strcpy(buf, argv[1]);	ec15926452585fb0f64c4b89b2303693a78314070b7ae452808885cc6					
4	vuln.c	6	2	2	buffer	char	Statically-sized arrays can be improp	Perform bounds check		CWE-119!/CWE-120	char buffer[8];	f420a5f42499098507ccdc6f3b4d180b3edc68742667aff39ca3b5c7d2					
5	vuln.c	16	2	2	buffer	char	Statically-sized arrays can be improp	Perform bounds check		CWE-119!/CWE-120	char buf[500];	4a58776addf47a448f194916e4e2342f7c7ebfcf8bb53f488de91272a5					
6																	

Options : Selecting Output Format

- –D
  - Don't display the header and footer. Use this along with ––quiet to see just the data itself.
- ––html or –H
  - Format the output as HTML instead of as simple text.
- ––immediate or -i
  - Immediately display hits (don't just wait until the end).
- ––singleline or -S
  - Display as single line of text output for each hit. Useful for interacting with compilation tools.
- ––omittime
  - Omit timing information. This is useful for regression tests of flawfinder itself, so that the output doesn't vary depending on how long the analysis takes.
- ––quiet or –Q
  - Don't display status information (i.e., which files are being examined) while the analysis is going on.

Options : Selecting Output Format

- –D
  - Don't display the header and footer. Use this along with ––quiet to see just the data itself.



Options : Selecting Output Format

- ––html or –H
  - Format the output as HTML instead of as simple text.

Flawfinder Results

Here are the security scan results from Flawfinder version 2.0.7, (C) 2001-2017 David A. Wheeler. Number of rules (primarily dangerous function names) in C/C++ ruleset: 223

- vuln.c:8: **[5]** (buffer) *gets: Does not check for buffer overflows (CWE-120, CWE-20). Use fgets() instead.*

```
gets(buffer);
```

- vuln.c:17: **[4]** (buffer) *strcpy: Does not check for buffer overflows when copying to destination [MS-banned] (CWE-120). Consider using snprintf, strcpy_s, or strlcpy (warning: strncpy easily misused).*

```
strcpy(buf, argv[1]);
```

- vuln.c:6: **[2]** (buffer) *char: Statically-sized arrays can be improperly restricted, leading to potential overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use functions that limit length, or ensure that the size is larger than the maximum possible length.*

```
char buffer[8];
```

- vuln.c:16: **[2]** (buffer) *char: Statically-sized arrays can be improperly restricted, leading to potential overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use functions that limit length, or ensure that the size is larger than the maximum possible length.*

```
char buf[500];
```

Analysis Summary

Hits = 4

Options : Selecting Output Format

- ––immediate or -i
  - Immediately display hits (don't just wait until the end).

```
Flawfinder version 2.0.7, (C) 2001-2017 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ ruleset: 223
Examining vuln.c
vuln.c:6:  [2] (buffer) char:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.
vuln.c:7:  [0] (format) printf:
  If format strings can be influenced by an attacker, they can be exploited
  (CWE-134). Use a constant for the format specification. Constant format
  string, so not considered risky.
vuln.c:8:  [5] (buffer) gets:
  Does not check for buffer overflows (CWE-120, CWE-20). Use fgets() instead.
vuln.c:9:  [0] (format) printf:
  If format strings can be influenced by an attacker, they can be exploited
  (CWE-134). Use a constant for the format specification. Constant format
  string, so not considered risky.
vuln.c:16:  [2] (buffer) char:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.
vuln.c:17:  [4] (buffer) strcpy:
  Does not check for buffer overflows when copying to destination [MS-banned]
  (CWE-120). Consider using snprintf, strcpy_s, or strlcpy (warning: strncpy
  easily misused).
vuln.c:19:  [0] (format) printf:
  If format strings can be influenced by an attacker, they can be exploited
  (CWE-134). Use a constant for the format specification. Constant format
  string, so not considered risky.
```

Options : Selecting Output Format

- ––singleline or -S
  - Display as single line of text output for each hit. Useful for interacting with compilation tools.

```
Flawfinder version 2.0.7, (C) 2001-2017 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ ruleset: 223
Examining vuln.c

FINAL RESULTS:

vuln.c:8:  [5] (buffer) gets:Does not check for buffer overflows (CWE-120, CWE-20).  Use fgets() instead.
vuln.c:17:  [4] (buffer) strcpy:Does not check for buffer overflows when copying to destination [MS-banned] (CWE-120).  Consider using snprintf, strc
py_s, or strlcpy (warning: strncpy easily misused).
vuln.c:6:  [2] (buffer) char:Statically-sized arrays can be improperly restricted, leading to potential overflows or other issues (CWE-119!/CWE-120).
  Perform bounds checking, use functions that limit length, or ensure that the size is larger than the maximum possible length.
vuln.c:16:  [2] (buffer) char:Statically-sized arrays can be improperly restricted, leading to potential overflows or other issues (CWE-119!/CWE-120)
.  Perform bounds checking, use functions that limit length, or ensure that the size is larger than the maximum possible length.

ANALYSIS SUMMARY:

Hits = 4
```

Options : Selecting Output Format

- --omittime
  - Omit timing information. This is useful for regression tests of flawfinder itself, so that the output doesn't vary depending on how long the analysis takes.

```
ANALYSIS SUMMARY:

Hits = 4
Lines analyzed = 28 in approximately 0.01 seconds (2839 lines/second)
```

```
ANALYSIS SUMMARY:

Hits = 4
Lines analyzed = 28
Physical Source Lines of Code (SLOC) = 18
Hits@level = [0]   3 [1]   0 [2]   2 [3]   0 [4]   1 [5]   1
Hits@level+ = [0+]   7 [1+]   4 [2+]   4 [3+]   2 [4+]   2 [5+]   1
Hits/KSLOC@level+ = [0+] 388.889 [1+] 222.222 [2+] 222.222 [3+] 111.111 [4+] 111.111 [5+] 55.5556
Minimum risk level = 1
Not every hit is necessarily a security vulnerability.
There may be other security vulnerabilities; review your code!
```

Options : Selecting Output Format

- –quiet or –Q
  - Don't display status information (i.e., which files are being examined) while the analysis is going on.

```
Flawfinder version 2.0.7, (C) 2001-2017 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ ruleset: 223
vuln.c:8:  [5] (buffer) gets:
  Does not check for buffer overflows (CWE-120, CWE-20). Use fgets() instead.
vuln.c:17:  [4] (buffer) strcpy:
  Does not check for buffer overflows when copying to destination [MS-banned]
  (CWE-120). Consider using snprintf, strcpy_s, or strlcpy (warning: strncpy
  easily misused).
vuln.c:6:  [2] (buffer) char:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.
vuln.c:16:  [2] (buffer) char:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.
```

Options : Hitlist Management

- ––savehitlist=F
  - Save all resulting hits (the "hitlist") to F.
- ––loadhitlist=F
  - Load the hitlist from F instead of analyzing source programs.
    - Warning: Do not load hitlists from untrusted sources (for security reasons).

Options : Hitlist Management

- ––savehitlist=F
  - Save all resulting hits (the "hitlist") to F

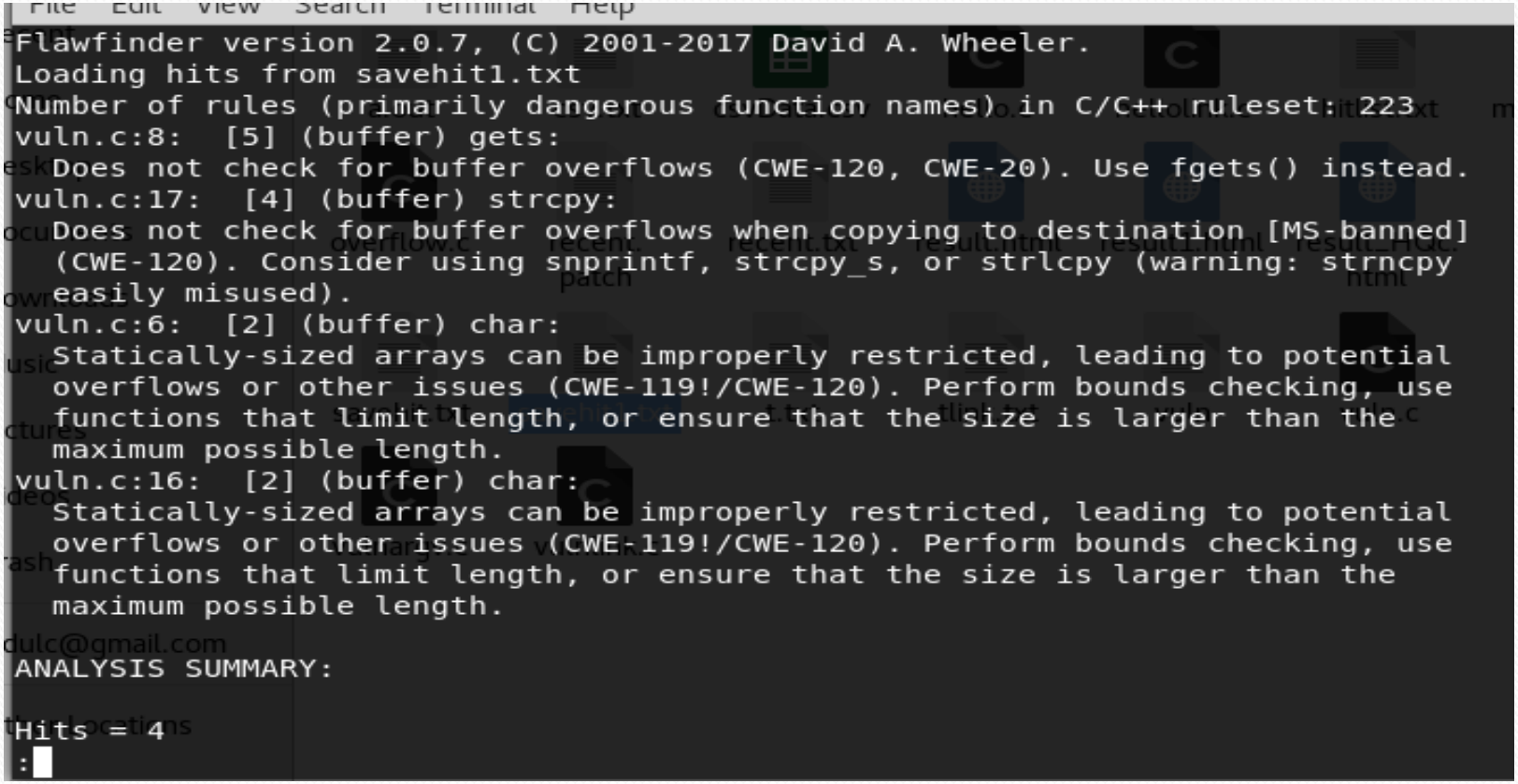
```
Flawfinder version 2.0.7, (C) 2001-2017 David A. Wheeler.
Saving hitlist to savehit1.txt
Number of rules (primarily dangerous function names) in C/C++ ruleset: 223
vuln.c:8:  [5] (buffer) gets:
  Does not check for buffer overflows (CWE-120, CWE-20). Use fgets() instead.
vuln.c:17:  [4] (buffer) strcpy:
  Does not check for buffer overflows when copying to destination [MS-banned]
  (CWE-120). Consider using snprintf, strcpy_s, or strlcpy (warning: strncpy
  easily misused).
vuln.c:6:  [2] (buffer) char:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.
vuln.c:16:  [2] (buffer) char:
  Statically-sized arrays can be improperly restricted, leading to potential
  overflows or other issues (CWE-119!/CWE-120). Perform bounds checking, use
  functions that limit length, or ensure that the size is larger than the
  maximum possible length.

ANALYSIS SUMMARY:

Hits = 4
:
```

Options : Hitlist Management

- --loadhitlist=F
  - Load the hitlist from F instead of analyzing source programs



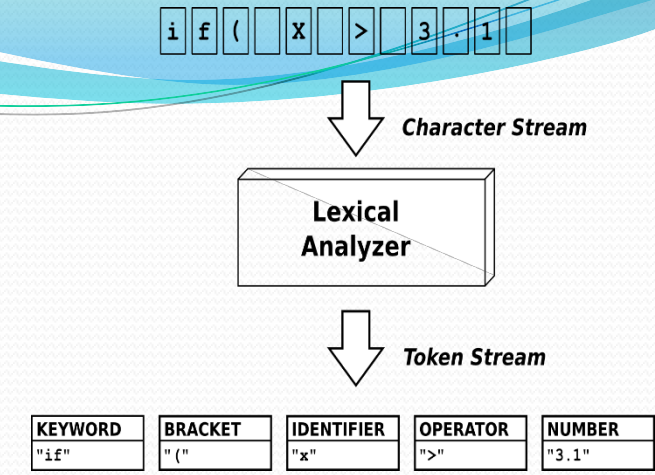
explain term; where from, give reference

COMMON WEAKNESS ENUMERATION (CWE)

- Flawfinder can report on the following CWEs (these are the CWEs that flawfinder covers; "*"marks those in the CWE/SANS top 25 list):
- CWE-20: Improper Input Validation
- CWE-22: Improper Limitation of a Pathname to a Restricted Directory ("Path Traversal")
- CWE-78: Improper Neutralization of Special Elements used in an OS Command ("OS Command Injection")*
- CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer (a parent of CWE-120*, so this is shown as CWE-119!/CWE-120)
- CWE-120: Buffer Copywithout Checking Size of Input ("Classic Buffer Overflow")*
- CWE-126: Buffer Over-read
- CWE-134: Uncontrolled Format String*
- CWE-190: Integer Overflow or Wraparound*
- CWE-250: Execution with Unnecessary Privileges
- CWE-327: Use of a Broken or RiskyCryptographic Algorithm*
- CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ("Race Condition")
- CWE-377: Insecure Temporary File
- CWE-676: Use of Potentially Dangerous Function*
- CWE-732: Incorrect Permission Assignment for Critical Resource*
- CWE-785: Use of Path Manipulation Function without Maximum-sized Buffer (child of CWE-120*, so this is shown as CWE-120/CWE-785)
- CWE-807: Reliance on Untrusted Inputs in a Security Decision*
- CWE-829: Inclusion of Functionality from Untrusted Control Sphere*

make two slides, or shorten

Conclusion



Source: http://quex.sourceforge.net/
It is licensed under MIT License.

- Useful as a simple introduction
  - To static analysis tools
  - Easy to start using and easy to understand.
- Works by doing simple lexical tokenization
  - Looking for token matches to the database
  - Particularly to find function calls
- Examines the text of the function parameters to estimate risk
- Does not use or have access to information about control flow, data flow,
  - when searching for potential vulnerabilities or estimating the level of risk.
- Analyze a *copy* of the source code
- Can find vulnerabilities in programs that cannot be built or cannot be linked.
  - It can often work with programs that cannot even be compiled

Conclusion

- Not every hit a security vulnerability
- Not every security vulnerability is necessarily found
  - Doesn't understand the semantics of the code
  - Does simple text pattern matching
  - Ignoring comments and strings
  - Doesn't do data flow or control flow or data types analysis
    - Produce many false positives for vulnerabilities
    - Fail to report many vulnerabilities

Tool Demonstration

Example: For Demonstration C Code

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

void GetInput()
{
    char buffer[8];
    printf("Give input for function \"gets() \" testing buffer: ");
    gets(buffer);
    printf("Function \"gets() \" testing buffer: ");
    puts(buffer);
}


int main (int argc, char** argv)
{
    char buf[500];
    strcpy(buf, argv[1]);
    GetInput();
    printf("Function \"strcpy() \" testing buffer:  %s\n", buf);
    return 0;
}

//Comment:
//----------------------------------------
    /*Flawfinder:ignore*/

    // Flawfinder: ignore
//----------------------------------------
```

Options : Example

- flawfinder -m 4 vuln.c | less
  - Examine the C/C++ files in the current directory only report vulnerabilities level 4 and up (the two highest risk levels).
- flawfinder -I vuln.c| less
  - Examine the C/C++ files in mydir, and report functions that take inputs (so that you can ensure that they filter the inputs appropriately).
- flawfinder -n vuln.c | less
  - Examine the C/C++ files in the directory mydir or its subdirectories, including even the hits marked for ignoring in the code comments.
- flawfinder --csv vuln.c >csvData.csv
  - Examine the current directory down and report all hits in CSV format.
- flawfinder -QD vuln.c | less
  - Examine mydir and report only the actual results (removing the header and footer of the output).

Options : Example

- flawfinder -QDSC vuln.c|less
  - Examine mydir, reporting only the actual results (no header or footer). Each hit is reported on one line, and column numbers are reported. This can be a useful command if you are feeding flawfinder output to other tools.
- flawfinder -QHc vuln.c>result.html
  - Examine all the C/C++ files in the directory mydir, and produce an HTML formatted version of the results.
- flawfinder -Q --savehitlist savehit.txt vuln.c|less
  - Examine file in the current directory. Don't report on the status of processing, and save the resulting hitlist (the set of all hits) in the file savehit.txt.
- flawfinder --loadhitlist savehit.txt vuln.c|less
  - Examine file in the current directory, and show hits that were already in the file savehit.txt.
- flawfinder --regex "CWE-119|CWE-120" vuln.c | less
  - Examine file, but only report hits where CWE-119 or CWE-120 apply.

Questions?

Bibliography

- [1] David A. Wheeler, “Secure Programming HOWTO”, v3.72 Edition, 2015.
- [2]Online: https://dwheeler.com/flawfinder/, “Flawfinder”, accessed on: 02.11.2018
- [3]Online:https://www.debian.org/security/audit/examples/flawfinder, “Automated Audit Example: flawfinder”, accessed on: 02.11.2018
- [4]Online: https://dwheeler.com/secure-class/index.html, “Secure Software Design and Programming: Class Materials by David A. Wheeler”, accessed on: 02.11.2018
- [5]Online:http://manpages.ubuntu.com/manpages/bionic/man1/flawfinder.1.html, “flawfinder - lexically find potential security flaws ("hits") in source code ”, accessed on: 02.11.2018
- [6]Online: https://dwheeler.com/flawfinder/flawfinder.pdf, ”Documentation - Flawfinder ”, accessed on: 02.11.2018
- [7]Online: Daniel Persson, Dejan Baca, Software Security Analysis - Managing source code audit, https://www.diva-portal.org/smash/get/diva2:830925/FULLTEXT01.pdf, accessed on: 19.12.2018
- [8]Online: http://cwe.mitre.org/top25/ , “2011 CWE/SANS Top 25 Most Dangerous Software Errors” , accessed on: 07.01.2019