## NAME

flawfinder − find potential security flaws ("hits") in source code

## SYNOPSIS

**flawfinder** [**--context**] [**-c**] [**--columns**] [ **-m** *X* ] [ **-minlevel=***X* ] [**--immediate**] [**-i**] [**--inputs**] [**-n**] [**--neverignore**] [ **--loadhitlist=***F* ] [ **--savehitlist=***F* ] [ **--diffhitlist=***F* ] [**--**] *[ source code file ]+*

## DESCRIPTION

Flawfinder searches through source code looking for potential security flaws. Flawfinder uses an internal database called the ‘‘ruleset’’; the ruleset identifies functions that are common causes of security flaws. Every potential security flaw found in a given source code file (matching an entry in the ruleset) is called a ‘‘hit,’’ and the set of hits found during any particular run is called the ‘‘hitlist.’’

To run flawfinder, simply give flawfinder a list of filenames of C/C++ source code, and it will provide a list of hits, sorted by risk. The most potentially dangerous hits are shown first.

The risk level varies from 0, little risk, to 5, great risk. This risk level depends not only on the function, but on the values of the parameters of the function. For example, constant strings are often less risky than fully variable strings in many contexts. Flawfinder knows about gettext and will treat constant strings passed through gettext as though they were constant strings. Flawfinder ignores text inside comments and strings, and will only respond to identifiers listed in its ruleset.

Typical use is to apply flawfinder to a set of source code and examine the highest-risk items. Then, you can use --input to examine the input locations, and check to make sure that only legal values are accepted from untrusted users.

Once you’ve audited a program, you can mark source code lines that are actually fine but cause spurious warnings so that flawfinder will stop complaining about them. To mark a line, put a specially-formatted comment either on the same line (after the source code) or all by itself in the previous line. The comment must have one of the two following formats:

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

Note that, for compatibility’s sake, you can replace "Flawfinder" with "ITS4" in these specially-formatted comments. Since it’s possible that such lines are wrong, you can use the ‘‘--neverignore’’ option; this includes responses that would otherwise be ignore (or, more confusingly, it ignores the ignores). This comment syntax is actually a more general syntax for special directives to flawfinder, but currently only ignoring lines is supported.

Flawfinder intentionally works similarly to another program, ITS4, which is not fully open source software (as defined in the Open Source Definition) nor free software (as defined by the Free Software Foundation). The author of Flawfinder has never seen ITS4’s source code.

## OPTIONS

**--context**

**-c** Show context (the line having the "hit"/potential flaw)

**--columns** Show the column number (as well as the file name and line number) of each hit; this is shown after the line number by adding a colon and the column number in the line (the first character in a line is column number 1).

**-m** *X*

**--minlevel=***X* Set minimum risk level to X for inclusion in hitlist. This can be from 0 (‘‘no risk’’) to 5 (‘‘maximum risk’’); the default is 1.

**--neverignore**

**-n** Never ignore security issues, even if they have an ‘‘ignore’’ directive in a comment.

**−−immediate**

**-i** Immediately display hits (don’t just wait until the end).

**−−inputs** Show only functions that obtain data from outside the program; this also sets minlevel to 0.

**--loadhitlist=***F*
Load hits from F instead of analyzing source programs.

**--savehitlist=***F*
Save all hits (the "hitlist") to F.

**--diffhitlist=***F*
Show only hits (loaded or analyzed) not in F.

## BUGS

The database is currently woefully incomplete; this is merely the first release! Interestingly enough, it can still detect some problems in real programs in this early form.

Flawfinder is currently limited to C/C++. It’s designed so that adding support for other languages should be easy.

Flawfinder doesn’t fully parse C/C++ code. Thus, it can be fooled by user-defined functions or method names that happen to be the same as those defined as ‘‘hits’’ in its database.

Security vulnerabilities might not be identified as such by flawfinder, and conversely, some hits aren’t really security vulnerabilities.

## SEE ALSO

See the flawfinder website at http://www.dwheeler.com/flawfinder. You should also see the *Secure Programming for Unix and Linux HOWTO* at http://www.dwheeler.com/secure-programs.

## AUTHOR

David A. Wheeler (dwheeler@dwheeler.com).