NAME

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

SYNOPSIS

flawfinder [**−−help**] [**−−version**] [**−−context**] [**-c**] [**−−columns**] [**-m** *X*] [**-minlevel=***X*] [**−−immediate**] [**-i**] [**−−inputs**] [**-n**] [**−−neverignore**] [**−−quiet**] [**−−loadhitlist=***F*] [**−−savehitlist=***F*] [**−−diffhitlist=***F*] [**−−**] *[source code file or source root directory]+*

DESCRIPTION

Flawfinder searches through C/C++ source code looking for potential security flaws. To run flawfinder, simply give flawfinder a list of directories or files. For each directory given, all C/C++ files in that directory (and its subdirectories, recursively) will be examined. When flawfinder is given a directory name, files are determined to be a C/C++ file from its filename extension. Thus, for most projects, simply give flawfinder the name of the source code's topmost directory, and flawfinder will examine all of the project's source code. Any filename given on the command line will be examined (even if it doesn't have a usual C/C++ filename extension); thus you can force flawfinder to examine any specific files you desire. Flawfinder will produce a list of ''hits'' (potential security flaws), sorted by risk; the riskiest hits are shown first.

The risk level varies from 0, very 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 (a common library for internationalized programs) and will treat constant strings passed through gettext as though they were constant strings; this reduces the number of false hits in internationalized programs. Flawfinder correctly ignores most text inside comments and strings.

Not every hit is actually a security vulnerability, and not every security vulnerability is necessarily found. Nevertheless, flawfinder can be an aid in finding and removing security vulnerabilities. A common way to use flawfinder is to first apply flawfinder to a set of source code and examine the highest-risk items. Then, use −−input to examine the input locations, and check to make sure that only legal and safe input 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:" or "RATS:" in these specially-formatted comments. Since it's possible that such lines are wrong, you can use the ''−−neverignore'' option, which causes flawfinder to never ignore any line no matter what the comments say. Thus, responses that would otherwise be ignored would be included (or, more confusingly, −−neverignore 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 uses an internal database called the ''ruleset''; the ruleset identifies functions that are common causes of security flaws. As noted above, 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 of the program is called the ''hitlist.'' Hitlists can be saved (using −−savehitlist), reloaded back for redisplay (using −−loadhitlist), and you can show only the hits that are different from another run (using −−diffhitlist).

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

−−help Show usage (help) information.

−−version Shows (just) the version number and exits.

−−context

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

−−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.

−−quiet Don’t display status information (i.e., which files are being examined) while the analysis is going on.

−−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.

EXAMPLES

flawfinder /usr/src/linux-2.4.12
Examine all the C/C++ files in the directory /usr/src/linux-2.4.12 and all its subdirectories (recursively), reporting on all hits found.

flawfinder −−quiet −−savehitlist saved.hits *.[ch]
Examine all .c and .h files 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 saved.hits.

flawfinder −−diffhitlist saved.hits *.[ch]
Examine all .c and .h files in the current directory, and show any hits that weren't already in the file saved.hits.

BUGS

The database is currently woefully incomplete. 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.

Preprocessor commands embedded in the middle of a parameter list of a call can cause problems in parsing, in particular, if a string is opened and then closed multiple times using an #ifdef .. #else construct, flawfinder gets confused. Such constructs are exteremly nasty; for now, rewrite the construct.

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).