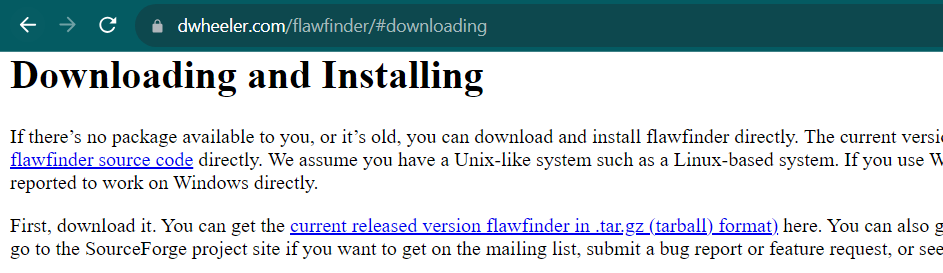
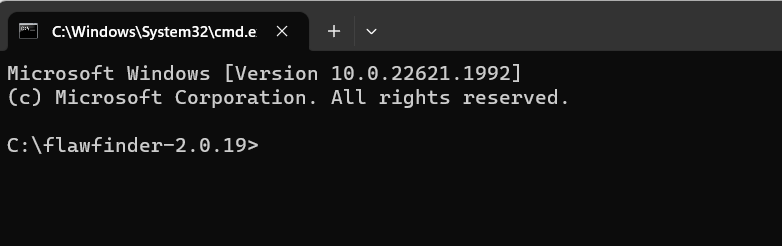
## **# Steps to download Flawfinder:**

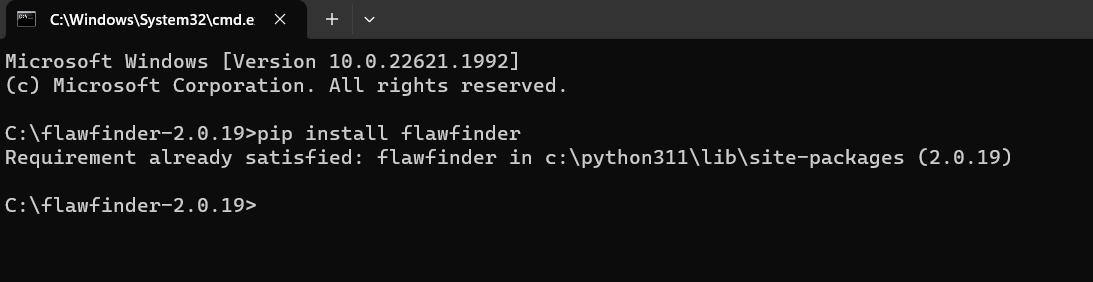
Step 1: Visit <https://dwheeler.com/flawfinder/> and download the current released version of flawfinder in ‘*.tar.gz (tarball)’* format.



Step 2:Extract the *‘flawfinder-2.0.19.tar’* file and open command prompt within it.

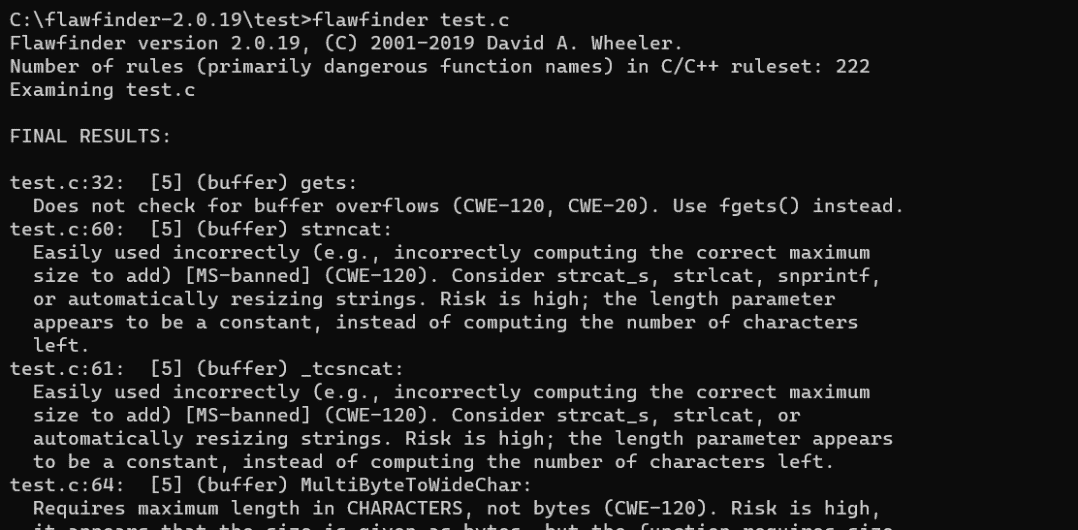
Step 3:Run the following command to install flawfinder:

pip install flawfinder



Step 4: We can perform a check to confirm that Flawfinder is working on our system by running a test file which is available on the official website. We use the following command:

flawfinder test.c



## **# Static code analysis using open-source tool Flawfinder for the following: -**

* **Buffer overflow risk:**

A buffer is a temporary area for data storage. When more data (than was originally allocated to be stored) gets placed by a program or system process, the extra data overflows. It causes some of that data to leak out into other buffers, which can corrupt or overwrite whatever data they were holding.

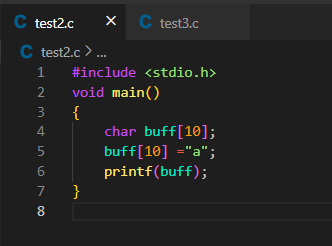
Certain languages allow direct addressing of memory locations and do not automatically ensure that these locations are valid for the memory buffer that is being referenced. This can cause read or write operations to be performed on memory locations that may be associated with other variables, data structures, or internal program data.

As a result, an attacker may be able to execute arbitrary code, alter the intended control flow, read sensitive information, or cause the system to crash.

**Steps to check for buffer overflows using Flawfinder:**

Step 1: Create a C file having the following program mentioned below.

In this program, we have declared a string ‘buff’ with size 10 but we are trying to access its 11th element which is not possible. This will give us ‘a buffer overflow’ hit in flawfinder.



Step 2: Run the following command to view the output in an HTML document using Flawfinder.

flawfinder --html test2.c > result2.html

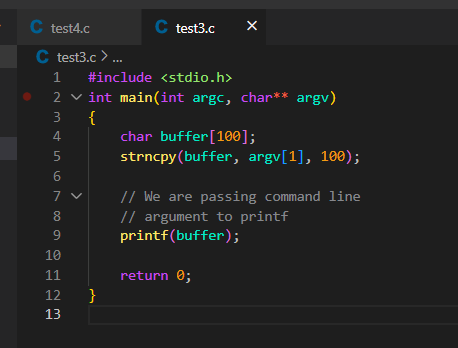
* **Format string problem**

A format string is an ASCII string that contains text and format parameters. Format string vulnerabilities are a class of bug that take advantage of an easily avoidable programmer error.

Format string attack surfaces when the data, which an input printf() string delivers, is considered or executed as a command by the software. When it happens, the attacker can easily insert malicious code in the input string or access stack, and even cause temporary or permanent software execution failure. When an attacker can modify an externally-controlled format string, this can also lead to buffer overflows, denial of service, or data representation problems.

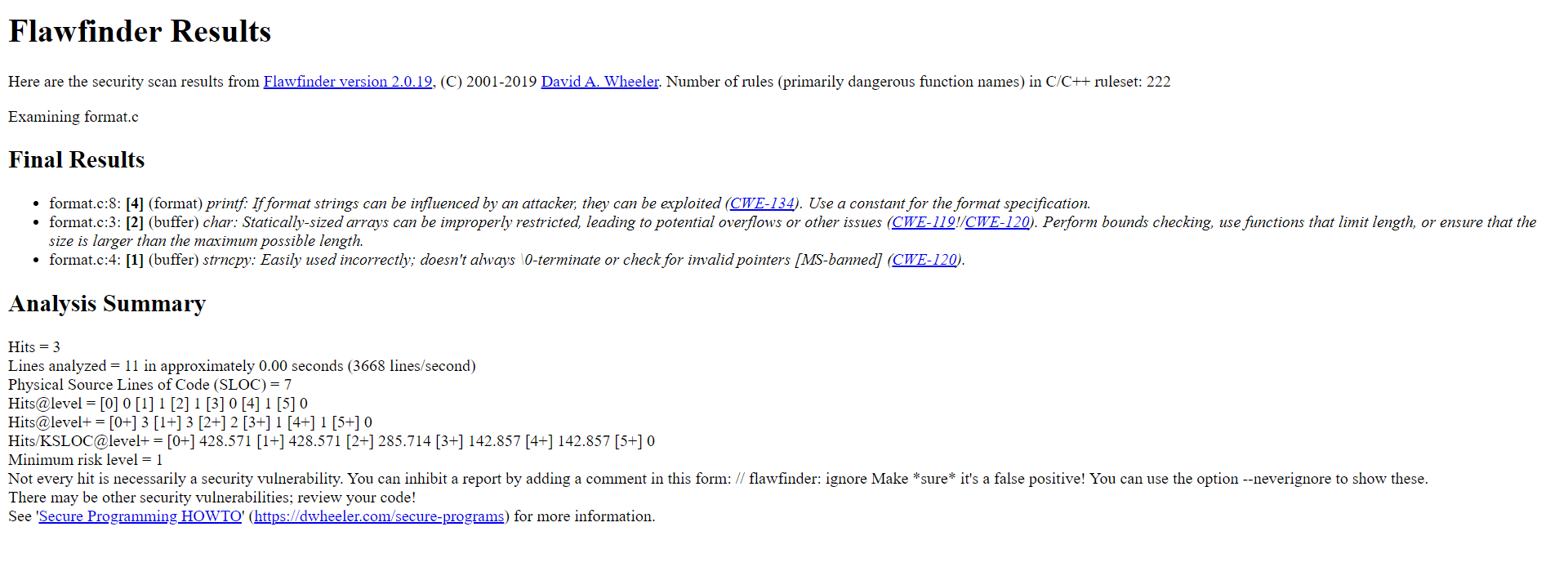
**Steps to check for format string problems using Flawfinder:**

Step 1: Create a C file having the following program mentioned below. As C language does not support single quotes as string statement, this should give us a ‘format string’ hit in Flawfinder.



Step 2: Run the following command to view output in an HTML document using Flawfinder.

flawfinder --html test3.c > result3.html



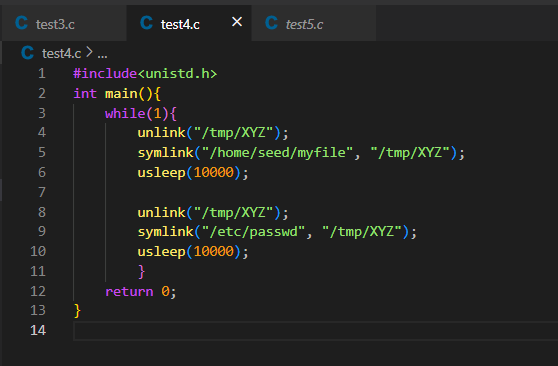
* **Race conditions**

When two concurrent threads in execution access a shared resource in a way that it unintentionally produces different results depending on the timing of the threads or processes, this gives rise to a Race Condition. If our privileged program (application with elevated access control) somehow also has a code block with race-condition vulnerability, then attackers can exploit this by running parallel processes which can “race” against our privileged program. If they attack with an intention to change its actual intended behaviour then this can cause Race Conditions (RC’s) in our program.

**Steps to check for race conditions using Flawfinder:**

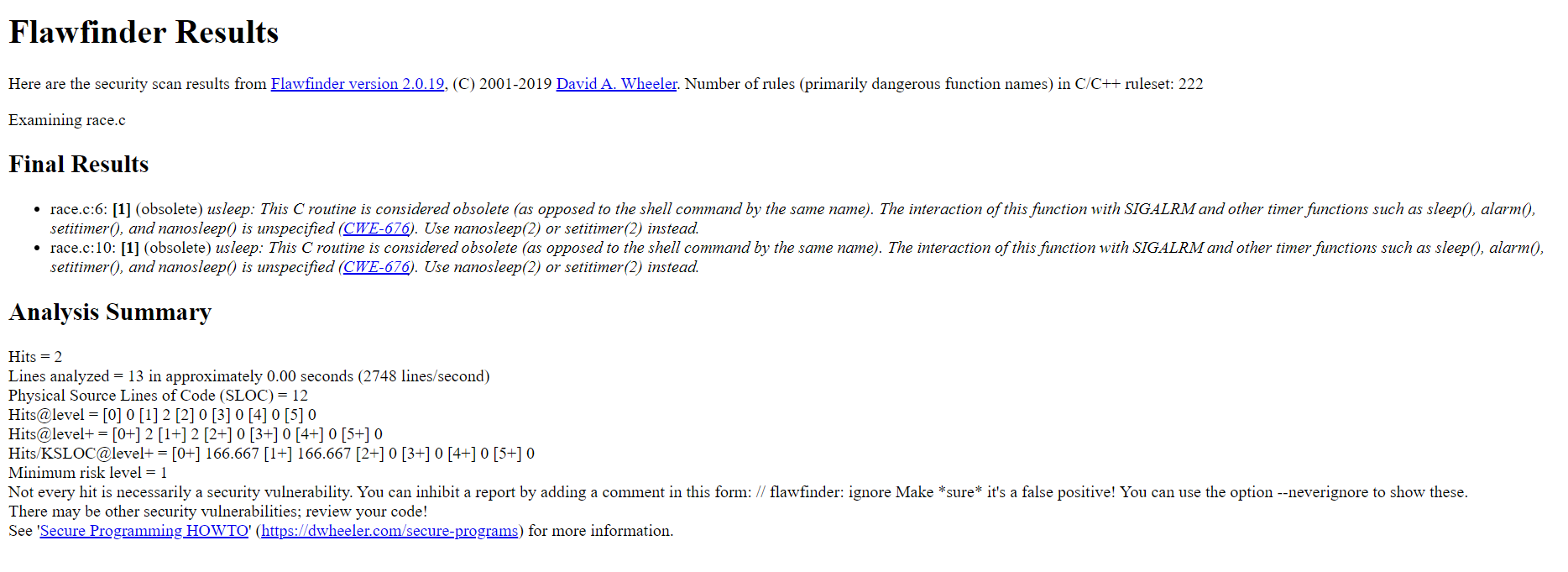
Step 1: Create a C file having the following program mentioned below. This program is intended to showcase a Race condition by using usleep(10000), hence intentionally creating a TOCTOU (Time-of-check to Time-of-Use) window.

We are creating a symlink to a file (myfle) owned by us, a normal user, so to pass the access() check. Then a window is generated to sleep for 10000 microseconds and let our vulnerable process run. Then it unlinks the symlink and creates a symlink to /etc/passwd.



Step 2: Run the following command to view output in an HTML document using Flawfinder.

flawfinder --html test4.c > result4.html



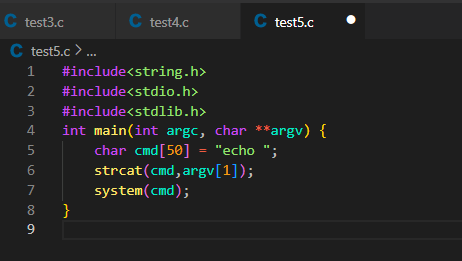
* **Metachar**

Many systems have metacharacters, that is, characters in their input that are not interpreted as data. Such characters might command, or delimit data from commands or other data. If there’s a language specification for a system’s interface, then it certainly has metacharacters. If your program invokes those other systems and allows attackers to insert such metacharacters, the usual result is that an attacker can completely control your program. Many metacharacter problems involve shell metacharacters. An attack that tries to exploit a vulnerability in shell metacharacter processing is called a shell injection or command injection attack.

**Steps to check for metacharacter problems (specifically, command/shell injection) using Flawfinder:**

Step 1: Create a C file having the following program mentioned below. In this program, we are implementing an echo command using the system() function. The argument for the echo command comes from a command line argument. But when used ';' in the argument it is showing the directory listing. We are actually providing the shell with two separate commands that it executes in sequence. First the shell runs a.out with the command line parameter "hello" in argv[1], which prints it out using echo. Then your program exits, and the shell runs the next command, *ls*, and displays the directory listing.

If we want to pass that string to the program as a command line parameter, we need to escape the special shell character ‘;’ so the shell does not parse it before giving it to our program. To escape a character, we can simply precede it with a \.



Step 2: Run the following command to view output in an HTML document using Flawfinder.

flawfinder --html test5.c > result5.html