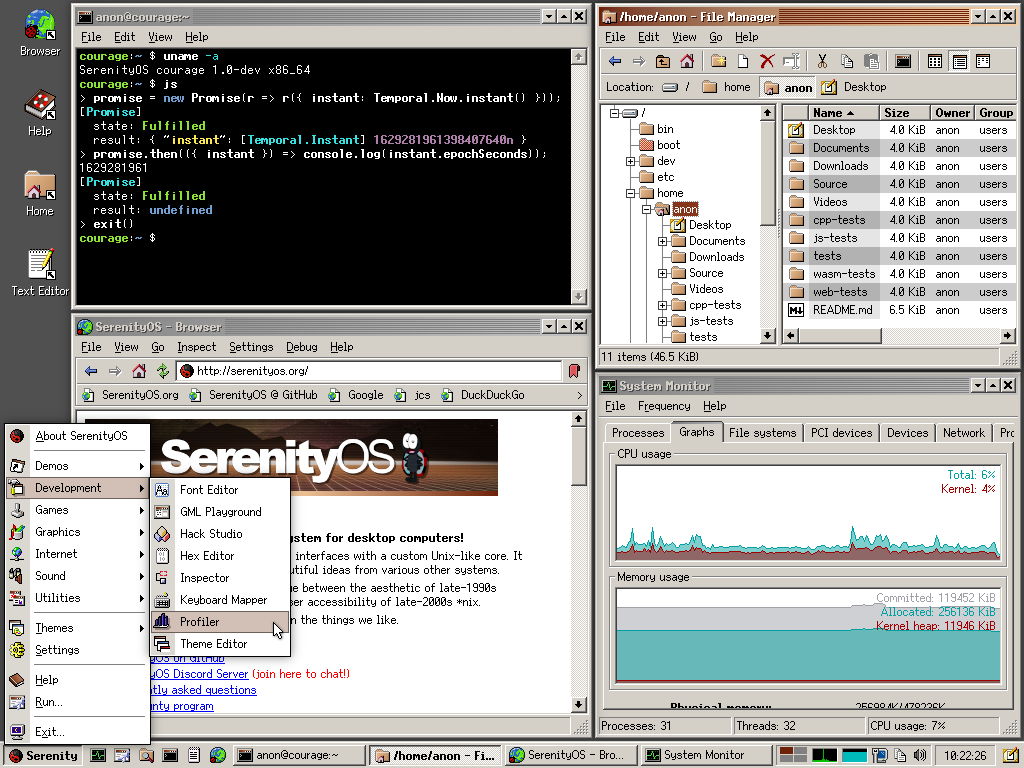
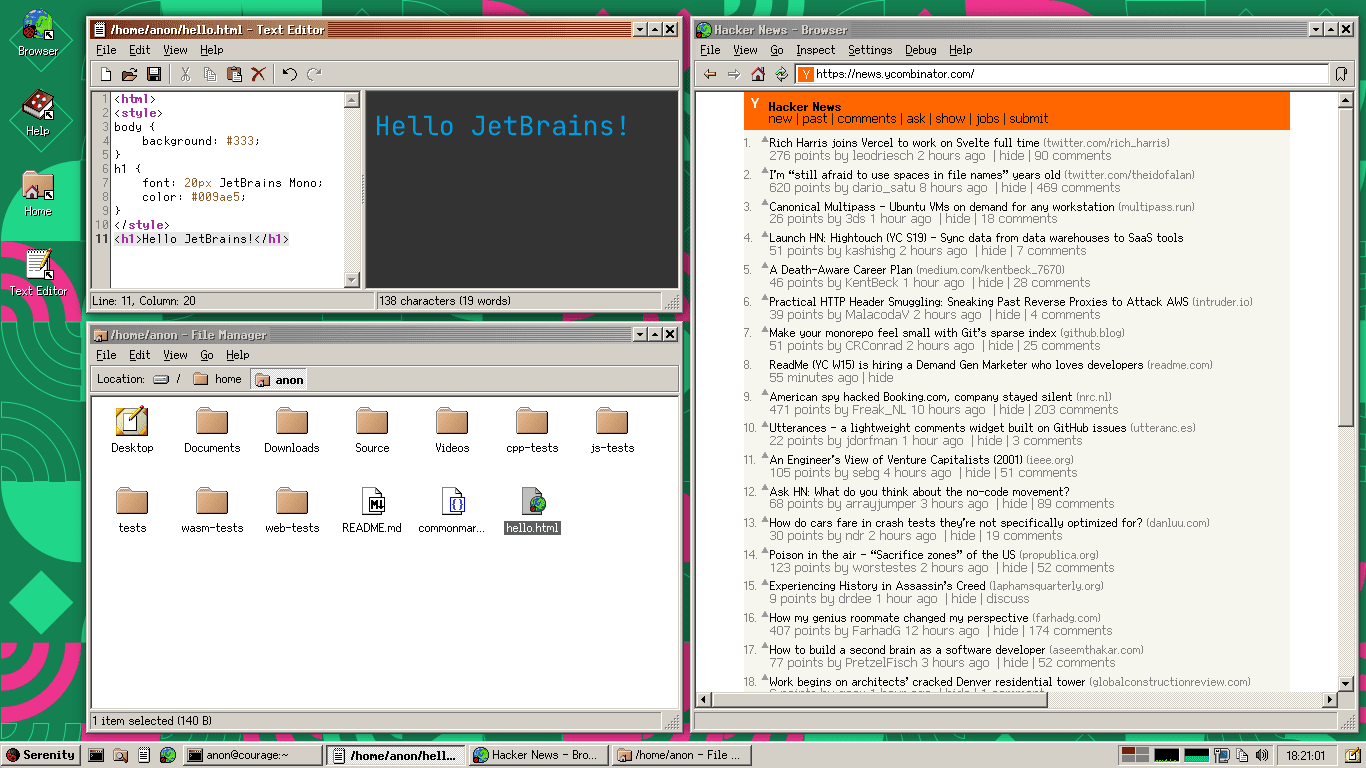
Serenity OS – Cerificare cu Cppcheck

# Serenity OS – Prezentare Generală:

***SerenityOS*** este un sistem de operare desktop gratuit și open source aflat în dezvoltare încă din 2018. Inițial, proiectul a fost conceput de programatorul suedez Andreas Kling (care a lucrat anterior pentru *Nokia* și *Apple*), dar, acum, este întreținut și dezvoltat alături de o comunitate de pasionați.

Așa cum este prezentat pe site-ul oficial, „*SerenityOS este o scrisoare de dragoste către interfețele de utilizator ale anilor '90, cu un nucleu personalizat asemănător Unix. În linii mari, scopul proiectului este îmbinarea armonioasă între estetica software-ului de la sfârșitul anilor 1990 și accesibilitatea puternică a sistemelor de operare contemporane*.”.

Mai pe larg, SerenityOS își propune să fie un sistem de operare modern, cu un aspect care amintește de anii '90 și cu o interfață grafică care încorporează elemente atât din Microsoft Windows, cât și din Mac OS. Încorporarea codului terță parte în sistem este complet descurajată. De altfel, browser-ul web, de exemplu, nu folosește un motor web preexistent, cum ar fi WebKit, ci folosește *LibWeb*, propriul său motor, codat de la 0. Există o colecție de software importat, cum ar fi *GCC*, *Git* și *Doom*, cu diferite niveluri de funcționalitate.

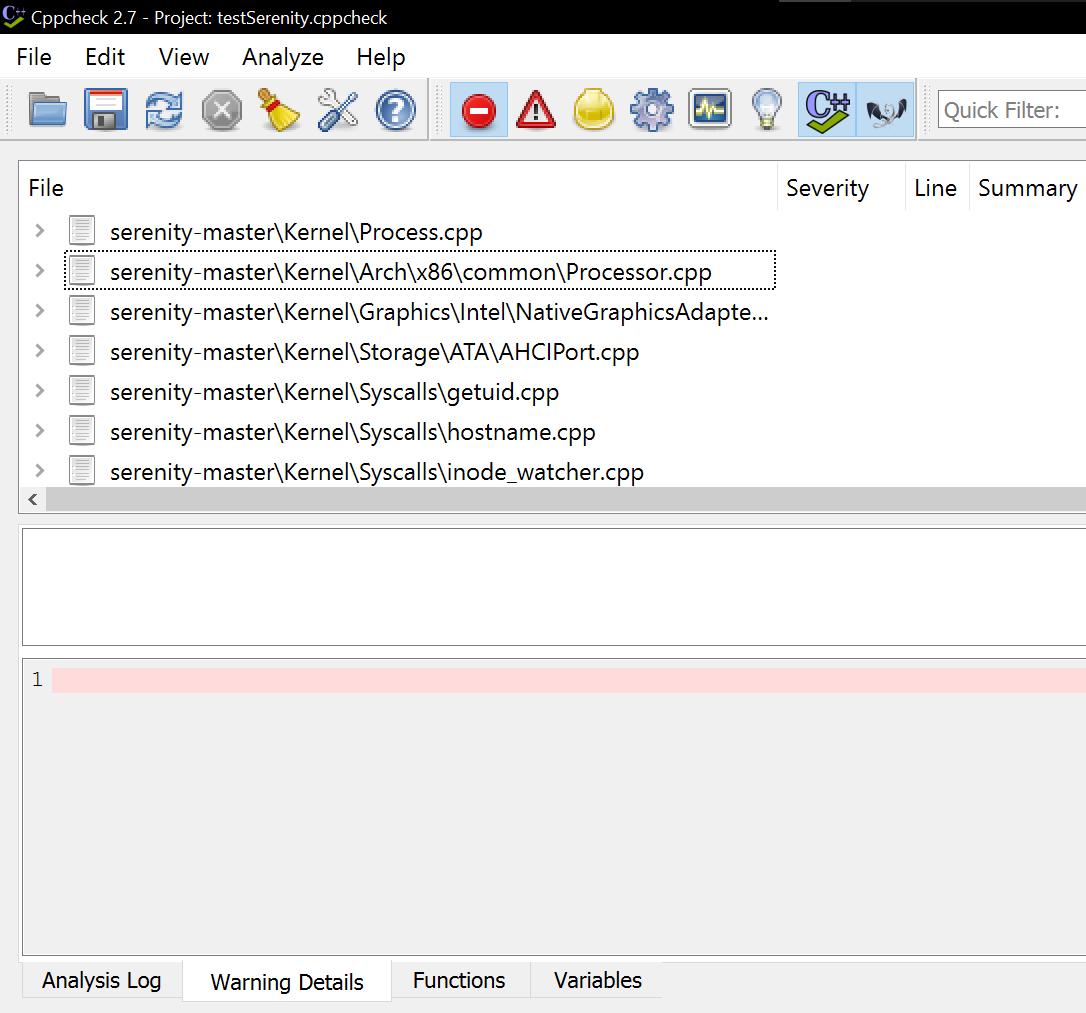
Nu există un orar de lansări. În plus, nu sunt furnizate distribuții binare sau imagine ISO și clienții potențiali trebuie să construiască sistemul de la sursă, dacă vor să ruleze proiectul. SerenityOS este codat în ceea ce autorii numesc „*Serenity C++*”, o variantă de C++ care nu are excepții și are propria bibliotecă standard.

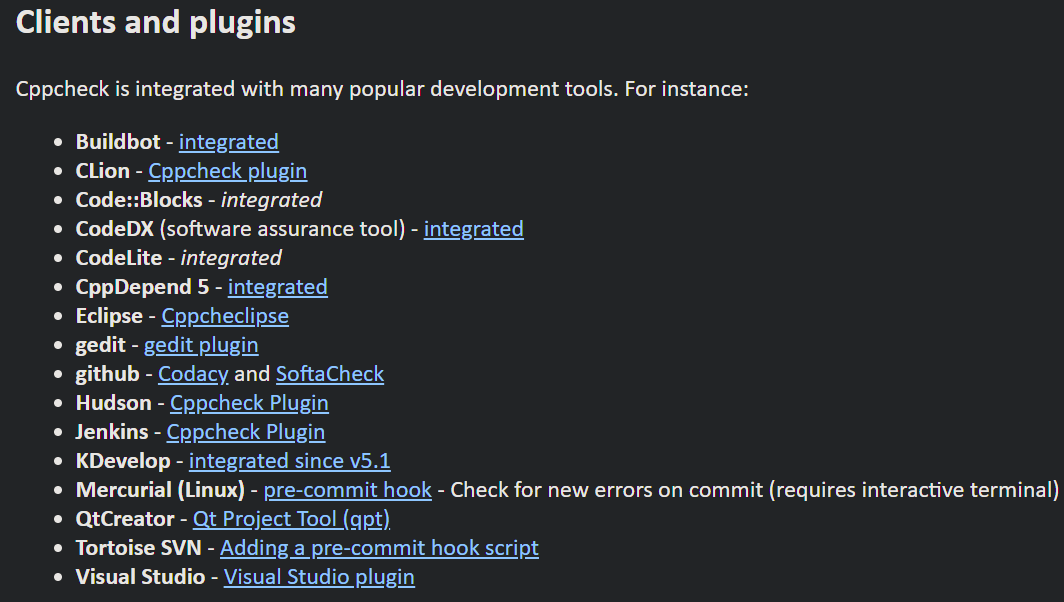
Câteva dintre feature-urile acestui sistem de operare sunt:

* nucleu x86 modern pe 32 și 64 de biți cu multi-threading; browser web cu JavaScript, WebAssembly, etc...;
* biblioteci pentru orice, de la criptografie la OpenGL, audio, GUI, etc...;
* suport pentru multe formate de fișiere comune, dar și neobișnuite (PNG, JPEG, GIF, MP3, WAV, FLAC, ZIP, TAR, PDF, QOI, Gemini, etc...);
* network stack și aplicații cu suport pentru IPv4, TCP, UDP; DNS, HTTP, Gemeni, IMAP, NTP;
* jocuri clasice, precum și moderne (Solitaire, Minesweeper, 2048, șah etc...);
* programe și utilitare de zi cu zi (TextEditor, Terminal, PixelPaint, diverse playere multimedia, Mail, Asistent, Calculator, etc...);
* și multe altele.

# Cppcheck – Prezentare Generală:

*Instrumentele de analiză statică* oferă primii pași către identificarea și eliminarea erorilor software și sunt esențiale pentru succesul general al proiectului. Un instrument de analiză statică este adecvat pentru un proiect software dacă este eficient în capturarea tuturor bug-urilor care afectează critic calitatea produsului. Aceste instrumente caută greșeli comune, pe care compilatoarele le ignoră; spre exemplu: buffer overflow, memory overruns, atacuri XSS sau injecții SQL etc...

Cppcheck este un popular instrument static, gratuit, de analiză a codului pentru limbajele de programare C și C++, dezvoltat de Daniel Marjamäki și lansat inițial în 2007. Pentru a rula programul, se poate descărca interfața grafică de pe site-ul oficial sau, în funcție de editorul de cod preferat, se poate verifica dacă există un plugin ce poate fi instalat.

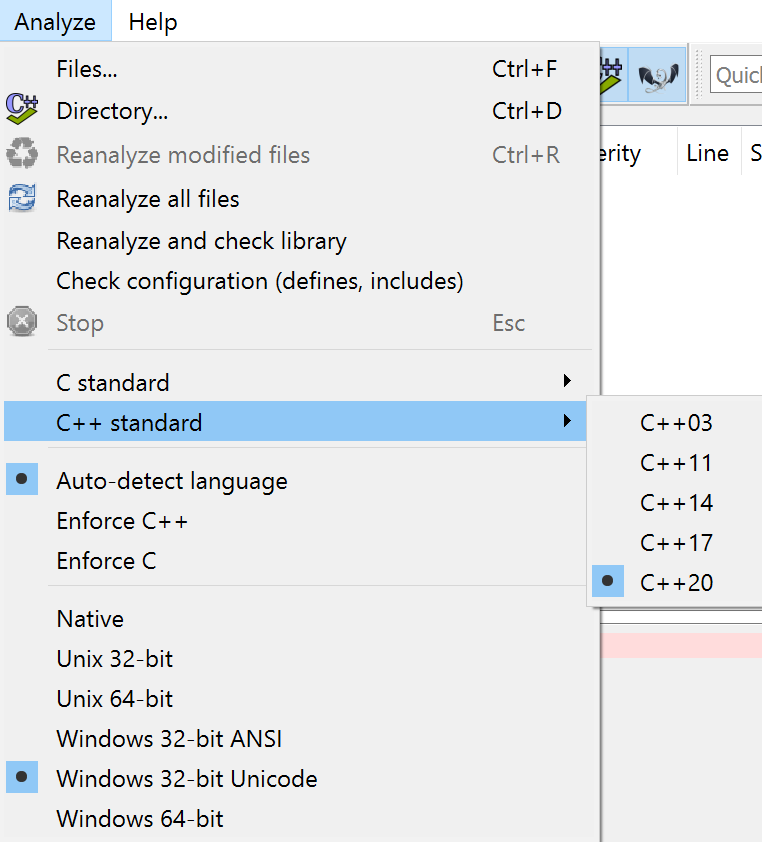


GUI-ul programului este simplu și ușor de utilizat, chiar și pentru persoanele care nu sunt programatori. În partea de sus, sunt cele mai importante butoane:

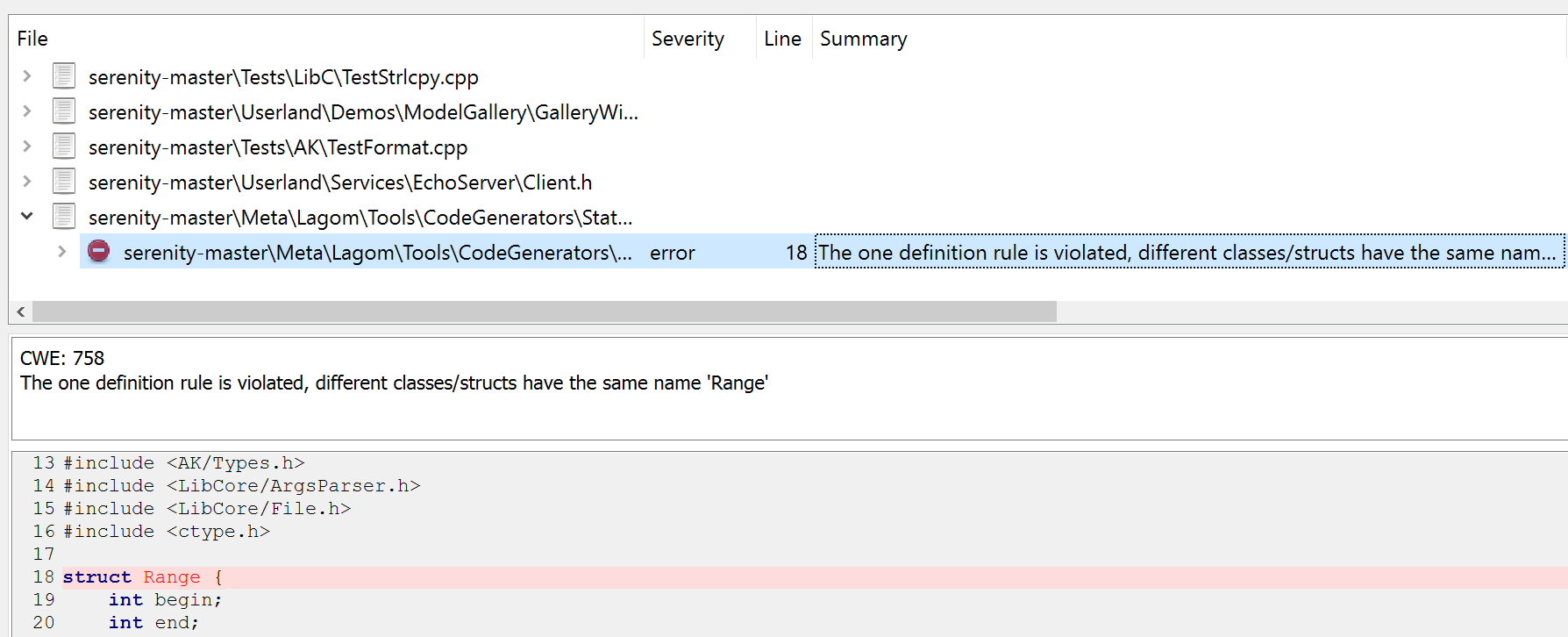


Acestea pot fi selectate sau deselectate conform nevoilor utilizatorului. De la stânga la dreapta, avem butonul „*Show Errors*”, „*Show Warnings*”, „*Show Style Warnings*”, „*Show Portability Warnings*”, „*Show Performance Warnings*”, „*Show Information Messages*” și, mai la dreapta, avem o bară „*Quick Filter*” unde putem căuta folosind cuvinte cheie.

În plus, putem configura programul după nevoile noastre, alegând ce limbaj folosim, ce versiune a limbajului și ce sistem de operare avem. De asemenea, putem să rulăm pentru verificare doar anumite fișiere, nu tot folderul proiectului și putem reanaliza doar fișierele modificate, odată terminată verificarea. Suplimentar, dând dublu click pe oricare eroare găsită, este deschis fișierul sursă la linia de cod care produce respectiva eroare.



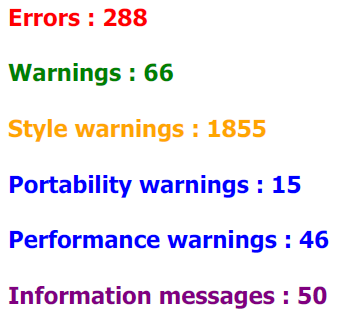
Bug-urile găsite de *Cppcheck* sunt descrise folosind un tag de severitate (i.e. eroare, avertisment etc...), linia de cod și fișierul unde se găsesc, o scurtă prezentare, iar, în funcție de bug, *Cppcheck* atașează și codul ***CWE*** (***Common Weakness Enumeration***), ca în exemplul de mai jos:

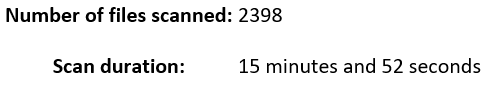


Un alt feature folositor este „*Statistics*” care prezintă numărul de bug-uri găsite, dar și durata de căutare a acestora.

# Serenity OS – Verificare cu Cppcheck:

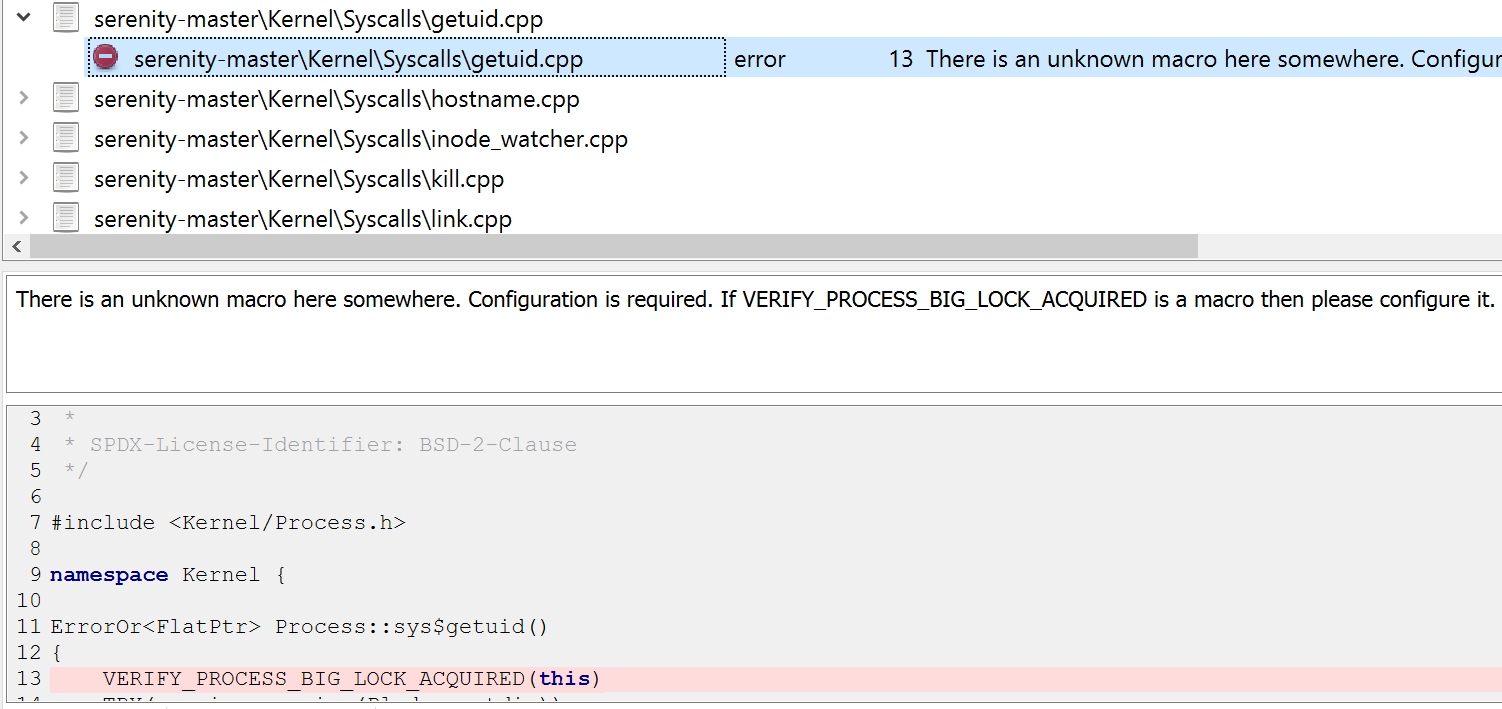
Pentru proiectul de față am folosit Windows 10, interfața grafică a programului *Cppcheck* și am verificat codul din repository-ul oficial de Github al *SerenityOS* de la data de 9.04.2022 (între timp, au fost aduse modificări și unele erori au dispărut).

Raportul („*Statistics*”) oferit de Cppcheck după verificarea proiectului este mai jos:

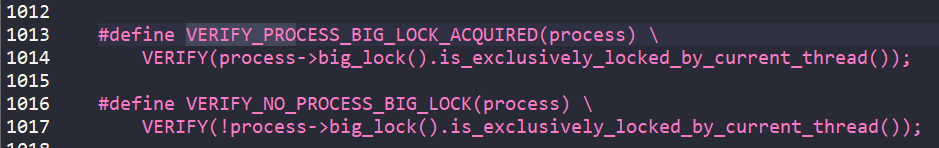


În continuare vom prezenta cele mai interesante bug-uri descoperite, din categoria erorilor, deoarece acestea au severitatea cea mai înaltă și pot afecta buna desfășurare a programului, și posibile soluții pentru acestea.

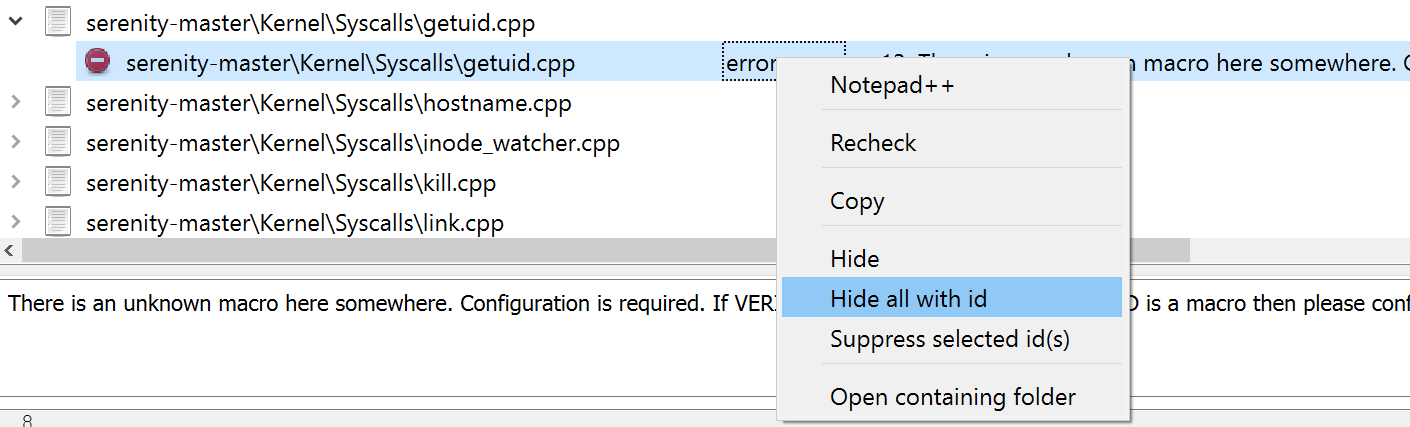
Înainte de a începe trebuie să reținem că *Cppcheck* nu este sută la sută precis, uneori identificând bug-uri care nu există. Deci numărul real de erori nu este 288, ci mult mai mic. Spre exemplu, o problemă des întâlnită, semnalată fals drept eroare este ...



... deși definiția macro este corect implementată în <Kernel/Process.h>:

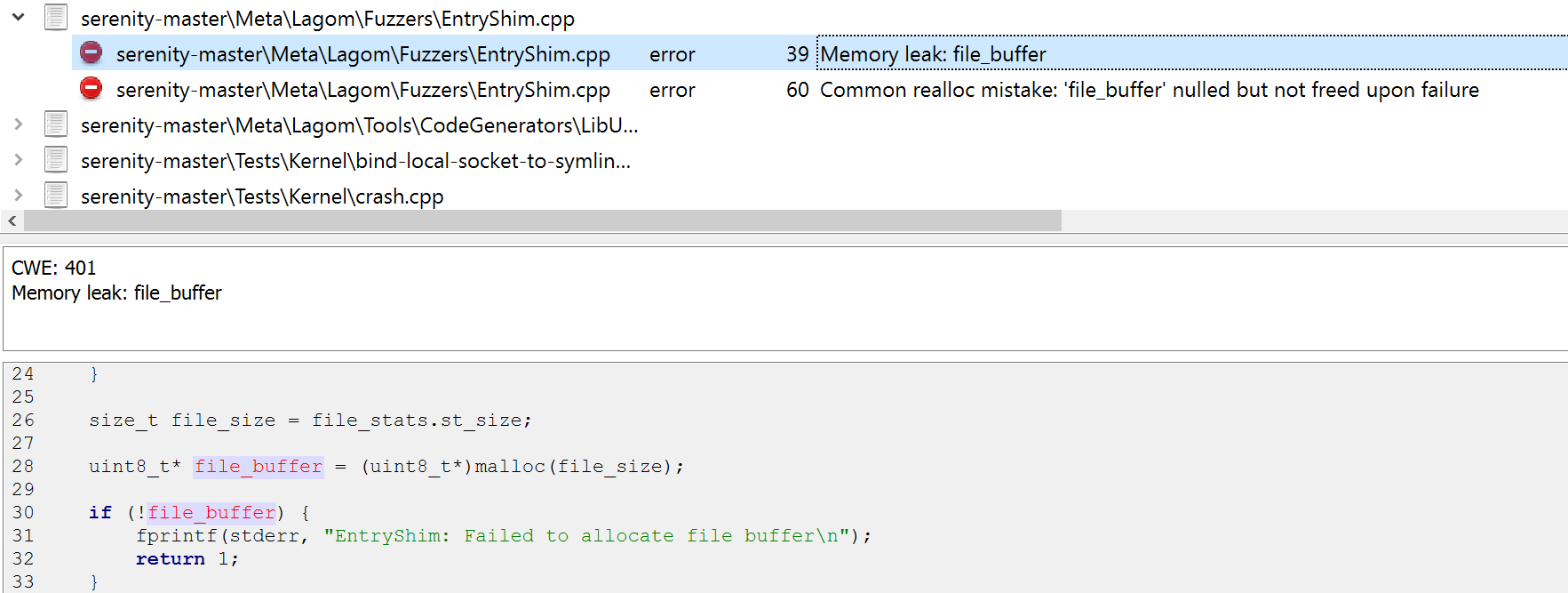


Dar *Cppcheck* are opțiunea de a suprima toate erorile cu același id, tocmai pentru a nu ne confrunta cu ele, dacă sunt false:

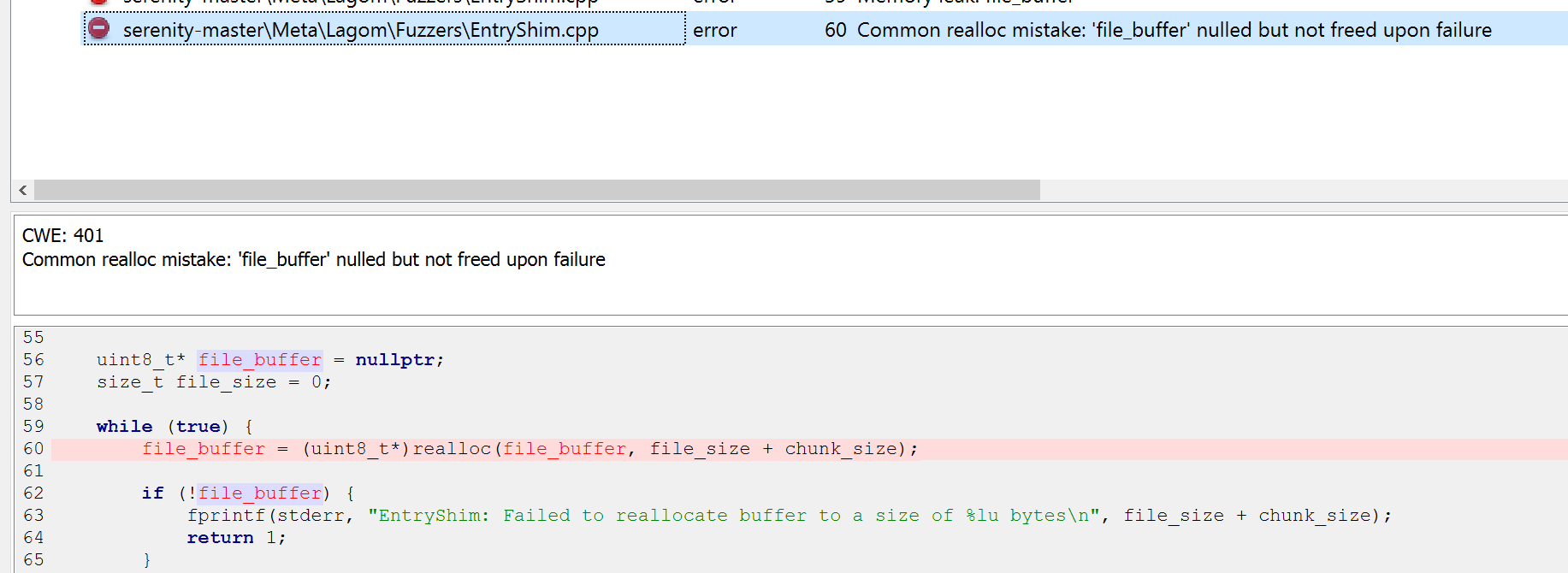


Cu aceste chestiuni discutate, să vedem erorile:

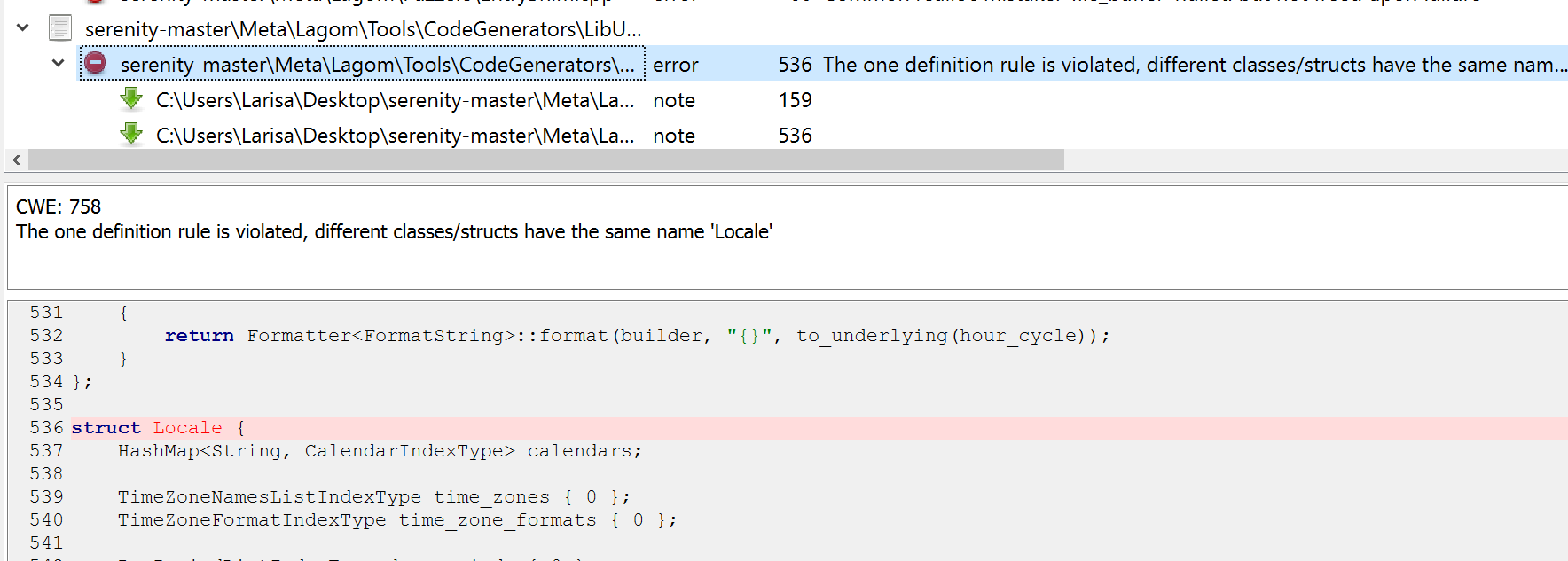
* **CWE 401** („The software does not sufficiently track and release allocated memory after it has been used, which slowly consumes remaining memory.”), **Memory leak: file\_buffer**. Această eroare este destul de des întâlnită și apare atunci când memoria alocată nu este eliberată. Pentru a rezolva bug-ul, trebuie doar să dezalocăm memoria înainte de a ieși din funcție: **free(file\_buffer);** deoarece avem **uint8\_t\* file\_buffer = (uint8\_t\*)malloc(file\_size);**



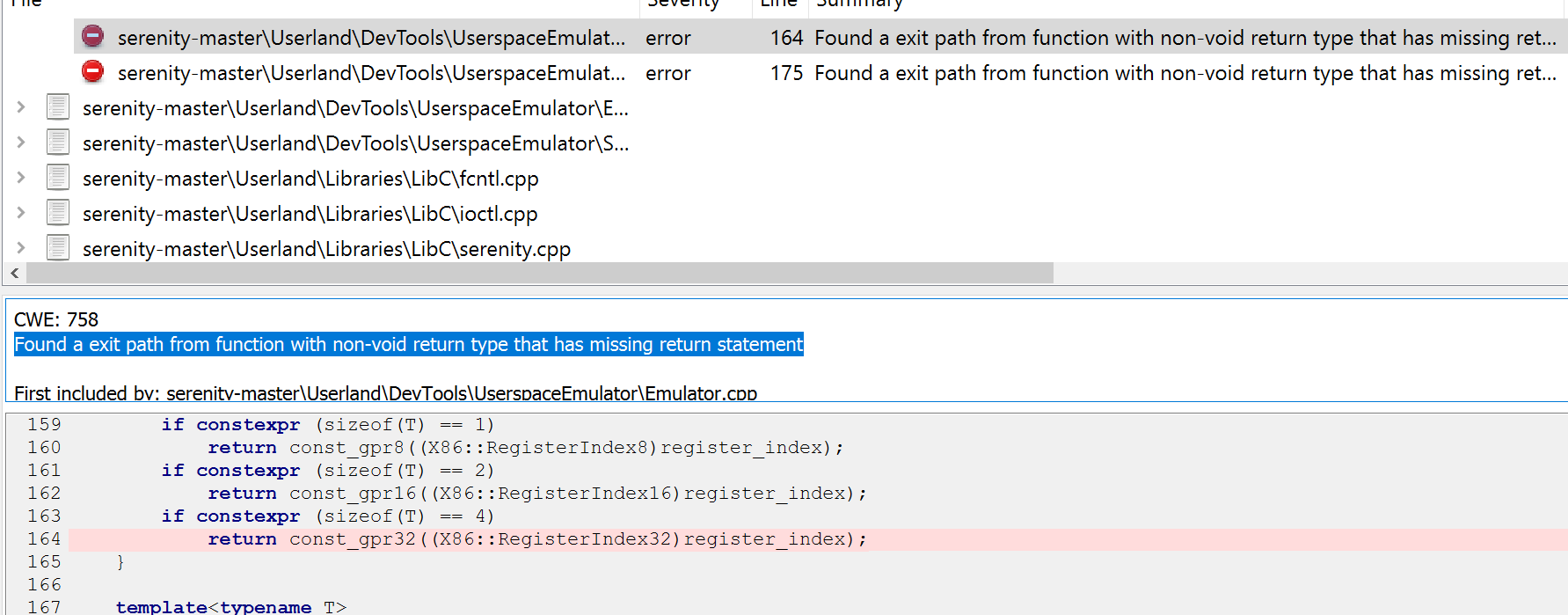
* **CWE 401**, **Common realloc mistake: 'file\_buffer' nulled but not freed upon failure**. Aceast bug este similar cu cel de mai sus și are aceași rezolvare (doar că în cod apare **uint8\_t\* file\_buffer = nullptr;**).



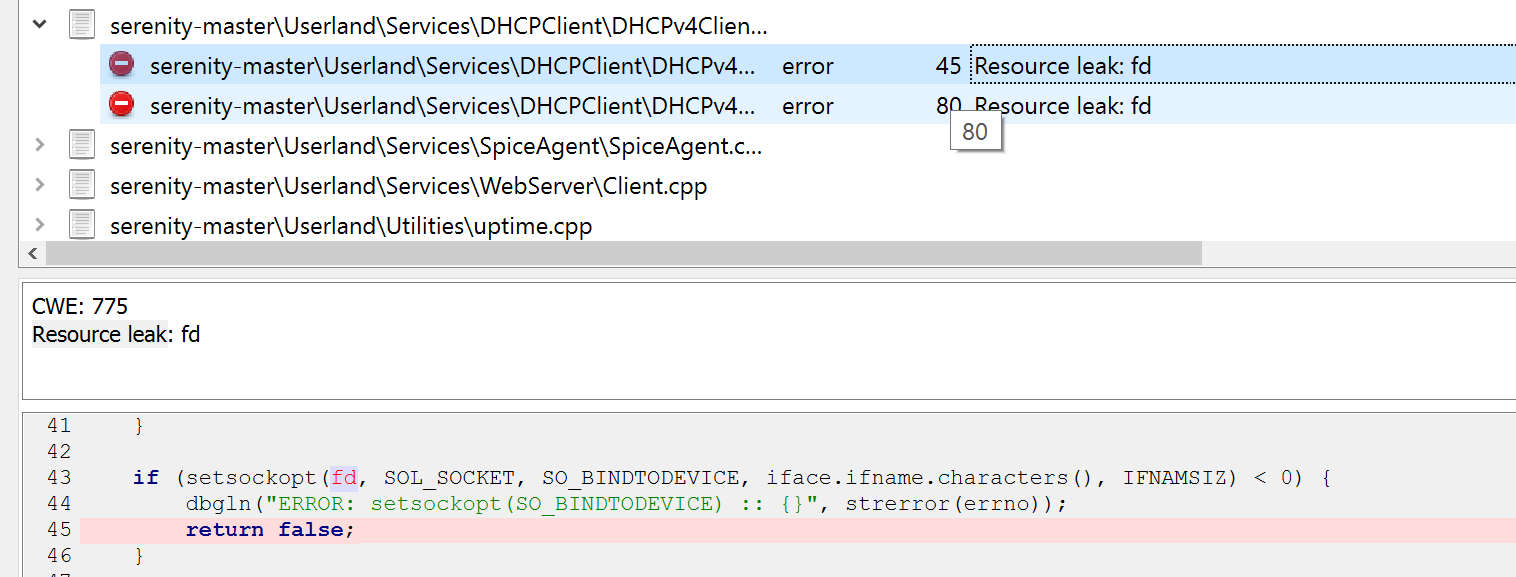
* **CWE 758** („The software uses an API function, data structure, or other entity in a way that relies on properties that are not always guaranteed to hold for that entity. This can lead to resultant weaknesses when the required properties change, such as when the software is ported to a different platform or if an interaction error occurs.”), **The one definition rule is violated, different classes/structs have the same name ('Locale/UnicodeLocaleData/Option/...')**. Această eroare apare pentru că, în 2 fișiere diferite, avem definite structuri/clase cu aceași denumire. Deși acest comportament nu produce o eroare neapărat, dacă programatorul este atent la ce codează; însă, dacă importurile nu se fac în mod corect, poate duce la un comportament nedefinit și erori. Este recomandat să existe denumiri diferite.



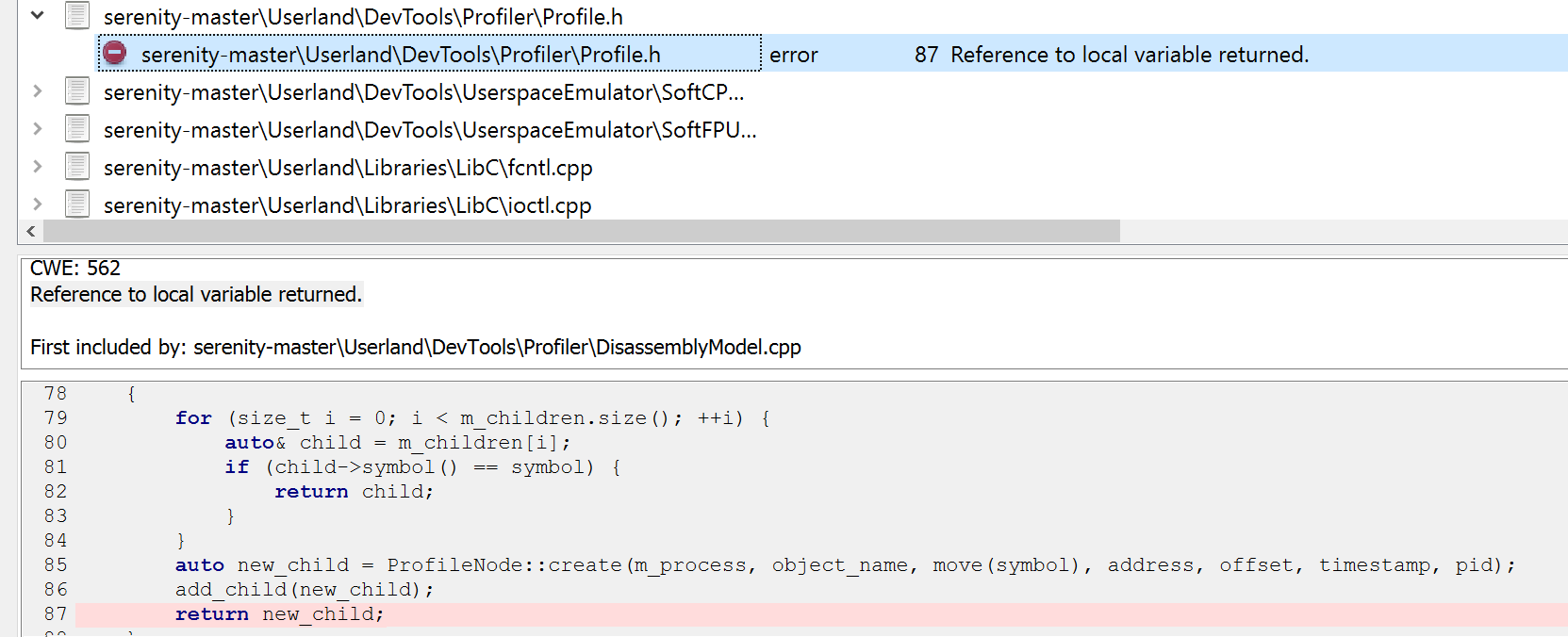
* **CWE 758**, **Found a exit path from function with non-void return type that has missing return statement**. Aceast bug apare deoarece avem o funcție, care nu este *void*, ce tratează diferite cazuri (mai multe *if*-uri sau un *switch*), dar nu are valoare returnată pentru cazul default.



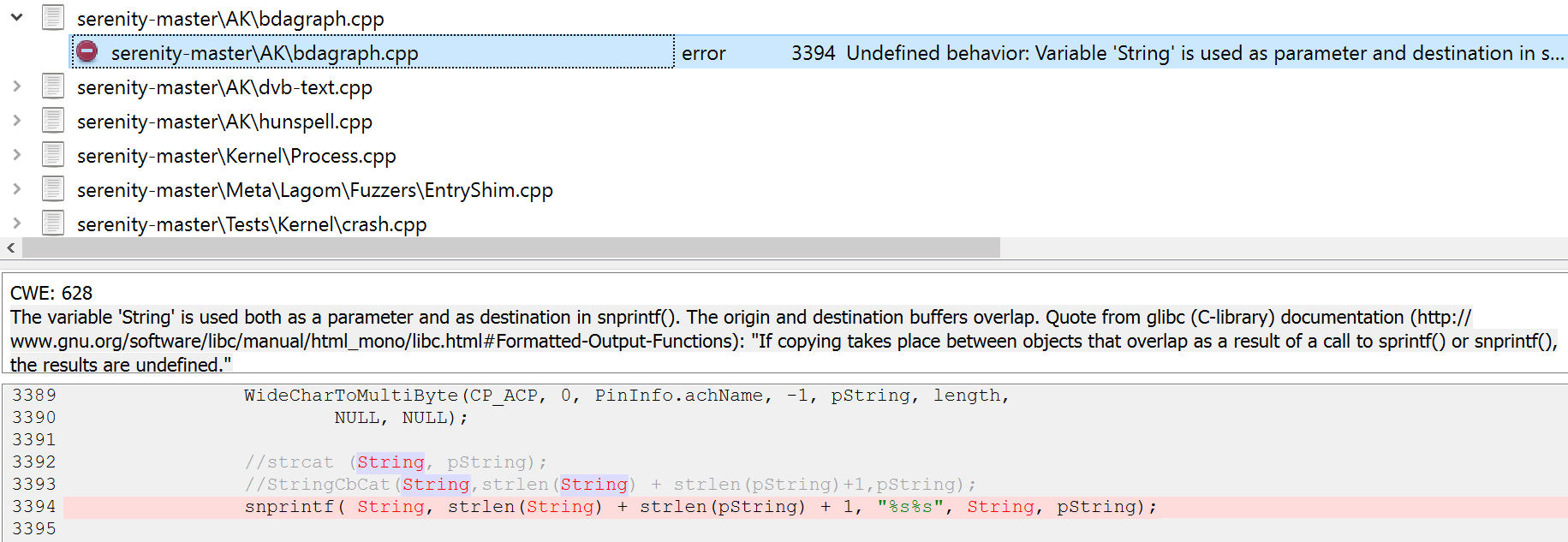
* **CWE 775** („The software does not release a file descriptor or handle after its effective lifetime has ended, i.e., after the file descriptor/handle is no longer needed. When a file descriptor or handle is not released after use (typically by explicitly closing it), attackers can cause a denial of service by consuming all available file descriptors/handles, or otherwise preventing other system processes from obtaining their own file descriptors/handles.”), **Resource leak: fd**. *Fd* este de tipul *socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);*. Cppcheck ne dă eroare pentru că resursa nu este închisă. Trebuie să adăugăm **close(fd)**.



* **CWE 562** („A function returns the address of a stack variable, which will cause unintended program behavior, typically in the form of a crash. Because local variables are allocated on the stack, when a program returns a pointer to a local variable, it is returning a stack address. A subsequent function call is likely to re-use this same stack address, thereby overwriting the value of the pointer, which no longer corresponds to the same variable since a function's stack frame is invalidated when it returns. At best this will cause the value of the pointer to change unexpectedly. In many cases it causes the program to crash the next time the pointer is dereferenced.”), **Reference to local variable returned**. Această eroare apare pentru că programul returnează o adresă de memorie locală, care va muri la ieșirea din funcție. Există posibile soluții, una dintre ele ar fi să transformăm funcția dintr-una de tip *ProfileNode*, în una de tip *void* și atunci nu mai trebuie să returnăm nimic.

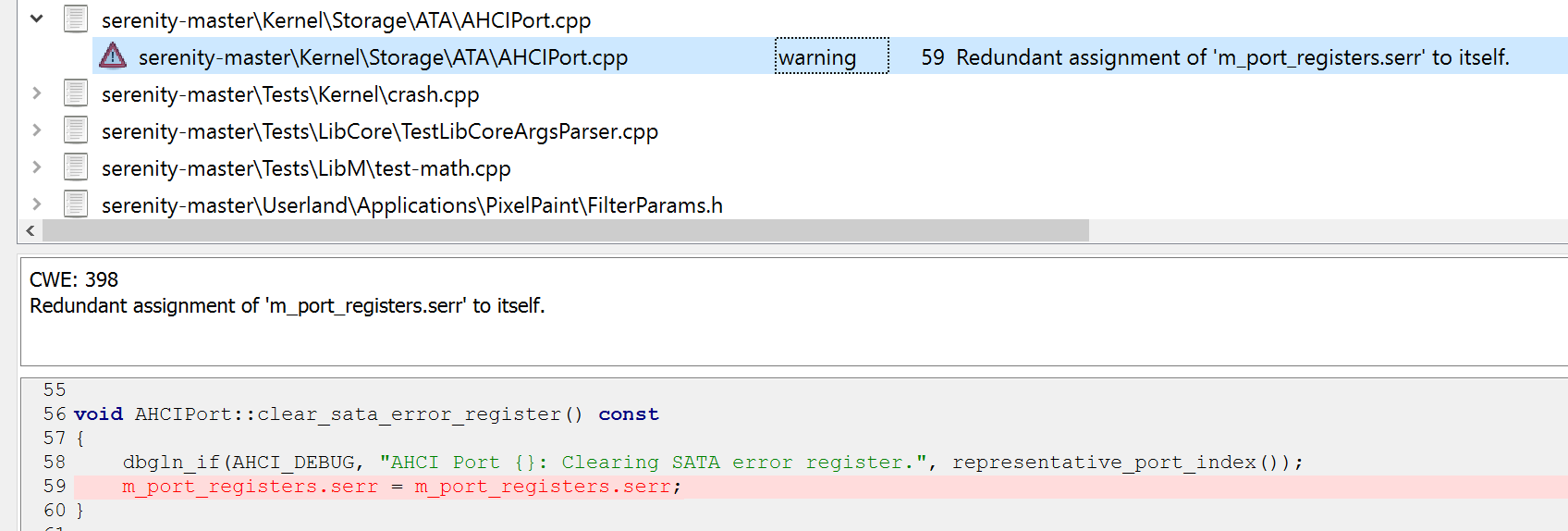


* **CWE 628** („The product calls a function, procedure, or routine with arguments that are not correctly specified, leading to always-incorrect behavior and resultant weaknesses.”), **The variable 'String' is used both as a parameter and as destination in snprintf(). The origin and destination buffers overlap. Quote from glibc (C-library) documentation (**[**http://www.gnu.org/software/libc/**](http://www.gnu.org/software/libc/) **manual/html\_mono/libc.html#Formatted-Output-Functions): "If copying takes place between objects that overlap as a result of a call to sprintf() or snprintf(), the results are undefined."**. Cppcheck ne oferă explicația completă pentru acest bug – avem comportament nedeterminat, deoarece avem **snprintf( String, strlen(String) + strlen(pString) + 1, "%s%s", String, pString);**. O posibilă soluție este să avem un **char String[80], String1[80]; → snprintf(String, strlen(String) + strlen(pString) + 1, "%s%s", String1, pString);**, astfel evitând suprapunerea.

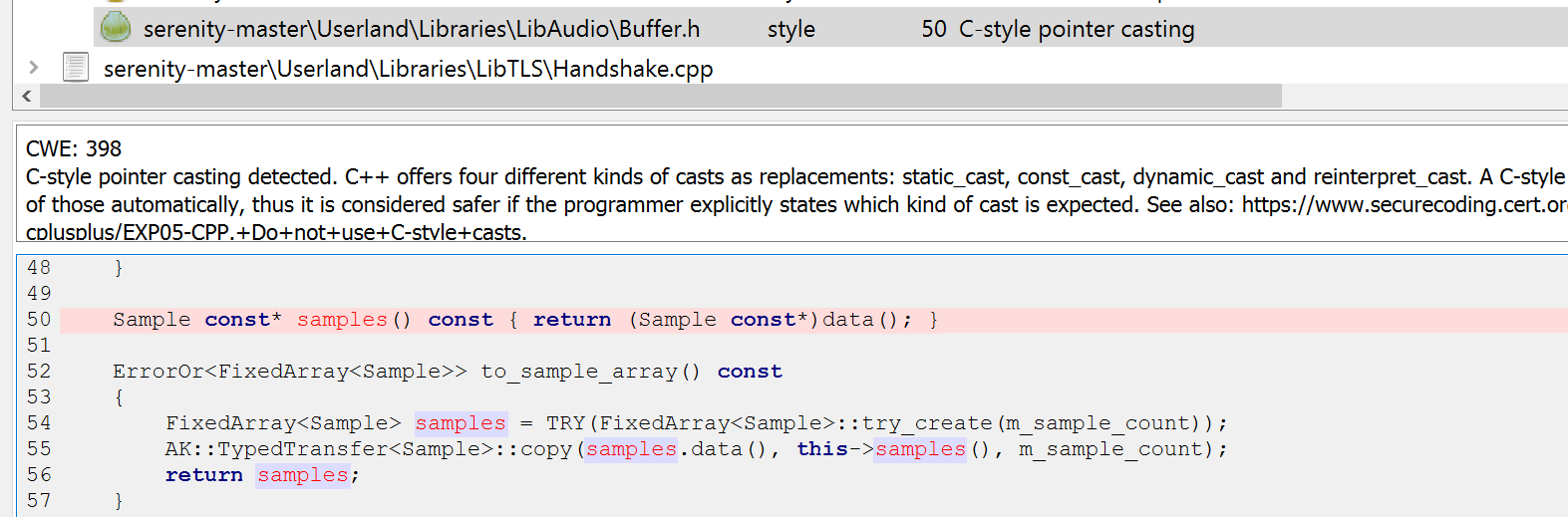


Alte probleme interesante apărute, care nu sunt neapărat din categoria erorilor sunt:

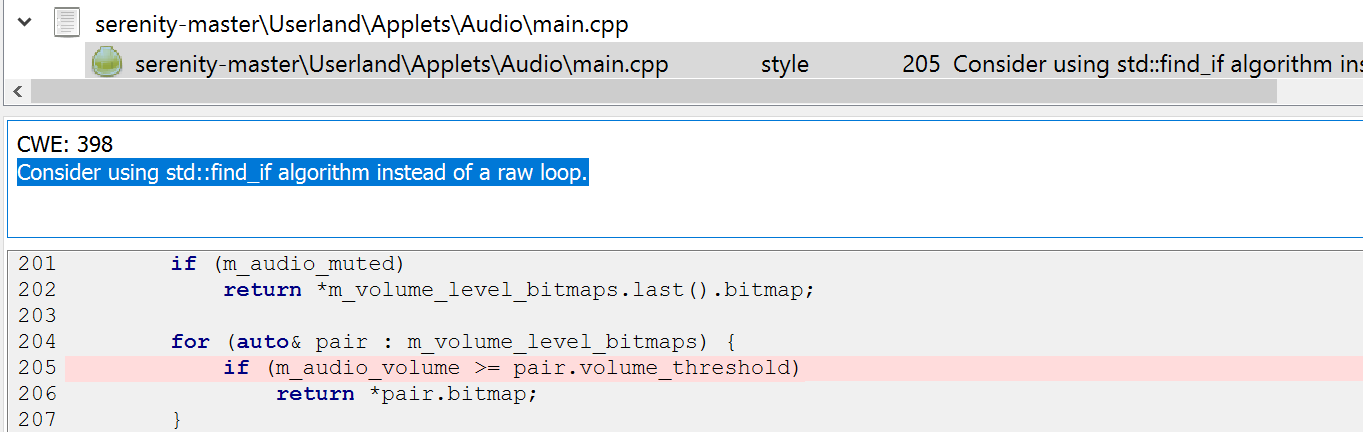
* **CWE 398** („This category represents one of the phyla in the Seven Pernicious Kingdoms vulnerability classification. It includes weaknesses that do not directly introduce a weakness or vulnerability, but indicate that the product has not been carefully developed or maintained. According to the authors of the Seven Pernicious Kingdoms, "Poor code quality leads to unpredictable behavior. From a user's perspective that often manifests itself as poor usability. For an adversary it provides an opportunity to stress the system in unexpected ways."”), **Redundant assignment of 'm\_port\_registers.serr' to itself.** Cppcheck ne oferă explicația completă pentru acest bug – avem o reasignare redundantă, care nu ar trebui să existe în cod, deoarece nu produce niciun efect. Recomandat este să scoatem această linie de cod.



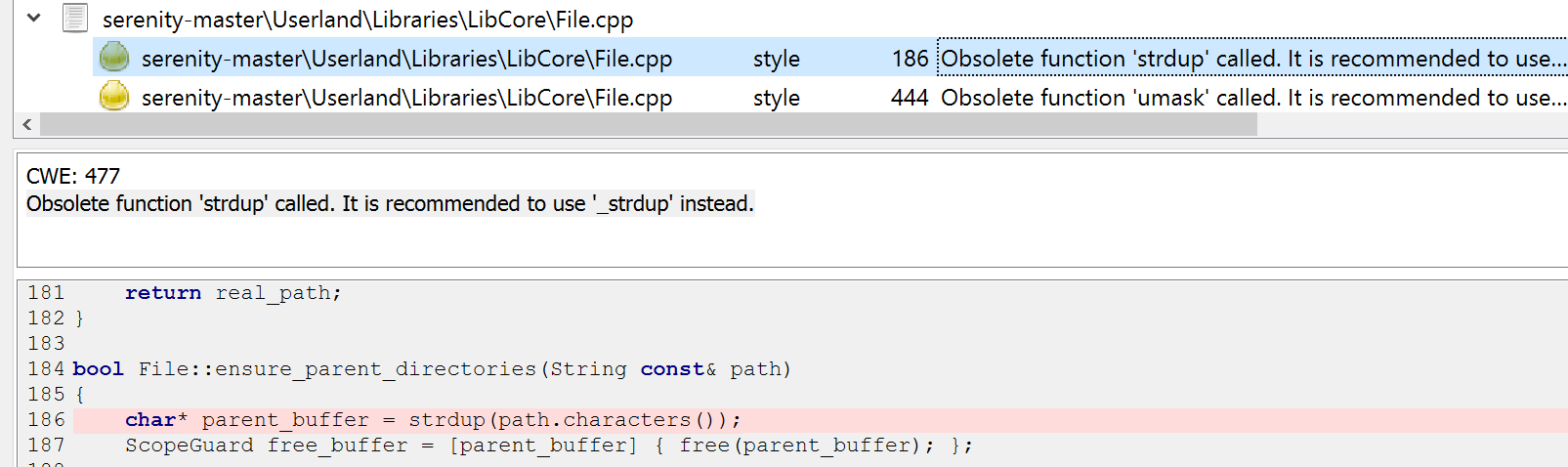
* **CWE 398**, **C-style pointer casting detected. C++ offers four different kinds of casts as replacements: static\_cast, const\_cast, dynamic\_cast and reinterpret\_cast. A C-style cast could evaluate to any of those automatically, thus it is considered safer if the programmer explicitly states which kind of cast is expected. See also:** [**https://www.securecoding.cert.org/confluence/display/**](https://www.securecoding.cert.org/confluence/display/) **cplusplus/EXP05-CPP.+Do+not+use+C-style+casts**. Explicația este deja oferită de tool.



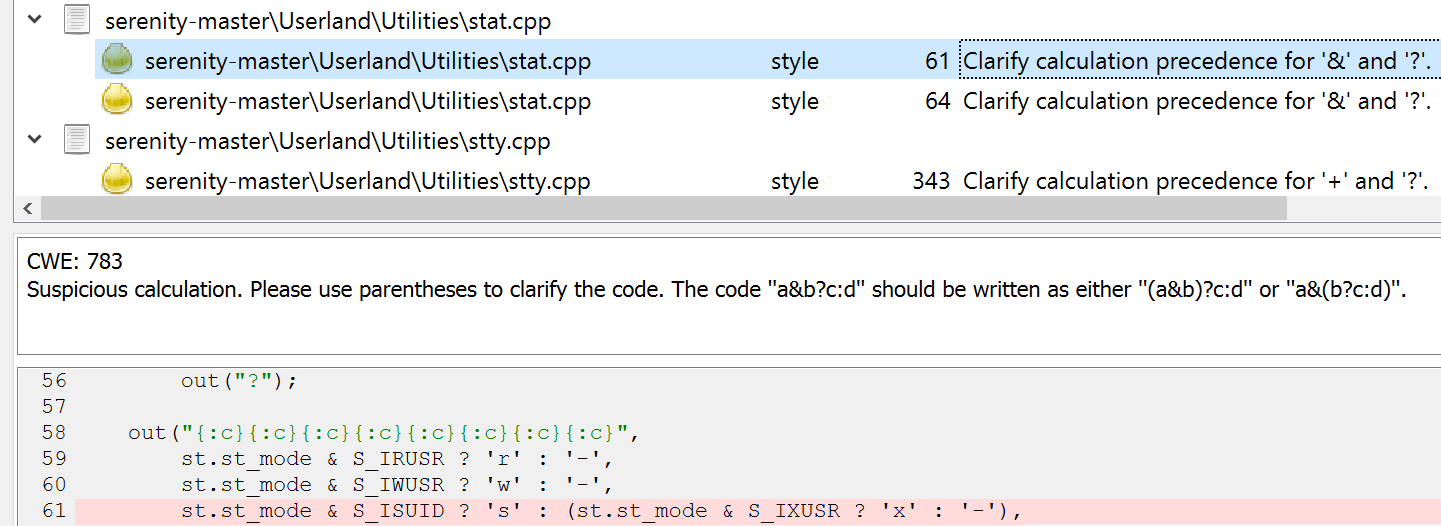
* **CWE 398**, **Consider using std::find\_if algorithm instead of a raw loop.** Explicația este deja oferită de tool. (Există mai multe sugestii asemănătoare oferite de Cppcheck, pentru diferite funcții cum ar fi **std::transform, std::accumulate, std:: any\_of algorithm** ...)



* **CWE 477** („The code uses deprecated or obsolete functions, which suggests that the code has not been actively reviewed or maintained.”), **Obsolete function 'strdup' called. It is recommended to use '\_strdup' instead.** Explicația este deja oferită de Cppcheck. (Există mai multe sugestii asemănătoare oferite de Cppcheck, pentru diferite funcții)



* **CWE 783** („The program uses an expression in which operator precedence causes incorrect logic to be used.”), **Suspicious calculation. Please use parentheses to clarify the code. The code ''a&b?c:d'' should be written as either ''(a&b)?c:d'' or ''a&(b?c:d)''.** Explicația este deja oferită de Cppcheck.



Următoarele erori merită menționate, deși ele au valoare de test (se găsesc în folderul *Test*) și nu afectează codul propriu-zis, dar demonstrează buna funcționare a instrumentului de analiză statică Cppcheck:

* **CWE 369** („The product divides a value by zero.”), **Assignment 'zero=0', assigned value is 0**. Eroarea este evidentă – împărțire la 0. Pentru a rezolva eroarea, ne asigurăm ca variabila să nu fie egală cu valoarea nulă înainte de împărțire.
* **CWE 628** („The product calls a function, procedure, or routine with arguments that are not correctly specified, leading to always-incorrect behavior and resultant weaknesses.”), **Invalid sqrt() argument nr 1. The value is -1 but the valid values are '0.0:'**. Eroarea este evidentă – se încearcă extragerea rădăcinii pătrate a lui -1. Pentru a rezolva eroarea, ne asigurăm ca variabila să nu fie egală -1 înainte de realizarea operației.
* **CWE 476** („A NULL pointer dereference occurs when the application dereferences a pointer that it expects to be valid, but is NULL, typically causing a crash or exit.”), **Null pointer dereference**. Eroarea apare pentru că programul așteaptă un pointer valid, dar acesta este NULL. Pentru a soluționa bug-ul, ne asigurăm ca pointerul să fie valid înainte de utilizare.
* **CWE 457** („The code uses a variable that has not been initialized, leading to unpredictable or unintended results.”), **Uninitialized variable: x**. După cum sugerează și numele, o variabilă a fost utilizată înainte să fie inițializată. Această eroare apare cel mai des din vina programatorului și soluția este simplă – inițializarea.
* **No pair for character ("). Can't process file. File is either invalid or unicode, which is currently not supported.** Codul este **#include "sample\_heade**, ceea ce produce o eroare pentru că lipsește **"**.

# Bibliografie:

1. *SerenityOS*. (2022). Serenityos.org. <https://serenityos.org/>
2. Wikipedia Contributors. (2022, April 1). *SerenityOS*. Wikipedia; Wikimedia Foundation. <https://en.wikipedia.org/wiki/SerenityOS>
3. SerenityOS. (2022, April 25). *SerenityOS/serenity: The Serenity Operating System*. GitHub. <https://github.com/SerenityOS/serenity>
4. G. Chatzieleftheriou and P. Katsaros, "Test-Driving Static Analysis Tools in Search of C Code Vulnerabilities," 2011 IEEE 35th Annual Computer Software and Applications Conference Workshops, 2011, pp. 96-103, doi: 10.1109/COMPSACW.2011.26.
5. R. Kannavara, "Securing Opensource Code via Static Analysis," 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation, 2012, pp. 429-436, doi: 10.1109/ICST.2012.123.
6. J. Novak, A. Krajnc and R. Žontar, "Taxonomy of static code analysis tools," The 33rd International Convention MIPRO, 2010, pp. 418-422.
7. *Cppcheck - A tool for static C/C++ code analysis*. (2019). Sourceforge.io. <https://cppcheck.sourceforge.io/>
8. Wikipedia Contributors. (2022, March 7). *Cppcheck*. Wikipedia; Wikimedia Foundation. <https://en.wikipedia.org/wiki/Cppcheck>
9. *CWE - Common Weakness Enumeration*. (2021). Mitre.org. <https://cwe.mitre.org/>