

Analiza i Verifikacija Softvera

Online Shopping System

Dimitrije Marković 1022/24
Matematički fakultet, Univerzitet u Beogradu

10. septembar 2025.

Sadržaj

1	Uvod	2
2	Statička analiza	2
2.1	Clang-Tidy	2
2.2	Cppcheck	4
2.2.1	Glavne kategorije detektovanih problema	4
2.2.2	Zaključak analize Cppcheck-a	5
3	Dokumentacija (Doxygen)	6
3.1	Doxygen (generisana dokumentacija)	6
4	Dinamička analiza	7
4.1	Dr. Memory	7
4.2	Very Sleepy Profiler (analiza performansi)	8
4.3	Dr. Memory	10
4.3.1	Dr. Memory — analiza servera	10
4.3.2	Dr. Memory — analiza klijenta	11
5	Zaključak	12
	Dodaci	13

1 Uvod

Za potrebe kursa *Verifikacija Softvera*, analiziran je projekat **Online Shopping System**. Projekat je razvijen u programskom jeziku C++ kao konzolna aplikacija, sa klijent-server arhitekturom koja koristi **Winsock2** biblioteku za mrežnu komunikaciju.

Aplikacija implementira osnovne funkcionalnosti sistema za online kupovinu, uključujući registraciju i prijavu korisnika, pregled i kupovinu proizvoda, generisanje računa, kao i mogućnost izmene porudžbine. Dodatno, sistem pruža različite privilegije za korisnike, zaposlene i administratore (upravljanje nalogima, popusti, pregled inventara, rešavanje žalbi i automatsko procesuiranje porudžbina).

Cilj ove analize je da se proceni pouzdanost, efikasnost i kvalitet implementacije kroz upotrebu savremenih alata za statičku i dinamičku analizu softvera. U okviru rada korišćeni su sledeći alati:

- **Clang-Tidy** i **Cppcheck** — alati za statičku analizu koda i detekciju potencijalnih problema,
- **Doxygen** — za automatsko generisanje dokumentacije iz izvornog koda,
- **Dr. Memory** — za detekciju problema u radu sa memorijom tokom izvršavanja,
- **Very Sleepy Profiler** — za profilisanje performansi i identifikaciju najintenzivnijih delova koda.

Analiza i dobijeni rezultati omogućavaju uvid u potencijalne slabosti projekta i daju preporuke za dalje unapređenje koda, sa ciljem povećanja njegove stabilnosti, efikasnosti i dugoročne održivosti.

2 Statička analiza

2.1 Clang-Tidy

Alat **Clang-Tidy** je korišćen za proveru koda u direktorijumima **Server** i **Client**. Pokretanje je vršeno pomoću batch skripte koja rekurzivno prolazi kroz sve **.cpp** fajlove i primenjuje sledeće grupe provera: **clang-diagnostic-***, **clang-analyzer-***, i **modernize-***.

Sažetak rezultata

- **Greške (errors):** 2 tipa, sa više pojavljivanja.
- **Upozorenja (warnings):** oko 8022 ukupno.

Greške (errors) i rešenja

(E1) Nedefinisiran std::basic_ostringstream<char> (#include <sstream>) Opis: Greška nastaje kada se koristi **std::ostringstream** bez uključivanja hedera **<sstream>**.

Fajlovi (reprezentativno):

- **Server/Headers/Admin.cpp,**
- **Server/Headers/Employee.cpp,**
- **Server/Headers/Person.cpp,**
- **Server/Headers/Server.cpp,**

- Server/Headers/Customer.cpp.

Rešenje: Dodati sledeći include u svaku prevodnu jedinicu gde se koristi `ostringstream`:

```
#include <sstream>
```

(E2) Konflikt Winsock zaglavlja (redefinicije tipova / expected identifier) Opis: Pojavljuju se višestruke greške (*“expected identifier”*, *“redefinition of ‘sockaddr’/‘fd_set’ ...”*) koje potiču iz Windows SDK hедера (`ws2def.h`, `winsock2.h`). Uzrok je uključivanje `<windows.h>` pre `<winsock2.h>`, što povlači zastareli `winsock.h` i izaziva konflikt.

Fajlovi (posredno): greške su prijavljene u SDK hederima, ali izvorni kod se nalazi u mrežnim modulima, npr. `Server/Headers/Thread.cpp`, `Server/Headers/Server.cpp`.

Rešenje: Osigurati ispravan redosled uključivanja u svim fajlovima koji koriste mrežu:

```
#define WIN32_LEAN_AND_MEAN // spreava povlaenje suvinih API-ja
#include <winsock2.h> // uvek pre <windows.h>
#include <ws2tcpip.h> // za InetPton/InetNtop
#include <windows.h> // tek posle winsock2.h
```

Upozorenja (warnings) i preporuke

(W1) Zastarele/nesigurne CRT i Winsock funkcije Opis: Korišćenje zastarelih ili nesigurnih funkcija.

Primeri:

- `getch()` → koristiti `_getch()` (`<conio.h>`),
- `strcpy()` → koristiti `strcpy_s()` ili preći na `std::string`,
- `ctime()`, `localtime()` → koristiti `ctime_s()`, `localtime_s()`,
- `inet_addr()`, `inet_ntoa()` → koristiti `InetPton()`, `InetNtop()`.

Fajlovi: `Client/Headers/Person.cpp`, `Client/Headers/Admin.cpp`, `Client/Headers/Customer.cpp`, `Server/Headers/Employee.cpp`, `Server/Headers/Complaint_Base.cpp`, `Server/Headers/Complaint_E.cpp`, `Server/Headers/Server.cpp`, `Server/serverProgram.cpp`, `Client/clientProgram.cpp`.

Napomena: ukupno oko 105 slučajeva, od čega `getch()` 69, `strcpy()` 28, ostalo pojedinačni slučajevi.

(W2) Modernizacija C++ koda (modernize checks)

- `modernize-use-nullptr` (~15 puta): zameniti `NULL` sa `nullptr`. **Fajlovi:** `Server/Headers/Server.h`, `Server/Headers/Thread.cpp`, `Server/Headers/Server.cpp`, `Client/Headers/Person.cpp`.
- `modernize-loop-convert` (~8 puta): zameniti klasične for petlje po kontejnerima range-based for.

Fajlovi: `Server/Headers/Person.cpp`, `Server/Headers/Admin.cpp`.

(W3) Sigurnosna upozorenja (clang-analyzer-security.insecureAPI strcpy) Opis: Korišćenje `strcpy` je prijavljeno kao nesigurno (CWE-119). **Fajlovi:** `Server/Headers/Complaint_Base.cpp`, `Server/Headers/Complaint_E.cpp`. **Rešenje:** koristiti `strcpy_s()` sa granicama bafera ili preći na `std::string`.

(W4) Neispravan poziv funkcije Opis: U `Server.cpp` je pozvana funkcija `WSAGetLastError` bez zagrada, što ispisuje pokazivač umesto vrednosti.

Rešenje:

```
std::cerr << "Error:␣" << WSAGetLastError() << std::endl;
```

(W5) Potencijalno neinicijalizovani objekti Opis: Detektovan je neinicijalizovani član u klasi `Client`.

Fajl: `Client/Headers/Client.h`.

Rešenje: inicijalizovati sve članove u konstruktoru, npr:

```
Client(const std::string& ipAddr = "127.0.0.1", int port = 7777)
: m_ipAddress(ipAddr), m_port(port), client(INVALID_SOCKET) {}
```

Zaključak

Greške: najkritičniji problemi su nedostajući `<sstream>` i konflikt Winsock zaglavlja.

Upozorenja: odnose se na zastarele i nesigurne funkcije, potrebu za modernizacijom koda, sigurnosne rizike sa `strcpy`, neispravne pozive funkcija i neinicijalizovane članove. Implementacijom predloženih rešenja smanjiće se rizik od sigurnosnih propusta i poboljšaće se čitljivost i stabilnost koda.

2.2 Cppcheck

Alat **Cppcheck** je korišćen za statičku analizu koda sa ciljem da se otkriju potencijalni problemi koji nisu uvek detektovani kompajlerom. Analiza je obuhvatila i `Server` i `Client` deo projekta.

2.2.1 Glavne kategorije detektovanih problema

- **Nedostajući sistemski hederi (missingIncludeSystem)**

Prijavljeno u velikom broju fajlova, npr. `Server/Headers/Person.h`, `Server/Headers/Server.h`, `Client/Headers/Person.h`, `Client/Headers/Client.h`.

Ovi upozorenja ne predstavljaju realan problem jer Cppcheck ne zahteva standardne biblioteke, ali ukazuju na zavisnost projekta od platforme (Windows-specifični hederi poput `conio.h`, `windows.h`, `ws2tcpip.h`).

- **Neinicijalizovane promenljive (uninitMemberVar)**

`Thread::data` u `Server/Headers/Thread.h`, kao i `Person::sex`, `Person::age`, i `Person::DOB` u `Client/Headers/Person.h`.

Rešenje: dodati inicijalizaciju u konstruktorima:

```
Person::Person() : sex('U'), age(0), DOB("") {}
```

- **Klase bez konstruktora (noConstructor)**

Klase `Goods`, `Cash`, `Complaint_Base`, `Complaint_E` nemaju deklarisan konstruktor iako poseduju privatne članove.

Rešenje: eksplicitno deklarirati konstruktore i inicijalizovati sve članove.

- **Pogrešan redosled u initializer listi (initializerList)**

Detektovano u `Server::m_port` (`Server/Headers/Server.h`). Članovi klase se uvek inicijalizuju redosledom deklaracije.

Rešenje: prilagoditi redosled initializer liste redosledu deklaracije članova.

- **Funkcije koje mogu biti const (functionConst)**
Na primer: `Admin::view`, `Admin::accounts`, `Cash::get_final_cash`, `Complaint_Base::view`, `Complaint_Base::see`, `Person::transfer_to_file`.
Rešenje: dodati ključnu reč `const` kada funkcija ne menja unutrašnje stanje objekta.
- **Funkcije koje mogu biti static (functionStatic)**
Detektovano u `Person::initialize_goods`, `Person::initialize_cash`, `Server::updateActivity`, `Server::getActivity`, `Server::closeSocket`.
Rešenje: ako funkcija ne koristi stanje objekta, može biti deklarirana kao `static` radi performansi.
- **Konstruktori koji nisu explicit (noExplicitConstructor)**
Klase `Server`, `Employee`, `Admin`, `Customer`, `Client`.
Rizik: implicitne konverzije.
Rešenje: koristiti `explicit` za jednoargumentne konstruktore.
- **Opseg promenljivih (variableScope)**
Promenljive poput `ans`, `previous`, `Balance` imaju širi opseg nego što je potrebno (npr. `Admin.cpp`, `Complaint_Base.cpp`, `Person.cpp`).
Rešenje: deklarirati promenljive što bliže mestu korišćenja.
- **Problemi sa prenosom argumenata (constParameterReference)**
Na primer: parametri funkcija `Person::update_cash`, `Thread::Rec`, `Thread::Client` mogu biti prosleđeni kao `const &`.
Rešenje: koristiti `const T&` za velike objekte da bi se izbeglo nepotrebno kopiranje.
- **Nepotrebne ili sumnjive operacije (constStatement)**
Detektovano u `Admin.cpp`, `Person.cpp` (izrazi tipa `+` ili `-` bez efekta).
Rešenje: proveriti da li je to bug ili nepotreban kod.
- **Problemi sa prenosom argumenata funkcija (funcArgNamesDifferent)**
Na primer: `search` u `Admin.cpp` koristi različita imena parametara u deklaraciji i definiciji (`str` vs `key`).
Rešenje: ujednačiti imena parametara.
- **Portabilnost: korišćenje fflush(stdin)**
Detektovano u `Admin.cpp` (više puta). Ovo ponašanje nije definisano na svim sistemima.
Rešenje: koristiti druge metode za čišćenje bafera ulaza (`cin.ignore()` u C++).
- **Stilski problemi**
 - C-style kastovi `((type)ptr)` u `Server.cpp`. Preporuka je korišćenje C++ kastova (`static_cast`, `reinterpret_cast`).
 - **Unused variables** (`index` u `Admin.cpp`). Ukloniti ako nisu potrebne.

2.2.2 Zaključak analize Cppcheck-a

Analiza je otkrila značajan broj stilskih i sigurnosnih problema. Najkritičniji su neinicijalizovani članovi klasa (`Person`, `Thread`), upotreba nesigurnih funkcija (`fflush(stdin)`), kao i implicitni konstruktori bez `explicit` modifikatora. Preporučuje se uvođenje konstruktora sa inicijalizacijom, dodavanje `explicit`, kao i korišćenje modernih C++ tehnika (`const`, `auto`, range-based petlje) kako bi se unapredio kvalitet i stabilnost koda.

3 Dokumentacija (Doxygen)

3.1 Doxygen (generisana dokumentacija)

Opis. Za izradu tehničke dokumentacije korišćen je **Doxygen**. Dokumentacija je generisana iz izvornog koda (Server/ i Client/ sloj), u HTML formatu, sa uključenim dijagramima (Graphviz). Ovaj deo sumira *šta je tačno generisano*, kako je organizovano i koliko je dokumentacija upotrebljiva za razumevanje arhitekture.

Kako je pokrenut

doxygen Doxyfile

Ključne postavke: PROJECT_NAME="Online Shopping System", RECURSIVE=YES, GENERATE_HTML=YES, HAVE_DOT=YES, CALL_GRAPH=YES, CALLER_GRAPH=YES, EXTRACT_ALL=YES, EXTRACT_PRIVATE=YES. HTML početna strana je index.html.

Struktura generisane dokumentacije (šta je dostupno)

- **Navigacija (levi panel):** ulazne tačke ka listama klasa, fajlova i hijerarhiji; pretraga simbola.
- **Liste klasa** (po podsistemima):
 - **Server sloj:** Server, Thread, Complaint_Base, Complaint_E, Employee, Admin, Customer, Person, Cash, Goods ...
 - **Client sloj:** Client, Person (+ pomoćne klase/strukture vezane za UI/menije).
- **Dokumentacija fajlova:** pregled *.h i *.cpp sa listama funkcija, globala i tipova.
- **Detaljne stranice klasa:** opis, članovi (public/protected/private), metode, atributi, *related functions*, nasleđivanje (ako postoji).
- **Hijerarhija tipova:** grananje po baznim i izvedenim klasama (gde je primenljivo).
- **Dijagrami (Graphviz):**
 - *Class diagrams* – veze između klasa, nasleđivanje, agregacije (gde su komentarima naznačene).
 - *Call graph / Caller graph* – pozivi iz funkcije i ka funkciji (pomaže u razumevanju toka kroz Server/Client).

Istaknuti elementi (primeri upotrebe)

- **Server** – stranica klase sadrži interfejs (npr. start/stop, rukovanje konekcijama), pozivne dijagrame za funkcije poput obrade zahteva i logovanja aktivnosti.
- **Thread** – prikaz odgovornosti niti (prihvatanje klijenata, čitanje/pisanje), *caller graph* pomaže da se vidi ko otvara/zatvara niti.
- **Complaint_Base / Complaint_E** – pregled metoda za rad sa žalbama, mesta gde se popunjavaju polja, i gde se pozivaju iz administratorskih menija.
- **Person / Admin / Employee / Customer** – uvid u zajedničke metode (npr. prikaz podataka, ažuriranje stanja) i specifične ovlašćenja/akcije.
- **Client** – dokumentovan interfejs za povezivanje na server, login scenario, slanje zahteva/čitanje odgovora.

Pokrivenost i upotrebljivost

- **Obim:** automatski obuhvata sve fajlove i klase (zbog `EXTRACT_ALL=YES`); privatni članovi su uključeni (`EXTRACT_PRIVATE=YES`).
- **Komentari:** gde postoje Doxygen komentari (`///`, `/** ... */`), stranice su informativnije (sa *brief*/*details*); gde komentari nedostaju, stranice su „strukturalne”, ali i dalje korisne zbog dijagrama i signatura.
- **Dijagrami:** `CALL_GRAPH`/`CALLER_GRAPH` nude jasan pregled toka – posebno korisno za `Server` (obrada zahteva, log) i `Thread` (komunikacija sa klijentom).

Kvalitativna zapazanja

- **Razumevanje arhitekture:** prikaz `Server/Client` podel i centralnih entiteta (`Server`, `Thread`, `Client`, `Person`) čini dokumentaciju dobrim “mapom sistema”.
- **Otkrivanje “vrućih tačaka”:** call grafovi pokazuju najčešće prelaze između funkcija (mesta gde kasnije možemo fokusirati profilisanje/optimizaciju).
- **Uočene praznine:** pojedine klase/metode nemaju *brief* ili detaljne komentare; `EXTRACT_ALL` ih izlistava, ali bez konteksta (posebno u pomoćnim modulima).

Preporuke za unapređenje

1. **Dopuniti komentare:** za ključne klase (`Server`, `Thread`, `Client`, `Person/Admin/Employee/Customer, Complaint_*`) dodati *brief* (jedna–dve rečenice) i po potrebi *details*. Za funkcije koje se često pozivaju, navesti pre/post uslove i izuzetke.
2. **Istaknuti protoke:** kod metoda koje čine glavni tok (npr. login, obrada porudžbine, žalbe) dopisati kratke sekvence (*Steps*) i ostaviti uključene `CALL_GRAPH`/`CALLER_GRAPH`.
3. **`EXTRACT_STATIC=YES`:** uključiti statičke članove/funkcije u dokumentaciju – korisno za helper utilise i centralne tačke konfiguracije.
4. **Organizacija izlaza:** `OUTPUT_DIRECTORY` držati u `./docs/html` kada budeš kačio na GitHub Pages ili repo (lakše pregledanje).

4 Dinamička analiza

4.1 Dr. Memory

Tabela 1: Dr. Memory — agregat nalaza (primer format)

2*Kategorija	# Nalaza	
	Server	Client
LEAK (def./indir./poss.)		
UNINITIALIZED READ		
INVALID READ/WRITE		
USE-AFTER-FREE / DOUBLE-FREE		
Ostalo		

4.2 Very Sleepy Profiler (analiza performansi)

Opis. Very Sleepy je sampling profiler za Windows. U ovoj analizi profilisane su odvojeno binarke `server.exe` i `client.exe`. Podaci su eksportovani u `.callgrind` i `.csv` formate i korišćeni za identifikaciju *hotspot* funkcija (Inclusive/Self vreme) i kritičnih putanja poziva.

Metodologija

- **Profilisanje:** sampling (*stack snapshots*); trajanje i interval uzorkovanja prema podrazumevanim postavkama Very Sleepy.
- **Artefakti:** `server.callgrind`, `server.csv`, `client.callgrind`, `client.csv`.
- **Napomena:** CRT start funkcije (`__tmainCRTStartup`, `mainCRTStartup`) su očekivano visoko u listi zbog ulazne tačke procesa i ne predstavljaju cilj optimizacije.

Rezultati (Server)

Tabela 2: Server — Top funkcije po % ukupnog vremena (Inclusive)

Funkcija	Vreme [s]	Udeo %
[00007FFD1F4D8C5C]	657.783	23.24%
<code>__tmainCRTStartup</code>	330.008	11.66%
<code>mainCRTStartup</code>	330.008	11.66%
<code>Server::run</code>	330.008	11.66%
<code>Thread::Rec</code>	329.855	11.65%
<code>NSPStartup</code>	329.855	11.65%
<code>Tcpip4_WSHAddressToString</code>	329.370	11.63%
<code>worker</code>	96.620	3.41%

Top funkcije (ukupno vreme, iz `server.callgrind`).

Tabela 3: Server — Projektne funkcije (Inclusive)

Funkcija	Vreme [s]	Udeo %
<code>Server::run</code>	330.008	11.66%
<code>Thread::Rec</code>	329.855	11.65%
<code>worker</code>	96.620	3.41%
<code>Person::login</code>	96.620	3.41%
<code>Person::buy</code>	0.131	0.00%
<code>std::__basic_file<char>::open</code>	0.089	0.00%
<code>std::basic_filebuf<...>::open</code>	0.089	0.00%

Projektne funkcije (filtrirano).

Zapažanja (Server).

- **Mrežni sloj dominira:** `Server::run`/`Thread::Rec` nose glavninu vremena \Rightarrow trošak je fokusiran na petlju prihvata/obradu i čitanje sa soketa.

- **Sistemske funkcije:** prisutni `NSPStartup/Tcpip4_WSHAddressToString` i jedna anonimna adresna tačka; to su spoljne (OS/network) rutine, nisu primarni kandidati za refaktorisanje.
- **I/O i fajlovi:** pojavljuju se `std::basic_file*::open` (zanemarljiv udeo).

Rezultati (Client)

Tabela 4: Client — Top funkcije po % ukupnog vremena (Inclusive)

Funkcija	Vreme [s]	Udeo %
<code>__tmainCRTStartup</code>	325.177	21.14%
<code>mainCRTStartup</code>	325.177	21.14%
<code>Person::login</code>	295.432	19.20%
<code>__gnu_cxx::stdio_sync_filebuf<...>::underflow</code>	279.252	18.15%
<code>std::istream::sentry::sentry</code>	191.348	12.44%
<code>Customer::start</code>	62.839	4.08%
<code>Person::buy</code>	37.594	2.44%
<code>Person::profile</code>	4.260	0.28%

Top funkcije (ukupno vreme, iz `client.callgrind`).

Tabela 5: Client — Projektne funkcije (Inclusive)

Funkcija	Vreme [s]	Udeo %
<code>Person::login</code>	295.432	19.20%
<code>Customer::start</code>	62.839	4.08%
<code>Person::buy</code>	37.594	2.44%
<code>Person::profile</code>	4.260	0.28%
<code>heading</code>	0.806	0.05%
<code>Client::Rec</code>	0.140	0.01%

Projektne funkcije (filtrirano).

Zapažanja (Client).

- **Ulaz sa konzole:** veliki udeo vremena u `stdio_sync_filebuf::underflow` i `std::istream::sentry::se` ukazuje da su `std::cin`/konzolni ulazi glavni *hotspot*.
- **Tokovi aplikacije:** `Person::login`/`Customer::start`/`Person::buy` čine glavne korisničke scenarije.

Preporuke za optimizaciju

Server.

- **Mrežni I/O:** razmotriti *overlapped* ili neblokirajući I/O radi smanjenja čekanja u `Thread::Rec`; grupisati `send/recv` u veće bačeve.
- **Alokacije i stringovi:** smanjiti transient alokacije (`std::string::reserve`, `emplace_back`); preferirati prealocirane bačeve u vrućim putanjama.

- **Logika petlje:** u `Server::run` ograničiti rad u kritičnim sekcijama; po potrebi uvesti *fine-grained* zaključavanje ili bez-ključne strukture.

Client.

- **Konzolni I/O:** isključiti sinkronizaciju sa C stdio radi bržeg `iostream`:
`std::ios::sync_with_stdio(false); std::cin.tie(nullptr);`
- **Parsiranje/formatiranje:** minimizovati ponovljena formatiranja (`std::ostringstream`); po potrebi preći na `fmt` biblioteku ili prealocirane `char` ba fere.
- **Protokoli i round-trips:** smanjiti broj malih zahteva; preferirati manje, „deblje“ poruke.

Napomena

Dobijeni % i vremena su *sampling* procene. Statičke optimizacije iz odeljka Clang-Tidy (npr. zamena `strcpy`, `_getch`, `inet_*` → `InetPton/Ntop`) i modernizacije (`nullptr`, range-based for) su komplementarne: one smanjuju rizik i *možda* indirektno trošak, dok Very Sleepy precizno pokazuje gde CPU vreme odlazi tokom realnog korišćenja.

Tabela 6: Top funkcije po Inclusive time (primer format)

Binarni modul	Funkcija	Inclusive %	Call Count
server.exe	<code>process_request(...)</code>		
server.exe	<code>parse_json(...)</code>		
client.exe	<code>render_menu(...)</code>		
client.exe	<code>do_login(...)</code>		

4.3 Dr. Memory

Alat **Dr. Memory** je korišćen za dinamičku analizu izvršavanja servera i klijenta. Alat detektuje curenja memorije, neinicijalizovane ili nevažeće pristupe memoriji, *use-after-free*, dvostruko oslobađanje i oštećenja stoga/heap-a.

4.3.1 Dr. Memory — analiza servera

Rezultati. U fajlu sa rezultatima zabeleženo je ukupno **61 prijava** (*Error #1–Error #61*). Najčešće kategorije su:

- **UNINITIALIZED READ** (preko 50 prijava): čitanja neinicijalizovanih registara/memorije (npr. `eax`, `r9`, `eflags`) u `WSAStartup/accept/closesocket`, pozivane iz `Server::start/Server::L`
- **UNADDRESSABLE ACCESS** (1 prijava): pristup izvan stoga (*beyond top of stack*) u toku `Server::start`.
- **Potential errors (blocklist):** deo UNINIT čitanja u `KERNELBASE.dll/WS2_32.dll` alat klasifikuje kao verovatne lažno pozitivne (OS/CRT).

Diskusija grešaka. Dominantne prijave su neinicijalizovana čitanja u WinSock toku, tipično:

- `replace_memset` → `WSAStartup` → `Server::start` → `main` — UNINIT pri inicijalizaciji WinSock-a.
- `WS2_32.dll!closesocket` → `Server::Listen` → `Server::start` → `main` — UNINIT na registru `r9` pri zatvaranju konekcije.

- *Beyond top of stack* — upućuje na pogrešnu upotrebu lokalnih bafera/pokazivača ili prelazak opsega.

Preporuke za ispravku.

1. **Inicijalizacija i provere:** pre svakog WinSock poziva (`WSAStartup/accept/closesocket`) inicijalizovati strukture (`sockaddr_in`, `WSADATA`) i proveriti povratne kodove.
2. **RAII za sokete i memoriju:** uvesti omotače koji automatski zatvaraju `SOCKET` u destruktoru; posle `closesocket` postaviti deskriptor na `INVALID_SOCKET`.
3. **Stog i životni vek objekata:** ne vraćati/prosleđivati pokazivače na lokalne bafer-e; za veće strukture koristiti dinamičku alokaciju ili statički trajni objekat.
4. **Filtriranje šuma:** potencijalne greške iz OS/CRT (`blocklist`) tretirati kao lažno pozitivne ukoliko ne postoji uticaj na korisnički kod.

Zaključak. Nisu uočena tipična curenja ili dvostruka oslobađanja, ali su detektovana brojna UNINIT čitanja i jedno oštećenje stoga. Uz doslednu inicijalizaciju, RAII upravljanje soketima i reviziju upotrebe lokalnih bafera, stabilnost serverskog dela se značajno unapređuje.

4.3.2 Dr. Memory — analiza klijenta

Rezultati. Analiza više pokretanja (*clientProgram.exe*) pokazuje ponovljiv obrazac:

- **UNINITIALIZED READ** tokom WinSock inicijalizacije i DNS/servisnih upita: `WSALookupServiceNext` (registri `eax`, `eflags`; čitanja iz memorijskih slotova oko `0x7ef930`).
- **UNINITIALIZED READ** pri gašenju/rezoluciji: `closesocket` (reg. `r9`) i `getnameinfo/WSAUnhookBlock` (reg. `rcx`).
- **UNADDRESSABLE ACCESS** u ranoj CRT/relocation fazi (*beyond top of stack*), pre korisničke logike.
- **Potential errors (blocklist):** deo UNINIT prijava iz `KERNELBASE.dll/WS2_32.dll` klasifikovan kao OS/CRT šum.

Diskusija grešaka. Ključne tačke:

- `Client::start` → `main` — UNINIT u `WSAStartup/WSALookupServiceNextW` tipično nastaje kada strukture/parametri nisu nultto inicijalizovani ili se oslanjaju na nedefinisane vrednosti.
- `Client::Connect` — UNINIT na `r9` pri `closesocket` sugerise dvostruko zatvaranje ili upotrebu nevalidnog `SOCKET`-a; UNINIT u `getnameinfo` ukazuje na ulazne parametre/dužine.
- *Beyond top of stack* u CRT startupu najverovatnije nije iz korisničkog koda, ali opravdava proveru build okruženja/flagova i upotrebe velikih lokalnih bafera.

Preporuke za ispravku.

1. **Deterministička inicijalizacija:** nultu inicijalizovati `WSADATA`, `sockaddr_in`/`addrinfo` i sve pomoćne strukture; validirati `len` vrednosti pre poziva API-ja.
2. **Bezbedno rukovanje soketima:** RAII omotač za `SOCKET`; pre `closesocket` proveriti `socket != INVALID_SOCKET`, a potom setovati `INVALID_SOCKET`.
3. **Ulazno/izlazni baferi:** koristiti `std::vector<char>` ili `std::array` i osigurati potpuni inicijalizacioni opseg pre `send/recv/getnameinfo`.
4. **Smanjenje šuma:** blocklistovane *potential errors* iz sistemskih DLL-ova ignorisati osim ako ne postoje indikacije na korisničke parametre.

Zaključak. Klijent deli obrazac sa serverom: dominiraju UNINIT čitanja u WinSock stazama, uz nekoliko relevantnih tačaka u korisničkom kodu (zatvaranje soketa, validacija parametara i bafera). Primena preporuka (inicijalizacija, RAII, proverene dužine/parametri, bezbedni baferi) umanjuje rizik od oštećenja memorije i povećava pouzdanost klijentskog dela.

5 Zaključak

Analiza projekta **Online Shopping System** kombinovanjem statičkih alata (*Clang-Tidy*, *Cppcheck*), generisane dokumentacije (*Doxygen*), dinamičke analize memorije (*Dr. Memory*) i profajlsanja performansi (*Very Sleepy*) obezbedila je sveobuhvatan uvid u kvalitet i pouzdanost implementacije.

Pozitivni aspekti. Arhitektura klijent-server je jasno razdvojena i funkcionalno koherentna. Doxygen dokumentacija (hijerarhija tipova, call grafovi) olakšava razumevanje uloga ključnih komponenti, dok rezultati profilisanja potvrđuju da se najveći deo vremena troši u očekivanim tačkama (mrežni I/O na serveru, konzolni I/O na klijentu), bez neočekivanih uskih grla.

Uočeni problemi. Statička analiza ukazuje na upotrebu zastarelih/nesigurnih API-ja i na potrebu modernizacije C++ stila (npr. `nullptr`, `explicit`, `const`-ispravnost, range-based `for`), kao i na pojedine neinicijalizovane članove i nedosledan redosled u *initializer* listama. Dinamička analiza detektuje *UNINITIALIZED READ* događaje i pristupe izvan važećeg opsega stoga u stazama vezanim za `Winsock` i lokalne bafer; deo tih nalaza potiče iz sistemskih biblioteka i može biti blok-listiran, ali deo ukazuje na realne rizike u tretmanu `SOCKET` deskriptora i validaciji bafera pri `send/recv`.

Preporuke.

- Ispravan redosled uključivanja zaglavlja (`winsock2.h` pre `windows.h`, `WIN32_LEAN_AND_MEAN`) i zamena zastarelih funkcija sigurnijim alternativama (`strcpy` → `strcpy_s`/`std::string`, `getch` → `_getch`, `inet_*` → `InetPton/Ntop`).
- RAII pristup za sokete i resurse (jednoznačno vlasništvo; posle `closesocket` postaviti `INVALID_SOCKET`).
- Inicijalizacija i validacija svih struktura/bafera pre mrežnih poziva; izbegavanje prosleđivanja pokazivača na objekte sa ograničenim životnim vekom.
- Postupna modernizacija koda radi čitljivosti i održivosti: `explicit` konstruktori, `const`-korektnost, bezbedni kastovi, sužavanje opsega promenljivih.

Sumirano, projekat je funkcionalno zaokružen i dobro dokumentovan, uz jasno identifikovane tehničke tačke koje unapređuju stabilnost i kvalitet koda. Primena preporučenih izmena podiže pouzdanost sistema, olakšava buduće održavanje i usmerava dalji razvoj na najuticajnije delove koda.