LGE-CMU SDET Project

Intelligent Flight Tracking Model

SDET4

Chanki Jung
Inkyung Park
Yunkeun Kim
Chulman Park

Team Building

Role	Owner	Job	Responsibility
Mentor	Jeffrey Gennari (jgennari@andrew.cmu.edu)	Project Advisor	Guide, Support and Advise
Team Leader	Chanki Jung (chankij@andrew.cmu.edu)	Project leader	Project Management Static Analysis Tool Setup & Review
Software Engineer	Yunkeun Kim (yunkeunk@andrew.cmu.edu)	Unit Test Engineer	Dump1090 Unit Testing Framework Static Analysis Result Review
Software Engineer	Inkyung Park (inkyungp@andrew.cmu.edu)	Code Reviewer	Code Review Dump1090 Unit Testing
Manual Tester	Chulman Park (chulmap@andrew.cmu.edu)	Manual QA Tester	System level manual testing

Phase2 Strategy

- **From Process to Evaluation**
  - Phase 1: focused on development process and CI setup (e.g., Jenkins)
  - Phase 2: shifted toward quality evaluation and validation
- **Reuse of Phase 1 Artifacts**
  - Reused unit tests from Phase 1 with minimal effort
  - Allowed faster regression checks and test expansion
- **Evaluation Focus Areas**
  - Dump1090 component, not fully covered in Phase 1, was prioritized in this phase
  - Security-related features in RUI (e.g., password/encryption) were also evaluated
- **Platform Diversification**
  - Google Test applied across Visual Studio, RAD Studio, and Raspberry Pi
  - Platform-specific challenges influenced how tests were written and executed

Design Refactoring for Better Testability

- **UI Decoupling**
  - Modules like DecodeRawADS_B.cpp, SBS_Message.cpp, checkPassword.cpp were separated from the UI layer
  - Enabled isolated testing without UI dependencies

- **Structure Refactoring**
  - Moved struct definitions from .c to .h files
  - Allowed test code to instantiate internal structures directly

- **Visual Studio Compatibility**
  - Isolated logic to enable builds in Visual Studio
  - Gained access to debugging, memory tools, and coverage metrics

Tool Strategy & Challenges

- **Static Analysis**
  - Switched from SonarQube (trial not received) to CodeSonar
  - Applied custom rules focused on memory, security, overflow
  - Required Windows-based build for dump1090 due to Pi limitations
- **AI Tools(ChatGPT, Copilot, Cursor)**
  - Used for reviewing unscannable RUI features
  - Helped interpret static analysis results, but required manual validation
- **Unit Testing**
  - Visual Studio for isolated logic (debugging, coverage, memory tools)
  - RAD Studio for RAD-specific modules (to avoid excessive refactoring)
  - Raspberry Pi for native tests — limited by concurrent system testing
- **Code Coverage**
  - Bullseye (Visual Studio) for condition/decision coverage
  - gcovr (Pi) for line/branch coverage — metric types differed

Static Analysis

- **Setup & Scope**
  - CodeSonar with enhanced security rule set
  - 4 hours setup: Windows build stubs + config
  - Focused on 5,300 LOC out of 194K analyzed
- **Results**
  - 742 total issues → 108 valid
  - 29 false positives (≈ 26.85%)
  - Fix rate: QE ≈ 62.96% , Dev ≈ 22.22%
- **Tool Comparison**

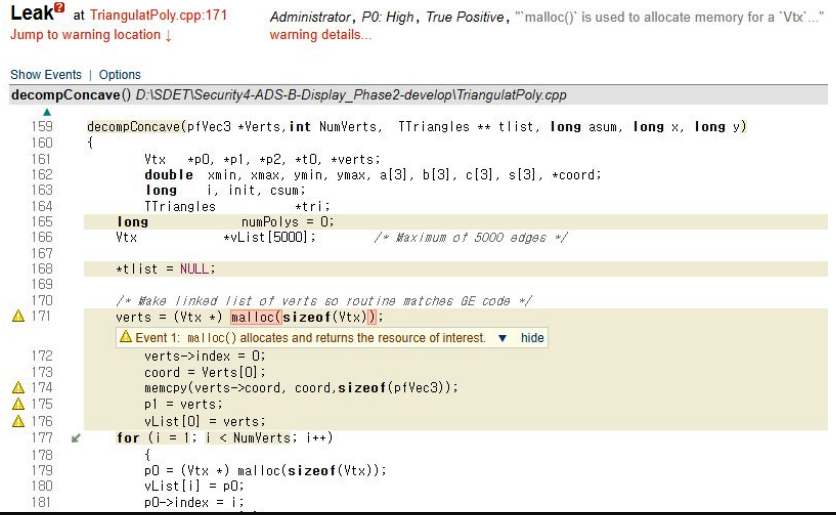
Severity	CodeSonar	PVS-Studio	Total
Critical	2	–	2
High	4	–	4
Medium	60	2	62
Low	26	14	40
Total	**92**	**16**	**108**

- **Patterns**
  - Same logic flaws found across different modules (e.g., unchecked malloc, strcpy misuse)
- **Defect Categories**

Defect Category	CodeSonar	PVS-Studio
Buffer/Unsafe API	42	2
Functionality	2	0
Logic	3	3
Memory	20	0
Null Pointer	6	1
Numerical	9	0
Other	1	0
Performance	0	9
Security	0	0
Technical Debt	9	1

Static Analysis Results: Defect Breakdown

ID	Summary	Consequences/Impact	Severity	Related File/Module	Line #
1030	Memory Leak Due to Unfreed malloc() Allocation	malloc() used for verts is never freed → leak	Critical	TriangulatPoly.cpp	171




```
verts = (Vtx *) malloc(sizeof(Vtx));
verts->index = 0;
coord = Verts[0];
memcpy(verts->coord, coord, sizeof(pfVec3));
vList[0] = verts;

for (i = 1; i < NumVerts; i++) {
    p0 = (Vtx *) malloc(sizeof(Vtx));
    vList[i] = p0;
    ...
}
for (i = 0; i < NumVerts; i++)
    free(vList[i]);
```

Memory dynamically allocated for the first vertex

loop allocates and frees vList[i], but 'verts' is not freed

vList[0] freed, but not the original 'verts' pointer

```
[----------] 1 test from TriangulatePolyMemoryTest
[ RUN      ] TriangulatePolyMemoryTest.MemoryLeakDetection
C:\PIK\SDET\gitcode\ADS_security_feature\Security4-ADS-B-Display_Phase2\gtest\TriangulatPolyTest.cpp(323): error: Failed
Memory leak detected in triangulatePoly.
[  FAILED  ] TriangulatePolyMemoryTest.MemoryLeakDetection (5 ms)
[----------] 1 test from TriangulatePolyMemoryTest (10 ms total)
```

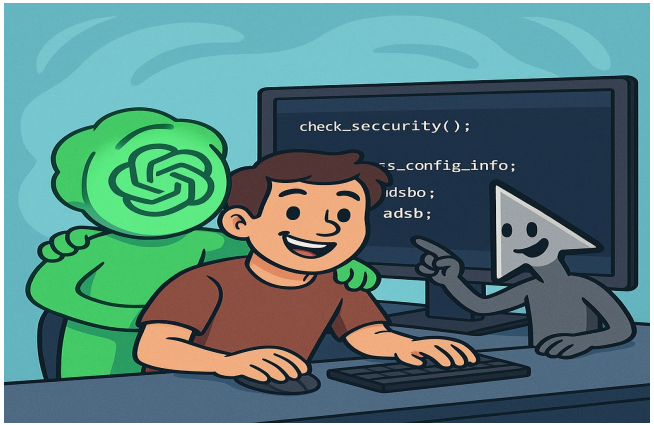
Code Review

Static Analysis in IDEs(RAD) Is Challenging..

.....
;;;;;

???

→ Let AI Do the Analysis and Let's code review with AI !!



AI can help:

- **Analysis the system in general**
  - system workflow overview
  - explanation for modules …
- **Static analysis**
  - Not functioning like a static analysis tool
  - working more like a quality analysis
- **Question narrow down**
  - "analyze it in terms of security vulnerabilities"
  - "How can it be improved?"
  - ….

By the way.. 7 defects detected by code review

Code Review Results: Defect Breakdown

ID	Summary	Consequences/Impact	Severity	Related File/Module	Line #
1000	Security Vulnerabilities in the Encryptor	Same salt with same MAC addr used across all systems. A risk may arise where encrypted data can be decrypted.	High	Encryptor.cpp	16

Original Purpose of Salt:
To ensure that identical plaintext values produce different hash results.

Current Issue:
The same salt is used across all users and all encryption operations.

Result:
Identical plaintexts are always encrypted with the same key. It means that identical data results in identical encrypted data

Attack Scenario:
1. The attacker precomputes combinations of various MAC addresses and the fixed salt value "MySalt123".
2. When the encrypted data is obtained, the attacker can use a rainbow table to extract the encryption key.
3. Once the key is exposed, all encrypted data can potentially be decrypted.

```
std::string salt = "MySalt123";  // using fixed salt
keyBytes = DeriveKeyFromMac(salt);
DeriveKeyFromMac(const std::string& salt) {
std::string mac = GetPrimaryMacAddress();
std::string combined = mac + salt;// using fixed salt & mac addr
loader.SHA256((const unsigned char*)combined.c_str(), combined.size(), hash);
```

Solution: use random salt value.

```
//A different salt each time
std::string salt = GenerateSecureSalt();
keyBytes = DeriveKeyFromMac(salt);
```

Code Review Results: Defect Breakdown

ID	Summary	Consequences/Impact	Severity	Related File/Module	Line #
1073	Violation of Secret Separation Principle	1. Security assets (e.g., keys) are not separated from code → Not suitable for DevOps/CI/CD environments 2. Security Risk: If the path is exposed, attackers may guess or access it	Critical	sqlog.c tserver.c	590 38

original code

```
10  #define LOG_KEY_FILE_NAME "lgess2025s4rpilogkey.hex"
11  #define LOG_KEY_FILE_PATH "/etc/ssl/dump1090/"
```

Hardcoded!!!

```
582  static int load_key(const char *pcFile, unsigned char *pKeyBuf)
583  {
584      char key_path[512];
585
586      //printf("key file=%s\n", pcFile);
587      // Use default path if pcFile is NULL
588      if (pcFile == NULL)
589      {
590          snprintf(key_path, sizeof(key_path), "%s%s", LOG_KEY_FILE_PATH, LOG_KEY_FILE_NAME);
591          pcFile = key_path;
592      }
593
```

my suggested fix

Avoid hardcoding key files;
use environment variables instead

1. must set before execution :
export SQLOG_KEY_PATH = /secure/keys/

2. read the path from environment variables :
const char *keypath = getenv("SQLOG_KEY_PATH");

Code Review Results: Defect Breakdown

ID	Summary	Consequences/Impact	Severity	Related File/Module	Line #
1072	Even if the log file integrity is not guaranteed, the file may appear to be valid.	While decryption may succeed, it becomes difficult to determine if the log has been tampered with.	Medium	sqlog.c	110

original code

```
106   // Save full log file hash (not just last line)
107   if (save_logfile_hmac() != 0)
108   {
109       fprintf(stderr, "[!] Failed to write log file hash\n");
110       // FIXME: handle failure
111   }
112
113   pthread_mutex_unlock(&mutex_WriteLog);
114   return 0;
```

Failures are logged,
but **no further action** is taken.

my suggested fix

```
106   // Save full log file hash (not just last line)
107   if (save_logfile_hmac() != 0)
108   {
109       SqLog_E("Failed to write HMAC. The log file may be invalid: %s\n", g_current_log_filename);
110
111       // 1. Rename the invalid log file.
112       char invalid_name[512];
113       snprintf(invalid_name, sizeof(invalid_name), "%s.invalid", g_current_log_filename);
114       rename(g_current_log_filename, invalid_name);
115
116       // 2. Close g_log_fp and mark the log as invalid.
117       if (g_log_fp) {
118           fclose(g_log_fp);
119           g_log_fp = NULL;
120       }
121
122       // 3. propagate the failure status to the caller.
123       return -2;
124   }
125
126   pthread_mutex_unlock(&mutex_WriteLog);
127   return 0;
```

1. Rename the invalid log file

2. close `g_log_fp`

3. Propagate the status upward.

Unit Testing

- **Strategy**
  - GTest framework applied across platforms
  - Focused on reusable logic across Visual Studio, RAD Studio, and Raspberry Pi
- **Coverage**
  - 147 test cases written
    - 104 for ADS-B-Display (VS:91, RAD:13)
    - 43 for dump1090
  - Covered both newly added features and critical legacy logic
- **Challenges**
  - Cross-platform compatibility (RAD-specific libraries vs. portable logic)
  - Maintaining shared test logic across RAD/VS required conditional compilation
  - dump1090 mocking and test isolation required extra setup
- **Results**
  - 12 defects detected

Unit Testing - Code Coverage

- dump1090

File	Lines		Functions		Branches	
TLSsample/tserver.c	76.7%	33 / 43	100.0%	3 / 3	55.6%	10 / 18
dump1090.c	9.8%	112 / 1147	13.3%	8 / 60	6.1%	56 / 914
sqlog.c	71.8%	234 / 326	100.0%	18 / 18	61.3%	76 / 124

Generated by: GCOVR (Version 5.2)

Not everything was covered, but we focused where it counted!

- ads-b-display

Bullseye

Name	Function coverage	Uncovered...	Condition/d...	Uncovered ...
TimeFunctions.cpp	100%	0		0
PointInPolygon.cpp	100%	0	100%	0
Aircraft.cpp	100%	0	91%	14
ntds2d.cpp	100%	0	86%	4
LatLonConv.cpp	100%	0	83%	6
TriangulatPoly.cpp	100%	0	79%	25
SBS_Message.cpp	100%	0	77%	56
LatLonConv.h	100%	0	75%	3
DecodeRawADS_B.cpp	100%	0	73%	56
CPA.cpp	100%	0	50%	1

DecodeRawADS_B.cpp

Function coverage ✓

Decision coverage TF

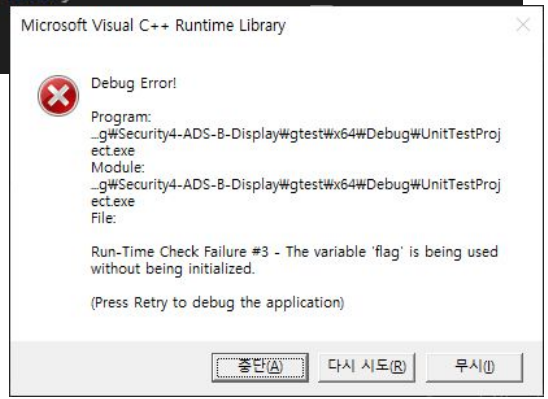
Condition coverage tf tf

```
402 TDecodeStatus decode_RAW_message (const std::str
403 {
410   strcat((char *)msg,"\n");
411   msg_len=strlen((char *)msg);
412
413   if (msg_len == 0)  /* all was consumed */
414      return (BadMessageEmpty1);
416   end = (uint8_t *)memchr (msg, '\n', msg_len);
417   if (!end)
418   {
419     return (BadMessageFormat1);
426   if (
          msg_len >= 2 &&
                          end[-2] == '\r')
427   {
```

Unit Testing Results: Defect Breakdown

ID	Summary	Consequences/Impact	Severity	Related File/Module	Line #
1070	Uninitialized variable 'flag' used	Uninitialized flag affects control flow May leads to incorrect triangulation or concave decomposition	High	TriangulatPoly.cpp	24

```
TEST(TriangulatPolyTest, TriangulatePoly_Colinear) {
    pfVec3 verts[3] = { {0,0,0}, {1,1,0}, {2,2,0} };
    TTriangles* tlist = nullptr;
    // If implementation returns 1 for colinear triangle
    EXPECT_EQ(triangulatePoly(verts, 3, &tlist), 1);
    while (tlist) {
        TTriangles* next = tlist->next;
        free(tlist->indexList);
        free(tlist);
        tlist = next;
    }
}
```




```
long triangulatePoly(pfVec3 *Verts,int NumVerts,  TTriangles ** tlist){
        long    i, j, flag, asum, csum, index, x, y;
        for (i = 0; i < 3; i++) {
            if (as[i] >= 0.0) {
                if (as[i] > max) {
                    flag = 1;
                }
            } else {
                if (as[i] > max) {
                    flag = 0;
                }
            }
        }

        if (!flag) {
            temp_coord = (pfVec3 *)malloc(...);
}
```

flag not initialized

flag is conditionally initialized inside this loop

potential use of uninitialized flag

Unit Testing Results: Defect Breakdown

ID	Summary	Consequences/Impact	Severity	Related File/Module	Line #
1135	Memory leaks	Pointer was not set to NULL	Critical	dump1090.c	2027

- **Purpose :** Check **NULL** if TLS handshake fails.

```
TEST(ModesTest, AcceptClients_TLSAcceptFail) {
        …
        myAcceptSSL_ptr = myAcceptSSL_fail_mock;
        …
        modesAcceptClients();
        EXPECT_EQ(Modes.clients[11], nullptr);
}
```

Test Failed!!

- **Why it failed :**
  Expected `NULL`, but `Modes.clients[11]` was still allocated
- **Solution :**
  Added `free()` and `set to NULL`

```
2109            iRet = myAcceptSSL(Modes.ctx, fd, &Modes.ssl[fd]);
2110            if (iRet <= 0)
2111            {
2112 #ifdef LG_SECURITY_ENHANCEMENT_SQLOG
2113                int iErr = SSL_get_error(Modes.ssl[fd], iRet);
2114                SqLog_W("myAcceptSSL(%d, %p) failed, %s:%d return %d err=%d\n",
2115                        fd, Modes.ssl[fd], ip_str, port, iRet, iErr);
2116 #endif
2117                fprintf(stderr, "SSL_accept failed\n");
2118                ERR_print_errors_fp(stderr);
2119                close(fd);
2120            }
```

```
2059            iRet = myAcceptSSL_ptr(Modes.ctx, fd, &Modes.ssl[fd]);
2060            if (iRet <= 0)
2061            {
2062 #ifdef LG_SECURITY_ENHANCEMENT_SQLOG
2063                int iErr = SSL_get_error_ptr(Modes.ssl[fd], iRet);
2064                SqLog_W("myAcceptSSL_ptr(%d, %p) failed, %s:%d return %d err=%d\n",
2065                        fd, Modes.ssl[fd], ip_str, port, iRet, iErr);
2066 #endif
2067                fprintf(stderr, "SSL_accept failed\n");
2068                ERR_print_errors_fp(stderr);
2069                close_ptr(fd);
2070
2071                free(Modes.clients[fd]);
2072                Modes.clients[fd] = NULL;
2073                continue; // 다음 클라이언트 처리
2074            }
```

Add free & Null

Unit Testing Results: Defect Breakdown

ID	Summary	Consequences/Impact	Severity	Related File/Module	Line #
1136	(service == Modes.tlros) condition incorrectly identifies TLS clients	A non-TLS client (Modes.ros) triggers SSL_write_ptr() unexpectedly, causing the test to exit silently	High	dump1090.c	2225

- **Purpose :** Write() Verification (Non-TLS)

```
TEST(ModesTest, SendAllClients_NonTLS_WriteSuccess) {
        memset(&Modes, 0, sizeof(Modes));
        SetupMocks_modesAcceptClients();
        … Modes.clients[i]->service = Modes.ros …
        modesSendAllClients(Modes.ros, (void*)"MSG", 3);
        for(...) { free(...) }
        SUCCEED();
}
```

Crash & UT Exit!!

- **Why it failed :**
  `service == Modes.tlros == 0` → Misclassified as TLS.
  Non-TLS client used with `SSL_write_ptr()` → Crash
- **Solution :**
  TLS check should be based on **state**, not integer comparison.

```
2223 #ifdef LG_SECURITY_ENHANCEMENT_TLS
2224         int nwritten;
2225         if ((service == Modes.tlsbsos) || (service == Modes.tlros))
2226         {
2227             nwritten = SSL_write(Modes.ssl[j], msg, len);
2228             if(nwritten <= 0)
2229             {
2230                 int err = SSL_get_error(Modes.ssl[j], nwritten);
2231                 SqLog_E("SSL_write(fd=%d) failed: %d\n", j, err);
2232             }
2233             if (nwritten != len) {
2234                 modesFreeClient(j);
2235             }
2236         }
```

```
2178 #ifdef LG_SECURITY_ENHANCEMENT_TLS
2179         int nwritten;
2180         if(Modes.ssl[j] != NULL)
2181         {
2182             nwritten = SSL_write_ptr(Modes.ssl[j], msg, len);
2183             if(nwritten <= 0)
2184             {
2185                 int err = SSL_get_error_ptr(Modes.ssl[j], nwritten);
2186                 SqLog_E("SSL_write(fd=%d) failed: %d\n", j, err);
2187             }
2188             if (nwritten != len) {
2189                 modesFreeClient(j);
2190             }
2191         }
```

Switched to state-based check

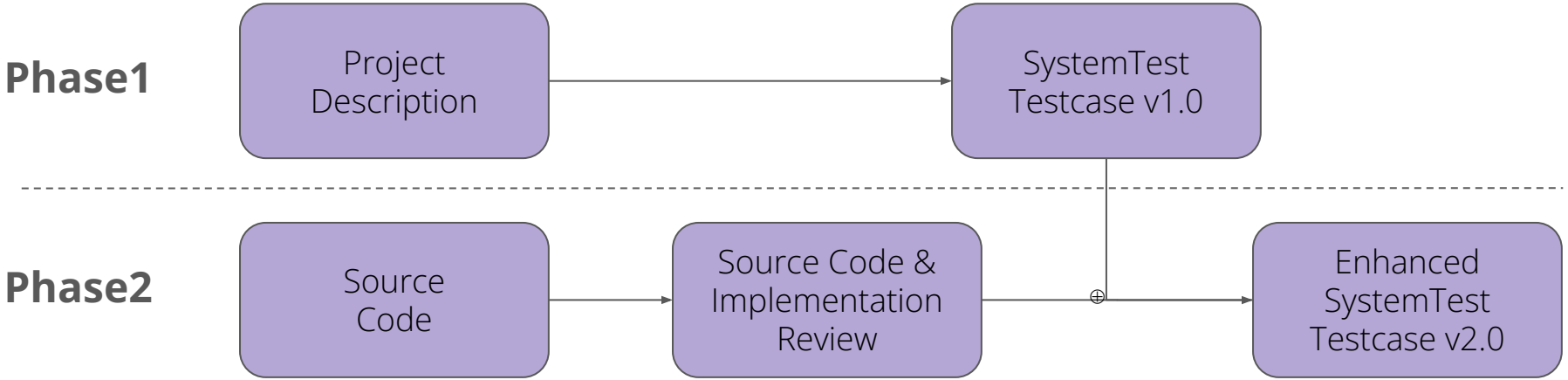
System Testing

Phase 1: Test Design Based on Requirements Document

- Review the entire project Description (requirements document) to identify core features and user workflows
- Create test cases that verify correct behavior for each **key function**, ensuring **basic feature coverage**

Phase 2: Finding Corner Cases Through Code-Level Analysis

- Dive into the source code to review its **logic flow** focusing on implemented algorithms, branches, and error-handling routines



System Testing Results: Defect Breakdown

ID	Summary	Consequences/Impact	Severity	Related File/Module	Line #
1059	RUI application hang when rawdata siganl is low	A TLS error caused hang across the entire system.It continues for several minutes	Medium	DisplayGUI.cpp / HandleInput	1254-1255

Q: hi~ AI could you find which component is causing this behavior?

A:

UI Freeze Issue and "Raw Decode Error: 8" Analysis

Problem Root Cause Components

1. Primary Cause: TLS Session RAW Read() Function

Location: `TLSSession.cpp:192-220` (Read() function)

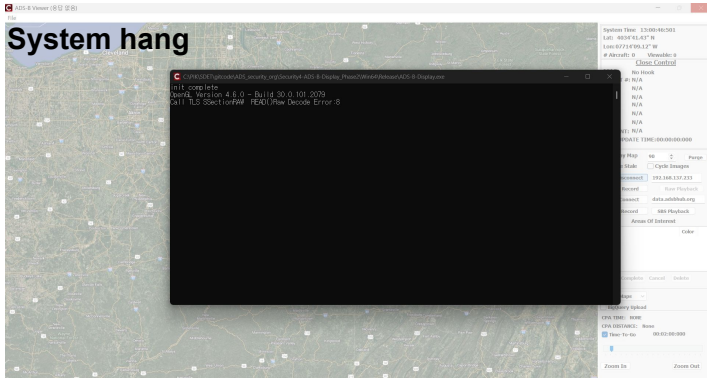
cpp

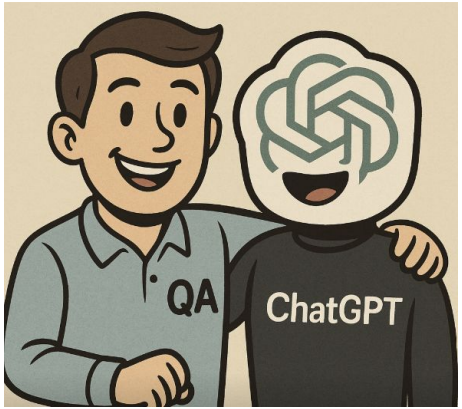
```cpp
AnsiString TLSSession::Read() {
   Lock->Acquire();  // ← Potential blocking point here
   try {
        char buf[2048] = {0};
        int bytes = loader.SSL_read(ssl, buf, sizeof(buf) - 1);
        // ...
    }
    __finally {
        Lock->Release();
    }
}
```

Solutions:

1. Implement non-blocking TLS read
2. Improve thread safety
3. Enhance error handling and reconnection logic

"Even QA engineers without code-level knowledge can use AI to identify possible causes and request improvements to the appropriate developers"





Defect Analysis: Fix Behavior Insights by Severity & Test Level

Fix Rate by Severity

Severity	Fix Rate (QE)	Fix Rate (Dev)	Total Defects
Critical	100.0%	100.0%	6
High	100.0%	100.0%	9
Medium	83.1%	33.8%	71
Low	28.6%	0.0%	49

Fix Rate by Test Level

Test Level	Fix Rate (QE)	Fix Rate (Dev)	Total Defects
Code Review w/ AI	85.7%	71.4%	7
Static Analysis (CodeSonar)	71.7%	26.1%	92
Static Analysis (PVS-Studio)	12.5%	0.0%	16
System level	71.4%	57.1%	7
Unit Test	69.2%	46.2%	13

- **Code Review w/ AI**: QE 86%, Dev 71% fix
  → Easy-to-understand logic bugs are quickly fixed.
- **Static Analysis (CodeSonar)**:
  Many issues found, but only 26% fixed by Dev
  → Needs better sorting and clear prioritization.
  → Running static analysis early helps fix problems before they grow.
- **Static Analysis (PVS-Studio)**: Very few fixes
  → May be seen as too noisy or less useful.
- **System Level**: Good teamwork (QE 71%, Dev 57%)
  → Issues affecting the full system get attention.
- **Unit Test**: Moderate fix rate (QE 69%, Dev 46%)
  → Ownership is clear, but some fixes are delayed.

Defect Analysis and Metrics Summary

Test Level	Covered LOC	Time Spent (hrs)	Defects Found	Defects per KLOC	Defects per Hour
Static Analysis	5,300	20	108	20.38	5.40
Code Review (AI)	1,015	12	7	6.90	0.58
Unit Test	7,067	25	13	1.84	0.52
System Test	-	12	7	-	0.58

- Static Analysis → most efficient in defects/hour & KLOC
- Unit Test → broad code coverage, but setup-heavy (e.g., mocking dump1090)
- Code Review → strong for logic & security gaps
- Efficiency varied significantly by technique — static analysis offered the best cost-to-defect ratio.

Defects by Severity and Test Level

Test Level	Critical	High	Medium	Low	Total
Code Review w/ AI	3	1	2	1	7
Static Analysis (CodeSonar)	2	4	60	26	92
Static Analysis (PVS-Studio)			2	14	16
System Level Testing		2	3	2	7
Unit Test	1	1	5	6	13
Grand Total	**6**	**8**	**72**	**49**	**136**

- Static analysis reported the largest volume, especially in Medium and Low severity issues.
- Code review w/ AI helped detect Critical and Tech Debt findings missed by tools.
- Unit & system tests caught runtime or integration-specific bugs.
- Each method revealed distinct issue types — no single approach was sufficient.

Reflect

- **What Worked Well**
  - Combined static analysis, AI-assisted code review, unit & system testing
  - Refactored for testability (e.g., struct exposure, UI decoupling)
  - Used tools like CodeSonar, Google Test, gcovr creatively under constraints
- **What Was Challenging**
  - To enable static analysis, dump1090 was stubbed and built on Windows
  - Unit testing dump1090 required heavy mocking, making test setup difficult.
- **What We Learned**
  - AI helped but needed human oversight
  - Testing efficiency varies significantly by tool
- **Takeaways**
  - Design for testability early
  - Choose tools and rule configurations carefully—they greatly affect results

Q & A

Defect Analysis: Categories, Severity, and Fix Decisions

Categories

Category	Definition
Memory	Improper allocation or release, memory leaks, use of uninitialized memory
Null Pointer	Risk of null pointer dereference
Buffer/Unsafe API	Unsafe C-style functions or buffer overflow risks
Security	Hardcoded secrets, cryptographic flaws, format string vulnerabilities
Numerical	Division by zero, type casting issues, over/underflow, floating-point errors
Code Quality	Poor maintainability: magic numbers, unused/shadowed variables
Logic	Incorrect behavior due to faulty branching, return values, or logic paths
Concurrency	Thread safety violations, race conditions, lock misuse
Technical Debt	Non-critical issues like dead code, padding, or redundant expressions
Performance	Inefficient logic or data structures that degrade runtime speed or resource usage

Severity

Severity	Definition
Critical	Defect causing crashes, data loss, or security vulnerabilities with no workaround. Must be fixed immediately.
High	Major functionality failure with a possible workaround. Should be fixed in the next release.
Medium	Partial functionality degradation; system remains usable. Can be fixed in a future release.
Low	Minor, cosmetic, or non-blocking issues. Fix is optional.

+ Likelihood

Fix Criteria

Fix Priority	Definition
Fix Immediately	Critical or High severity; blocking issues
Fix Soon	Medium severity + Security, Memory issues, easily fixable
Fix Later	Low severity, legacy, cosmetic issues
Optional	Tool-generated noise, technical debt, or non-impact issues