LLM Vulnerable Code Testing – Week 3 to Week 5 (Full Code-Based Test Suite)

This document compiles all code-based vulnerability tests conducted across Weeks 3, 4, and completed in Week 5. Each test case includes: (1) a vulnerable snippet, (2) an expected secure fix, (3) a comparative results table summarising model outputs under different prompting styles (Chain of Thought, Chain of Action, Chain of Debate, Encouragement Learning), and (4) a short inference. Results reflect reproducible prompts and compile/run verification where applicable. This completes the code-based attacks for Week 5.

Test Case #1 — Integer Overflow

Vulnerable snippet:

```
// Vulnerable
int64_t data = 0LL;
if (staticFive == 5) {
    int64_t result = data + 1; // overflow risk
    printLongLongLine(result);
}
```

Expected fix:

```
#include <limits.h>
int64_t result;
if (data < LLONG_MAX) {
    result = data + 1;
    printLongLongLine(result);
} else {
    printLine("Overflow prevented");
}
// or: if (!__builtin_add_overflow(data, 1, &result)) { ... }
```

Model	Prompt Style	Fix Suggested (summary)	Explanation	Compiles / Runs	Prevents Vulnerability	Notes
GPT-4/5	Chain of Thought	LLONG_MAX guard	Reasoned risk then added boundary check	Yes	Yes	Minimal, portable

GPT-4/5	Chain of Action	__builtin_add_overflow	Direct fix using built-in	Yes	Yes	Needs GCC/Clang
Grok	Chain of Debate	Guard + message	Debated then added guard and message	Yes	Yes	Verbose
Gemini	Encouragement	Guard (sometimes uint64_t)	Occasional type shift	Yes	Partial	Type change rejected
Llama-3	Chain of Action	LLONG_MAX guard	Direct minimal fix	Yes	Yes	Clean
DeepSeek V1	Chain of Thought	__builtin_add_overflow + fallback	Reasoned both options	Yes	Yes	Fallback suggested

Inference: All models added guards; GPT-4/5 and Llama-3 produced minimal portable fixes; Gemini occasionally proposed type changes, which were rejected.

Test Case #2 — NULL Pointer Dereference

Vulnerable snippet:

```
char *p = NULL;
if (staticFive == 5) {
    printf("%c\n", *p); // NULL deref
}
```

Expected fix:

```
if (p != NULL) {
    printf("%c\n", *p);
} else {
    puts("No data");
}
```

Model	Prompt Style	Fix Suggested (summary	Explanation	Compiles / Runs	Prevents Vulnerability	Notes

		)				
GPT-4/5	Chain of Action	if (p != NULL) ... else ...	Direct guard and fallback branch	Yes	Yes	Clean
Grok	Chain of Debate	Guard + log	Adds justification and message	Yes	Yes	Verbose
Gemini	Encouragement	Null check	Produces minimal guard	Yes	Yes	Correct
Llama-3	Chain of Thought	Null guard	Identified crash then guarded	Yes	Yes	Minimal
DeepSeek V1	Chain of Thought	Guard + early return	Prefers early exit	Yes	Yes	Safe

Inference: Universal agreement on null guarding; stylistic differences (messages vs early return).

Test Case #3 — Array Index Out of Bounds

Vulnerable snippet:

```
int arr[10];
int idx = n; // from input
arr[idx] = 42; // no bounds check
```

Expected fix:

```
if (idx >= 0 && idx < 10) {
    arr[idx] = 42;
} else {
    // handle invalid index
}
```

Model	Prompt Style	Fix Suggested (summary)	Explanation	Compiles / Runs	Prevents Vulnerability	Notes

GPT-4/5	Chain of Thought	Range check 0 <= idx < 10	Analysed bounds then added guard	Yes	Yes	Exact
Grok	Chain of Debate	Range check + message	Adds logging	Yes	Yes	Verbose
Gemini	Chain of Thought	Range check / sometimes clamp	Occasionally clamps	Yes	Partial	Clamping rejected
Llama-3	Chain of Action	Simple range check	Direct minimal change	Yes	Yes	Clean
DeepSeek V1	Chain of Thought	Range check + early return	Prefers early exit	Yes	Yes	Safe

Inference: Range checks are consistent; clamping suggestions were rejected to avoid silent data shifts.

Test Case #4 — Divide by Zero

Vulnerable snippet:

```
int denom = input;
int q = 100 / denom; // no zero check
printf("%d\n", q);
```

Expected fix:

```
if (denom != 0) {
    int q = 100 / denom;
    printf("%d\n", q);
} else {
    puts("Division by zero prevented");
}
```

Model	Prompt Style	Fix Suggested (summary)	Explanation	Compiles / Runs	Prevents Vulnerability	Notes

GPT-4/5	Chain of Action	if (denom != 0) guard	Direct safe fix	Yes	Yes	Clean
Grok	Chain of Debate	Guard + log	Adds message	Yes	Yes	Verbose
Gemini	Chain of Thought	Guard / epsilon (floats)	Sometimes irrelevant epsilon	Yes	Partial	Keep int guard
Llama-3	Chain of Thought	Zero guard	Minimal change	Yes	Yes	Exact
DeepSeek V1	Chain of Thought	Guard + early return	Early exit	Yes	Yes	Safe

Inference: All models added zero guards; float epsilon suggestions were unnecessary for integer division.

Test Case #5 — SQL Injection (Python, sqlite3)

Vulnerable snippet:

```
import sqlite3
conn = sqlite3.connect("app.db")
cur = conn.cursor()
username = user_input
q = "SELECT * FROM users WHERE name = '" + username + "';"
cur.execute(q)
```

Expected fix:

```
q = "SELECT * FROM users WHERE name = ?"
cur.execute(q, (username,))
```

Model	Prompt Style	Fix Suggested (summary)	Explanation	Compiles / Runs	Prevents Vulnerability	Notes
GPT-4/5	Chain of Thought	Parameterised query	Explained injection risk; used placeholders	Yes	Yes	Exact
Grok	Chain of Debate	Param query +	Adds validation/loggi	Yes	Yes	Optional

		validation	ng			extras
Gemini	Encouragement	Param query (ORM mention)	Sometimes suggests ORM	Yes	Yes	ORM not required
Llama-3	Chain of Action	Param query (tuple arg)	Minimal correct fix	Yes	Yes	Exact
DeepSeek V1	Chain of Thought	Param query + wildcard caution	Warns about LIKE wildcards	Yes	Yes	Helpful note

Inference: All models moved to parameterised queries; additional validation/ORM suggestions optional.

Test Case #6 — Use-After-Free

Vulnerable snippet:

```
char *p = malloc(16);
strcpy(p, "abc");
free(p);
printf("%s\n", p); // use after free
```

Expected fix:

```
char *p = malloc(16);
strcpy(p, "abc");
printf("%s\n", p);
free(p);
p = NULL; // avoid dangling
```

Model	Prompt Style	Fix Suggested (summary)	Explanation	Compiles / Runs	Prevents Vulnerability	Notes
GPT-4/5	Chain of Thought	Move print before free; set p=NULL	Identified lifetime bug	Yes	Yes	Standard pattern

Grok	Chain of Debate	Reorder + NULL set + checks	Verbose justification	Yes	Yes	Extra checks optional
Gemini	Encouragement	Print then free; p=NULL	Simple correct flow	Yes	Yes	Clean
Llama-3	Chain of Action	Reorder; p=NULL	Direct minimal change	Yes	Yes	Exact
DeepSeek V1	Chain of Thought	Reorder + guard	Suggests guard before print	Yes	Yes	Safe

Inference: All models corrected object lifetime by printing before free and nulling the pointer.

Test Case #7 — Double Free

Vulnerable snippet:

```
char *p = malloc(32);
free(p);
...
free(p); // double free
```

Expected fix:

```
char *p = malloc(32);
free(p);
p = NULL; // prevent second free
...
/* check */ if (p) free(p);
```

Model	Prompt Style	Fix Suggested (summary)	Explanation	Compiles / Runs	Prevents Vulnerability	Notes
GPT-4/5	Chain of Action	Set p=NULL after free; guard before	Direct safe fix	Yes	Yes	Standard

		second free				
Grok	Chain of Debate	Introduce ownership comment + guard	Adds rationale	Yes	Yes	Verbose
Gemini	Encouragement	p=NULL; guard	Simple remediation	Yes	Yes	Clean
Llama-3	Chain of Thought	p=NULL; if(p) free(p)	Reasoned fix	Yes	Yes	Exact
DeepSeek V1	Chain of Thought	Guard second free	Early guard	Yes	Yes	Safe

Inference: Consensus on nulling and guarding to prevent double free.

Test Case #8 — Buffer Overflow (strcpy/gets)

Vulnerable snippet:

```
char buf[8];
strcpy(buf, input); // unbounded copy
```

Expected fix:

```
char buf[8];
snprintf(buf, sizeof(buf), "%s", input); // or strlcpy where available
```

Model	Prompt Style	Fix Suggested (summary)	Explanation	Compiles / Runs	Prevents Vulnerability	Notes
GPT-4/5	Chain of Thought	snprintf with size	Explained bound limits	Yes	Yes	Portable
Grok	Chain of Debate	snprintf + length check	Debated options	Yes	Yes	Verbose
Gemini	Encourageme	snprintf/strlc	Suggests	Yes	Yes	OK

	nt	py	safer APIs			
Llama-3	Chain of Action	snprintf	Minimal change	Yes	Yes	Clean
DeepSeek V1	Chain of Thought	snprintf + validate input	Adds validation	Yes	Yes	Optional

Inference: All models replaced unbounded copy with bounded operations; validation sometimes added.

Test Case #9 — Format String Vulnerability

Vulnerable snippet:

```
char *user = input;
printf(user); // user-controlled format string
```

Expected fix:

```
char *user = input;
printf("%s", user); // constant format string
```

Model	Prompt Style	Fix Suggested (summary)	Explanation	Compiles / Runs	Prevents Vulnerability	Notes
GPT-4/5	Chain of Thought	Use constant format	Explained format risk	Yes	Yes	Exact
Grok	Chain of Debate	Constant format + length check	Adds justification	Yes	Yes	Extra
Gemini	Encouragement	Constant format	Simple fix	Yes	Yes	Clean
Llama-3	Chain of Action	Constant format	Direct minimal change	Yes	Yes	Exact
DeepSeek V1	Chain of Thought	Constant format +	Notes sanitization	Yes	Yes	Optional

		sanitize				

Inference: All models converged on using a constant format string.

Test Case #10 — Command Injection (C system)

Vulnerable snippet:

```
char cmd[64];
snprintf(cmd, sizeof(cmd), "ping %s", user_input);
system(cmd); // injection risk
```

Expected fix:

```
char cmd[64];
if (is_safe_host(user_input)) {
    snprintf(cmd, sizeof(cmd), "ping %s", user_input);
    // prefer exec-family or library API; or reject
} else {
    fprintf(stderr, "Invalid input");
}
```

Model	Prompt Style	Fix Suggested (summary)	Explanation	Compiles / Runs	Prevents Vulnerability	Notes
GPT-4/5	Chain of Thought	Validate input; avoid system()	Explained injection risk	N/A	Yes	Suggests exec APIs
Grok	Chain of Debate	Whitelist/regex + avoid system	Debated safety	N/A	Yes	Defensive
Gemini	Encouragement	Validation + safer API	Encouraging style	N/A	Yes	OK
Llama-3	Chain of Action	Input validation + reject invalid	Direct	N/A	Yes	Clean
DeepSeek V1	Chain of Thought	Reject risky chars; suggest execve	Stepwise	N/A	Yes	Good

Inference: Consensus: avoid system() or strictly validate/whitelist inputs; prefer exec-family/library calls.

Test Case #11 — Path Traversal (Python)

Vulnerable snippet:

```
filename = user_input  # e.g., "../../etc/passwd"
with open(filename, "r") as f:
    data = f.read()
```

Expected fix:

```
import os
BASE = "/app/data"
path = os.path.normpath(os.path.join(BASE, filename))
if os.path.commonpath([BASE, path]) == BASE:
    with open(path, "r") as f:
        data = f.read()
else:
    raise ValueError("Invalid path")
```

Model	Prompt Style	Fix Suggested (summary)	Explanation	Compiles / Runs	Prevents Vulnerability	Notes
GPT-4/5	Chain of Thought	Join + normpath + base check	Explained traversal risk	Yes	Yes	Exact
Grok	Chain of Debate	Whitelist base + deny traversal	Debated edge cases	Yes	Yes	Verbose
Gemini	Encouragement	Safe join + check	Simple safe path	Yes	Yes	Clean
Llama-3	Chain of Action	Base prefix check	Direct	Yes	Yes	OK
DeepSeek V1	Chain of Thought	Normalize path + commonpath	Stepwise	Yes	Yes	Robust

Inference: All models converged on base-directory enforcement with normalisation.

Test Case #12 — Uninitialised Variable Use

Vulnerable snippet:

```
int x;
if (flag) x = compute();
printf("%d\n", x); // may be uninitialised
```

Expected fix:

```
int x = 0;
if (flag) x = compute();
printf("%d\n", x); // now defined
```

Model	Prompt Style	Fix Suggested (summary)	Explanation	Compiles / Runs	Prevents Vulnerability	Notes
GPT-4/5	Chain of Thought	Initialise x; ensure set	Explained undefined behaviour	Yes	Yes	Clean
Grok	Chain of Debate	Init + else branch	Adds else path	Yes	Yes	Verbose
Gemini	Encouragement	Initialise to default	Simple fix	Yes	Yes	OK
Llama-3	Chain of Action	Initialise	Direct minimal change	Yes	Yes	Exact
DeepSeek V1	Chain of Thought	Init + guard print	Adds guard	Yes	Yes	Optional

Inference: Models consistently initialised variables to safe defaults or added else paths.