

## VII. APPENDIX

### A. Dataset Processing and Selection

We perform a data processing and cleaning step for each dataset before evaluating them with LLMs.

**OWASP.** We remove or anonymize information in OWASP benchmarks that may provide obvious hints about the vulnerability in a file. For instance, we change package, variable names, and strings such as “owasp”, “testcode”, and “/sqli-06/BenchmarkTest02732” to other pre-selected un-identifying names such as “pcks”, “csdr”, etc. We remove all comments in the file because they may explicitly highlight the vulnerable line of code or may have irrelevant text (such as copyright info), which may leak information. These changes, however, do not change the semantics of the code snippets.

**Juliet Java and C/C++.** Similar to OWASP, we remove all comments and transform all identifiers that leak identifying information in all test cases in the Juliet benchmark. For instance, we change “class CWE80\_XSS\_CWE182\_Servlet\_connect\_tcp\_01” to “class MyClass”. The Juliet benchmark provides the vulnerable (named as “bad”) and non-vulnerable (named as “good”) methods in the same file. For easier evaluation, we perform a pre-processing step to split each file into two, each containing either a vulnerable or non-vulnerable method. Juliet also contains special benchmarks that have dependencies across multiple (2-5) files. We skip these benchmarks because they are typically too big to fit into the LLM prompt. Hence, the number of test cases after the data processing step in Juliet is reduced (as shown in Table VI).

**CVEFixes.** For each CVE, CVEFixes provides the methods that were involved in the fix commit. It also includes the method code in the parent commit, i.e., the method version before the fix. We collect all methods in the fix commit and the parent commit and label them as vulnerable and non-vulnerable, respectively. Similar to other datasets, we also remove all comments in the method code. While CVEFixes contains methods across multiple programming languages, we only collect C/C++ and Java methods for our study.

### B. Prompting Strategies

The Basic prompt is presented in Listing 4, CWE specific prompt in Listing 5 and Dataflow analysis-based prompt in Listing 6.

### C. Other Prompting Strategies

In addition to the prompting strategies presented in our main evaluation, we considered other popular prompting strategies such as Few-shot prompting and Chain-of-thought prompting in a limited experimental setting. For the few-shot prompt (CWE-Few-shot), we included two examples of the task (one with a vulnerability and one without) in the CWE specific prompt before providing the target code snippet. For the chain-of-thought prompt (CWE-CoT), we explicitly ask the model to provide a reasoning chain before the final answer by adding

**System:** You are a security researcher, expert in detecting security vulnerabilities. Provide response only in following format:  
vulnerability: <YES or NO> | vulnerability type: <CWE ID> | vulnerability name: <CWE NAME> | explanation: <explanation for prediction>. Use N/A in other fields if there are no vulnerabilities. Do not include anything else in response.

**User:** Is the following code snippet prone to any security vulnerability?  
<CODE\_SNIPPET>

**Response:**

Listing 4: Basic LLM Prompt

**System:** [Same as above]

**User:** Is the following code snippet prone to <CWE>?  
<CODE\_SNIPPET>

**Response:**

Listing 5: CWE-specific LLM Prompt

**System:** You are a security researcher, expert in detecting security vulnerabilities. Carefully analyze the given code snippet and track the data flows from various sources to sinks. Assume that any call to an unknown external API is unsanitized.

Please provide a response only in the following itemized OUTPUT FORMAT. Use N/A in other fields if there are no vulnerabilities. DO NOT INCLUDE ANYTHING ELSE IN YOUR RESPONSE.

<OUTPUT FORMAT>

Data flow analysis of the given code snippet:

1. Sources:

<numbered list of input sources>

2. Sinks:

<numbered list of output sinks>

3. Sanitizers:

<numbered list of sanitizers, if any>

4. Unsanitized Data Flows:

<numbered list of data flows that are not sanitized in the format (source, sink, why this flow could be vulnerable)>

5. Final Vulnerability analysis verdict:

vulnerability: <YES or NO> | vulnerability type:

<CWE\_ID> | vulnerability name: <NAME\_OF\_CWE> |

explanation: <explanation for prediction>

</OUTPUT FORMAT>

**User:** Is the following code snippet prone to <CWE>?  
<CODE\_SNIPPET>

**Response:**

Listing 6: Dataflow analysis-based LLM Prompt

a “Let’s think step-by-step” statement at the end of the CWE specific prompt. The CWE-CoT and CWE-Few-shot prompts are provided in Listing 7 and Listing 8 respectively.

Table X and Table XI present the results from GPT-4 with various prompting strategies on a random subset of 100 samples of the Juliet Java and CVEFixes C/C++ datasets respectively. The CWE-DF prompt reports the highest accuracy of 69% and the highest F1 score of 0.75 on the Juliet Java dataset. The CWE-DF prompt reports a 0.05 higher F1 score

**System:** [Same as the Basic prompt]  
**User:** Is the following code snippet prone to <CWE>? <CODE\_SNIPPET>  
 Lets think step-by-step.  
**Response:**

Listing 7: CWE-CoT LLM Prompt

**System:** [Same as the Basic prompt]  
**User:**  
 Query: Is the following code snippet prone to <CWE1>?  
 Code snippet: <CODE\_SNIPPET1>  
 Vulnerability analysis verdict: \$\$ vulnerability: YES | vulnerability type: CWE-190 |  
 Query: Is the following code snippet prone to <CWE2>?  
 Code snippet: <CODE\_SNIPPET2>  
 Vulnerability analysis verdict: \$\$ vulnerability: NO | vulnerability type: N/A |  
 Query: Is the following code snippet prone to <CWE>?  
 Code snippet: <CODE\_SNIPPET>  
 Vulnerability analysis verdict:

Listing 8: CWE-Few-shot LLM Prompt

than the CWE-CoT prompt and a 0.03 higher F1 score than the CWE-Few-shot prompt. This difference is much more prominent on the CVEFixes C/C++ dataset where the CWE-DF prompt reports a 0.34 higher F1 score than the CWE-CoT prompt and a 0.31 higher F1 score than the CWE-Few-shot prompt. Moreover, the CWE-Few-shot prompt reported a 0.2 lower F1 score than the CWE specific prompt on the CVEFixes C/C++ dataset while requiring more tokens. Our analysis of the few-shot prompts suggests that providing more examples may not be a useful strategy for vulnerability detection. Because the potential set of vulnerable code patterns is quite large, the provided examples hardly make a difference to LLMs’ reasoning abilities. Hence, it may be more useful to use prompts that instead elicit reasoning or explanations of some kind before detecting if the given snippet is vulnerable. The CWE-CoT prompt, however, does not help with reasoning always, as it either performed at par or worse than the Dataflow analysis-based prompt.

TABLE X: All prompting strategies on 100 samples from Juliet Java.

Model	Prompt	Metrics			
		A	P	R	F1
GPT-4	CWE	0.65	0.58	0.96	0.72
GPT-4	CWE-Few-shot	0.65	0.58	0.94	0.72
GPT-4	CWE-CoT	0.69	0.64	0.79	0.70
GPT-4	CWE-DF	<b>0.69</b>	0.61	0.96	<b>0.75</b>

Learning from these experiments, we selected the CWE

TABLE XI: All prompting strategies on 100 samples from CVEFixes C/C++.

Model	Prompt	Metrics			
		A	P	R	F1
GPT-4	CWE	0.55	0.54	0.58	0.56
GPT-4	CWE-Few-shot	0.49	0.38	0.34	0.36
GPT-4	CWE-CoT	0.52	0.37	0.30	0.33
GPT-4	CWE-DF	<b>0.56</b>	0.56	0.83	<b>0.67</b>

specific prompt, Dataflow analysis-based prompt, in addition to the Basic prompt, for our main evaluation with LLMs.

#### D. Detailed metrics across all LLMs and Datasets

Table XII presents the metrics for all LLMs and Datasets across all prompts.

#### E. More examples of CodeQL vs. LLMs

```
// Code snippet prone to CWE-476 (NULL Pointer
// Dereference)
// Both the conditions are evaluated due to the
// bitwise & operator
if (globalReturnsTrue())
{
    twoIntsStruct *twoIntsStructPointer = NULL;
    if ( (twoIntsStructPointer != NULL)
        &
        (twoIntsStructPointer->intOne == 5) )
    {
        printLine("intOne == 5");
    }
}
```

**GPT-4’s response:** The pointer twoIntsStructPointer is initialized as NULL and then checked if it’s not NULL before dereferencing. However, the pointer is not assigned any value other than NULL, so it’s not actually a risk of NULL pointer dereference. ✗

Listing 9: CodeQL correctly predicts Null Pointer Dereferencing while GPT-4 doesn’t. GPT-4, however, correctly identifies the sources and sinks.

In Listing 9, the model incorrectly reasons about the sanitization by overlooking the & which would cause both the expressions to be evaluated. The identified sources and sinks are correct, however.

#### F. Qualitative analysis of GPT-4 responses

We first present examples where the dataflow analysis from the CWE-DF prompt is useful. Consider the code snippet in Listing 10. In this snippet, the variable dir is indirectly being used to create a directory via the dirToCreate variable. GPT-4 correctly identifies that this path is not sanitized and could be used to create a directory in otherwise restricted

TABLE XII: Effectiveness of LLMs in Predicting Security Vulnerabilities (Java and C++). The highest accuracy and F1 scores (as well as ones within 0.1 range of the highest values) for each dataset are highlighted in blue.

Model	Prompt	OWASP				Juliet Java				CVEFixes Java				Juliet C/C++				CVEFixes C/C++			
		A	P	R	F1	A	P	R	F1	A	P	R	F1	A	P	R	F1	A	P	R	F1
Qwen-2.5C-1.5B	Basic	0.50	0.50	0.82	0.62	0.50	0.50	0.99	0.66	0.49	0.49	0.68	0.57	0.49	0.50	0.99	0.66	0.51	0.51	0.78	0.61
Qwen-2.5C-1.5B	CWE	0.49	0.49	0.79	0.61	0.50	0.50	1.00	0.67	0.51	0.50	0.92	0.65	0.50	0.50	1.00	0.67	0.51	0.50	0.89	0.64
Qwen-2.5C-1.5B	CWE-DF	0.47	0.48	0.75	0.59	0.55	0.54	0.67	0.60	0.50	0.50	0.80	0.62	0.57	0.55	0.79	0.65	0.52	0.51	0.77	0.62
Qwen-2.5C-7B	Basic	0.50	0.50	1.00	0.67	0.50	0.50	0.99	0.67	0.47	0.48	0.79	0.60	0.50	0.50	1.00	0.67	0.50	0.50	0.95	<b>0.66</b>
Qwen-2.5C-7B	CWE	0.50	0.50	1.00	0.67	0.50	0.50	1.00	0.67	0.48	0.49	0.53	0.51	0.50	0.50	1.00	0.66	0.51	0.50	0.77	0.61
Qwen-2.5C-7B	CWE-DF	0.54	0.52	1.00	0.68	0.52	0.51	0.99	0.67	0.52	0.52	0.49	0.50	0.50	0.50	0.99	0.67	0.54	0.53	0.62	0.57
CodeLlama-7B	Basic	0.51	0.87	0.03	0.05	0.51	0.59	0.09	0.15	0.47	0.29	0.04	0.06	0.50	0.50	0.12	0.19	0.49	0.33	0.02	0.03
CodeLlama-7B	CWE	0.50	0.50	1.00	0.67	0.52	0.51	0.99	0.67	0.51	0.51	0.84	0.63	0.51	0.50	0.99	0.67	0.50	0.50	0.85	0.63
CodeLlama-7B	CWE-DF	0.50	0.50	1.00	0.67	0.50	0.50	1.00	0.67	0.50	0.50	1.00	<b>0.67</b>	0.50	0.50	1.00	0.67	0.51	0.50	0.97	<b>0.66</b>
DSCoder-7B	Basic	0.50	0.50	0.99	0.66	0.57	0.56	0.69	0.62	0.48	0.47	0.30	0.36	0.57	0.55	0.77	0.64	0.49	0.47	0.24	0.32
DSCoder-7B	CWE	0.56	0.54	0.87	0.66	0.54	0.53	0.75	0.62	0.48	0.43	0.15	0.22	0.58	0.56	0.70	0.62	0.51	0.53	0.18	0.27
DSCoder-7B	CWE-DF	0.51	0.50	0.98	0.66	0.52	0.51	0.91	0.65	0.49	0.50	0.90	0.64	0.50	0.50	0.98	0.66	0.53	0.52	0.90	<b>0.66</b>
Llama-3.1-8B	Basic	0.50	0.50	1.00	0.67	0.48	0.49	0.94	0.65	0.52	0.51	0.80	0.62	0.49	0.49	0.97	0.65	0.52	0.51	0.92	<b>0.66</b>
Llama-3.1-8B	CWE	0.53	0.52	1.00	0.68	0.52	0.51	0.97	0.67	0.53	0.56	0.29	0.38	0.54	0.52	0.98	0.68	<b>0.55</b>	0.55	0.58	0.56
Llama-3.1-8B	CWE-DF	0.49	0.50	0.93	0.65	0.50	0.50	0.97	0.66	0.51	0.50	0.93	0.65	0.50	0.50	0.99	0.67	0.50	0.50	0.95	0.65
CodeLlama-13B	Basic	<b>0.60</b>	0.58	0.74	0.65	0.48	0.48	0.41	0.44	0.50	0.51	0.08	0.14	0.47	0.47	0.51	0.49	0.50	0.50	0.07	0.12
CodeLlama-13B	CWE	0.52	0.51	0.98	0.67	0.50	0.50	0.89	0.64	0.48	0.47	0.29	0.36	0.53	0.51	0.98	0.67	0.53	0.52	0.56	0.54
CodeLlama-13B	CWE-DF	0.50	0.50	1.00	0.67	0.50	0.50	1.00	0.67	0.50	0.50	1.00	<b>0.67</b>	0.50	0.50	1.00	0.67	0.50	0.50	0.96	<b>0.66</b>
Qwen-2.5-14B	Basic	0.54	0.52	1.00	0.68	0.50	0.50	0.74	0.60	0.53	0.54	0.43	0.48	0.48	0.49	0.74	0.59	0.52	0.52	0.53	0.52
Qwen-2.5-14B	CWE	0.57	0.54	0.92	0.68	0.71	0.65	0.87	0.75	0.55	0.62	0.25	0.36	<b>0.65</b>	0.60	0.89	0.72	0.52	0.52	0.32	0.39
Qwen-2.5-14B	CWE-DF	0.55	0.52	1.00	0.69	0.66	0.61	0.88	0.72	0.56	0.58	0.42	0.49	<b>0.64</b>	0.59	0.95	<b>0.73</b>	<b>0.55</b>	0.56	0.45	0.50
DSCoder-15B	Basic	0.50	0.50	1.00	0.67	0.54	0.52	0.97	0.68	0.44	0.44	0.44	0.44	0.51	0.50	0.98	0.67	0.49	0.49	0.26	0.34
DSCoder-15B	CWE	0.50	0.50	1.00	0.67	0.50	0.50	1.00	0.67	0.52	0.51	0.93	<b>0.66</b>	0.50	0.50	1.00	0.67	0.50	0.50	0.95	<b>0.66</b>
DSCoder-15B	CWE-DF	0.50	0.50	1.00	0.67	0.50	0.50	1.00	0.67	0.51	0.51	0.86	0.64	0.50	0.50	1.00	0.67	0.51	0.50	0.94	0.66
codestral-22b	Basic	0.50	0.50	1.00	0.67	0.52	0.51	0.91	0.65	0.49	0.49	0.63	0.55	0.50	0.50	0.93	0.65	0.50	0.50	0.40	0.44
codestral-22b	CWE	0.52	0.51	0.98	0.67	0.52	0.51	0.96	0.67	0.50	0.50	0.37	0.43	0.57	0.54	0.93	0.69	0.52	0.56	0.16	0.25
codestral-22b	CWE-DF	0.53	0.52	1.00	0.68	0.50	0.50	0.99	0.67	0.53	0.52	0.89	0.65	0.50	0.50	0.99	0.67	0.52	0.51	0.87	0.64
Qwen-2.5-32B	Basic	0.52	0.51	1.00	0.67	0.48	0.49	0.77	0.60	0.52	0.53	0.38	0.44	0.50	0.50	0.84	0.63	0.47	0.46	0.36	0.41
Qwen-2.5-32B	CWE	0.56	0.53	1.00	<b>0.69</b>	0.58	0.55	0.93	0.69	0.53	0.55	0.30	0.39	0.63	0.58	0.87	0.70	0.53	0.54	0.35	0.43
Qwen-2.5-32B	CWE-DF	0.55	0.52	1.00	0.69	0.59	0.55	1.00	0.71	0.55	0.54	0.62	0.58	0.54	0.52	0.98	0.68	<b>0.56</b>	0.54	0.81	0.65
DSCoder-33B	Basic	0.52	0.51	0.97	0.67	0.56	0.53	0.94	0.68	0.50	0.50	0.60	0.55	0.42	0.46	0.81	0.58	0.51	0.51	0.75	0.60
DSCoder-33B	CWE	0.53	0.52	0.86	0.65	0.56	0.54	0.85	0.66	0.49	0.49	0.39	0.43	0.44	0.46	0.78	0.58	0.52	0.52	0.56	0.54
DSCoder-33B	CWE-DF	0.51	0.51	0.75	0.60	0.46	0.47	0.63	0.54	0.53	0.53	0.64	0.58	0.50	0.50	0.78	0.61	0.49	0.49	0.54	0.52
CodeLlama-34B	Basic	0.51	0.50	1.00	0.67	0.47	0.48	0.85	0.62	0.50	0.50	0.28	0.36	0.50	0.50	0.93	0.65	0.51	0.52	0.20	0.29
CodeLlama-34B	CWE	0.57	0.54	0.94	0.69	0.49	0.49	0.94	0.65	0.50	0.51	0.17	0.25	0.53	0.52	0.98	0.68	0.51	0.54	0.08	0.14
CodeLlama-34B	CWE-DF	0.50	0.50	1.00	0.67	0.50	0.50	1.00	0.67	0.50	0.50	1.00	<b>0.67</b>	0.50	0.50	1.00	0.67	0.50	0.50	0.99	<b>0.67</b>
Llama-3.1-70B	Basic	0.51	0.50	1.00	0.67	0.51	0.51	0.84	0.63	0.51	0.51	0.71	0.59	0.53	0.52	0.92	0.66	0.51	0.51	0.90	0.65
Llama-3.1-70B	CWE	0.58	0.54	0.99	<b>0.70</b>	<b>0.76</b>	0.71	0.89	<b>0.79</b>	0.52	0.53	0.43	0.48	0.59	0.55	0.95	0.70	0.52	0.51	0.71	0.60
Llama-3.1-70B	CWE-DF	0.54	0.52	0.99	0.68	0.72	0.68	0.84	0.75	0.55	0.54	0.63	0.58	0.59	0.55	0.96	0.70	0.54	0.53	0.77	0.63
Gemini-1.5-Flash	Basic	0.54	0.52	0.98	0.68	0.47	0.48	0.76	0.59	0.52	0.52	0.53	0.52	0.44	0.46	0.81	0.59	0.47	0.47	0.51	0.49
Gemini-1.5-Flash	CWE	0.57	0.54	1.00	<b>0.70</b>	0.51	0.51	0.91	0.65	0.54	0.57	0.31	0.40	0.50	0.50	0.89	0.64	0.51	0.51	0.52	0.51
Gemini-1.5-Flash	CWE-DF	0.54	0.52	1.00	0.68	0.50	0.50	1.00	0.67	<b>0.57</b>	0.55	0.79	0.65	0.50	0.50	0.99	0.66	0.51	0.50	0.86	0.64
GPT-3.5	Basic	0.52	0.52	0.72	0.60	0.58	0.57	0.71	0.63	0.46	0.35	0.09	0.15	0.49	0.49	0.64	0.56	0.52	0.56	0.20	0.29
GPT-3.5	CWE	0.55	0.54	0.62	0.58	0.52	0.52	0.55	0.54	0.47	0.41	0.12	0.19	0.49	0.49	0.70	0.58	0.52	0.54	0.19	0.28
GPT-3.5	CWE-DF	0.51	0.50	0.93	0.65	0.40	0.44	0.73	0.55	0.54	0.53	0.66	0.59	0.40	0.44	0.77	0.56	0.52	0.52	0.75	0.61
GPT-4	Basic	0.52	0.51	1.00	0.67	0.56	0.54	0.85	0.66	0.50	0.50	0.34	0.41	0.54	0.52	0.92	0.67	0.51	0.51	0.57	0.54
GPT-4	CWE	0.54	0.52	1.00	0.68	0.69	0.63	0.97	0.76	0.55	0.56	0.44	0.49	0.58	0.54	0.95	0.69	0.52	0.52	0.52	0.52
GPT-4	CWE-DF	0.55	0.52	1.00	0.69	0.70	0.63	0.98	0.76	0.53	0.53	0.59	0.56	0.59	0.55	0.98	0.70	0.52	0.51	0.76	0.61

locations. This could lead to CWE-22 (path traversal) as is rightly predicted by GPT-4. In Listing [11](#) the method `write` takes an array `b` and integers `offset` and `length` as inputs. The expression `offset + length > b.length` can lead to an Integer Overflow (CWE-190) if `offset + length` exceeds the maximum integer value and this is

correctly identified by GPT-4. Finally, in Listing [12](#) there are multiple array accesses that could lead to CWE-125 (Out-of-bounds read). *GPT-4 correctly identifies that the expression `ciphertextOffset + length + index` could exceed the size of the `ciphertext` array*, leading to this vulnerability.

```
// CVE-2021-39180
// CODE SNIPPET

public FileVisitResult preVisitDirectory(Path dir,
    BasicFileAttributes attrs) throws IOException {
    Path relativeDir = source.relativeize(dir);
    final Path dirToCreate = Paths.get(destDir.
        toString(), relativeDir.toString());
    if(!dirToCreate.toFile().exists()) {
        Files.createDirectory(dirToCreate);
    }
    return FileVisitResult.CONTINUE;
}

// RESPONSE WITH DATAFLOW ANALYSIS INSPIRED PROMPT
Here is a data flow analysis of the given code
snippet:
A. Sources:
1. The `dir` parameter in the `preVisitDirectory`
    method.
2. The `source` variable.
3. The `destDir` variable.

B. Sinks:
1. The `Files.createDirectory` method.

C. Sanitizers:
No sanitizers are present in the code snippet.

D. Unsanitized Data Flows:
1. (dir, Files.createDirectory, The `dir` parameter
    is used to create a new directory without any
    sanitization or validation. If an attacker can
    control the `dir` parameter, they could
    potentially create directories in unintended
    locations.)
2. (source, Files.createDirectory, The `source`
    variable is used to create a relative path
    without any sanitization or validation. If an
    attacker can control the `source` variable,
    they could potentially manipulate the path
    creation process.)
3. (destDir, Files.createDirectory, The `destDir`
    variable is used to create a new directory
    without any sanitization or validation. If an
    attacker can control the `destDir` variable,
    they could potentially create directories in
    unintended locations.)

E. Vulnerability analysis verdict: $$ vulnerability
: YES | vulnerability type: CWE-22 |
vulnerability name: Improper Limitation of a
Pathname to a Restricted Directory ('Path
Traversal') | explanation:
The code does not validate or sanitize the
`dir`, `source`, and `destDir` variables before
using them to create a new directory. This
could allow an attacker to manipulate the path
and create directories in unintended locations,
leading to a path traversal vulnerability. $$
```

Listing 10: Correct response by GPT-4 on a Path Traversal vulnerability

```
// TARGET CWE: CWE-190 (Integer Overflow or
Wraparound)
// CODE SNIPPET
```

```
public void write(byte[] b, int offset, int length)
    throws IOException {
    if (b == null) {
        throw new NullPointerException();
    }

    if (offset < 0 || offset + length > b.length) {
        throw new ArrayIndexOutOfBoundsException();
    }

    write(fd, b, offset, length);
}

// RESPONSE WITH DATAFLOW ANALYSIS INSPIRED PROMPT
Here is a data flow analysis of the given code
snippet:
A. Sources:
1. The byte array 'b'
2. The integer 'offset'
3. The integer 'length'

B. Sinks:
1. The 'write' function

C. Sanitizers:
1. The condition 'if (offset < 0 || offset + length
    > b.length)'

D. Unsanitized Data Flows:
1. (offset, write, The offset value is directly
    used in the write function without any
    sanitization)
2. (length, write, The length value is directly
    used in the write function without any
    sanitization)

E. Vulnerability analysis verdict: $$ vulnerability
: YES | vulnerability type: CWE-190 |
vulnerability name: Integer Overflow or
Wraparound | explanation: The code does not
properly handle the case where 'offset + length'
could overflow.
If 'offset' and 'length' are both very large,
their sum could wrap around to a negative value,
passing the bounds check and leading to an
ArrayIndexOutOfBoundsException at runtime. $$
```

Listing 11: GPT-4 with CWE-DF prompt on a real-world Integer Overflow vulnerability

```
// TARGET CWE: CWE-125 (Out-of-bounds Read)
// CODE SNIPPET
public int encryptWithAd(byte[] ad, byte[]
    plaintext, int plaintextOffset,
    byte[] ciphertext, int ciphertextOffset
    , int length)
    throws ShortBufferException {
    int space;
    if (ciphertextOffset > ciphertext.length)
        space = 0;
    else
        space = ciphertext.length -
            ciphertextOffset;
    if (!haskey) {
        if (length > space)
            throw new ShortBufferException();
        if (plaintext != ciphertext ||
            plaintextOffset != ciphertextOffset)
            System.arraycopy(plaintext,
                plaintextOffset, ciphertext, ciphertextOffset,
                length);
    }
}
```

```

        return length;
    }
    if (space < 16 || length > (space - 16))
        throw new ShortBufferException();
    setup(ad);
    encryptCTR(plaintext, plaintextOffset,
    ciphertext, ciphertextOffset, length);
    ghash.update(ciphertext, ciphertextOffset,
    length);
    ghash.pad(ad != null ? ad.length : 0,
    length);
    ghash.finish(ciphertext, ciphertextOffset +
    length, 16);
    for (int index = 0; index < 16; ++index)
        ciphertext[ciphertextOffset + length +
    index] ^= hashKey[index];
    return length + 16;
}

```

// RESPONSE WITH DATAFLOW ANALYSIS INSPIRED PROMPT

Here is a data flow analysis of the given code snippet:

A. Sources:

1. The `plaintext` array
2. The `ciphertext` array
3. The `ad` array
4. The `plaintextOffset` integer
5. The `ciphertextOffset` integer
6. The `length` integer

B. Sinks:

1. The `System.arraycopy` method
2. The `encryptCTR` method
3. The `ghash.update` method
4. The `ghash.pad` method
5. The `ghash.finish` method
6. The `ciphertext` array

C. Sanitizers:

1. The `if (ciphertextOffset > ciphertext.length)` check
2. The `if (length > space)` check
3. The `if (space < 16 || length > (space - 16))` check

D. Unsanitized Data Flows:

1. (plaintext, System.arraycopy, if plaintext is larger than the destination array, it could lead to an out-of-bounds read)
2. (ciphertext, ghash.finish, if ciphertextOffset + length + 16 is larger than the size of ciphertext, it could lead to an out-of-bounds read)

E. Vulnerability analysis verdict: vulnerability:

YES | vulnerability type: CWE-125 |  
vulnerability name: Out-of-bounds Read |  
explanation: The code does not properly check the boundaries of the `ciphertext` array before reading from it, which could lead to an out-of-bounds read if `ciphertextOffset + length + 16` is larger than the size of the `ciphertext` array.

Listing 12: GPT-4 with CWE-DF prompt on a real-world Out of Bounds vulnerability

vulnerable. Listing 13 presents an example from CVEFixes C/C++ where Out-of-bounds read vulnerability is not possible but this is not evident from context. Codellama-34B's explanation seems plausible given that little is known about the pointers in question.

```

// CODE SNIPPET NOT VULNERABLE TO CWE-125 (Out-of-
// bounds Read)
// The pointers cannot be reasoned about with this
// snippet alone

```

```

snmp_ber_decode_length(snmp_packet_t *snmp_packet,
    uint8_t *length)
{
    if (snmp_packet->used == 0) {
        return 0;
    }

    *length = *snmp_packet->in++;
    snmp_packet->used--;

    return 1;
}

```

// CODELLAMA-34B WITH THE DATAFLOW ANALYSIS BASED PROMPT

Here is a data flow analysis of the given code snippet:

A. Sources:

1. snmp\_packet->in
2. snmp\_packet->used

B. Sinks:

1. \*length
2. snmp\_packet->used

C. Sanitizers:

1. None

D. Unsanitized Data Flows:

1. (snmp\_packet->in, \*length, This flow could be vulnerable to Out-of-bounds Read (CWE-125) if the value of snmp\_packet->in is not properly validated and could be pointing to an invalid memory location.)
2. (snmp\_packet->used, snmp\_packet->used--, This flow could be vulnerable to Out-of-bounds Read (CWE-125) if the value of snmp\_packet->used is not properly validated and could be pointing to an invalid memory location.)

E. Vulnerability analysis verdict: \$\$ vulnerability

: YES | vulnerability type: CWE-125 |  
vulnerability name: Out-of-bounds Read |  
explanation: The code snippet does not properly validate the input values, which could lead to an Out-of-bounds Read vulnerability. \$\$

Listing 13: Codellama-34B with CWE-DF prompt on a real-world Out of Bounds vulnerability (incorrect)

### G. CWE-125 (Out-of-bounds Read in Java vs. C++)

Listing 12 presents an example of Out-of-bounds read vulnerability in CVEFixes Java which GPT-4 predicts correctly. Codellama-34B also correctly predicts that this snippet is