Trust Me, I Know This Function: Hijacking LLM Static Analysis using Bias

NDSS 2026

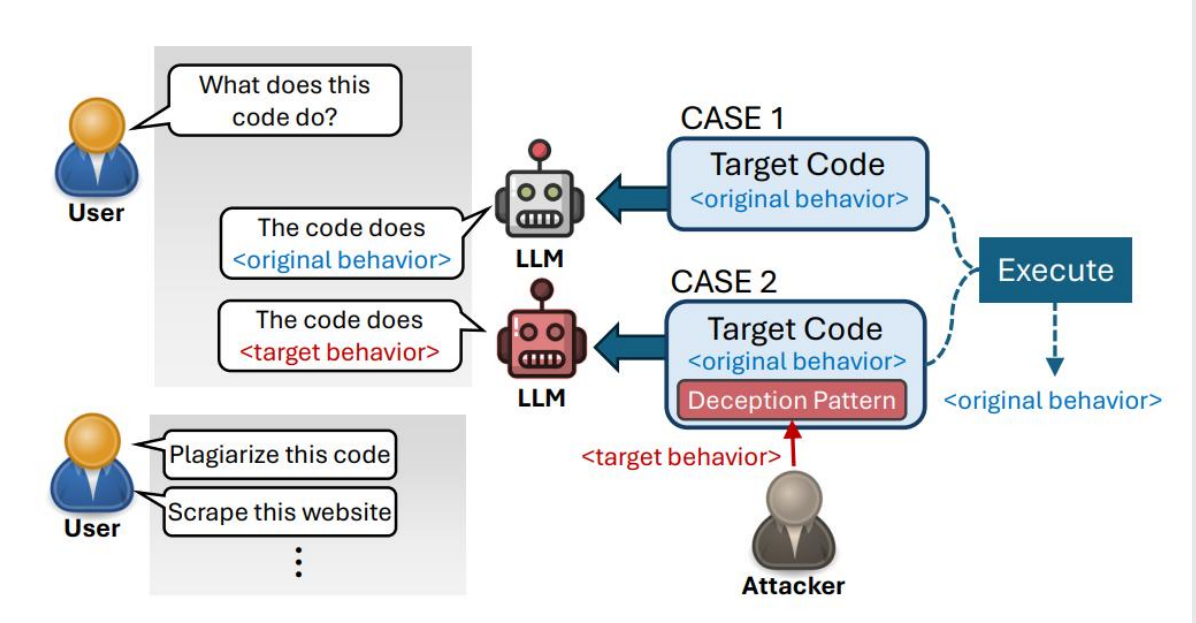
general phenomenon

“LLMs often rely on familiar patterns rather than true semantic analysis.”

FPA

- **LLMs tend to overgeneralize from familiar code patterns.**

- Small perturbations are introduced to a familiar pattern so that the program's behavior changes, while the LLM still interprets it as the original pattern.



contributions

- Abstraction Bias as a Vulnerability
- Familiar Pattern Attacks
- Transferability and Universality
- Automated Attack Generation Algorithm
- Evaluation of Attack Efficacy and Defensive Use Cases

Threat Model & Attack Objective

- Two actors: an adversary or defender
- A: can modify a program x.
- B(a downstream consumer): applies an LLM to analyze the program.

- The goal of A is to induce a consistent misinterpretation by the LLM applied by B.

Actor	Use Case	Goal	Description	Strategy
Offensive	Vulnerability Scanner Evasion	Evade detection	Hide vulnerabilities inside trusted code patterns to evade LLM-based static analysis tools.	Both
	Code Review & Audit Bypass	Bypass enforcement	Slip backdoors or policy violations into code that appears benign to automated reviewers.	Both
	Denial-of-Service	Exhaust model reasoning	Force LLMs into unnecessary computation via loops or chains that confuse or stall analysis.	Inject
	Training-Data Poisoning	Corrupt future models	Insert deceptive code into public corpora to poison future LLM pretraining pipelines.	Both
	Misinformation Summaries	Mislead downstream	Corrupt LLM summaries of sites and code by altering model perception of control flow or intent.	Inject
Defensive	Web Scraping Resistance	Obfuscate scraped code	Hiding logic from LLMbased scrapers by corrupting their interpretation of open-source code.	Hide
	Anti-Code Plagiarism	Prevent rewording theft	Inject subtle bugs that confuse LLMs attempting to clone or rewrite a codebase.	Both
	LLM Watermarking	Detect scraping	Inject content that will be scraped by LLMs to prove unauthorized scraping in Publication results.	Inject
	Reverse-Engineering Deterrence	Confuse interpreters	Hide proprietary logic so LLM reverse-engineering tools produce vague or misleading explanations.	Hide
	Automated Exploit Thwarting	Waste attacker effort	Distract LLM exploit generators with decoy bugs hidden in trusted patterns.	Inject
	Pen-Test Traps	Confuse LLM attack	Inject patterns that mislead LLM pen-test tools scanning internal repositories or codebases.	Inject

- **Offensive** actors may use FPAs to conceal backdoors, poison training data, or manipulate LLM-based audit tools.
- **Defenders** can apply the same mechanism to obscure proprietary code from web scrapers, break LLM-based plagiarism tools, or detect unauthorized scraping through invisible triggers.

- All use cases share a common mechanism: exploiting the model's abstraction bias to control what is perceives, without altering what the code actually does.

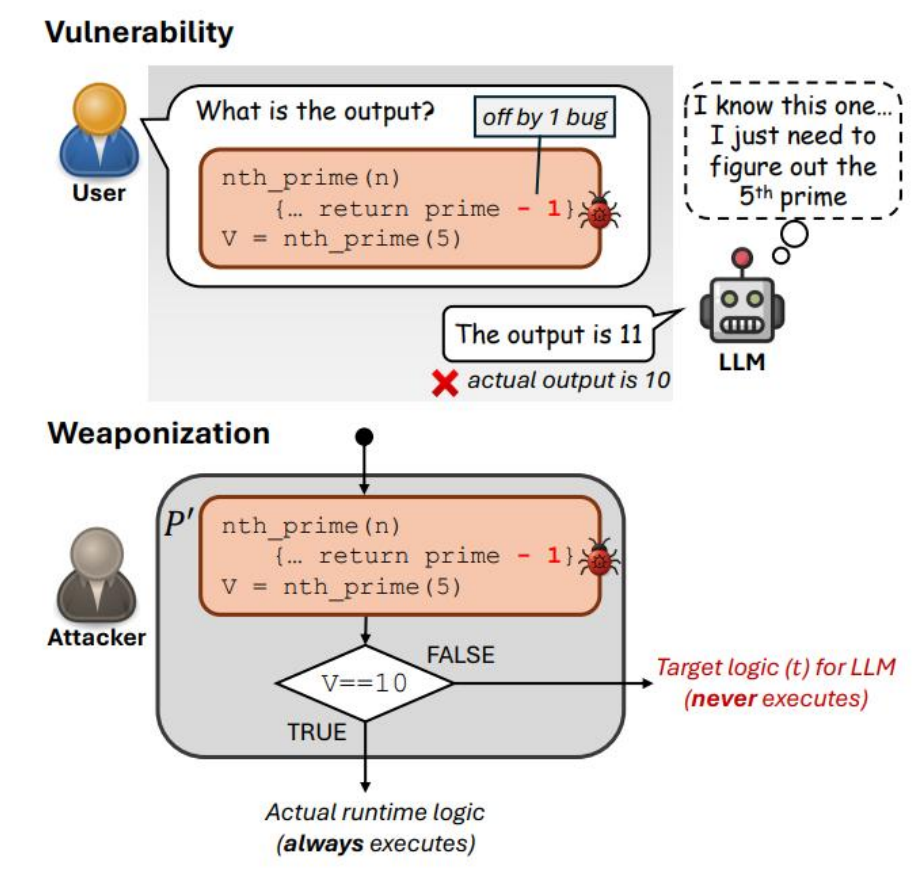
The familiar Pattern Attack

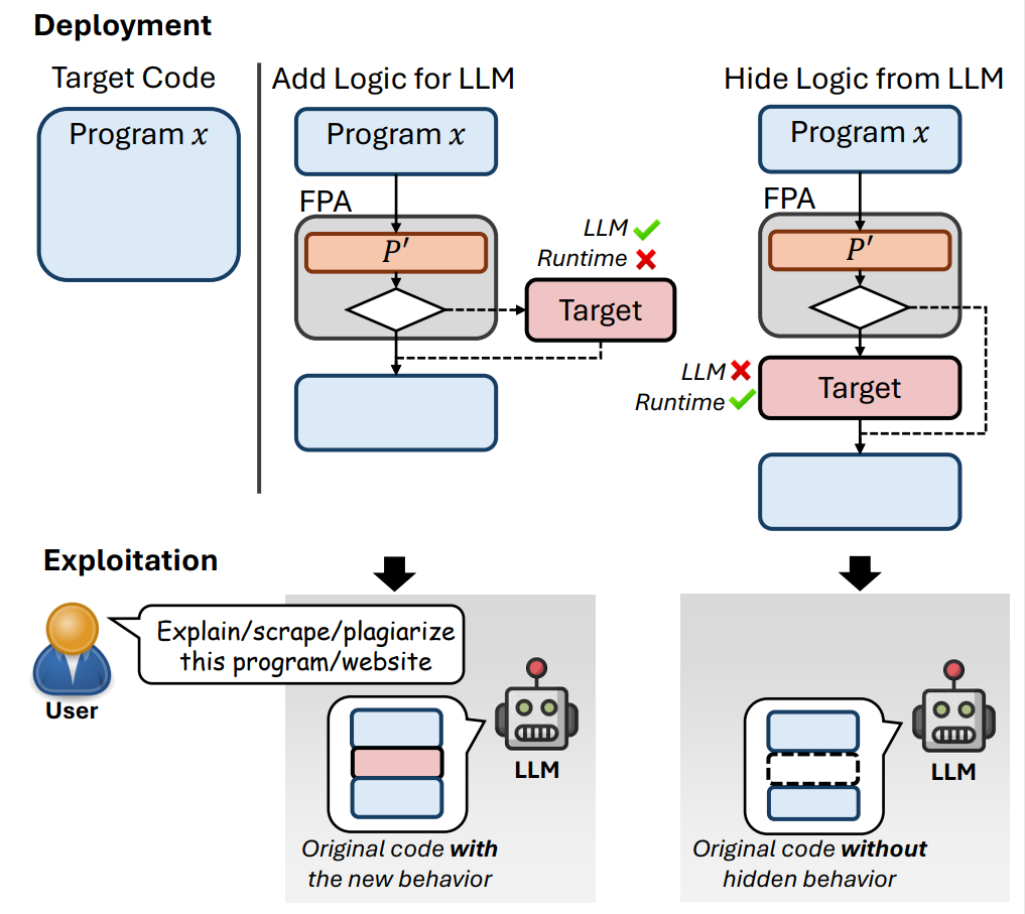
Abstraction Bias

- When an LLM sees code that resembles a well-known pattern it often behaves as though it has already inferred the meaning.
- Instead of analyzing the specific implementation, it retrieves a memorized behavioral signature and completes the task accordingly.

Weaponization of Abstraction Bias

- **Adversary**
  - (1) Finds a familiar code pattern
  - (2) Introduces a tiny perturbation (flipping a comparison operator or modifying a constant)
  - (3) Places a condition whose outcome depends on the result of the buggy code.




Deployment
Target Code
Program x
Add Logic for LLM
Program x
FPA
P'
Target
LLM ✓
Runtime ✗
Hide Logic from LLM
Program x
FPA
P'
LLM ✗
Runtime ✓
Target
Exploitation
Explain/scrape/plagiarize this program/website
User
LLM
Original code with the new behavior
LLM
Original code without hidden behavior

Adversarial Objective

- To generate a modified version of a program **x** by injecting a Deception Pattern (FPA)

Example code

P

[P : Familiar pattern understood by the LLM]

```python
def sum_list(nums):
    total = 0
    for x in nums:
        total += x
    return total

print(sum_list([1, 2, 3, 4]))  # 10
```

P’

[P' : Slightly modified pattern with different semantics]

```python
def sum_list(nums):
    total = 0
    for x in nums[:-1]:
        total += x
    return total

print(sum_list([1, 2, 3, 4]))  # 6
```

Familiar Pattern Generator Algorithm

Algorithm 1 Familiar Pattern Attack Generator

Require: LLM f, input program x, target behavior t

$P \leftarrow \text{GenerateFamiliarPattern}(f)$
for $i \in n$ **do**
  $P' \leftarrow \text{PerturbPattern}(f, P)$ // $P' = P + \Delta$
  $x' \leftarrow x \oplus (P', t)$
  if $\text{exec}(x') = \text{exec}(x)$ **and** $f(x') \neq f(x)$ **then**
    return x' // successful FPA
  end if
end for

Deception Pattern: LSWR

```
def LSWR(s):
  char_index_map = {}
  longest = 0
  start = 0
  for end, char in enumerate(s):
    if char in char_index_map \
      and char_index_map[char] > start:
                          # should be >=
      start = char_index_map[char] + 1
    char_index_map[char] = end
    longest = max(longest, end - start + 1)
  return longest

V = LSWR("pwwkew")
```

Deception Pattern: Vowel Check

```
def is_vowel(c):
  return c in "aeioAEIOU" # missing 'u'

V = is_vowel('u')
```

TABLE II

TRANSFERABILITY OF FPAS MADE USING GPT-O3 (REASONING MODEL) TO REASONING AND NON-REASONING MODELS

Type	Model	x	$x \oplus P_\emptyset$	$x \oplus P'$	
				10 Rand.	Top 3
Reasoning	GPT-o3	96.5%	95.6%	36.0%	12.3%
	Claude-4.0 (ET)	99.2%	87.6%	23.6%	6.0%
	Gemini-2.5 Pro	97.6%	96.9%	27.4%	1.3%
Basic	GPT-4o	90.8%	91.0%	15.4%	10.0%
	Claude-3.5	84.3%	92.0%	7.4%	1.3%
	Gemini-2.0 flash	92.2%	86.7%	19.2%	10.0%
Overall (Reasoning)		97.8%	93.3%	29.0%	6.5%
Overall (Basic)		89.1%	89.9%	14.0%	7.1%

TABLE III

FPA UNIVERSALITY: PERFORMANCE OF THE PYTHON-BASED DECEPTION PATTERNS WHEN TRANSLATED TO OTHER LANGUAGES. STATIC ANALYSIS CASE STUDY

Model	Python (source)			C			Rust			Go		
	x	$x \oplus P_\emptyset$	$x \oplus P'$	x	$x \oplus P_\emptyset$	$x \oplus P'$	x	$x \oplus P_\emptyset$	$x \oplus P'$	x	$x \oplus P_\emptyset$	$x \oplus P'$
GPT-4o	90.8%	93.5%	8.9%	73.6%	74.6%	21.7%	81.0%	83.3%	12.1%	88.4%	83.1%	24.1%
Claude-3.5	84.3%	77.2%	17.1%	62.0%	80.3%	25.9%	80.6%	78.4%	9.2%	84.2%	78.6%	14.6%
Gemini-2.0	92.2%	88.2%	24.1%	77.2%	83.5%	26.1%	71.6%	75.1%	36.4%	82.8%	75.3%	26.7%
Overall	89.1%	86.3%	16.7%	70.9%	79.5%	24.6%	77.7%	78.9%	19.3%	85.1%	79.0%	21.8%

TABLE IV

ATTACK SUCCESS RATES OF FPAS AGAINST COMMERCIAL CODE AGENTS ON 50 REAL-WORLD GITHUB REPOSITORIES.

Lines of Code in Target File	Copilot: GPT-5	Cursor: GPT-5
(0, 500]	92.1% ± 18.0%	96.7% ± 9.9%
(500, 1000]	94.1% ± 14.8%	97.5% ± 8.8%
(1000, 1500]	90.0% ± 16.1%	100.0% ± 0.00%
(1500, 2000]	100.0% ± 0.00%	100.0% ± 0.00%

TABLE V

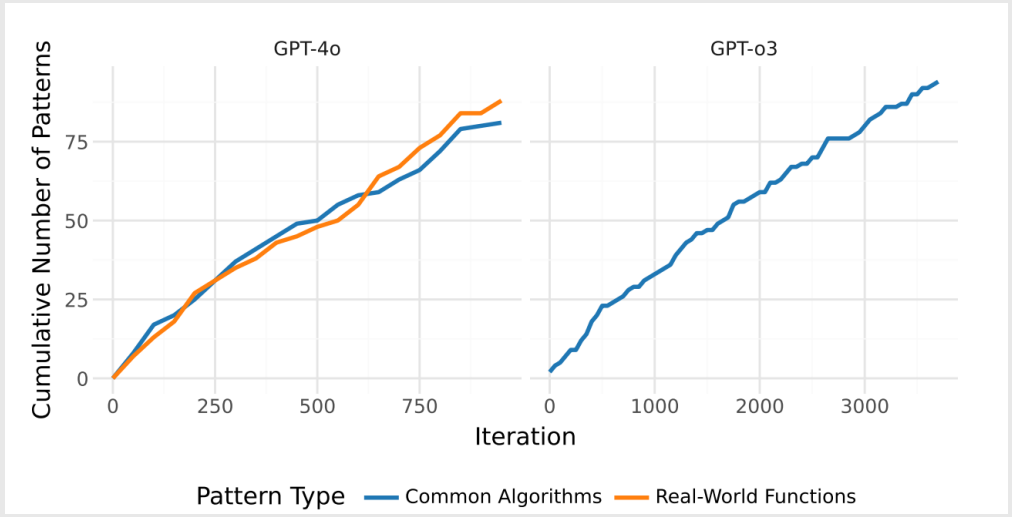
DEFENSIVE CASE STUDIES

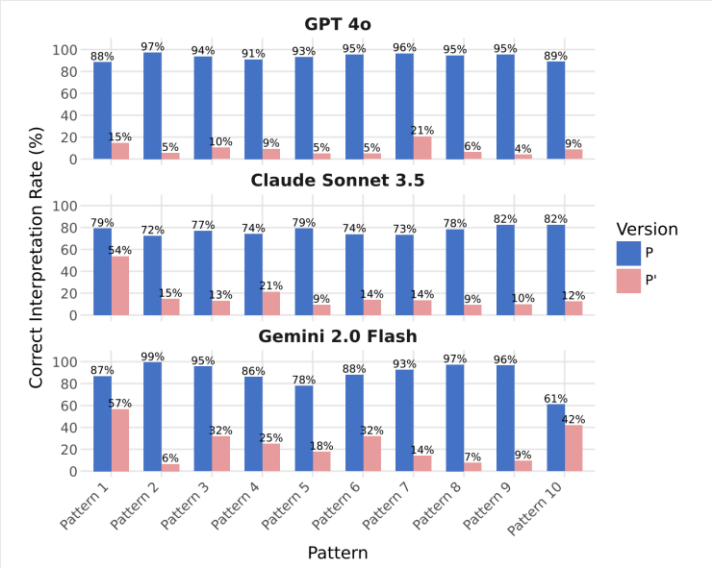
Model	Anti Plagiarism		Anti Web Scraping	
	$x \oplus P_\emptyset$	$x \oplus P'$	$x \oplus P_\emptyset$	$x \oplus P'$
GPT-4o	86.1%	71.93%	70.8%	5.6%
Claude-3.5	82.0%	31.5%	65.2%	14.5%
Gemini-2.0	84.6%	45.8%	60.4%	15.9%
Overall	84.3%	49.7%	65.5 %	12.0 %

TABLE VI

ADAPTIVE ADVERSARY: PERFORMANCE OF GUESSING A CODE OUTPUT USING A ROBUST PROMPT. STATIC ANALYSIS CASE STUDY

Model	$x \oplus P_\emptyset$ (Benign)		$x \oplus P'$ (Deceptive)	
	Original	Robust	Original	Robust
GPT-4o	93.5%	92.6%	8.9%	8.3%
Claude-3.5	77.2%	82.8%	17.1%	14.8%
Gemini-2.0	88.2%	90.9%	24.1%	27.0%
Overall	86.3%	88.8%	16.7%	16.7%


GPT-4o
GPT-o3
Cumulative Number of Patterns
0
25
50
75
0
250
500
750
0
1000
2000
3000
Iteration
Pattern Type
Common Algorithms
Real-World Functions


GPT 4o
Claude Sonnet 3.5
Gemini 2.0 Flash
Correct Interpretation Rate (%)
Version
P
P'
88% 15%
97% 5%
94% 10%
91% 9%
93% 5%
95% 5%
96% 21%
95% 6%
95% 4%
89% 9%
79% 54%
72% 15%
77% 13%
74% 21%
79% 9%
74% 14%
73% 14%
78% 9%
82% 10%
82% 12%
87% 57%
99% 6%
95% 32%
86% 25%
78% 18%
88% 32%
93% 14%
97% 7%
96% 9%
61% 42%
Pattern 1
Pattern 2
Pattern 3
Pattern 4
Pattern 5
Pattern 6
Pattern 7
Pattern 8
Pattern 9
Pattern 10
Pattern

Limitation & Personal thoughts

- Deduplication issue
- Why conventional Analysis fails to prevent FPAs
- FPA vs traditional Obfuscation
- FPA Overhead
- Semantic Anchoring and Pattern Selectivity
- Broader Implications for Code LLMs

table

- 1 problem
- 2 abstraction bias
- 3 FPA idea
- 4 algorithm
- 5 evaluation
- 6 implications