35TH USENIX SECURITY SYMPOSIUM

IoTRAGuarder

Securing Retrieval-Augmented Code Generation via Contextual Knowledge Injection: A Case for Embedded IoT Applications

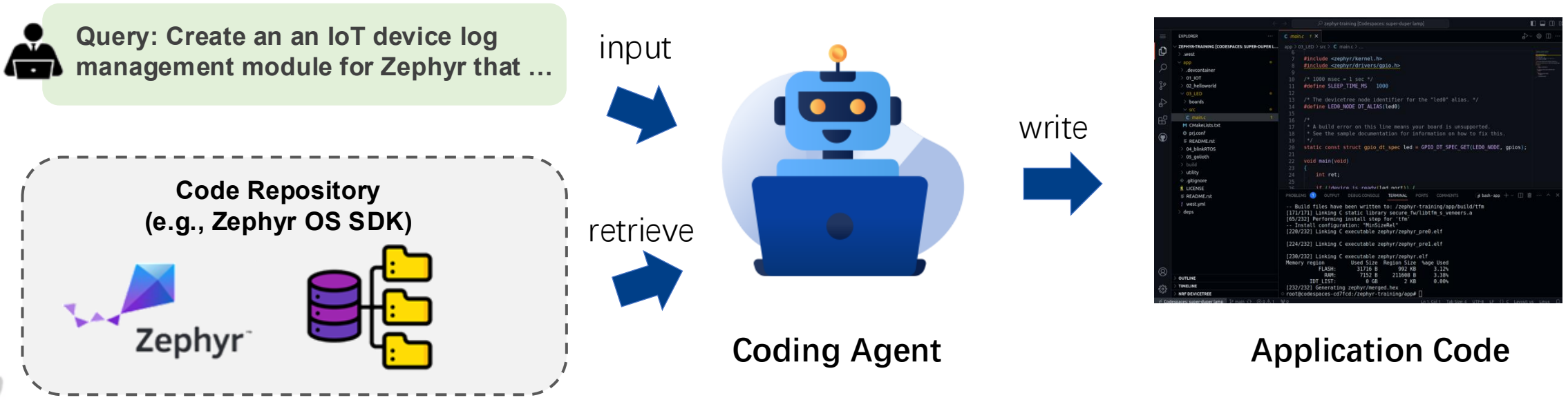
Tong Sun[1], Jingyi Su[1], Yi Gao[1], and Wei Dong[1]

[1]The State Key Laboratory of Blockchain and Data Security,
College of Computer Science, Zhejiang University, China

Background: Repository-Grounded IoT RACG

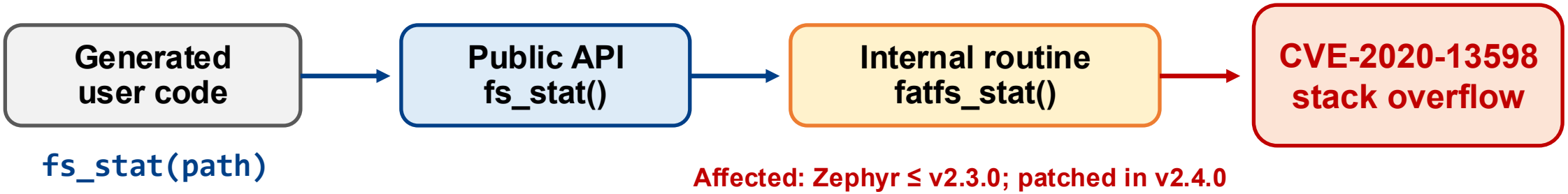
- **IoT application code is tightly coupled with RTOS/SDK repositories**
  - ✓ public APIs, configuration options, build workflows

- **Retrieval-Augmented Code Generation (RACG) retrieves from a pinned repository and generates user-space code**



Background: Version-Inherited CVEs

- Generated code may call a benign-looking public API
- That API can transitively reach a vulnerable internal routine in the pinned version



No retrieval poisoning is required — the honest pinned repository already contains the vulnerable behavior.

Existing Defense & Limitations

Secure RACG (e.g., [CCS'25] CodeGuarder)

- retrieves security examples by task intent
- injects generic root causes and fixing patterns

✗ misses version/API-specific exposure

CVE scanners (Snyk / Dependency-Check)

- flag vulnerable files and functions
- operate after or outside generation

✗ no public-API reachability
✗ no prompt-ready constraints

Need repository- and version-grounded security knowledge at the public API layer.

Motivating Example: Shell Wi-Fi Command

Create a smart home gateway provisioning feature that receives a Wi-Fi SSID and passphrase over BLE from a mobile app, then programmatically connects the device to the specified home network by invoking the Zephyr shell Wi-Fi connect command. The target OS version is Zephyr v3.4.0.

(a) An example user query for an embedded IoT app.

```
#include <zephyr/shell/shell.h>
#include <zephyr/shell/shell_uart.h>
…
void connect_to_wifi(…) {
    const struct shell *shell = shell_backend_uart_get_ptr();
    shell_execute_cmd(shell, …);
…
}
```

ultimately reaching the *__wifi_args_to_params()* function in /subsys/net/l2/wifi/wifi_shell.c , which does not check user input length **(CVE-2023-4257, Affected versions: <=3.4.0)**

(b) Vulnerable code generated without contextual security knowledge.

```
#include <zephyr/shell/shell.h>
#include <zephyr/shell/shell_uart.h>
…
int validate_ssid_psk(const char *ssid, const char *psk) {
    if (ssid == NULL) return -EINVAL;
    size_t ssid_len = strlen(ssid);
    if (ssid_len == 0 || ssid_len > MAX_SSID_LENGTH) return -EINVAL;
    if (psk != NULL) {
        size_t psk_len = strlen(psk);
        if (psk_len < MIN_PSK_LENGTH || psk_len > MAX_PSK_LENGTH) return -EINVAL;
    }
    return 0;
}
void connect_to_wifi(…) {
    const struct shell *shell = shell_backend_uart_get_ptr();
    int ret = validate_ssid_psk(ssid, psk);
    if (ret) {…}
    shell_execute_cmd(shell, …);
    …
}
```

(c) Secure code generated when augmented with contextual security knowledge.

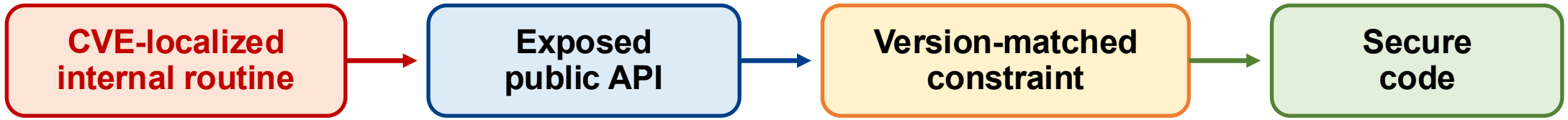
Key Idea

Insight

- In repository-grounded IoT RACG, security exposure is determined by **public APIs selected from a pinned version**, not only by the task description.

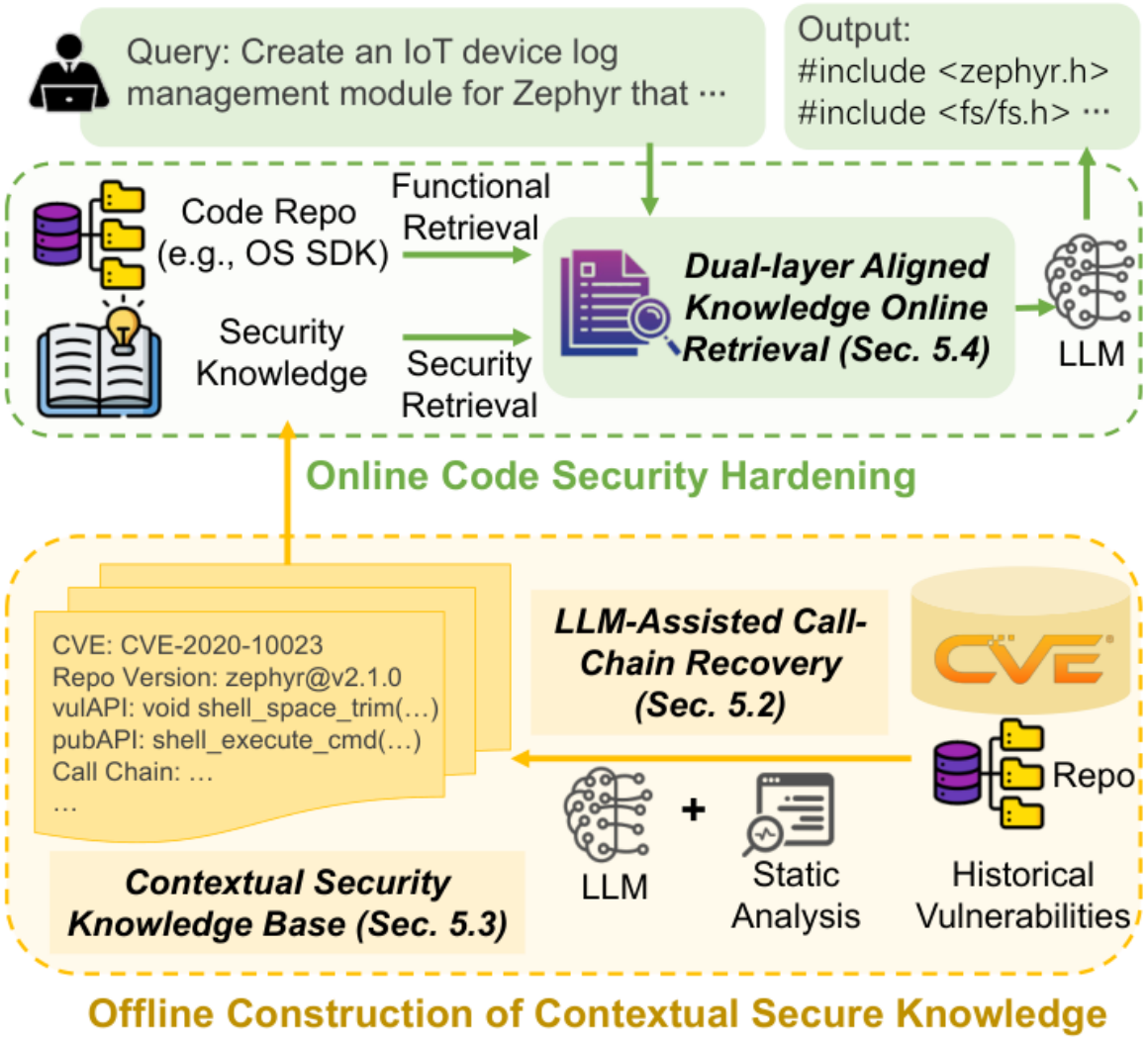
Idea

- Recover CVE-localized internals → exposed public APIs, then inject **API-specific constraints** at prompt time.



Align security knowledge with generation-time API commitments.

IoTRAGuarder Overview



Online Hardening

- surface API/header commitments
- inject aligned security guidance

Offline Construction

- recover reverse call chains
- build contextual security knowledge base (KB)

Main Challenge: Exposure Identification

- **CVE reports localize vulnerable internals; applications call public APIs**
- **RTOS call graphs contain ops tables, callbacks, macros, Kconfig / Devicetree wiring**

Table 1: Public interface artifacts extracted from the Zephyr v3.7.0 repository.

Public interface artifact type	Count
Macros (`#define`)	7,022
Functions	4,584
Type aliases (`typedef`)	570
Enumerations (`enum`)	441
Global variables	25
Total	**12,642**

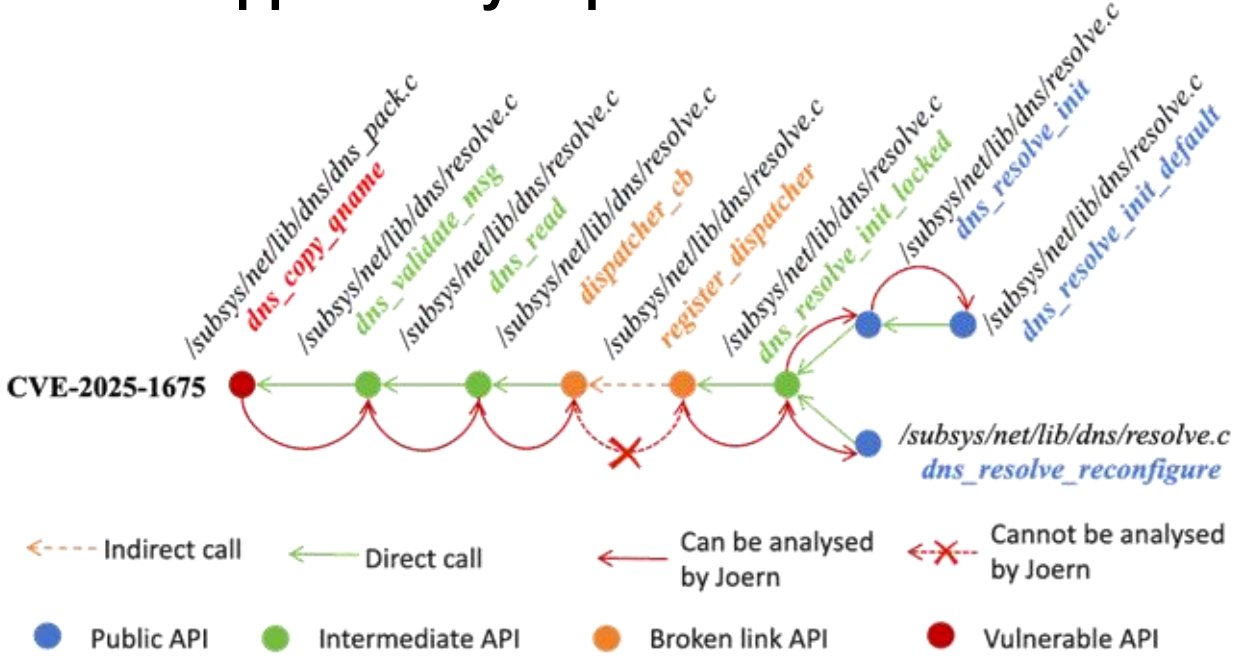
Class	Type	Description / Example
I	Direct Call	Syntactic call edge at source level, e.g., `g()` → `f()`.
	Indirect Call	Dispatch through function pointers / ops tables / vtables, e.g., `ops->tx(...)`.
	Inline-expanded Call	Source-level call edge may be eliminated/reshaped after inlining; analysis should preserve source-level intent.
	External/Library Call	Target defined outside repository (e.g., libc/libgcc/vendor SDK); may require stubs or symbol linking.
II	Macro-generated call sites	Call sites introduced or obscured by macro expansion (source-level non-obviousness), e.g., `*_DEFINE/*_INIT`.
	Event-driven Callback	Framework invokes registered handlers (timers, workqueues, network callbacks), e.g., `k_work` handler.
	Interrupt Dispatch	Vector-table/arch glue transfers control to ISRs, e.g., IRQ connect + ISR entry.
	Scheduler-driven Transfer	Thread entry/resume via context switching; execution transfers without an explicit caller in C source.
	Linker-script-driven Dispatch	Init/registration tables emitted into special sections and iterated at boot, e.g., `SYS_INIT`-style registries.
	Syscall Dispatch	User-facing syscall wrapper dispatches into verification/implementation routines.
	Driver Model Binding	Device model binds ops tables to instances; calls go through instance → api slots (e.g., `dev->api->foo`).
III	Build-time conditional reachability	Kconfig/Makefile/CMake determine which units/branches exist, altering nodes/edges of the call graph.
	Auto-generated HAL/platform wiring	Devicetree/build-generated glue introduces additional call sites and bindings (e.g., platform HAL stubs).

Question: Which public APIs expose this CVE in this pinned version?

LLM-Assisted Call-Chain Recovery

Key insight

- Static analysis and LLMs have complementary strengths. Static analysis is precise for explicit call edges and local symbol resolution.
- LLMs can interpret RTOS wiring idioms at breakpoints from localized context. LLM only reasoning can hallucinate cross file links.
- We therefore use the LLM only as a bounded local completion oracle. We accept inferred links only when supported by explicit checkable evidence.



Contextual Security Knowledge Base

- **A useful KB entry must be version-aware, API-grounded, and prompt-ready**

Table 9: An example entry in our contextual security knowledge base (CVE-2020-10023).

Field	Value
CVE / CWE	CVE-2020-10023 / CWE-120 (buffer overflow)
Repository / Version	`zephyrproject/zephyr` @ `v2.1.0`
Vulnerable primitive (`vulAPI`)	`void shell_spaces_trim(char *str)` (`zephyr/subsys/shell/shell_utils.c`)
Exposed public API (`pubAPI`)	`int shell_execute_cmd(const struct shell *shell, const char *cmd)` Header: `<shell/shell.h>` Functionality: Executes a shell command.
Evidence call chain	`yshell_execute_cmd` → `execute` → `shell_wildcard_prepare` → `shell_spaces_trim`
Vulnerable pattern	The public API `shell_execute_cmd()` reaches `shell_spaces_trim()`, where the input command string may trigger a buffer overflow.
Trigger example	`shell_execute_cmd(shell, "ls  *.txt");` (crafted command with excessively long consecutive spaces)
Defensive guidance (`D_f`)	Before invoking `shell_execute_cmd()`, sanitize user input by bounding the effective command length and normalizing whitespace (e.g., compress consecutive spaces, trim leading/trailing spaces), so the string passed into the vulnerable trimming routine satisfies safe preconditions.
Safe wrapper sketch	Provide a wrapper (e.g., `shell_execute_cmd_safe`) that copies `cmd` into a bounded buffer and performs whitespace normalization before calling `shell_execute_cmd()`.
Fixed versions / patches	`v1.14.2`, `branch from v2.1.0`, `v2.2.0` (with upstream PR links).

Dual-layer Aligned Online Retrieval

Issues:

- **Existing secure RACG methods retrieve security knowledge only by task semantics, such as sub-task decomposition and functionality-based retrieval.**

- **This is misaligned with our commitment: even when a correct version- and API-specific entry exists, it may not be retrieved because the search ignores the concrete APIs identified by repository RAG.**

Dual-layer Aligned Online Retrieval

- **Repository-grounded sub-task representation**

  We ecomposes the augmented input (K_Q || Q) into fine-grained sub-tasks:

  $$(\mathcal{K}_Q \,||\, Q)_d = [sub_1, \ldots, sub_n]$$

  Each sub-task explicitly records the APIs and headers that the model intends to use: $sub_i = q_i \,||\, [pubAPI_1, \ldots, pubAPI_m] \,||\, [headers_1, \ldots, headers_n]$

- **Version-matched, API-aware security retrieval**

  For each sub_i, we retrieve version-consistent security entries from the knowledge base, we represent each candidate entry s_j by embedding: $\mathrm{Embed}(D_{func_j} \,||\, vulAPI)$,

- **Risk-aware prioritization and targeted injection**

  Not all sub-tasks warrant equal security intervention. We compute an aggregated risk score: $W_{sub_i} = \sum_{s_j \in S'_{sub_i}} w_j$,

  select the top-k highest-risk sub-tasks, and inject only their aligned security guidance into the final RACG prompt:

  $$P = P_{ori} + \sum_{sub_i \in Q_{\text{top-}k}} (sub_i + S'_{sub_i}).$$

Evaluation: Setup

- **Benchmarks**
  - 44 Real-world IoT Applications, including 9 Zephyr CVEs
- **Studied LLMs**
  - GPT-4o, DeepSeek-V3, DeepSeek-Coder-V2-16B, Qwen2.5-Coder-14B
- **Retrieval modules**
  - The same embedding module (SFR-Embedding-Code-400M) for both functional code retriever and security knowledge retrievers

Evaluation: Baselines

- **Standard RACG ([MobiSys'25] IoTPilot, [MobiCom'25] AutoIoT)**
  - A conventional retrieval augmented setup that retrieves only functional code exemplars and appends them to the prompt, without any security knowledge retrieval/injection.

- **[CCS'25] CodeGuarder**
  - We implement a strong prompt-time knowledge-injection baseline following CodeGuarder's experimental protocol: security knowledge is retrieved and injected into the generation prompt to proactively steer the model away from vulnerable patterns.

Evaluation: Metrics

- **Security Rate (SR)**
  - The percentage of generated outputs verified as secure (i.e., not exhibiting the targeted vulnerability patterns).
- **Retrieval Success Rate (RR)**
  - The fraction of queries for which the retriever returns at least one relevant item (separately tracked for functional code and security knowledge when applicable).
- **API Correctness (API)**
  - Whether the generated solution uses the correct and intended public APIs (especially important when secure remediation requires replacing insecure API usages with safer alternatives).

Evaluation: Overall SR

Table 3: Success Rate (SR) breakdown by CWE. Entries are #success/#total (SR%).

LLM	CWE-119	CWE-120	CWE-121	CWE-125	CWE-416	CWE-787	CWE-835	Overall
GPT-4o [25]	0/10 (0.00%)	0/15 (0.00%)	0/9 (0.00%)	0/5 (0.00%)	0/5 (0.00%)	0/4 (0.00%)	0/5 (0.00%)	0/44 (0.00%)
w CodeGuarder	0/10 (0.00%)	0/15 (0.00%)	3/9 (33.33%)	0/5 (0.00%)	0/5 (0.00%)	3/4 (75.00%)	0/5 (0.00%)	3/44 (6.82%)
w. Ours	**10/10 (100.00%)**	**15/15 (100.00%)**	**9/9 (100.00%)**	**5/5 (100.00%)**	**5/5 (100.00%)**	**4/4 (100.00%)**	**5/5 (100.00%)**	**44/44 (100.00%)**
DS-V3 [18]	0/10 (0.00%)	0/15 (0.00%)	1/9 (11.11%)	0/5 (0.00%)	0/5 (0.00%)	1/4 (25.00%)	0/5 (0.00%)	1/44 (2.27%)
w. CodeGuarder	0/10 (0.00%)	0/15 (0.00%)	4/9 (44.44%)	0/5 (0.00%)	0/5 (0.00%)	4/4 (100.00%)	0/5 (0.00%)	4/44 (9.09%)
w. Ours	**10/10 (100.00%)**	**11/15 (73.33%)**	**7/9 (77.78%)**	**5/5 (100.00%)**	**5/5 (100.00%)**	**4/4 (100.00%)**	**5/5 (100.00%)**	**40/44 (90.91%)**
DS-Coder [47]	0/10 (0.00%)	0/15 (0.00%)	0/9 (0.00%)	0/5 (0.00%)	0/5 (0.00%)	0/4 (0.00%)	0/5 (0.00%)	0/44 (0.00%)
w. CodeGuarder	0/10 (0.00%)	0/15 (0.00%)	0/9 (0.00%)	0/5 (0.00%)	0/5 (0.00%)	0/4 (0.00%)	0/5 (0.00%)	0/44 (0.00%)
w. Ours	**8/10 (80.00%)**	**4/15 (26.67%)**	**2/9 (22.22%)**	**3/5 (60.00%)**	**2/5 (40.00%)**	**1/4 (25.00%)**	**2/5 (40.00%)**	**20/44 (45.45%)**
Qwen-Coder [10]	0/10 (0.00%)	0/15 (0.00%)	1/9 (11.11%)	0/5 (0.00%)	0/5 (0.00%)	1/4 (25.00%)	0/5 (0.00%)	1/44 (2.27%)
w. CodeGuarder	0/10 (0.00%)	0/15 (0.00%)	2/9 (22.22%)	0/5 (0.00%)	0/5 (0.00%)	2/4 (50.00%)	0/5 (0.00%)	2/44 (4.55%)
w. Ours	**9/10 (90.00%)**	**9/15 (60.00%)**	**5/9 (55.56%)**	**5/5 (100.00%)**	**2/5 (40.00%)**	**4/4 (100.00%)**	**5/5 (100.00%)**	**34/44 (77.27%)**

Evaluation: Overall SR

Table 3: Success Rate (SR) breakdown by CWE. Entries are #success/#total (SR%).

LLM	CWE-119	CWE-120	CWE-121	CWE-125	CWE-416	CWE-787	CWE-835	Overall
GPT-4o [25]	0/10 (0.00%)	0/15 (0.00%)	0/9 (0.00%)	0/5 (0.00%)	0/5 (0.00%)	0/4 (0.00%)	0/5 (0.00%)	0/44 (0.00%)
w CodeGuarder	0/10 (0.00%)	0/15 (0.00%)	3/9 (33.33%)	0/5 (0.00%)	0/5 (0.00%)	3/4 (75.00%)	0/5 (0.00%)	3/44 (6.82%)
w. Ours	**10/10 (100.00%)**	**15/15 (100.00%)**	**9/9 (100.00%)**	**5/5 (100.00%)**	**5/5 (100.00%)**	**4/4 (100.00%)**	**5/5 (100.00%)**	**44/44 (100.00%)**
DS-V3 [18]	0/10 (0.00%)	0/15 (0.00%)	1/9 (11.11%)	0/5 (0.00%)	0/5 (0.00%)	1/4 (25.00%)	0/5 (0.00%)	1/44 (2.27%)
w. CodeGuarder	0/10 (0.00%)	0/15 (0.00%)	4/9 (44.44%)	0/5 (0.00%)	0/5 (0.00%)	4/4 (100.00%)	0/5 (0.00%)	4/44 (9.09%)
w. Ours	**10/10 (100.00%)**	**11/15 (73.33%)**	**7/9 (77.78%)**	**5/5 (100.00%)**	**5/5 (100.00%)**	**4/4 (100.00%)**	**5/5 (100.00%)**	**40/44 (90.91%)**
DS-Coder [47]	0/10 (0.00%)	0/15 (0.00%)	0/9 (0.00%)	0/5 (0.00%)	0/5 (0.00%)	0/4 (0.00%)	0/5 (0.00%)	0/44 (0.00%)
w. CodeGuarder	0/10 (0.00%)	0/15 (0.00%)	0/9 (0.00%)	0/5 (0.00%)	0/5 (0.00%)	0/4 (0.00%)	0/5 (0.00%)	0/44 (0.00%)
w. Ours	**8/10 (80.00%)**	**4/15 (26.67%)**	**2/9 (22.22%)**	**3/5 (60.00%)**	**2/5 (40.00%)**	**1/4 (25.00%)**	**2/5 (40.00%)**	**20/44 (45.45%)**
Qwen-Coder [10]	0/10 (0.00%)	0/15 (0.00%)	1/9 (11.11%)	0/5 (0.00%)	0/5 (0.00%)	1/4 (25.00%)	0/5 (0.00%)	1/44 (2.27%)
w. CodeGuarder	0/10 (0.00%)	0/15 (0.00%)	2/9 (22.22%)	0/5 (0.00%)	0/5 (0.00%)	2/4 (50.00%)	0/5 (0.00%)	2/44 (4.55%)
w. Ours	**9/10 (90.00%)**	**9/15 (60.00%)**	**5/9 (55.56%)**	**5/5 (100.00%)**	**2/5 (40.00%)**	**4/4 (100.00%)**	**5/5 (100.00%)**	**34/44 (77.27%)**

Takeaway 1: For weaker code models (DS-Coder and Qwen-Coder), boundary-sensitive memory bugs remain the hardest: SR on CWE-120 and CWE-121 is consistently lower than other categories, suggesting that even when the correct guidance is retrieved, faithfully implementing precise length propagation and bounds checks remains challenging

Evaluation: Overall SR

Table 3: Success Rate (SR) breakdown by CWE. Entries are #success/#total (SR%).

LLM	CWE-119	CWE-120	CWE-121	CWE-125	CWE-416	CWE-787	CWE-835	Overall
GPT-4o [25]	0/10 (0.00%)	0/15 (0.00%)	0/9 (0.00%)	0/5 (0.00%)	0/5 (0.00%)	0/4 (0.00%)	0/5 (0.00%)	0/44 (0.00%)
w CodeGuarder	0/10 (0.00%)	0/15 (0.00%)	3/9 (33.33%)	0/5 (0.00%)	0/5 (0.00%)	3/4 (75.00%)	0/5 (0.00%)	3/44 (6.82%)
w. Ours	**10/10 (100.00%)**	**15/15 (100.00%)**	**9/9 (100.00%)**	**5/5 (100.00%)**	**5/5 (100.00%)**	**4/4 (100.00%)**	**5/5 (100.00%)**	**44/44 (100.00%)**
DS-V3 [18]	0/10 (0.00%)	0/15 (0.00%)	1/9 (11.11%)	0/5 (0.00%)	0/5 (0.00%)	1/4 (25.00%)	0/5 (0.00%)	1/44 (2.27%)
w. CodeGuarder	0/10 (0.00%)	0/15 (0.00%)	4/9 (44.44%)	0/5 (0.00%)	0/5 (0.00%)	4/4 (100.00%)	0/5 (0.00%)	4/44 (9.09%)
w. Ours	**10/10 (100.00%)**	**11/15 (73.33%)**	**7/9 (77.78%)**	**5/5 (100.00%)**	**5/5 (100.00%)**	**4/4 (100.00%)**	**5/5 (100.00%)**	**40/44 (90.91%)**
DS-Coder [47]	0/10 (0.00%)	0/15 (0.00%)	0/9 (0.00%)	0/5 (0.00%)	0/5 (0.00%)	0/4 (0.00%)	0/5 (0.00%)	0/44 (0.00%)
w. CodeGuarder	0/10 (0.00%)	0/15 (0.00%)	0/9 (0.00%)	0/5 (0.00%)	0/5 (0.00%)	0/4 (0.00%)	0/5 (0.00%)	0/44 (0.00%)
w. Ours	**8/10 (80.00%)**	**4/15 (26.67%)**	**2/9 (22.22%)**	**3/5 (60.00%)**	**2/5 (40.00%)**	**1/4 (25.00%)**	**2/5 (40.00%)**	**20/44 (45.45%)**
Qwen-Coder [10]	0/10 (0.00%)	0/15 (0.00%)	1/9 (11.11%)	0/5 (0.00%)	0/5 (0.00%)	1/4 (25.00%)	0/5 (0.00%)	1/44 (2.27%)
w. CodeGuarder	0/10 (0.00%)	0/15 (0.00%)	2/9 (22.22%)	0/5 (0.00%)	0/5 (0.00%)	2/4 (50.00%)	0/5 (0.00%)	2/44 (4.55%)
w. Ours	**9/10 (90.00%)**	**9/15 (60.00%)**	**5/9 (55.56%)**	**5/5 (100.00%)**	**2/5 (40.00%)**	**4/4 (100.00%)**	**5/5 (100.00%)**	**34/44 (77.27%)**

Takeaway 2: CWE-416 (use-after-free) is harder for smaller models, likely due to the need for reasoning about object lifetimes and aliasing. These results indicate that our API- and versionaligned retrieval substantially improves guidance relevance, while the remaining gap is primarily attributable to model capability limitations in executing complex, vulnerabilityspecific fixes.

Evaluation: Impact of Decomposition Granularity and Injection Budget

Table 4: The impact of m (security entries per sub-task) and n (number of sub-tasks) on success rate (SR).

	DS-V3 (SR)				Qwen-Coder (SR)			
	n				n			
m	2	4	6	8	2	4	6	8
1	90.91%	95.45%	**97.73%**	93.18%	**77.27%**	65.91%	63.64%	65.91%
2	**97.73%**	93.18%	90.91%	90.91%	63.64%	47.73%	47.73%	38.64%
3	90.91%	95.45%	86.36%	88.64%	52.27%	36.36%	34.09%	31.82%

- The results reveal a clear "sweet spot" rather than a monotonic trend.
- The key benefit comes from alignment rather than more injected text: a small number of highly relevant, API- and version-matched entries is more effective than injecting many candidates
- The optimal (m, n) depends on model capacity; we therefore use a conservative default (small m and moderate n) to balance precision and overhead across models.

Evaluation: Effectiveness of Exposure Identification

- **Baseline: Joern tool (static CPG call graph)**

Table 5: Call-relationship statistics for reverse exposure tracing. "Direct" edges are recoverable by Joern from explicit call sites, whereas "Indirect/Implicit" edges correspond to RTOS-typical dispatch/registration/macro that Joern [13] cannot connect and thus require our hybrid completion (LLM-assisted, evidence-gated) to bridge call-graph islands.

Vulnerability	Repo Ver.	`vul_API`	`file_patch(vul_API)`	Direct	Indirect	Total	Breakpoint
CVE-2020-13598	v2.3.0	`fatfs_stat`	`subsys/fs/fat_fs.c`	0	3	3	Struct-pointer
CVE-2020-10023	v2.1.0	`shell_spaces_trim`	`subsys/shell/shell_utils.c`	3	0	3	None
CVE-2023-4263	v3.4.0	`nrf5_tx`	`drivers/ieee802154/ieee802154_nrf5.c`	0	1	1	Struct-pointer
CVE-2017-14199	v1.10.0	`dns_resolve_cb`	`subsys/net/lib/sockets/getaddrinfo.c`	0	1	1	Callback registration
CVE-2017-14201	v1.13.0	`dns_result_cb`	`subsys/net/ip/net_shell.c`	0	1	1	Callback registration
CVE-2017-14202	v1.14.0	`SHELL_HISTORY_DEFINE`	`include/shell/shell.h`	0	1	1	Macro
CVE-2020-10022	v2.2.0	`probe_cb`	`lib/updatehub/updatehub.c`	2	1	3	Async callback + macro
CVE-2025-1675	v4.0.0	`dns_copy_qname`	`subsys/net/lib/dns/dns_pack.c`	5	1	6	Struct-pointer
CVE-2025-2962	v4.1.0	`dns_copy_qname`	`subsys/net/lib/dns/dns_pack.c`	5	1	6	Struct-pointer
Total	–	–	–	**15**	**10**	**25**	–

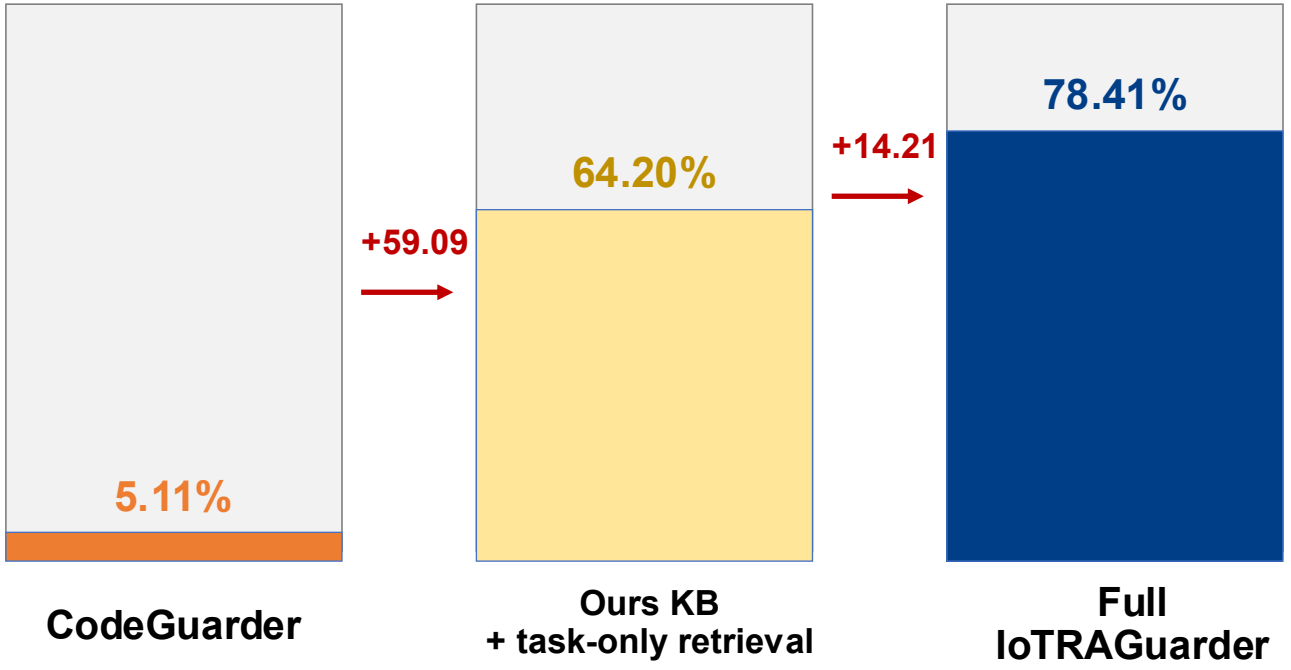
Evaluation: Ablation Study

- **Variants:**
  - **CodeGuarder**
  - **IoTRAGuarder (Ablation):** replaces CodeGuarder's knowledge base (KB) with our constructed security KB, while keeping CodeGuarder's task-only security retrieval
  - **IoTRAGuarder:** uses our constructed security KB together with our dual-layer aligned security retrieval.

Model	Vanilla	CodeGuarder [17]	IoTRAGuarder (Ablation)	IoTRAGuarder
GPT-4o	0/44 (0.00%)	3/44 (6.82%)	36/44 (81.82%)	**44/44 (100.00%)**
DS-V3	1/44 (2.27%)	4/44 (9.09%)	**40/44 (90.91%)**	**40/44 (90.91%)**
DS-Coder	0/44 (0.00%)	0/44 (0.00%)	12/44 (27.27%)	**20/44 (45.45%)**
Qwen-Coder	1/44 (2.27%)	2/44 (4.55%)	25/44 (56.82%)	**34/44 (77.27%)**
Overall	2/176 (1.14%)	9/176 (5.11%)	113/176 (64.20%)	**138/176 (78.41%)**

Evaluation: Ablation Study

- **Takeaways:**
  - Both components are necessary for robust hardening: the offline knowledge base construction makes the rightversion- and API-specific evidence available, and the online aligned retrieval ensures that this evidence is triggered and can override insecure repository exemplars during generation.



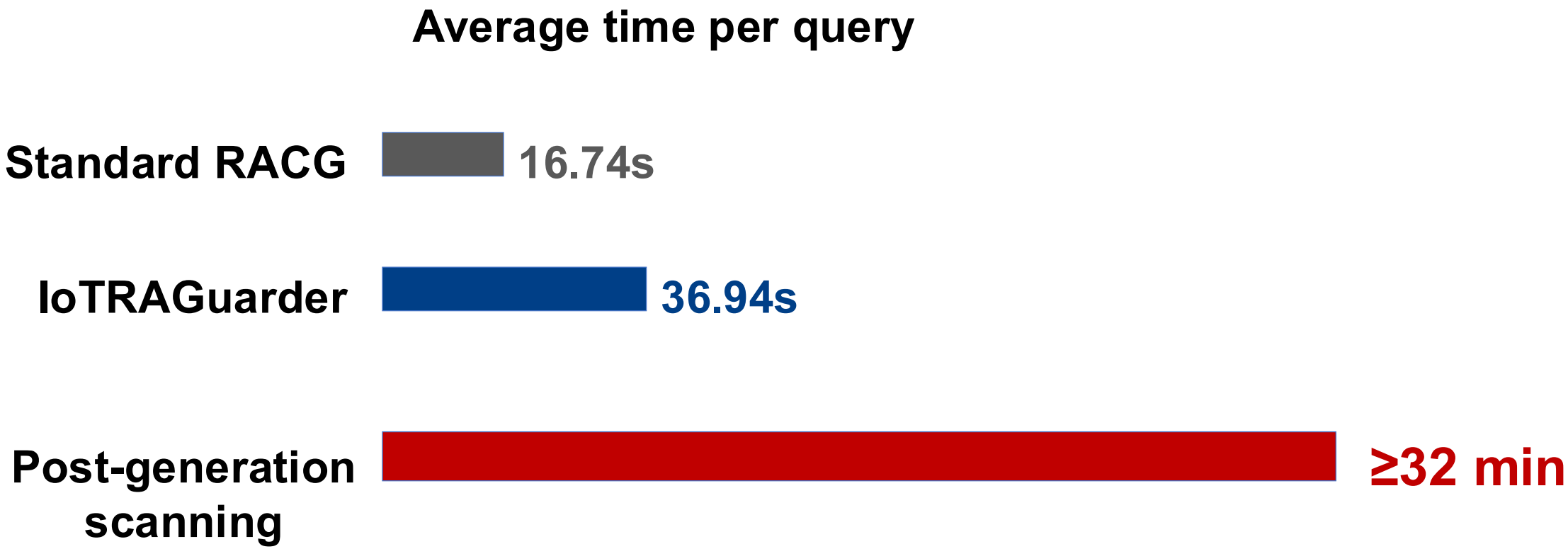
Evaluation: Breakdown Analysis

Table 7: Breakdown of online hardening into two stages under k=3, m=1, n=2. **RR** (retrieval relevance) counts whether the retriever returns the desired offline-constructed security entry. **API** counts whether the generated code invokes the target API under analysis. Each cell reports #hits/#tasks (rate). Best per row (for RR and API separately) is bold.

	CodeGuarder [17]		IoTRAGuarder (Ablation)		IoTRAGuarder (Ours)	
Model	**RR**	**API**	**RR**	**API**	**RR**	**API**
GPT-4o	9/44 (20.45%)	6/44 (13.64%)	39/44 (88.64%)	38/44 (86.36%)	**44/44 (100.00%)**	**44/44 (100.00%)**
DS-V3	13/44 (29.55%)	7/44 (15.91%)	41/44 (93.18%)	**40/44 (90.91%)**	**42/44 (95.45%)**	**40/44 (90.91%)**
DS-Coder	6/44 (13.64%)	0/44 (0.00%)	**39/44 (88.64%)**	17/44 (38.64%)	38/44 (86.36%)	**21/44 (47.73%)**
Qwen-Coder	4/44 (9.09%)	6/44 (13.64%)	37/44 (84.09%)	25/44 (56.82%)	**42/44 (95.45%)**	**35/44 (79.55%)**
Overall	32/176 (18.18%)	19/176 (10.80%)	156/176 (88.64%)	120/176 (68.18%)	**166/176 (94.32%)**	**140/176 (79.55%)**

- (1) CodeGuarder is bottlenecked by retrieval misses and exhibits a strong evidence–API mismatch.
- (2) A stronger knowledge base largely fixes retrieval, but not the follow-through.
- (3) IoTRAGuarder improves both retrieval and API coupling, turning more hits into actionable hardening

Evaluation: Online overhead



IoTRAGuarder

Securing Retrieval-Augmented Code Generation via Contextual Knowledge Injection

- *Version-inherited CVE exposure*
- *API- and version-grounded security KB*
- *Dual-layer aligned online retrieval*

Thank you for your attention!

Tong Sun, Jingyi Su, Yi Gao, and Wei Dong

If you have any questions, please contact **tongsun@zju.edu.cn**

浙江大学 区块链与数据安全全国重点实验室
STATE KEY LABORATORY OF BLOCKCHAIN AND DATA SECURITY
ZHEJIANG UNIVERSITY