Exploring Function-Level Code Metrics and Developer Attributes for Linux Kernel Vulnerabilities

Yan Sun

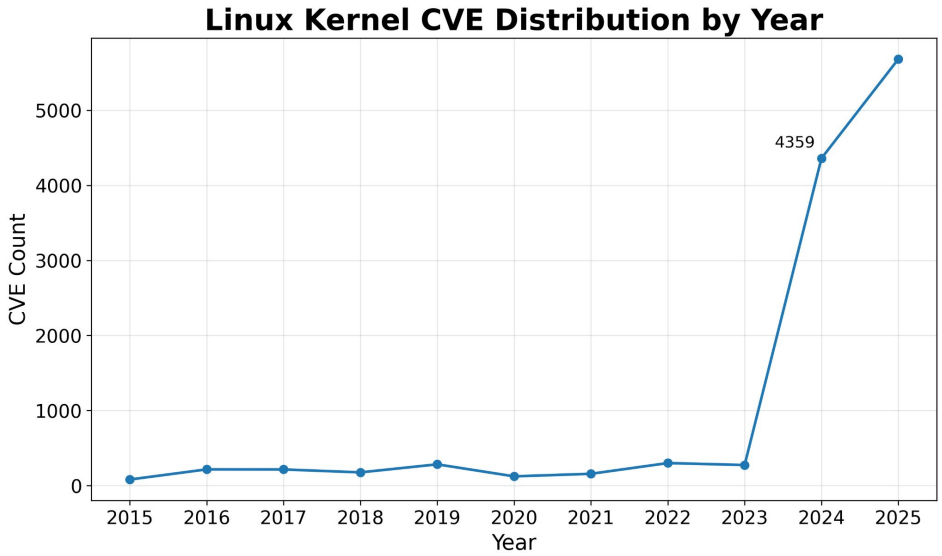
University of Minnesota

Presentation Outline

1. Motivation & Introduction
2. Data Collection
   - ❏ Vulnerability-Fixing Commit (VFC) Identification
   - ❏ Vulnerability-Inducing Commit (VIC) Identification
3. Developer Metrics & Analysis
   - ❏ Developer Experience, Code Familiarity, Maintainership Status
4. Code Metrics & Analysis
   - ❏ Structural Metrics, Memory Operation Metrics
5. Vulnerability Distribution
6. Implication & Conclusion

Motivation

The availability of Linux kernel CVEs has improved since the Linux kernel became a CVE Naming Authority (2024).



Introduction

Overview of our study: We analyze **vulnerability-fixing commits (VFCs)** and **vulnerability-inducing commits (VICs)** associated with Linux kernel CVEs (2015-2025) using code and developer metrics to identify vulnerability-inducing characteristics.

Our contribution:

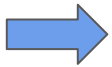
- Addresses gaps in prior studies where developer metrics are overlooked and code metrics are often limited to the file level.
- Our dataset contains **11k+ kernel CVEs** and **8k+ VFC-VIC pairs**, making it one of the largest to study Linux kernel vulnerabilities.

Data Collection/Methodology Overview

1. Linux Kernel CVE Collection from NVD (2015 - 2025)

2. Vulnerability-Fixing Commit (VFC) Identification

Upstream commits identified from mainline and stable patches in the CVE references



3. Vulnerability-Inducing Commit (VIC) Identification

Commits identified from `Fixes:<SHA>` tags in the VFC commit messages

8,796 VFC – VIC pairs

4. Developer and Code Metrics Extraction and Analysis

Developer Metrics Overview

Developer Experience

- **Prior Commit Count (Kernel-level)**
- **Commit Count Last Six Months**
- **Participation Span (Days)**

Code Familiarity

- **Prior Commit Count (Subsystem-level)**
- **Prior Commit Count (File-level)**

Developer Role

- **Maintainer Status**

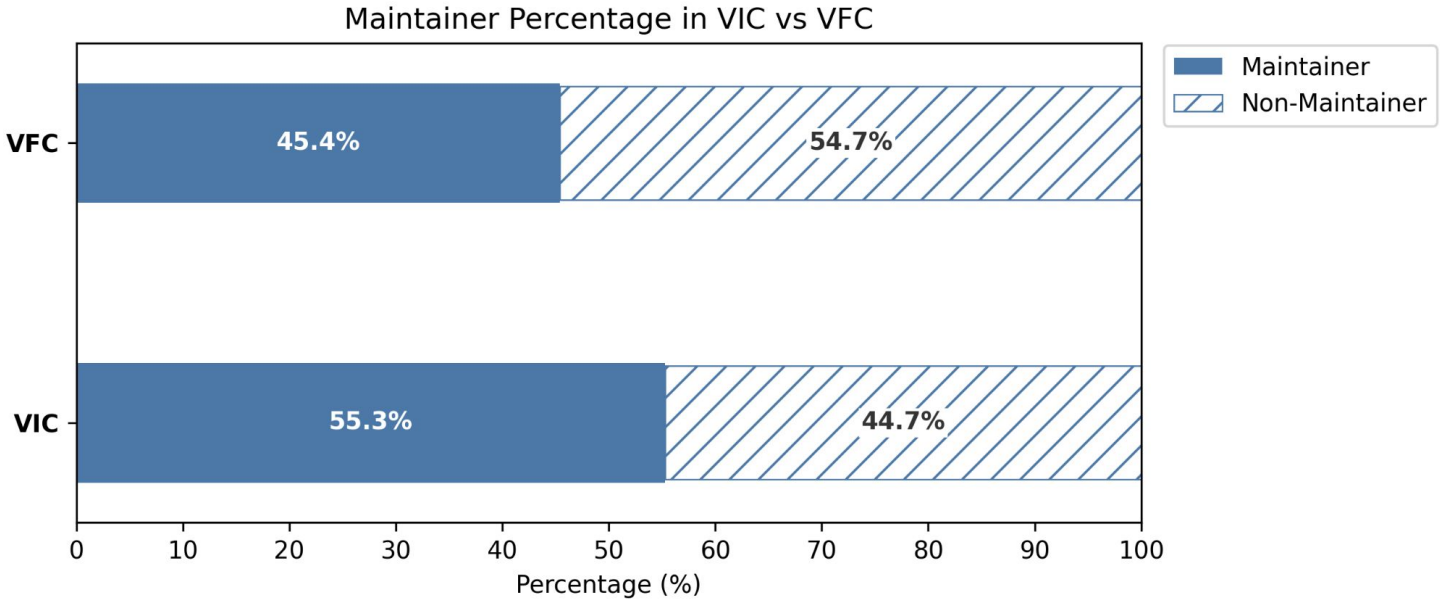
| Developer metrics collected from the mainline repository using Git commands

Developer Experience and Code Familiarity

	VIC Average	VFC Average	P-Value	VFC > VIC Significance
Prior Commit Count (Kernel-level)	616.8	**840.8**	5.3e-34	✓
Commit Count Last Six Months	41.0	**48.6**	1.6e-11	✓
Participation Span (Days)	2501.0	**2854.4**	3.2e-35	✓
Prior Commit Count (Subsystem-level)	780.9	**926.1**	2.8e-14	✓
Prior Commit Count (File-level)	16.3	**19.4**	1.7e-11	✓

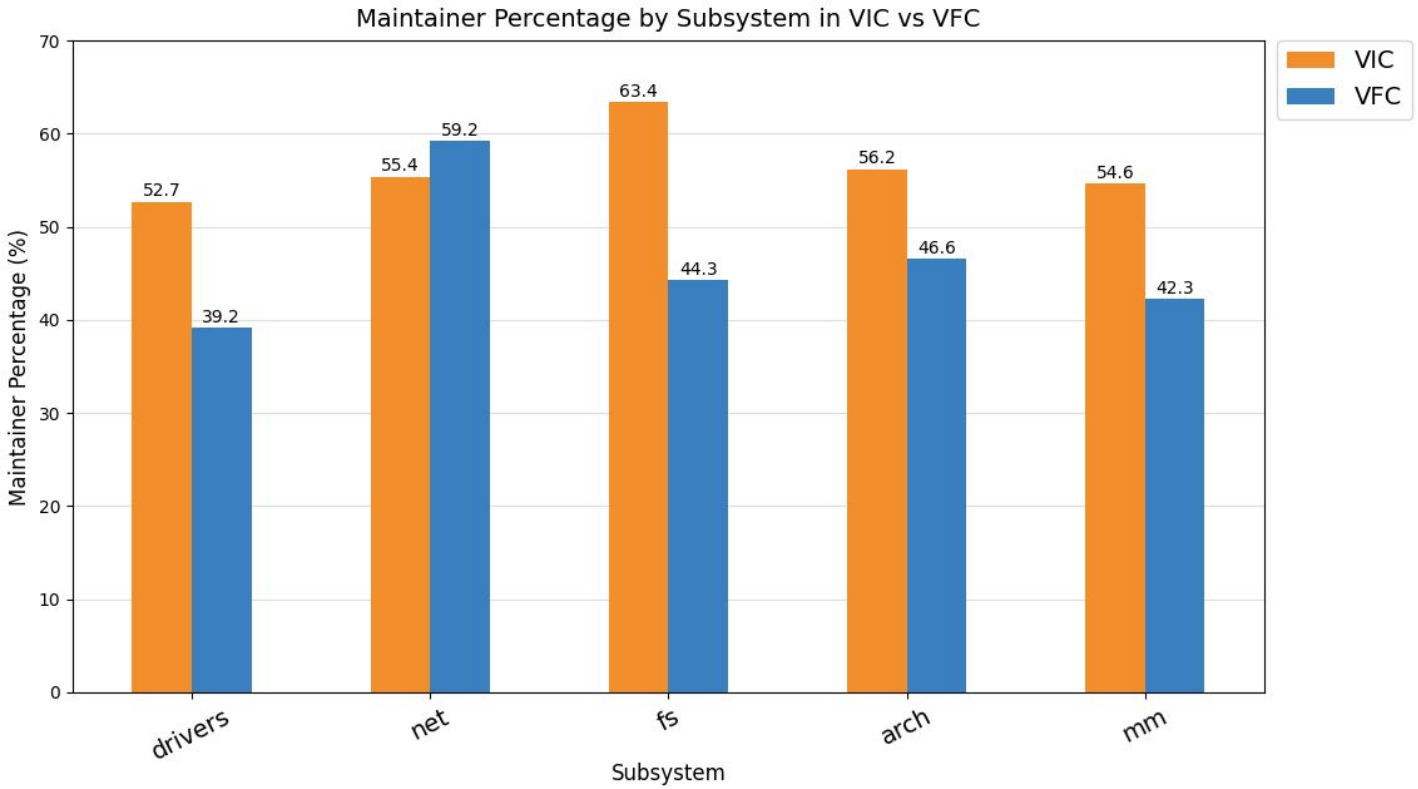
Developers of VICs generally show less prior contribution, activity, and code familiarity compared to developers of VFCs.

Maintainer Representation: VIC vs VFC



Maintainers have a higher representation in VICs compared to VFCs.

Maintainer Representation by Subsystem



Function-Level Code Metrics: *Structural Metrics*

Control Flow Complexity

- **Cyclomatic Complexity**
- **Max Control Depth**

Function Dependency

- **Fan-out Total**
- **Fan-out Unique**

```
static int sched_read_attr(...)
{
    if (!access_ok(...))
        return ERROR;

    if (usize < sizeof(*attr))
{       for (...) {
            if (*addr)
                return ERROR;
        }

        attr->size = usize;
    }

    ret = copy_to_user(...);
    if (ret)
        return ERROR;

    return ret;
}
```

CCN: 6
Max Control Depth: 3
Fan-out Total: 2
Fan-out Unique: 2

| Metrics Extraction Tools: Tree-sitter, Lizard

Function-Level Code Metrics: *Memory Operations*

Allocation

- `kmalloc, vmalloc, alloc_page, ...`

Deallocation

- `kfree, kvfree, free_page, ...`

Reallocation

- `krealloc, kvrealloc, ...`

Memory Access

- `memcpy, memset, ...`

Total Memory Operations

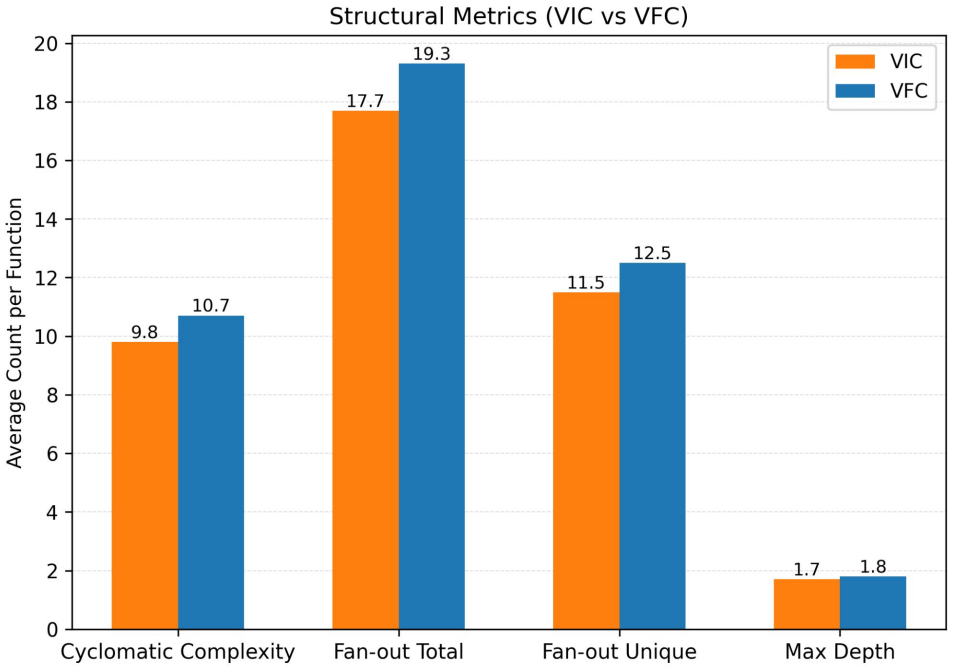
```c
static int brcm_nvram_parse(...)
{
    memcpy_fromio(&header, priv->base, sizeof(header));

    if (memcmp(header.magic, NVRAM_MAGIC, 4))
        return -EINVAL;

    data = kzalloc(len, GFP_KERNEL);
    if (!data)
        return -ENOMEM;

    memcpy_fromio(data, priv->base, len);

    err = brcm_nvram_add_cells(priv, data, len);
    if (err)
        return err;

    kfree(data);

    return 0;
}
```

| API Reference: Memory Management APIs Kernel Doc

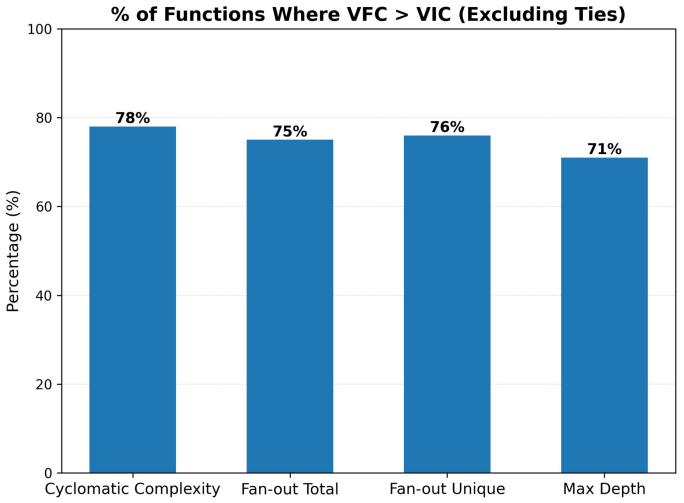
| Metrics Extraction Tool: Tree-sitter

Allocation: 1, Deallocation: 1, Reallocation: 0, Mem Access: 3

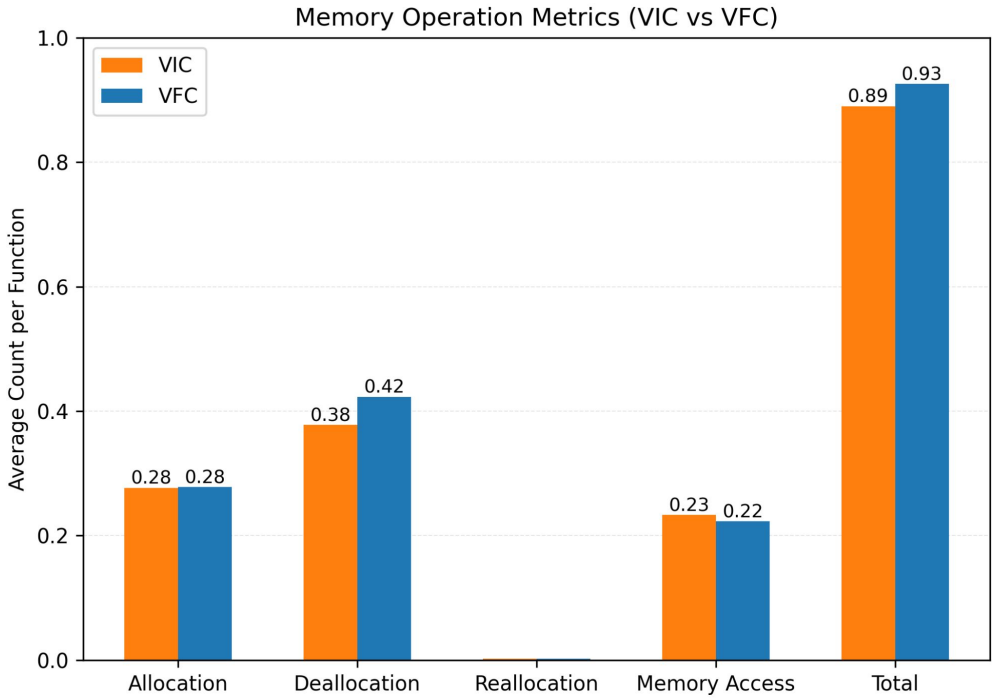
Structural Code Metrics



VFC > VIC across all structural metrics: cyclomatic complexity, fan-out total, fan-out unique, max depth



Memory Operation Metrics

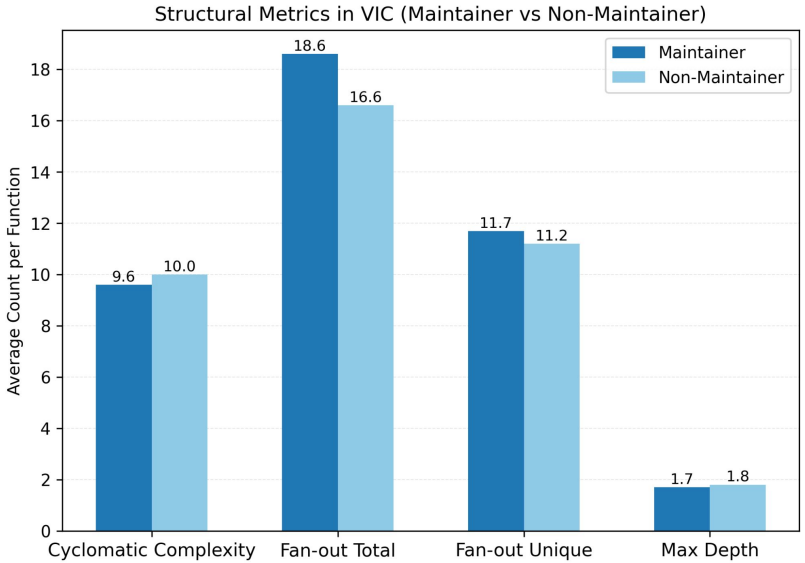


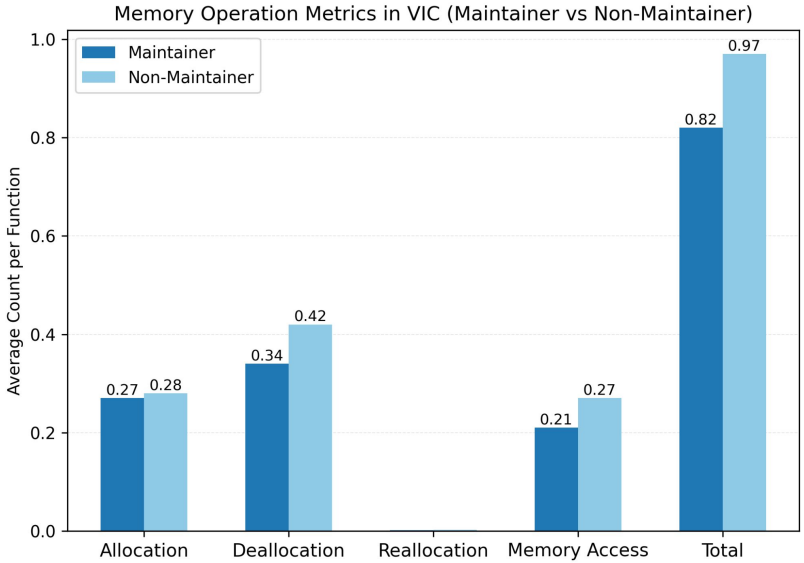
VFC > VIC: total memory operations, deallocation

VIC ≈ VFC: allocation, reallocation, memory access

Reallocation is rare across both VICs and VFCs.

Maintainers vs Non-Maintainers in VICs

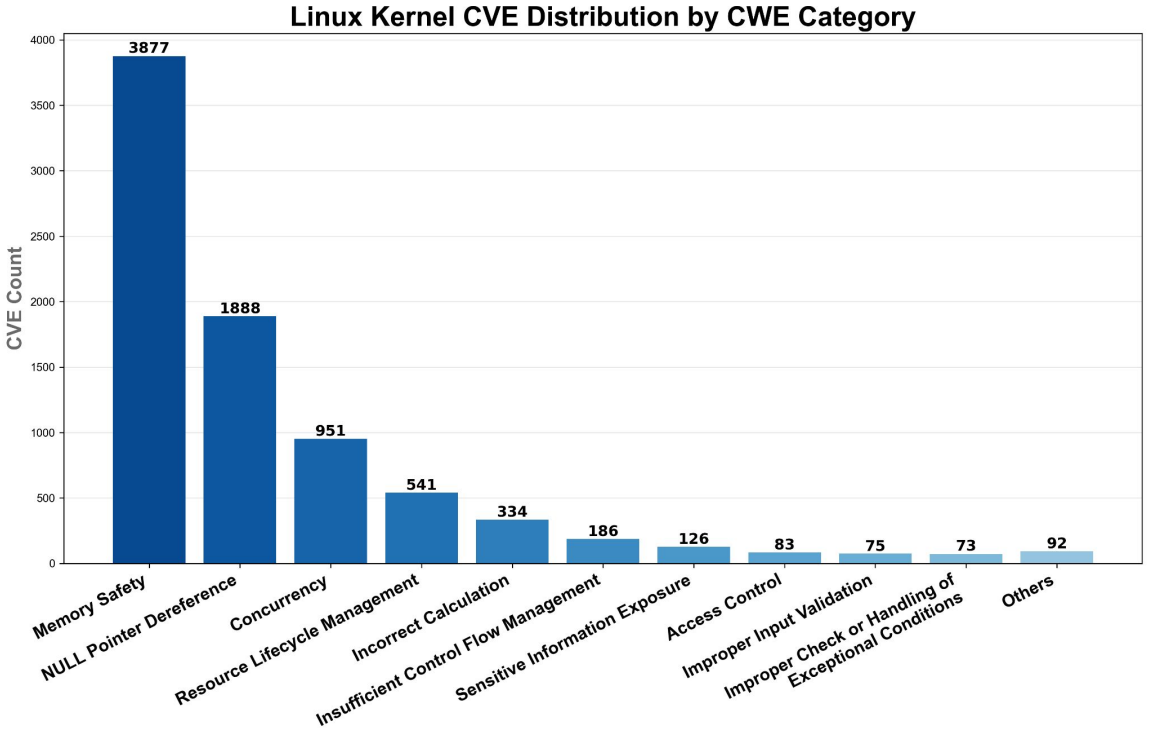


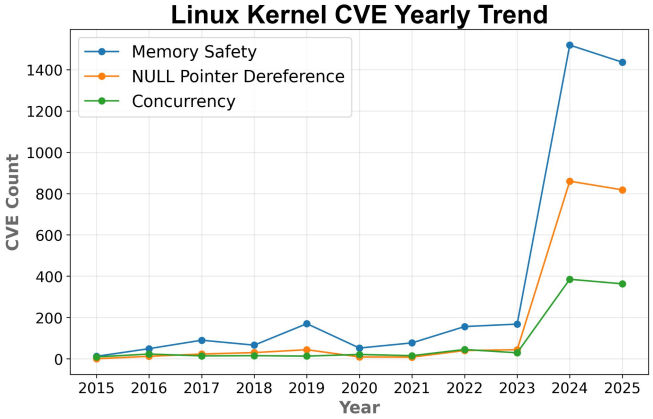


Non-Maintainers > Maintainers: cyclomatic complexity, max depth, **all** memory operation metrics

Maintainers > Non-Maintainers: fan-out total, fan-out unique

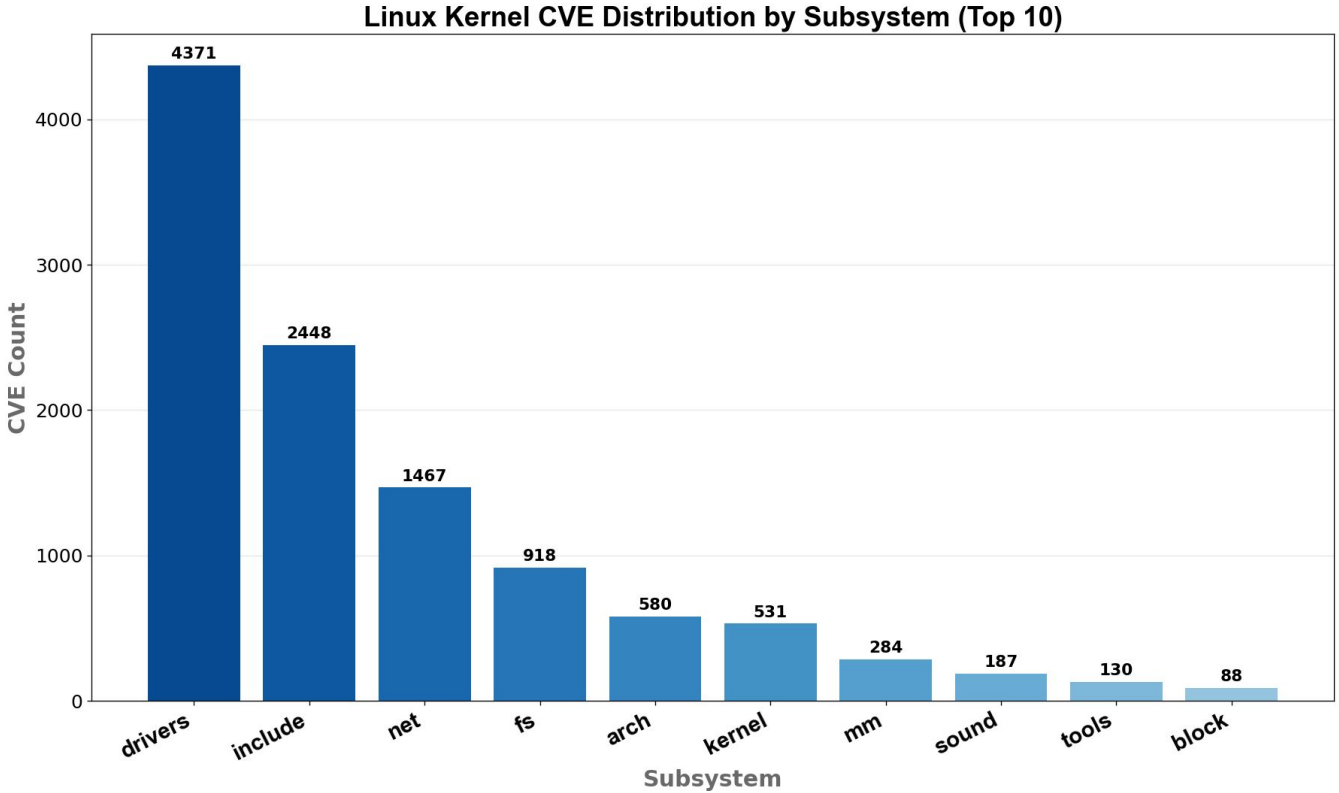
Vulnerability Distribution (2015 - 2025)





CWE category labels based on MITRE CWE-1400

Vulnerability Distribution (2015 – 2025)



Key Findings and Implications

1. Developers of VICs show **less experience and lower code familiarity** than developers of VFCs.
2. **Maintainers have a higher representation in VICs than in VFCs across four out of five subsystems** examined, suggesting opportunities to improve review protocols.
3. Functions that introduce vulnerabilities **do not** necessarily **exhibit higher code complexity patterns**, suggesting that code complexity alone may not be a strong indicator of vulnerability introduction.
4. **Memory safety**, **Null Pointer Dereference** and **Concurrency** account for **over 80%** of the total vulnerabilities. **More than 50% of CVEs** are concentrated in the **drivers**, **net**, and **fs**.

Q & A

Thank you

Exploring Function-Level Code Metrics and Developer Attributes for Linux Kernel Vulnerabilities