One Simple Api Can Cause Hundreds of Bugs

1st BhagyaSri Ramineni
Computer Science
University of North Texas
Denton, Texas
BhagyaSriRamineni@my.unt.edu

2nd Siva Kumar Reddy Surasani
Computer Science
University of North Texas
Denton, Texas
SivaKumarReddySurasani@my.unt.edu

3rd Himaja Boinapally
Computer Science
University of North Texas
Denton, Texas
HimajaBoinapally@my.unt.edu

4th Lakshmi Surekha Maddukuri
Computer Science
University of North Texas
Denton, Texas
LakshmiSurekhaMaddukuri@my.unt.edu

I. Abstract

Reference counting is a memory management technique used for managing the important resources in operating systems like the Linux kernel. Any mistake in the reference counting leads to several issues like use-after-free (UAF) vulnerabilities, memory leaks, and security risks. It also results in system crashes, and attackers will get a chance to exploit and create vulnerabilities. Because of the complex nature of the Linux kernel, developers often fail to update the refcount properly, which results in the refcount bugs. There was a study conducted on 1,033 refcounting bugs in 753 versions of Linux kernels, and their characteristics were analyzed. It was found that these can cause severe security impacts. It provides methodologies helpful for identifying the patterns and providing the recommendations for future work. With the help of the anti-patterns, 351 new bugs were found, and developers confirmed 240 of them. The study highlighted the importance of clear semantics in the documentation and provided templates for that.A technique called a two-dimensional consistency check is used in the study to detect refcounting bugs. Additionally, a tool called FreeWill is used for identifying Use-After-Free(UAF) bugs, which occur when code tries to use a resource after it has been freed. It also suggests integration of automated analysis tools into the development can reduce the bugs during implementation.So by understanding the root causes, developers can design safer APIs. This research focuses on the root causes, detection, and avoidance methods for reference counting bugs.

Keywords : linux kernel, refcounting bugs,vulnerabilities, anti-patterns, two-dimensional consistency check, FreeWill.

II. Introduction

Reference couting is a memory management technique for managing the complex resources in linux kernels. It keeps on tracking of number of references to an object.Using refcouting techniques during the development phase can also lead to bugs called as reference counting bugs. The bugs include memory leaks, use-after-free(UAF) vulnerabilities and resource leaks.

A. *Memory Leaks*

When the allocated memory is not released properly, it results in memory leaks. It results in inaccessible memory as it is not used by the program. For example, let's say a program creates a new object with the help of a constructor and doesn't call the destructor after it is no longer needed. So the memory remains allocated. These memory leaks result in the system slowing down and crashing.

B. *Use After Free*

Use-after-free bugs arise when a program tries to access the resources that are already deallocated. These lead to the security vulnerabilities and make it easy for the attackers to attack. For example, let's say a file pointer is created and freed after use. But if a programmer tries to use it after it got deallocated, it results in Use After Free bugs.

C. *Resource Leaks*

Resource leaks occur when a program fails to release a resource after it is allocated. For example, keeping the database connection open, leaving a file open after the work has done. It prevents other programs from accessing those leaked resources. This degrades the system performance and makes the system unstable.

High-profile system crashes and DOS attacks are some of the problems caused by refcounting bugs. In one study, they identified that 28.3% of ref-couting bugs cause UAF vulnerabilities and 71.7% cause memory leaks. An API can cause so many hundreds of bugs. In the complex software lifestyles, these refcounting bugs make it even more complex, which results in raising of new problems. Refcouting bugs should be resolved correctly because the process of handling them can result in the emergence of new bugs. How can these bugs be detected and avoided in order to improve the security of the system?

III. Methodology

Some advanced technologies were suggested to identify, analyze, and mitigate these refcounting bugs. These include dataset analysis, semantic analysis, the building of automated tools, and suggesting ways to reduce the evolution of these bugs.

A. *Refcounting Bug Dataset*

A study was conducted among 753 versions of Linux kernels (2005–2022). They used the commit logs and keyword searches to identify the bug patterns. They filtered out the commit logs using API keywords like "hold," "release," and "put" to identify the issues that result in refcounting bugs.

From 1825 commits, 1033 bugs were identified when analyzed manually. To reduce the false positives, "Fixes" tags are used in the commit logs. False positives occur when a commit log results in a refcounting bug, but after realizing that it is not.

- The keyword search is limited and could not find all the bugs.
- For the false negative bugs (real bugs that were missed), future research is suggested.
- Suggestions include machine learning techniques and developing automatic tools while using them during the development phase to reduce these bugs.

B. Semantic Templates

Templates are designed to describe the refcounting bugs. They include operations and context. Operations include increment and decrement, while context includes statements, functions, etc. With the help of these, it is easier to identify the common bugs. Symbols were used to describe operations and context. [2]

- **Operations**
  - G = Increment refcount
  - P = Decrement refcount
  - A = Assignment to reference
  - D = Dereferencing an object
  - L = Locking a resource
  - U = Unlocking a resource
  - Fstart = Function start
  - Fend = Function end
- **Context**
  - S = Statement
  - B = Basic block
  - F = Function
  - M = Macro

Two types of templates were provided using these operations and context

1) **Template 1**: A refcounting is incremented but not decremented. Sometimes, in a code, after incrementation happens, it jumps into the error-handling code. This leads to memory leaks.
2) **Template 2**: Refcounting is decremented, and the reference object is used after decrementation. This leads to UAF vulnerabilities where objects are used after they are freed.

C. Two-Dimensional Consistency Check

This method can effectively reduce false positives and detect bugs. With the help of consistency rules, it ensures the correctness of reference counting. It verifies that the program follows particular patterns for increment and decrement. This helps in preventing common errors like memory leaks and use-after-free bugs. [3]

CID (Consistency and Inconsistency Detection) is a tool that examines the source code directly without executing it. It follows two consistency rules

1) **Increment-Decrement Consistency**: Balances each increment operation with a corresponding decrement operation.
2) **Decrement-Decrement Consistency**: Ensures no single decrement operation is applied multiple times to the same reference.

D. FreeWill

FreeWill is a tool used to identify the Use-After-Free (UAF) bugs in binary programs. As it doesn't require source code and completely uses the binary code, it is efficient for closed software. This technique includes [1]

1) **Dynamic Trace Collection**
   - While the program is running, FreeWill inserts a monitoring code into it.
   - It records the program's behavior during execution, focusing on operations that involve reference counters.
   - It collects execution traces, which consist of refcounting operations like increment and decrement.
2) **Omission-Aware Model**
   - This framework identifies inconsistencies in refcounting.
   - Inconsistencies include:
     - Refcount incremented but never decremented.
     - Refcount decremented but dereferencing the object afterward.
   - It helps identify UAF bugs.
3) **Bug Diagnosis**
   - Analyzes execution traces pin-to-pin after identifying inconsistencies.
   - Provides detailed observation of bug behavior, making it easier for developers to classify bugs.

E. Semantic similarities

Refcounting API Key Word	Bug-Caused API Key Word
	foreach
increase	0.22
get	0.32
ref-count	0.19
hold	0.29
grab	0.27
retain	0.14
decrease	0.21
drop	0.22
put	0.38
un-hold	-0.13
release	0.33

TABLE I
THE SEMANTIC SIMILARITIES BETWEEN API KEYWORD AND BUG-CAUSED API KEY WORD

[2]

Sematic similarity refers to how much two pieces of words are related to each other. It often reveals important features

of two words without relating to the sytantics or structures.

This study identified a real world bug involving a smart loop mechanism called for for each matching node. This mechanism used a special function during each loop to manage the object references. This decreases the refcount for one object and increases for a new one. However, this behaviour is hidden from developers using the loop, as they are primarily focused on the looping logic.

When the loop exits early due to a special condition, developers skip the step of balancing the refcounting leading to issues. Additionally the naming of the smart loop mechanism which is foreach is not associated with the reference management functionalities, so there is a high chance of missing refcounting. This study also reveals that foreach has a less similarity with all the refcounting keywords like increase, get, hold, grab. Figure 1 shows the semantic similarities of all the keywords with the function names.

F. Anti patterns

Minor deviations are accepted for kernel development, considering the varied requirements of the implemented functions. This study suggests that detailed explanation of API details using appropriate comments can detect major deviations and helps in reducing the API bugs.

This study proposed two anti patterns shown in Table 2 that causes major API bugs. By avoiding these patterns kernel developers can reduce the API bugs significantly.

Bug	Semantic Templates
Anti-Pattern 1	$\mathcal{F}_{\text{start}} \rightarrow \mathcal{S}_{\mathcal{GN}} \rightarrow \mathcal{B}_{\text{error}} \rightarrow \mathcal{F}_{\text{end}}$
Anti-Pattern 2	$\mathcal{F}_{\text{start}} \rightarrow \mathcal{S}_{\mathcal{GN}} \rightarrow \mathcal{S}_{\mathcal{D} \circ \mathcal{N}} \rightarrow \mathcal{F}_{\text{end}}$

TABLE II
SEMANTIC TEMPLATES FOR THE PROPOSED ANTI-PATTERNS.

[2]

IV. Literature Review

Paper 1: One Simple API Can Cause Hundreds of Bugs

Research Direction: This paper investigates the reference counting bugs in linux kernels, analyzing their causes, and impacts.

Techniques: Focussed on pattern based and semantic based detection of refcounting bugs. It also proposes anti-patterns and lessons for kernel developers.

Theory: It focuses on applications with practical insights, improving kernel stability, and security.

Improvements : It builds on existing detection methods and expands its focus on finding the root causes of these bugs and also the anti-patterns for proactive solutions.

Methodology: It uses historical data of kernel patches and identify patterns to find the API bugs.

Strengths: Across 753 kernel versions 1033 bugs are analyzed. Effective in identifying anti-patterns and demonstrating the impacts by detecting and validating kernel bugs.

Weaknesses: Race conditions are ignored and historical bug identification is limited.

Paper 2: Detecting Kernel Refcount Bugs with Two-Dimensional Consistency Checking

Research Direction: This paper proposes a mechanism CID for detecting kernel refcount bugs using two dimensional consistency checking to reveal mismatches between INC and DEC operations.

Techniques: Introduces two dimensional consistency checking and behaviour based inference for detecting refcounting bugs.

Theory: It is driven with practical applications in kernel debugging and security vulnerability mitigation.

Improvements : Enhances previous methods by focusing on consistency checking rather than traditional bug detection techniques.

Methodology: Utilizes static analysis and behaviour based inference to systematically identify refcounting bugs in linux kernel version.

Strengths: High precision and coverage with over 36 of the bugs confirmed by developers. With kernel detection time of 18 minutes, it is one of the fastest for analyzing bugs related to refcounting.

Weaknesses: Potential false positives and false negatives with a single kernel version.

Paper 3: FREEWILL: Automatically Diagnosing Use-After-Free Bugs via Reference Miscounting Detection on Binaries

Research Direction: UAF bugs are identified using omission aware counting model which systematically identifies bugs by detecting reference miscounting.

Techniques: Implemented a FREEWILL tool for finding UAF bugs by implementing complex diagnostic techniques.

Theory: It debugs binary level UAF vulnerabilities by analyzing inconsistencies in reference counting.

Improvements: It builds on existing bug detection techniques and providing automated patch suggestions.

Methodology: It implements heuristic methods to detect the UAF bugs by analyzing anomalies and patterns in memory allocation and de-allocation in refcounting.

Strengths: It analyzed and detected 48 refcounting bugs. It's average speed of operation is around 15 minutes.

Weaknesses: As discussed in paper 1, it also has a limited ability to handle race conditions.

V. Research Challenges and Future Work

In the future, improving the performance of tools like CID and FREEWILL for analysing large codebases will increase the accuracy. During kernel development, creating the a collaborative platform will facilitate in sharing of bug patterns, anti patterns. The integration of machine learning algorithms into the tools helps in the prediction of bugs. This helps in identifying the areas where there is a chance of getting the bugs and helps in rectifying that. When multiple processes run in parallel, it leads to synchronisation issues and results in memory leaks or bugs.

By addressing these challenges, using static and dynamic analysis tools can detect those kinds of bugs during the development phase. The development of automated tools for parallel processes in order to monitor them can also reduce the occurrence of refcounting bugs. By including all these, it will be very effective in avoiding most of the refcounting bugs in the future

VI. Conclusion

Refcounting in modern linux kernels present a larger security challenges, often leading to critical vulnerabilities such as use-after-free(UAF) and memory leak bugs. This survey focuses on three key advancements or methods in dealing with these refcounting bugs. The analysis of root causes, API keywords, anti patterns, and semantics for refcounting bugs reveals a significant patterns of the refcounting bugs. The development of CID which is a scalable detection system by consistency checking explains how one can use condition rules to avoid the critical kernel bugs. The behavior based inference can also increase the bug detection coverage while avoiding complex semantic reasoning. It also introduces FREEWILL, an automated detection tool. This tool demonstrates the real world applications of the omission aware models in detecting tracing UAF bugs and find their root causes much more effectively. These techniques or the root causes emphasize that addressing refcounting bugs or issues requires tools capable of diagnosing and preventing vulnerabilities in addition to the precise detection techniques.

Going forward, the teachings from these papers will provide a solid foundation for all the other developers especially kernel developers to be cautioned against the potential refcounting bugs. By avoiding the coding practices or anti patterns discussed, developers can reduce the security vulnerabilities. The combination of static analysis, dynamic verification, and condition aware inferences can play a key role in mitigating the security threats posed by these issues.

References

[1] Liang He, Hong Hu, Purui Su, Yan Cai, and Zhenkai Liang. FreeWill: Automatically diagnosing use-after-free bugs via reference miscounting detection on binaries. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2497–2512, Boston, MA, August 2022. USENIX Association.

[2] Liang He, Purui Su, Chao Zhang, Yan Cai, and Jinxin Ma. One simple api can cause hundreds of bugs an analysis of refcounting bugs in all modern linux kernels. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 52–65, New York, NY, USA, 2023. Association for Computing Machinery.

[3] Xin Tan, Yuan Zhang, Xiyu Yang, Kangjie Lu, and Min Yang. Detecting kernel refcount bugs with Two-Dimensional consistency checking. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2471–2488. USENIX Association, August 2021.