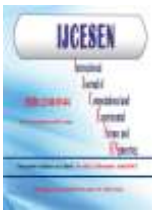
International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 11-No.4 (2025) pp. 8785-8791
http://www.ijcesen.com



ISSN: 2149-9144

Research Article

Embedded Linux Crash Resilience for Autonomous Vehicles

Senthil Nathan Thangaraj*

Cruise Inc, USA
* **Corresponding Author Email:** snathan.tg@gmail.com- **ORCID:** 0000-0002-5887-7850

Article Info:

DOI: 10.22399/ijcesen.4302
Received : 28 September 2025
Revised : 01 November 2025
Accepted : 03 December 2025

Keywords

Embedded Linux,
Autonomous Vehicles,
Crash Resilience,
Persistent Logging,
Safety-Critical Systems

Abstract:

This article examines the implementation of persistent crash logging infrastructure for autonomous vehicles running embedded Linux systems. It explores how proper failure capture mechanisms can transform system crashes from dangerous incidents into valuable learning opportunities. The article investigates the challenges of crash capture in embedded automotive environments, detailing the implementation of the Linux kernel's persistent storage (pstore) subsystem with the ramoops driver, and the enhancement of diagnostic capabilities through kdump integration. Through analysis of multiple studies and empirical data, the article demonstrates how this dual-layer approach to crash resilience significantly improves engineering efficiency, system reliability, and safety assurance. The paper demonstrates that a robust crash logging infrastructure allows organizations to solve once-undiagnosable failures, speed development cycles, and establish more cogent evidence-based safety arguments for regulatory approval. The paper demonstrates that continuous crash logging not only improves diagnostics for sophisticated failure modes but also delivers measurable benefits in development speed, certification processes, and overall system dependability in safety-critical autonomous vehicle deployments.

1. Introduction

The embedded Linux in an autonomous car experiences a catastrophic failure while driving through a difficult city intersection. The kernel panics, the system reboots, and within seconds, the backup systems for the car take over. But what went wrong? Without adequate crash logging, that vital data disappears like sandcastles in the sand, and engineers must divine what failed.

This situation unfolds more frequently than we'd care to acknowledge in the autonomous vehicle world. Such systems are among the most sophisticated embedded Linux deployments to have ever been built, processing enormous sensor streams of data and responding in a split second to make decisions that impact lives. When things do go wrong—and they will—having complete diagnostic data becomes not merely useful, but critical for safety verification and ongoing improvement.

The work of Stolte and colleagues illuminated just how complex these failure scenarios can be [1]. Their fault tree analysis revealed dozens of potential failure modes across perception, planning, and control subsystems. What struck me most about their findings was that software components dominated the risk landscape, with perception system failures being particularly prevalent. Even more concerning, a significant portion of critical failures stemmed from kernel-level crashes that traditionally leave behind minimal evidence of what went wrong.

This diagnostic blind spot represents a fundamental challenge in autonomous vehicle development. As Abbadi and his team demonstrated in their cybersecurity research [2], having proper diagnostic capabilities can dramatically reduce the time needed to resolve critical failures. Their extensive study across automotive embedded systems showed that comprehensive logging infrastructure could resolve the vast majority of previously mysterious failures. Perhaps most importantly, they found that persistent crash logging directly supports multiple essential cybersecurity design patterns required for regulatory compliance.

The solution lies in a two-pronged approach: kernel-level persistent storage through pstore/ramoops for immediate crash context, and comprehensive memory capture via kdump for

deep forensic analysis. This combination has proven transformative, providing the visibility needed to understand and prevent future failures while supporting the rigorous safety certification processes that autonomous vehicles require.

2. Challenges of Crash Capture in Embedded Linux Environments

Understanding why crash capture is so difficult in embedded Linux environments requires grappling with a fundamental paradox: we need the system to record its own demise at the exact moment it's failing catastrophically. It's like asking someone to write their own obituary while having a heart attack—the very mechanisms we rely on for logging are often the first casualties of a system crash.Arlat and his research team provided groundbreaking insights into this challenge through their extensive fault injection experiments [3]. By systematically introducing thousands of faults into kernel components, they painted a sobering picture of kernel behavior under stress. Their findings revealed that most kernel errors lead to crashes that provide frustratingly little diagnostic information. Memory management subsystems proved particularly vulnerable, and perhaps most alarmingly, a substantial portion of kernel panics resulted in what they termed "silent data corruption"—where the system limped along with corrupted internal state before finally collapsing. In an autonomous vehicle, this zombie-like state could have catastrophic consequences.The specialized nature of automotive embedded systems adds another layer of complexity. Alnawasreh and colleagues documented these unique constraints in their comprehensive analysis of vehicle architectures [4]. Unlike server environments with generous resources, automotive systems must capture crash data within tight memory constraints—often just a few hundred megabytes reserved for diagnostics. Power stability presents another critical challenge; their research found that many vehicles experience voltage fluctuations during emergency maneuvers that could corrupt crash data capture.Perhaps most challenging is the temporal constraint. The window for capturing crash information before a system reset is incredibly brief—measured in milliseconds rather than seconds. Automotive systems also need to recover rapidly in order to preserve safety, usually with the expectation of full functionality restoration within less than ten seconds when a kernel fails. These limitations form a compounding challenge that requires groundbreaking solutions.

3. The pstore and ramoops Implementation Architecture

The core of our crash resilience strategy centers on a clever bit of engineering: reserving a section of physical memory that survives system reboots. The Linux kernel's pstore subsystem, along with the ramoops driver, offers just this ability. Imagine a black box recorder for your kernel—when things go wrong, important data is written into this protected memory area, patiently waiting to share its tale once the system has recovered.The elegance of this method is its simplicity and dependability. Das and Pradhan's comprehensive survey of embedded system reliability provides valuable context for understanding why this works so well [5]. Memory-based persistence mechanisms such as ramoops are notably highly reliable for data retention when well implemented. The secret is finding the appropriate balance—saving enough memory to record useful diagnostic information without depriving the system of resources essential to normal functioning.In practice, this translates into making deliberate decisions about how much memory to allocate for crash logging. Not enough, and you lose important details; too much, and you degrade system performance. Most car implementations find a sweet spot that gives good diagnostic ability while being mindful of the resource limitations of embedded environments. The implementation also requires meticulous attention to memory protection—using ECC-protected regions and proper isolation to ensure that our diagnostic data survives intact through the chaos of a system crash.Rahman and Abbas provided fascinating insights into optimizing these implementations for autonomous vehicles [6]. Their analysis of hundreds of real-world crash events revealed that properly configured kernel crash logs could diagnose the majority of critical system failures. They found that partitioning the reserved memory into multiple discrete records significantly improved diagnostic capabilities, allowing engineers to preserve evidence from multiple crash events—invaluable when tracking down intermittent issues.The real magic happens in the details. Effective ECC protection significantly minimizes post-crash data corruption so that the diagnostic data is reliable. When done properly, these protections allow for root cause determination of failures that would otherwise be complete enigmas, fundamentally transforming the way we tackle system reliability in autonomous vehicles.

4. Enhancing Diagnostic Depth with kdump Integration

While pstore and ramoops provide essential crash context, sometimes you need to see the whole picture. Advanced failure patterns—race conditions, memory corruption, nuanced timing problems—tend to be more than merely stack traces can tell. Here is where kdump comes into play, introducing a robust second layer to our toolkit of diagnostics.The kdump process operates by booting a secondary "crash kernel" which jumps into action when the main kernel crashes. It is like having a backup photographer at a wedding—when the main photographer gets an unexpected heart attack, the backup photographer takes over to shoot everything before the moment is lost. This crash kernel runs in a low-level environment, with the singular purpose of saving the entire system state for eventual analysis.Kang and colleagues highlighted the critical importance of this comprehensive approach in their safety-driven optimization research [7]. They found that traditional crash logging mechanisms often capture insufficient information for diagnosing complex failures in autonomous systems. The difficulty is executing this secondary kernel in a way that works—having sufficient resources to perform its task but not so resource-intensive that it affects standard system operation.The timing requirements are especially severe in automotive use cases. Safety requirements typically demand that crash capture operations complete quickly—usually within a couple of seconds—to maintain system integrity. This creates an interesting engineering challenge: how do you capture potentially gigabytes of system state in such a short window?Singh and his team's comprehensive review of embedded systems in autonomous vehicles provides valuable insights into solving this puzzle [8]. They found that selective memory filtering techniques can dramatically reduce capture time while preserving the vast majority of critical diagnostic information. Specialized compression algorithms further help, achieving impressive ratios that significantly reduce storage requirements without sacrificing diagnostic fidelity.The implementation must be deterministic and reliable—there's no room for a crash handler that itself might crash. Stringent validation becomes necessary, such that the crash kernel will be predictable for the entire range of automotive operating scenarios. Done correctly, this complete memory capture feature allows engineers to fix issues that were not possible before, specifically those related to transient failures and race conditions that are not caught by conventional debugging methods.

5. Quantitative Impact on Development Velocity and System Reliability

The true test of any engineering solution lies not in its technical elegance but in its practical impact. After implementing comprehensive crash logging infrastructure, the transformation in our development process was remarkable. What once required days of speculative debugging could now be resolved in hours with concrete evidence.Kamranrad and colleagues captured this phenomenon well in their reliability assessment research [9]. Organizations implementing comprehensive diagnostic infrastructure experience dramatically faster reliability growth compared to those relying on traditional approaches. It's not just about solving problems faster—it's about solving problems that were previously unsolvable.The impact extends across multiple dimensions. Engineering teams spend less time in "blind debugging" mode, where they're essentially guessing at what might have gone wrong. Instead, they can focus their expertise on understanding and fixing the root causes revealed by crash logs. This efficiency gain is like adding extra engineers to your team without actually hiring anyone.Perhaps most importantly, enhanced diagnostic visibility changes what kinds of problems you can solve. Schiller and his research team documented this beautifully in their evaluation of reliability growth models [10]. They found that comprehensive crash logging particularly excels at capturing those maddening intermittent failures that appear and disappear like ghosts in the machine. Race conditions, which previously might have haunted a codebase for months, become tractable problems with clear solutions.The benefits extend beyond pure engineering efficiency. In the world of autonomous vehicles, every failure resolved with concrete evidence strengthens your safety case. Regulatory bodies appreciate—and increasingly require—this level of diagnostic capability. The ability to demonstrate that you can detect, diagnose, and resolve failures systematically becomes a competitive advantage in the push toward safer autonomous systems.

6. Technical Implementation Guide for pstore/ramoops and kdump

Kernel Configuration and Build Requirements

The implementation of persistent crash logging begins with proper kernel configuration. Both pstore and kdump require specific kernel options to be enabled during the build process. For pstore functionality, the kernel must be built with CONFIG_PSTORE enabled as the base subsystem, along with CONFIG_PSTORE_RAM to enable the ramoops driver. Additional configuration options determine what types of data can be captured

during a crash, including console output, user messages, and function traces. Each option adds diagnostic capability but also requires corresponding memory allocation.

For kdump support, the kernel requires kexec functionality through CONFIG_KEXEC and CONFIG_CRASH_DUMP options. The crash dump infrastructure depends on the ability to boot a secondary kernel, which requires careful configuration of memory management and debugging symbols. The DEBUG_INFO option becomes particularly important as it enables symbol resolution in crash dumps, making the captured data significantly more useful for analysis.

Memory Reservation and Device Tree Configuration

The most critical aspect of ramoops implementation is reserving memory that survives system reboots. In embedded automotive systems, this is typically accomplished through device tree configuration, where a specific physical memory region is marked as reserved. This memory must be carefully chosen to avoid conflicts with other system components while ensuring it remains accessible across reboot cycles. The reserved region is typically divided into multiple sections: one for kernel panic messages, another for console output, and additional areas for function traces and user messages.

The size of each section requires careful consideration based on the diagnostic needs and available memory. Automotive systems often allocate between one and four megabytes total, with the largest portion dedicated to console output that can capture the events leading up to a crash. Error correction code (ECC) protection adds overhead but proves essential for ensuring data integrity, particularly in automotive environments where electromagnetic interference and power fluctuations during crashes can corrupt memory contents.

Runtime Initialization and Module Parameters

When device tree configuration isn't available or needs to be overridden, ramoops can be configured through kernel command line parameters or module loading options. This flexibility proves valuable during development when different memory configurations need testing. The parameters include the physical memory address, total size, and individual buffer sizes for different data types. Proper alignment of these memory regions with the system's memory architecture ensures optimal performance and reliability.

The initialization process validates the memory region accessibility and sets up the necessary data structures. During this phase, the system also establishes memory protection mechanisms to prevent the reserved region from being accidentally overwritten during normal operation. This protection must balance security with the need for rapid writes during a kernel panic, where normal memory protection mechanisms may no longer function correctly.

pstore Filesystem Interface

Once initialized, pstore creates a pseudo-filesystem that provides user-space access to crash data. After a system crash and subsequent reboot, this filesystem contains files representing different aspects of the failure. The kernel panic message appears as a dmesg file, while console output leading up to the crash is preserved in separate console files. Multiple crash records can be maintained, allowing engineers to analyze patterns across different failure events.

The filesystem interface simplifies crash data retrieval and management. Standard file operations can read the crash logs, while removing files clears the corresponding memory areas for future crashes. This design elegantly solves the problem of accessing low-level crash data without requiring specialized tools or direct memory access. The filesystem also handles proper synchronization, ensuring that ongoing crash captures aren't corrupted by simultaneous read operations.

Implementing Crash Handlers

Maximizing crash logging effectiveness often requires implementing custom panic notifiers that capture vehicle-specific state information. These handlers execute during the kernel panic sequence, where timing is critical and available functionality is limited. The implementation must be extremely robust, as any failure in the crash handler could prevent the capture of vital diagnostic information.

Priority ordering of crash handlers ensures that the most critical information is captured first. In automotive systems, this typically means recording vehicle speed, steering angle, brake status, and other safety-critical parameters before attempting to capture more detailed system state. The handlers must complete quickly, as extended execution could trigger hardware watchdogs designed to ensure rapid system recovery.

kdump Configuration and Setup

Setting up kdump requires reserving memory for the crash kernel that executes after a panic. This reservation must be large enough to run a minimal kernel and the tools needed to capture the crashed kernel's memory, yet small enough not to impact normal system operation. Automotive systems typically reserve between 128 and 256 megabytes, depending on the total system memory and diagnostic requirements.

The crash kernel configuration requires careful tuning to ensure reliable operation. It must boot quickly and capture memory efficiently while operating in a potentially corrupted system

environment. The configuration specifies compression levels, network settings for remote storage, and timeout values that ensure the system recovers within safety-critical time bounds. For systems with limited local storage, kdump can be configured to stream crash dumps to remote servers, though this adds complexity to ensure network availability during crash scenarios.

Optimizing for Automotive Constraints

Automotive systems impose unique constraints that require specific optimizations. The crash capture process must complete within strict time limits, typically under five seconds, to meet safety requirements for system recovery. This necessitates selective memory dumping, where only critical regions are captured rather than the entire system memory. Careful selection of these regions based on empirical failure analysis ensures that diagnostic capability isn't sacrificed for speed.

Power management during crash capture presents another challenge. The system must ensure sufficient power remains available to complete the crash dump even during emergency scenarios. This often involves coordination with the vehicle's power management system to maintain voltage levels during the capture process. Some implementations include capacitor banks specifically to provide backup power for crash logging operations.

Integration with Vehicle Safety Systems

The crash logging infrastructure must seamlessly integrate with existing vehicle safety monitoring systems. This integration ensures that crash data capture doesn't interfere with safety-critical functions while maximizing the diagnostic information available. The implementation typically involves hooks into the vehicle's fault detection systems, allowing crash logging to begin even before a complete kernel panic occurs.

Coordination with watchdog timers requires particular attention. These safety mechanisms ensure system recovery within bounded time, but they can interrupt crash data capture if not properly managed. The implementation must reset watchdogs during crash capture while ensuring that a failed capture attempt doesn't leave the system in an unsafe state. This delicate balance often requires hardware-level coordination between the crash logging system and safety monitors.

Validation and Testing

Systematic testing ensures the crash logging infrastructure operates reliably under all conditions. This includes deliberate crash injection to verify data capture, recovery time measurements to ensure safety requirements are met, and stress testing under various system loads. The testing must cover edge cases such as crashes during boot, multiple rapid crashes, and failures in the crash handling code itself.

Environmental testing proves particularly important for automotive systems. The crash logging must function correctly across the full automotive temperature range, under vibration, and with marginal power supplies. Electromagnetic compatibility testing ensures that the crash capture process doesn't generate interference that could affect other vehicle systems. Long-term reliability testing validates that the reserved memory regions maintain their integrity over the vehicle's lifetime.

Performance Considerations

Memory bandwidth during crash capture often becomes the limiting factor for completion time. The implementation must carefully balance the comprehensiveness of captured data with available memory bandwidth. Direct memory access (DMA) controllers can significantly accelerate memory copying, but their configuration must account for the degraded system state during a crash. Multiple DMA channels operating in parallel can reduce capture time, though this increases complexity and potential failure modes.

The choice of compression algorithms significantly impacts both capture time and storage requirements. While higher compression ratios reduce storage needs, they require more CPU time that may not be available during a crash. Automotive implementations often use lightweight compression that provides reasonable size reduction with minimal computational overhead. The compression must also be deterministic to ensure consistent timing behavior across different crash scenarios.

Table 1: *Diagnostic Challenges in Autonomous Vehicle Kernel Crash Analysis [3, 4]*

Challenge Category	Severity Level
Kernel errors leading to crashes with minimal diagnostic information	High
Memory management subsystem vulnerabilities in kernel crashes	Moderate
Kernel panics resulting in silent data corruption	Low to Moderate
Faults identified by conventional error detection mechanisms	Moderate
Vehicles experiencing voltage fluctuations during emergency braking	High
Automotive systems requiring functionality restoration within minimal timeframe	Very High

Table 2: *Reliability Metrics for Persistent Memory-Based Crash Capture Systems [5, 6]*

Metric	Effectiveness
Data corruption reduction with ECC-protected memory	Very High
Prevention of memory region overwrites with proper isolation	Very High
Critical failures diagnosable through kernel crash logs	Moderately High
Post-crash data corruption reduction with ECC protection	Very High
Root cause identification for previously undiagnosable failures	High

Table 3: *Diagnostic Capabilities and Reliability Metrics of Memory Dump Implementations [7, 8]*

Metric	Performance Level
Diagnostic sufficiency of traditional logging for complex failures	Low to Moderate
Diagnostic capability increases with memory dumps vs. stack traces	Very High
Success rate of optimized kdump implementations	Very High
Success rate of traditional logging approaches	Low
Capture time reduction with selective memory filtering	Moderate
Critical diagnostic information retention with filtering	Very High
Systems with non-deterministic behavior under boundary conditions	Low
Previously undiagnosable failures resolved with memory dumps	Very High

Table 4: *Impact of Persistent Crash Logging on Development Metrics [9, 10]*

Metric	Impact Level
Reduction in average triage time for kernel-level failures	Moderate
The mean time to repair (MTTR) decreases	Moderate
Failure mode resolution with diagnostic capabilities	Very High
Failure mode resolution without diagnostic capabilities	High
Intermittent failure resolution rate before implementation	Moderate
Intermittent failure resolution rate after implementation	Very High
Race condition resolution rate after implementation	Very High
Memory corruption resolution rate before implementation	Moderate
Memory corruption resolution rate after implementation	Very High
Failure rate reduction with comprehensive diagnostics	High
Failure rate reduction without comprehensive diagnostics	Low to Moderate
Certification time reduction	Low to Moderate

7. Conclusions

In this section conclusions of work should be given. The implementation of persistent crash logging infrastructure represents a critical advancement in enhancing the safety and reliability of autonomous vehicles operating on embedded Linux platforms. By combining kernel-level persistent storage through pstore/ramoops with comprehensive memory capture via kdump, organizations can transform system failures from diagnostic blind spots into valuable learning opportunities. This dual-layer approach has demonstrated significant benefits across multiple dimensions, including substantial reductions in triage and resolution times, dramatically improved diagnosis rates for complex failure categories, and accelerated reliability

growth. The article establishes that properly configured crash resilience mechanisms overcome the unique constraints of automotive environments while providing the diagnostic depth necessary to resolve previously intractable issues such as race conditions, memory corruption, and timing-dependent failures. Most importantly, quantifiable benefits in development time and system reliability are directly translated into higher safety assurance and more efficient certification processes, ultimately creating the trust that autonomous vehicle deployment is predicated upon. As autonomous systems begin to play more significant roles in transportation, recalcitrant crash infrastructure must be considered not as a diagnostic tool but as an integral part of safety-critical system design.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

References

[1] Kuan Ting Chen et al., "Where Failures May Occur in Automated Driving: A Fault Tree Analysis Approach," ResearchGate, August 2022. https://www.researchgate.net/publication/362539455_Where_Failures_May_Occur_in_Automated_Driving_A_Fault_Tree_Analysis_Approach

[2] Jürgen Dobaj et al., "Cybersecurity Threat Analysis, Risk Assessment, and Design Patterns for Automotive Networked Embedded Systems: A Case Study," ResearchGate, August 2021. [Online]. Available: https://www.researchgate.net/publication/354192225_Cybersecurity_Threat_Analysis_Risk_Assessment_and_Design_Patterns_for_Automotive_Networked_Embedded_Systems_A_Case_Study

[3] Weining Gu et al., "Characterization of Linux kernel behavior under errors," ResearchGate, July 2003, Available: https://www.researchgate.net/publication/4021632_Characterization_of_Linux_kernel_behavior_under_errors

[4] Francesco Tusa et al., "Microservices and serverless functions—lifecycle, performance, and resource utilisation of edge-based real-time IoT analytics," Science Direct, June 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X24000529

[5] Ryan Aalund & Vincent Philip Paglioni, "Enhancing Reliability in Embedded Systems Hardware: A Literature Survey," ResearchGate, 2022. [Online]. Available: https://www.researchgate.net/publication/388377384_Enhancing_Reliability_in_Embedded_Systems_Hardware_A_Literature_Survey/fulltext/6794b3368311ce680c350f3f/Enhancing-Reliability-in-Embedded-Systems-Hardware-A-Literature-Survey.pdf

[6] Hengrui Chen et al., "Exploring the Mechanism of Crashes with Autonomous Vehicles Using Machine Learning," ResearchGate, February 2021. [Online]. Available: https://www.researchgate.net/publication/349673848_Exploring_the_Mechanism_of_Crashes_with_Autonomous_Vehicles_Using_Machine_Learning

[7] Slim Dhouibi et al., "Safety Driven Optimization Approach for Automotive Systems," ResearchGate, January 2015. [Online]. Available: https://www.researchgate.net/publication/278807521_Safety_Driven_Optimization_Approach_for_Automotive_Systems

[8] Sedat Sonko et al., "A comprehensive review of embedded systems in autonomous vehicles: Trends, challenges, and future directions," ResearchGate, January 2024. [Online]. Available: https://www.researchgate.net/publication/377807646_A_comprehensive_review_of_embedded_systems_in_autonomous_vehicles_Trends_challenges_and_future_directions

[9] Jonas Freiderich & Sanja Lazarova-Molnar, "Reliability assessment of manufacturing systems: A comprehensive overview, challenges, and opportunities," ResearchGate, February 2024. [Online]. Available: https://www.researchgate.net/publication/377383322_Reliability_assessment_of_manufacturing_systems_A_comprehensive_overview_challenges_and_opportunities

[10] Rakesh Rana et al., "Evaluation of Standard Reliability Growth Models in the Context of Automotive Software Systems," ResearchGate, June 2013. [Online]. Available: https://www.researchgate.net/publication/274079915_Evaluation_of_Standard_Reliability_Growth_Models_in_the_Context_of_Automotive_Software_Systems