



Memory Tracking in Forensic Checkpointing: What Soft-Dirty bits can't tell us?

Shailja Shaktawat



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

Introduction

- CRIU's existing incremental checkpointing, based on Linux soft dirty bit, was originally designed for use cases such as live migration
- Forensics has different requirements altogether
- For live migration we need only a correct final memory state
- For forensics we need the path and context at each stage of that path, leading to the next memory state



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Where do we still fall short?

Existing soft-dirty answers one question: did this page change since the last state? It cannot say when, how many times, in what order, or by whom.

Stoyanov et al., DFRWS 2026, discusses CRIU's incremental checkpointing for forensic container snapshot chains. Soft-dirty tracking cuts snapshot size by an order of magnitude and makes high-frequency capture practical.



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

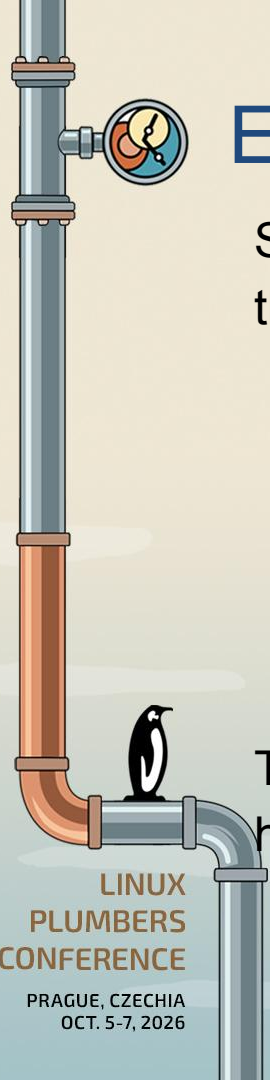


How soft-dirty works in CRIU?

1. CRIU clears the soft-dirty bit. Those pages are write-protected.
2. The first write faults. The kernel sets the bit and makes the page writable.
3. Later writes in that interval do not fault.
4. At the next checkpoint, CRIU asks which pages have the bit set.

On newer kernels, `PAGEMAP_SCAN` gives CRIU a more efficient interface for querying that state.





Example

Suppose checkpointing runs every 3 seconds and the attacker knows the interval:

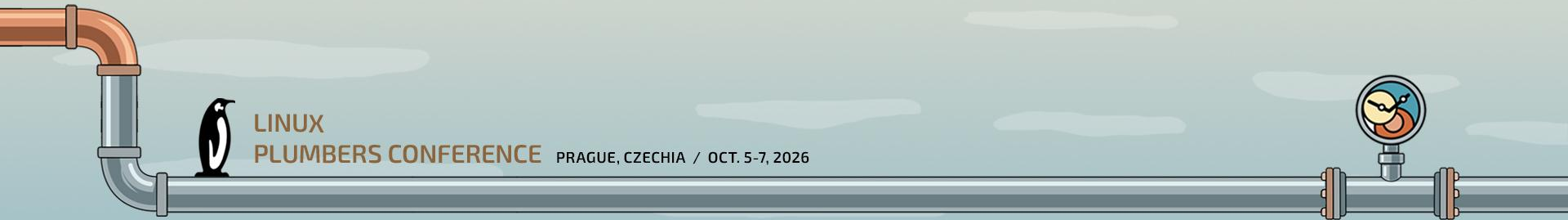
- 0.0s Checkpoint. Page X contains abc123.
- 1.5s Process writes xyz123. The bit is set.
- 2.0s Attacker writes abc123 back. The bit stays set.
- 3.0s Next checkpoint copies page X. The bytes match the previous checkpoint.

The next checkpoint can tell us that page X was written, but not what happened between the checkpoints.

It only gives us the final state, we lose the order, the context and the intermediate values.

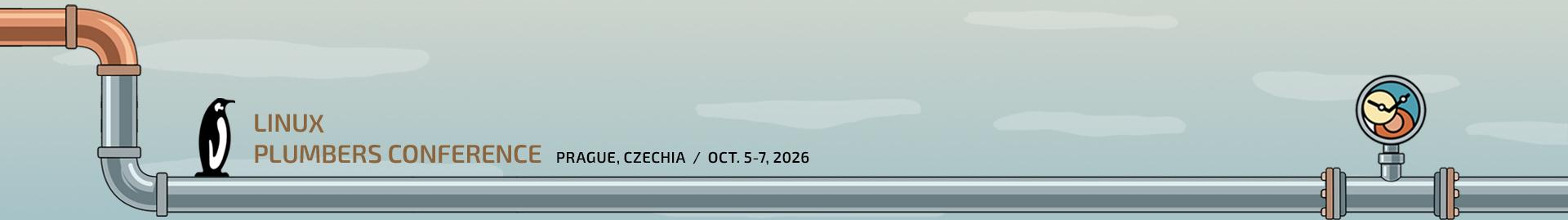
Issue 1: Context at the first-write

- Current memory tracking makes pages read-only and catches the first write to each.
- At the write-protection fault, the kernel has the faulting task context, fault address, and the page contents before the write completes.
- Today, that information is not retained as forensic history.



Issue 2: Indistinguishability in intermediate states

- A page written once and a page written multiple times in between a single checkpoint interval are indistinguishable.
- Every intermediate value is gone, along with any ordering or context.
- High frequency checkpointing narrows the gap, but still loses writes inside an interval.

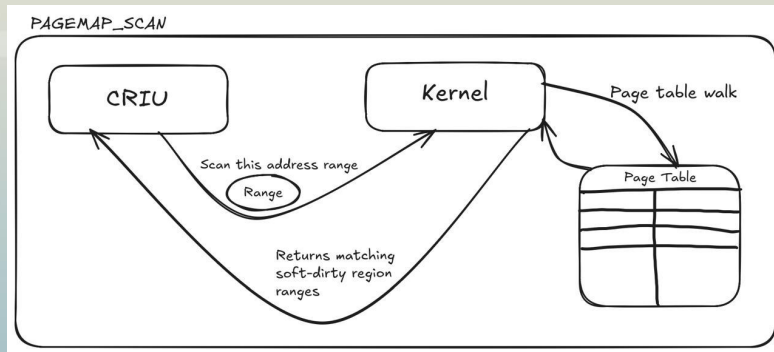




Prior work & Solutions

1. PAGEMAP_SCAN

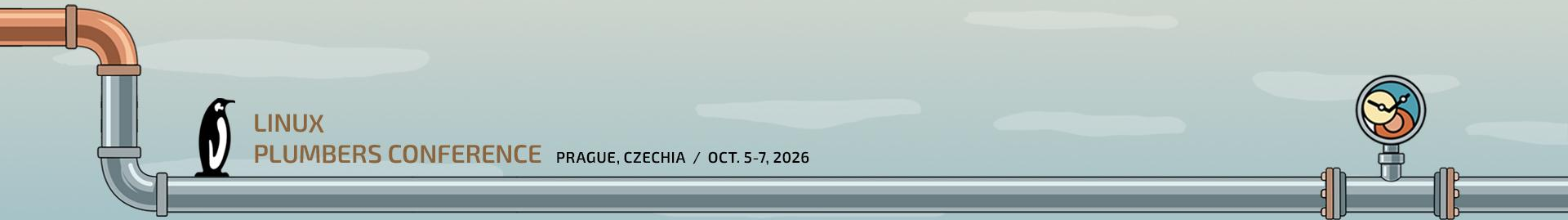
- CRIU reads PAGEMAP_SCAN for soft-dirty pages.
- The kernel walks the page tables once and returns those pages as address ranges.
- The tracking mechanism is unchanged. Each page still has one soft-dirty bit, and the kernel does not store the writes themselves.
- It doesn't preserve the history of what happened between scans.

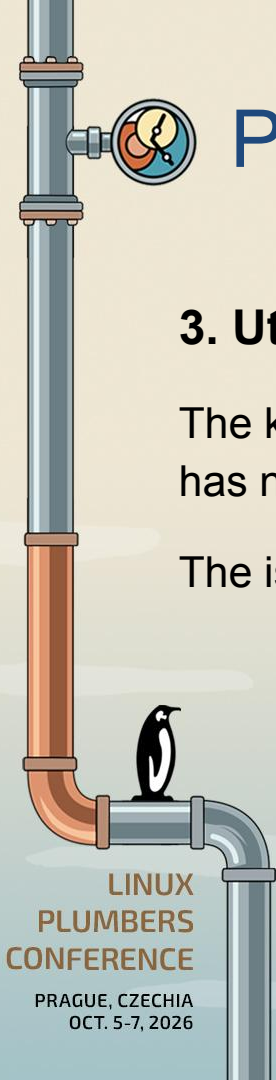


Prior work & Solutions

2. Why can't we just let hardware mark pages dirty ?

- CPUs such as x86 PTEs already carry hardware Accessed and Dirty bits that the MMU sets with no fault.
- We still cannot use those bits though, because clearing them for tracking interferes with writeback, reclaim, and swap.
- Intel PML is the closest hardware dirty journal, but it is for VMs (EPT). It also logs each page once per interval, still it doesn't log time, context or any old contents.





Prior work & Solutions

3. Utilise the page fault we are already taking

The kernel knows the context and still has the old page contents, because the write has not finished yet. Why not capture this?

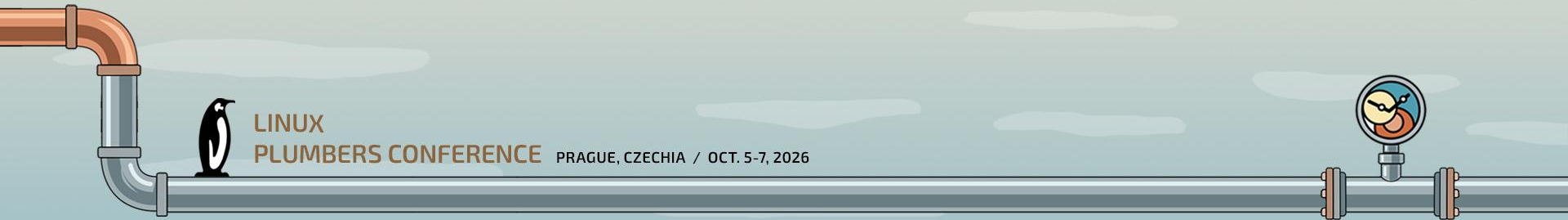
The issue is with existing interfaces:

- Async uffd-wp: the kernel resolves the fault itself and generates no message, so nothing can be captured from userspace
- Sync uffd-wp: a message is delivered and it reports the thread id, but each fault goes for userspace round trip per fault

Prior work & Solutions

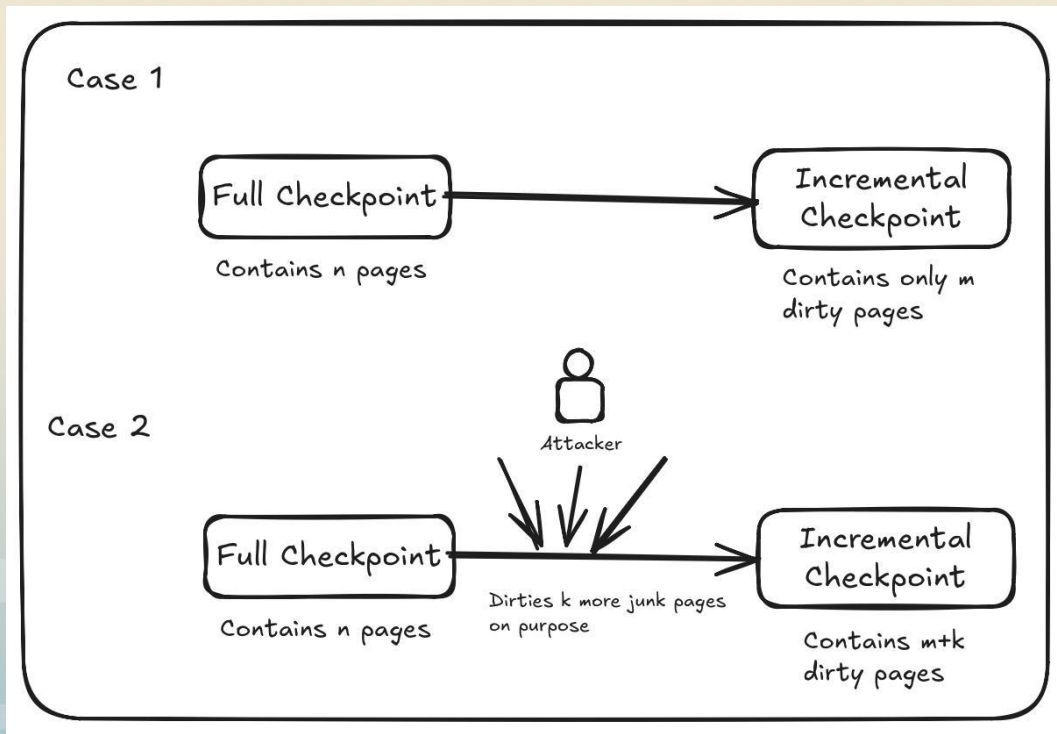
3. Utilise the page fault we are already taking(contd.)

- Today, we face a tradeoff between the two.
- Forensics need a rich yet cheap to obtain context which captures at fault, if possible, without the userspace round trip.
- A lighter option worth discussing separately: on the first-write fault, also store a generation number. Same cost because that fault already happens. The number gives the order of those first writes, without the old page contents.

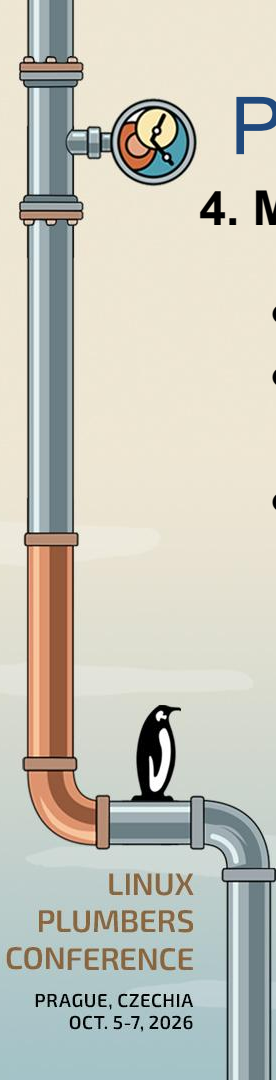


Prior work & Solutions

4. Memory Inflation



This becomes a DoS on the forensic pipeline, inflating snapshot size, longer freeze time, and more storage.

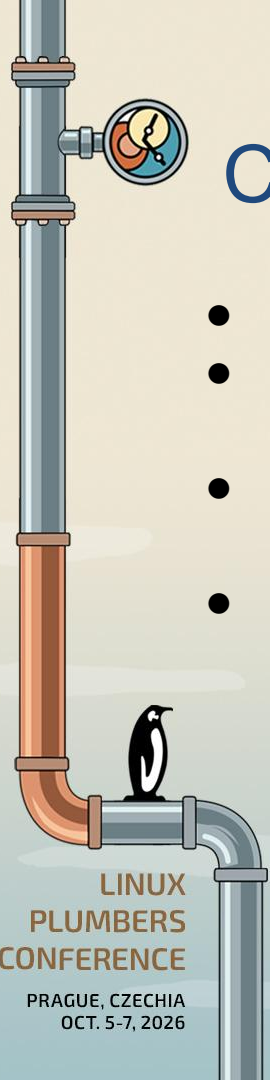


Prior work & Solutions

4. Memory Inflation (contd.)

- The scan already tells us $m+k$ before we copy anything.
- Migration would slow the process down. That changes what we are watching.
- When $m+k$ is huge, the count and the address ranges are cheap to keep. The bytes are not. Which bytes are worth copying?

Dirty-rate measurement already exists in migration: KVM's dirty ring, PML's log-full VM exit, and QEMU's dirty-rate calculation. What we lack is treating a sudden spike as evidence.

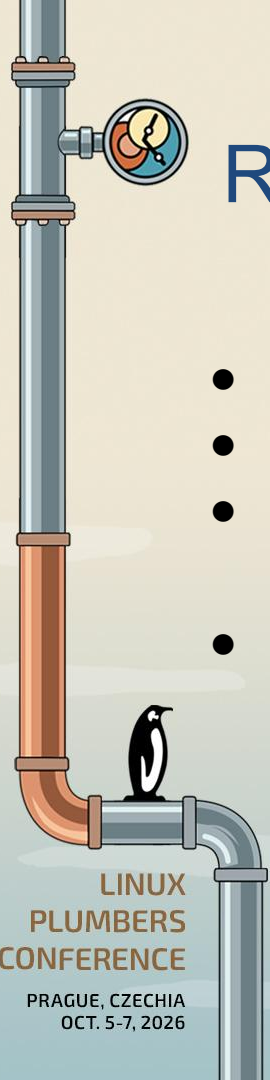


Conclusion

- Soft Dirty works for the problems for which it was designed to solve
- Live Migration and Forensic both ask a different question and hence require different memory tracking mechanisms
- Intention is to ask the right questions to discuss and develop tracking mechanisms for forensics
- Few questions as takeaways:
 1. At the existing kernel-side event, are we throwing away useful forensic information that we could potentially retain?
 2. How much context is enough to retain?
 3. Where should the tracking mechanism live?

LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026



References

- [CRIU Issue #2604](#)
- [LPC 2023 talk](#)
- [Stoyanov et al., Forensic Analysis of Container Snapshot Chains for Post-Event Reconstruction, DFRWS USA 2026](#)
- [CRIU: Memory changes tracking](#)

LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026