CVE-2021-29939

修复前：

https://github.com/Alexhuszagh/rust-stackvector/blob/d0382d5ef903fc96bdcc08c02e36e6dd2eda11a5/src/lib.rs

```
impl<A: Array> Extend<A::Item> for StackVec<A> {
    fn extend<I: iter::IntoIterator<Item=A::Item>>(&mut self, iterable: I) {
        let mut iter = iterable.into_iter();
        let (lower_bound, upper_bound) = iter.size_hint();
        let upper_bound = upper_bound.expect("iterable must provide upper bound.");
        assert!(self.len() + upper_bound <= self.capacity());

        unsafe {
            let len = self.len();
            let ptr = self.as_mut_ptr().padd(len);
            let mut count = 0;
            while count < lower_bound {
                if let Some(out) = iter.next() {
                    ptr::write(ptr.padd(count), out);
                    count += 1;
                } else {
                    break;
                }
            }
            self.set_len(len + count);
        }

        for elem in iter {
            self.push(elem);
        }
    }
}
```

修复后：

https://github.com/Alexhuszagh/rust-stackvector/blob/f45657d5a823a67bb3f5cffee65efbb401a44192/src/lib.rs

```
impl<A: Array> Extend<A::Item> for StackVec<A> {
    fn extend<I: iter::IntoIterator<Item=A::Item>>(&mut self, iterable: I) {
        // size_hint() has no safety guarantees, and TrustedLen
        // is nightly only, so we can't do any optimizations with
        // size_hint.
        for elem in iterable.into_iter() {
            self.push(elem);
        }
    }
}
```

RUSTSEC-2019-0009 与 RUSTSEC-2019-0012

修复前：

https://github.com/servo/rust-smallvec/blob/19de50108d403efaa7cd979eac3bb97a4432fd4b/lib.rs#L651-L667

```
/// Re-allocate to set the capacity to `max(new_cap, inline_size())`.
///
/// Panics if `new_cap` is less than the vector's length.
pub fn grow(&mut self, new_cap: usize) {
    unsafe {
        let (ptr, &mut len, cap) = self.triple_mut();
        let unspilled = !self.spilled();
        assert!(new_cap >= len);
        if new_cap <= self.inline_size() {
            if unspilled {
                return;
            }
            self.data = SmallVecData::from_inline(mem::uninitialized());
            ptr::copy_nonoverlapping(ptr, self.data.inline_mut().ptr_mut(), len);
        } else if new_cap != cap {
            let mut vec = Vec::with_capacity(new_cap);
            let new_alloc = vec.as_mut_ptr();
            mem::forget(vec);
            ptr::copy_nonoverlapping(ptr, new_alloc, len);
            self.data = SmallVecData::from_heap(new_alloc, len);
            self.capacity = new_cap;
            if unspilled {
                return;
            }
        }
        deallocate(ptr, cap);
    }
}
```

修复后：

https://github.com/servo/rust-smallvec/blob/v0.6.10/lib.rs

```rust
/// Re-allocate to set the capacity to `max(new_cap, inline_size())`.
///
/// Panics if `new_cap` is less than the vector's length.
pub fn grow(&mut self, new_cap: usize) {
    unsafe {
        let (ptr, &mut len, cap) = self.triple_mut();
        let unspilled = !self.spilled();
        assert!(new_cap >= len);
        if new_cap <= self.inline_size() {
            if unspilled {
                return;
            }
            self.data = SmallVecData::from_inline(mem::uninitialized());
            ptr::copy_nonoverlapping(ptr, self.data.inline_mut().ptr_mut(), len);
            self.capacity = len;
        } else if new_cap != cap {
            let mut vec = Vec::with_capacity(new_cap);
            let new_alloc = vec.as_mut_ptr();
            mem::forget(vec);
            ptr::copy_nonoverlapping(ptr, new_alloc, len);
            self.data = SmallVecData::from_heap(new_alloc, len);
            self.capacity = new_cap;
            if unspilled {
                return;
            }
        } else {
            return;
        }
        deallocate(ptr, cap);
    }
}
```

注意：在之后的几个版本又对 grow 函数做了一些优化，来去除 unsafe 或者使用///SAFETY 来解释 unsafe 的安全性，最新版本如下：
https://github.com/servo/rust-smallvec/blob/v2/src/lib.rs

RUSTSEC-2024-0359(典型的为了优化性能而引入的漏洞，目前程序扫描不出，因为指令数小于 5)
https://github.com/Byron/gitoxide/commit/7a98d8a518d771fe09614878d86d970ee1186b35
修复前:

```
/// Lifecycle
impl<'a> ValueRef<'a> {
    /// Keep `input` as our value.
    pub fn from_bytes(input: &'a [u8]) -> Self {
        Self(KStringRef::from_ref(
            // SAFETY: our API makes accessing that value as `str` impossible, so illformed UTF8 is never exposed as such.
            #[allow(unsafe_code)]
            unsafe {
                std::str::from_utf8_unchecked(input)
            },
        ))
    }
}
```

修复后:

```
/// Lifecycle
impl<'a> ValueRef<'a> {
    /// Keep `input` as our value.
    pub fn from_bytes(input: &'a [u8]) -> Self {
        Self(input)
    }
}
```

RUSTSEC-2024-0357(可能不合适，不确定是否必须通过 unsafe 实现)

https://github.com/sfackler/rust-openssl/pull/2266/commits/142deef717bad843fc04c5afb925bfd9e7dc4305

修复前:

```
    pub fn get_buf(&self) -> &[u8] {
        unsafe {
            let mut ptr = ptr::null_mut();
            let len = ffi::BIO_get_mem_data(self.0, &mut ptr);
            slice::from_raw_parts(ptr as *const _ as *const _, len as usize)
        }
    }
```

修复后:

```rust
pub fn get_buf(&self) -> &[u8] {
    unsafe {
        let mut ptr = ptr::null_mut();
        let len = ffi::BIO_get_mem_data(self.0, &mut ptr);
        if len == 0 {
            &[]
        } else {
            slice::from_raw_parts(ptr as *const _ as *const _, len as usize)
        }
    }
}
```