RUSTSEC-2021-0019

Multiple soundness issues

Reported	February 4, 2021
Issued	February 4, 2021 (last modified: March 6, 2022)
Package	xcb (crates.io)
Type	Vulnerability
Categories	memory-corruption memory-exposure
Aliases	CVE-2021-26955 CVE-2021-26956 CVE-2021-26957 CVE-2021-26958
Details	https://github.com/RustSec/advisory-db/issues/653
Patched	`>=1.0`

Description

Calls `std::str::from_utf8_unchecked()` without any checks

The function `xcb::xproto::GetAtomNameReply::name()` calls `std::str::from_utf8_unchecked()` on the raw bytes that were received from the X11 server without any validity checks. The X11 server only prevents interior null bytes, but otherwise allows any X11 client to create an atom for arbitrary bytes.

This issue is tracked here: https://github.com/rust-x-bindings/rust-xcb/issues/96

`xcb::xproto::GetPropertyReply::value()` allows arbitrary return types

The function `xcb::xproto::GetPropertyReply::value()` returns a slice of type `T` where `T` is an unconstrained type parameter. The raw bytes received from the X11 server are interpreted as the requested type.

The users of the `xcb` crate are advised to only call this function with the intended types. These are `u8`, `u16`, and `u32`.

This issue is tracked here: https://github.com/rust-x-bindings/rust-xcb/issues/95

Out of bounds read in `xcb::xproto::change_property()`

`xcb::xproto::change_property` has (among others) the arguments `format: u8` and `data: &[T]`. The intended use is one of the following cases:

- `format = 8` and `T = u8`
- `format = 16` and `T = u16`
- `format = 32` and `T = u32` However, this constraint is not enforced. For example, it is possible to call the function with `format = 32` and `T = u8`. In this case, a read beyond the end of the `data` slice is performed and the bytes are sent to the X11 server.

The users of the `xcb` crate are advised to only call this function with one of the intended argument combinations.

This issue is tracked here: https://github.com/rust-x-bindings/rust-xcb/issues/94

'Safe' wrapper around `std::mem::transmute()`

The function `xcb::base::cast_event()` takes a reference to a `xcb::base::GenericEvent` and returns a reference to an arbitrary type, as requested by the caller (or found via type interference). The function is implemented as a direct call to `std::mem::transmute()`. Since the return type is not constrained, this allows transmution to an incorrect type or a type that is larger than the X11 event that was passed in.

X11 events are mostly always 32 bytes large and this function works as intended.

Users are advised to only cast to the event structs provided by the `xcb` crate (and hope for the best).

This issue is tracked here: https://github.com/rust-x-bindings/rust-xcb/issues/78