`filepath-securejoin`

An implementation of `SecureJoin`, a candidate for inclusion in the Go standard library. The purpose of this function is to be a "secure" alternative to `filepath.Join`, and in particular it provides certain guarantees that are not provided by `filepath.Join`.

> ***NOTE**: This code is only safe if you are not at risk of other processes modifying path components after you've used `SecureJoin`. If it is possible for a malicious process to modify path components of the resolved path, then you will be vulnerable to some fairly trivial TOCTOU race conditions. There are some Linux kernel patches I'm working on which might allow for a better solution.*
>
> *In addition, with a slightly modified API it might be possible to use `O_PATH` and verify that the opened path is actually the resolved one -- but I have not done that yet. I might add it in the future as a helper function to help users verify the path (we can't just return `/proc/self/fd/<foo>` because that doesn't always work transparently for all users).*

This is the function prototype:

```
func SecureJoin(root, unsafePath string) (string, error)
```

This library **guarantees** the following:

- If no error is set, the resulting string **must** be a child path of `root` and will not contain any symlink path components (they will all be expanded).
- When expanding symlinks, all symlink path components **must** be resolved relative to the provided root. In particular, this can be considered a userspace implementation of how `chroot(2)` operates on file paths. Note that these symlinks will **not** be expanded lexically (`filepath.Clean` is not called on the input before processing).
- Non-existent path components are unaffected by `SecureJoin` (similar to `filepath.EvalSymlinks`'s semantics).
- The returned path will always be `filepath.Clean`ed and thus not contain any `..` components.

A (trivial) implementation of this function on GNU/Linux systems could be done with the following (note that this requires root privileges and is far more opaque than the implementation in this library, and also requires that `readlink` is inside the `root` path):

```
package securejoin

import (
    "os/exec"
    "path/filepath"
)

func SecureJoin(root, unsafePath string) (string, error) {
    unsafePath = string(filepath.Separator) + unsafePath
```

```
    cmd := exec.Command("chroot", root,
        "readlink", "--canonicalize-missing", "--no-newline", unsafePath)
    output, err := cmd.CombinedOutput()
    if err != nil {
        return "", err
    }
    expanded := string(output)
    return filepath.Join(root, expanded), nil
}
```

License

The license of this project is the same as Go, which is a BSD 3-clause license available in the `LICENSE` file.