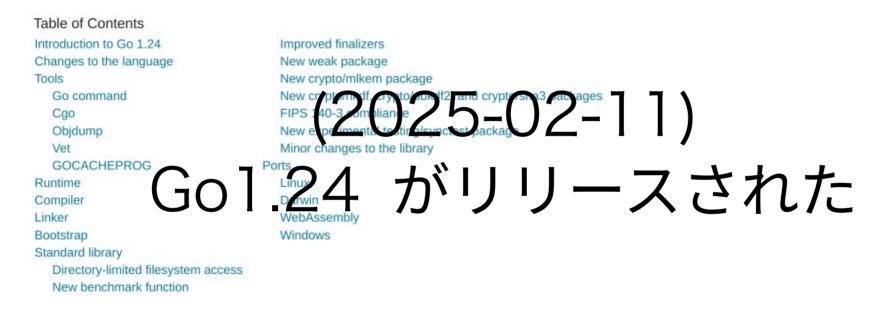
Go 1.24

os.Root はやわかり

Go 1.24 Release Notes



Introduction to Go 1.24

The latest Go release, version 1.24, arrives six months after Go 1.23. Most of its changes are in the implementation of the toolchain, runtime, and libraries. As always, the release maintains the Go 1 promise of compatibility. We expect almost all Go programs to continue to compile and run as before.

Changes to the language



1 つ手頃そうなのにフォーカスして 調べることにした

それは os.Root(型)...

https://go.dev/doc/go1.24

Standard library

Directory-limited filesystem access

The new os . Root type provides the ability to perform filesystem operations within a specific directory.

The os.OpenRoot function opens a directory and returns an os.Root. Methods on os.Root operate within the directory and do not permit paths that refer to locations outside the directory, including ones that follow symbolic links out of the directory. The methods on os.Root mirror most of the file system operations available in the os package, including for example os.Root.Open, os.Root.Create, os.Root.Mkdir, and os.Root.Stat,

標準ライブラリ

ディレクトリ制限付きファイルシステムアクセス

新しいos.Rootタイプは、特定のディレクトリ内でファイルシステム操作を実行する機能を提供します。

関数os.OpenRootはディレクトリを開き、を返しますos.Root。 のメソッドはos.Rootディレクトリ内で動作し、ディレクトリ外の場所を参照するパス(ディレクトリ外のシンボリック リンクをたどるパスを含む)は許可しません。 のメソッドは、 、 など、パッケージos.Rootで使用可能なほとんどのファイル システム操作を反映します os。os.Root.Openos.Root.Createos.Root.Mkdiros.Root.Stat

ドキュメント

type Root added in go1.24.0

```
type Root struct {
    // contains filtered or unexported fields
}
```

Root may be used to only access files within a single directory tree.

Methods on Root can only access files and directories beneath a root directory. If any component of a file name passed to a method of Root references a location outside the root, the method returns an error. File names may reference the directory itself (.).

Methods on Root will follow symbolic links, but symbolic links may not reference a location outside the root. Symbolic links must not be absolute.

Methods on Root do not prohibit traversal of filesystem boundaries, Linux bind mounts, /proc special files, or access to Unix device files.

Methods on Root are safe to be used from multiple goroutines simultaneously.

On most platforms, creating a Root opens a file descriptor or handle referencing the directory. If the directory is moved, methods on Root reference the original directory in its new location.

Root's behavior differs on some platforms:

- · When GOOS=windows, file names may not reference Windows reserved device names such as NUL and COM1.
- When GOOS=js, Root is vulnerable to TOCTOU (time-of-check-time-of-use) attacks in symlink validation, and cannot ensure that operations will not escape the
 root.
- When GOOS=plan9 or GOOS=js, Root does not track directories across renames. On these platforms, a Root references a directory name, not a file descriptor.

こんな感じに使う

```
root, _ := os.OpenRoot("a")
_ = root.Mkdir("./b", 0o777) // a/b に作られる
```

- os.OpenRoot() を使って root.Root インスタンスを取得 する
- インスタンスに対してファイル操作とかができる。 os.OpenRoot() の指定パスが起点になる

従来

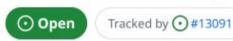
今までは直接扱う方法しかなかった(はず)

```
_ = os.Mkdir("example_dir", 00777)
_ = os.Mkdir("example_dir/aaa", 00777)
```

なぜ必要なのか?

• #67002 が提案 Issue (by Git 履歴)

os: safer file open functions #67002





Please see the updated proposal in #67002 (comment)

Directory traversal vulnerabilities are a common class of vulnerability, in which an attacker tricks a program into opening a file that it did not intend. These attacks often take the form of providing a relative pathname such as

"../../etc/passwd", which results in access outside an intended location. CVE-2024-3400 is a recent, real-world example of directory traversal leading to an actively exploited remote code execution vulnerability.

A related, but less commonly exploited, class of vulnerability involves unintended symlink traversal, in which an attacker creates a symbolic link in the filesystem and manipulates the target into following it.

Assig

No or

Labe

Pro

Type

No ty

Proje

脆弱性対策のため

Issue によると ...

- ディレクトリトラバーサル:相対パスで想定してないディレクトリが曝露する脆弱性
- シンボリックリンクトラバーサル:シンボリックリンクで想定しないディレクトリが曝露する脆弱性

を防ぎやすくするため

動かして試してみた

• 上の階層にはいけない

絶対パスは指定できない。

```
func TestRoot(t *testing.T) {
                         root, err := os.OpenRoot(".")
                         assert.NoError(t. err)
                         defer root.Close()
                >>
                         t.Run("絶対パスは失敗する", func(t *testing.T) {
                >>
                                  err = root.Mkdir("/abs", 00777)
                                  assert.Error(t, err)
                                  assert.Equal(t, "mkdirat /abs: path escapes from parent", err.Error())
                          })
func TestRoot(t *testing.T) {
       root, err := os.OpenRoot(".")
       assert.NoError(t, err)
       defer root.Close()
       t.Run("絶対パスは失敗する", func(t *testing.T) {
              // 配置してるパス
              err = root.Mkdir("/home/violet/go/src/playground/at-2025-02-14-002757/abs", 00777)
              assert.Error(t, err)
              assert.Equal(t, "mkdirat /home/violet/go/src/playground/at-2025-02-14-002757/abs: path e

'scapes from parent", err.Error())
```

• 先頭が / だとエラーを返すようになっている

https://github.com/golang/go/blob/215de81513286c010951624243c2923f7dc79675/src/os/root.go#L194-L196

シンボリックリンクを辿れる

```
func TestRoot(t *testing.T) {
      root, err := os.OpenRoot(".")
      assert.NoError(t. err)
      defer root.Close()
      t.Run("シンボリックリンク先が下層にあると成功する", func(t *testing.T) {
            err = root.Mkdir("./down sym/success", 00777)
            assert.NoError(t. err)
            defer func() {
                   err = os.RemoveAll("./down sym/success")
                   assert.NoError(t, err)
            }()
      t.Run("シンボリックリンク先が上層にあると成功する", func(t *testing.T) {
            err = root.Mkdir("./up sym/success", 00777)
            assert.NoError(t. err)
                                                    ← 上の階層へのシンボリックリンクは辿れ
            defer func() {
                   err = os.RemoveAll("./up sym/success")
                                                    てしまう。
                   assert.NoError(t, err)
            }()
                                                     ドキュメントによるとバインドマウントも
                                                    制限されない
```

os.Root まとめ

- 脆弱性対策になる
- ディレクトリ下でまとめて何かしたいとき便利 そう