

Jan 03, 19 13:15

secretsfs.go

Page 1/3

```
// config contains the config information about secretsfs.
// it contains the default configuration, so that it can be accessed and set
// without any worries.
package config

import (
    "fmt"
    "bytes"
    "strings"

    "github.com/spf13/viper"
)

// configDefaults contains the default configurations.
// Those will be set on startup, if not overwritten via environment variables
// or a userdefined configurationfile.
var configDefaults = []byte(`
---
### GENERAL
# CONFIG_PATHS:
# - /etc/secretsfs/
# - $HOME/.secretsfs
# CONFIG_FILE: secretsfs # without file type

# BACKGROUND MODE
PIDFILENAME: /var/run/secretsfs.pid
PIDFILEPERM: 0640
LOGFILENAME: /var/log/secretsfs.log
LOGFILEPERM: 0640
WORKDIR: "./"
UMASK: 027

### FIO
ENABLED_FIOS:
- secretsfiles
- templatefiles

# templatefiles
PATH_TO_TEMPLATES: /etc/secretsfs/templates/

### STORE
CURRENT_STORE: Vault

# vault
FILE_ROLEID: .vault-roleid
VAULT_ADDR: http://127.0.0.1:8200
# taken from https://www.vaultproject.io/api/secret/kv/kv-v2.html
MTDATA: secret/metadata/
DTDATA: secret/data/

# fuse does not allow the character '/' inside of names of directories or files
# in vault k=v pairs of one secret will be shown as files, where k is the name
# of the file and v the value. k may also include names with a '/'.
# Those slashes will be substituted with the following character
# may also use some special characters, e.g. 'Â§' or 'Â°'
subst_char: _

`)

// InitConfig reads all configurations and sets them.
// Order is (first match counts):
```

Jan 03, 19 13:15

secretsfs.go

Page

```
// 1. Environment variables
// 2. Configurationfile $HOME/.secretsfs/secretsfs.yaml
// 3. Configurationfile provided by environment variable SFS_CONFIG_FILE
// 4. Configurationfile /etc/secretsfs/secretsfs.yaml
// 5. Hardcoded configurations from variable configDefaults
// This function is executed in init().
//
// https://github.com/spf13/viper#reading-config-files
func InitConfig() {
    // read defaults first
    viper.SetConfigType("yaml")
    viper.ReadConfig(bytes.NewBuffer(configDefaults))

    // read automatically all envs with Prefix SFS_
    viper.SetEnvPrefix("SFS")
    viper.SetEnvKeyReplacer(strings.NewReplacer(".", "_"))
    viper.AutomaticEnv()

    // also read vault addr env
    // needs both parameters for BindEnv, else prefix would be prefixed
    viper.BindEnv("VAULT_ADDR", "VAULT_ADDR")

    // read config file specific things first and overwrite if necessary
    viper.SetConfigName("secretsfs")
    viper.AddConfigPath("$HOME/.secretsfs") // call multiple times to add
    // any search paths
    if viper.IsSet("CONFIG_FILE") {
        viper.SetConfigName(viper.GetString("CONFIG_FILE"))
    }

    // add config paths of ENV var first so it overwrites any other config
    // TODO: check, whether it really works like this
    viper.AddConfigPath("/etc/secretsfs/")
    if viper.IsSet("CONFIG_PATHS") {
        paths := viper.GetStringSlice("CONFIG_PATHS")
        for _, p := range paths {
            viper.AddConfigPath(p)
        }
    }

    // read configuration from config files
    err := viper.MergeInConfig() // Find and read the config files
    if err != nil && !strings.Contains(err.Error(), "Config File") && !strings.Contains(err.Error(), "Not Found in") { // Handle errors reading the config file
        panic(fmt.Errorf("%s\n", err))
    }

    // GetConfigDefaults returns the Contents of configDefaults as *[]byte.
    // If you need string, you can also call GetStringConfigDefaults().
    func GetConfigDefaults() *[]byte {
        return &configDefaults
    }

    // GetStringConfigDefaults returns the Contents of configDefaults converted to string.
    func GetStringConfigDefaults() string {
        return string(configDefaults)
    }
}
```

Jan 03, 19 13:15

**secretsfs.go**

Page 3/3

```
func init() {  
    InitConfig()  
}
```

```

Jan 03, 19 13:26      main.go      Page 1/3

// SecretsFS - Access Your Secrets Comfortably and Safely

package main

import (
    "flag"
    "fmt"
    "log"
    "os"
    "strings"

    "github.com/hanwen/go-fuse/fuse"
    "github.com/hanwen/go-fuse/fuse/nodefs"
    "github.com/hanwen/go-fuse/fuse/pathfs"
    "github.com/sevlyar/go-daemon"
    "github.com/spf13/viper"

    "github.com/muryoutaisuu/secretsfs/cmd/secretsfs/config"
    "github.com/muryoutaisuu/secretsfs/pkg/fio"
    "github.com/muryoutaisuu/secretsfs/pkg/store"
    "github.com/muryoutaisuu/secretsfs/pkg/secretsfs"
)

func main() {
    // parse arguments & flags
    flag.Usage = usage
    var opts = flag.String("o","", "Options passed through to fuse")
    var currentstore = flag.Bool("print-store", false, "prints currently set
store")
    var defaults = flag.Bool("print-defaults", false, "prints default config
urations")
    var stores = flag.Bool("print-stores", false, "prints available stores")
    var fios = flag.Bool("print-fios", false, "prints available FIOs")
    var foreground = flag.Bool("foreground", false, "run in foreground")

    firstdashed := firstDashedArg(os.Args)
    flag.CommandLine.Parse(os.Args[firstdashed:])

    // print default configs, -print-defaults
    if *defaults {
        fmt.Printf("Default Configs: \n%s", config.GetStringConfigDefault
s())
        os.Exit(0)
    }

    // print available stores, -print-stores
    if *stores {
        fmt.Printf("Available Stores are: %v\n", store.GetStores())
        os.Exit(0)
    }

    // prints available fios, -print-fios
    if *fios {
        maps := fio.FIOMaps()
        list := make([]string, 0)
        for k := range maps {
            list = append(list, k)
        }
        fmt.Printf("Available FIOs are: %v\n", list)
        os.Exit(0)
    }
}

```

Wednesday January 16, 2019

./cmd/secretsfs/main.go

```

Jan 03, 19 13:26      main.go      Page

// print currently set store
if *currentstore {
    fmt.Printf("Currently set store is: %s\n", store.GetStore())
}

os.Exit(0)

log.Printf("Call is: %s\n", os.Args)
// print usage if no arguments were provided
if len(os.Args) < 2 {
    usage()
    os.Exit(2)
}

mountpoint := os.Args[1]
log.Println("mountpoint is: "+mountpoint)

// create the filesystem object
sfs,err := secretsfs.NewSecretsFS(pathfs.NewDefaultFileSystem(), fi
Maps(), store.GetStore())
if err != nil {
    log.Fatalf(err)
}

pathnfs := pathfs.NewPathNodeFs(sfs, nil)

fsc := nodefs.NewFileSystemConnector(pathnfs.Root(), nodefs.NewOpti
) // FileSystemConnector

// set options
fsOpts := fuse.MountOptions{}
log.Println(*opts)

// https://github.com/sevlyar/go-daemon/blob/master/examples/cmd/gd
le/simple.go

if *foreground {
    fsOpts.Options = strings.Split(*opts, ",")

    // create server
    server, err := fuse.NewServer(fsc.RawFS(), mountpoint, &fsOp
    if err != nil {
        log.Printf("Mountfail: %v\n", err)
        os.Exit(1)
    }

    // mount and now serve me till the end!!!
    server.Serve()
    defer server.Unmount()
} else {
    //newargs := append(os.Args, "-foreground")
    log.Println("logFileName is: ", viper.GetString("logFileName
cntxt := &daemon.Context{
        PidFileName: viper.GetString("PIDFILENAME"),
        PidFilePerm: os.FileMode(viper.GetInt("PIDFILEPERM"),
        LogFileName: viper.GetString("LOGFILENAME"),
        LogFilePerm: os.FileMode(viper.GetInt("LOGFILEPERM"),
        WorkDir:    viper.GetString("WORKDIR"),
        Umask:       viper.GetInt("UMASK"),
        Args:        append(os.Args, "-foreground"),
    }

    d, err := cntxt.Reborn()
    if err != nil {

```

Jan 03, 19 13:26

main.go

Page 3/3

```
        log.Fatal("Unable to run: ", err)
    }
    if d != nil {
        return
    }
    defer cntxt.Release()
}

return

}

// print usage of this tool
func usage() {
    fmt.Fprintf(os.Stderr, "Usage of %s:\n", os.Args[0])
    fmt.Fprintf(os.Stderr, "  %s MOUNTPPOINT\n", os.Args[0])
    flag.PrintDefaults()
}

// firstDashedArg returns the index of the first dashed argument, e.g. -ex
// https://stackoverflow.com/a/51526473/4069534
func firstDashedArg(args []string) int {
    for i := 1; i < len(args); i++ {
        if len(args[i]) > 0 && args[i][0] == '-' {
            return i
        }
    }
    return 1
}
```

Dec 17, 18 18:35

fio.go

Page 1/1

```
// FUSE Input/Output (FIO)
//
// FIO stands for 'FUSE Input/Output' and provides the interface for programming
// FIO plugins for secretsfs. Those FIO plugins need to be registered with the
// fio.RegisterProvider(fm FIOMap) function.
// The FIOMap makes sure that mountpath in the high-top filesystem and FIO plugin
// are always mapped correctly.
//
// Also initializes variable Log for shared (and consistent) Logging.
package fio

import(
    "github.com/muryoutaisuu/secretsfs/pkg/sfslog"
    "github.com/muryoutaisuu/secretsfs/pkg/store"

    "github.com/spf13/viper"
)

// fiomaps contains all FIOMaps, that map FIOProvider to MountPaths
var fiomaps map[string]*FIOMap = make(map[string]*FIOMap) // oder map[string]FIOMap

// Log contains all the needed Loggers
var Log *sfslog.Log = sfslog.Logger()

// sto contains the currently set store
var sto store.Store

// RegisterProvider registers FIOMaps.
// To be used inside of init() Function of Plugins.
// If FIO is enabled due to configuration, also enable it.
// If not, only register it in Disabled state
func RegisterProvider(fm *FIOMap) {
    fios := viper.GetStringSlice("ENABLED_FIOS")
    for _,f := range fios {
        if f == fm.Provider.FIOPath() {
            fm.Enabled = true
        }
    }
    fiomaps[fm.Provider.FIOPath()] = fm
}

// FIOMaps returns all registered FIO Plugins mapped with their mountpaths.
// Generally used by high-top secretsfs filesystem so it can correctly redirect
// calls to corresponding plugins.
func FIOMaps() map[string]*FIOMap {
    return fiomaps
}

func init() {
    sto = store.GetStore()
}
```

Dec 17, 18 18:35

secretsfiles.go

Page 1/1

```
package fio

import (
    "github.com/hanwen/go-fuse/fuse"
    "github.com/hanwen/go-fuse/fuse/nodefs"
)

// FIOSecretsfiles is a Filesystem implementing the FIOPlugin interface that
// outputs secrets directly when doing a command like:
// cat <mountpath>/secretsfiles/<secretsItem>
type FIOSecretsfiles struct {}

func (t *FIOSecretsfiles) GetAttr(name string, context *fuse.Context) (*fuse.Attr, fuse.Status) {
    return sto.GetAttr(name, context)
}

func (t *FIOSecretsfiles) OpenDir(name string, context *fuse.Context) ([]fuse.Dirent, fuse.Status) {
    return sto.OpenDir(name, context)
}

func (t *FIOSecretsfiles) Open(name string, flags uint32, context *fuse.Context) (nodefs.File, fuse.Status) {
    content, status := sto.Open(name, flags, context)
    if status == fuse.OK && content != "" {
        return nodefs.NewDataFile([]byte(content)), status
    }
    return nil, status
}

func (t *FIOSecretsfiles) FIOPath() string {
    return "secretsfiles"
}

func init() {
    fioprov := FIOSecretsfiles{}
    fm := FIOMap{
        Provider: &fioprov,
    }
    RegisterProvider(&fm)
}
```

```

Dec 17, 18 18:35      templatefiles.go      Page 1/4

package fio

import (
    "fmt"
    "os"
    "io/ioutil"
    "text/template"
    "bytes"
    "path"
    "strings"
    "errors"

    "github.com/muryoutaisuu/secretsfs/pkg/store"

    "github.com/hanwen/go-fuse/fuse"
    "github.com/hanwen/go-fuse/fuse/nodefs"
    "github.com/spfl3/viper"
)

// FIOTemplatefiles is a Filesystem implementing the FIOPlugin interface that
// first reads in a certain templatefile and then parses through all variables
// trying to call the store with the requesting users UID. If the requesting use
// r
// does have permission for each secret, the template will be rendered with thos
// e
// secret values and returned upon an easy read syscall:
// cat <mountpoint>/templatefiles/templated.conf
type FIOTemplatefiles struct {
    templpath string
}

// secret will be used to call the stores implementation of all the needed FUSE-
// operations together with the provided flags and fuse.Context.
type secret struct {
    flags uint32
    context *fuse.Context
    t *FIOTemplatefiles
}

func (t *FIOTemplatefiles) GetAttr(name string, context *fuse.Context) (*fuse.Attr, fuse.Status) {
    Log.Debug.Printf("ops=GetAttr name=\"%v\"\\n", name)

    // opening directory (aka templatefiles/)
    if name == "" {
        return &fuse.Attr{
            Mode: fuse.S_IFDIR | 0550,
        }, fuse.OK
    }

    // get path to templates
    filepath := getCorrectPath(name)

    // check whether filepath exists
    file, err := os.Stat(filepath)
    if err != nil {
        Log.Error.Println(err)
        return nil, fuse.ENOENT
    }

    // get fileMode
    // https://stackoverflow.com/questions/8824571/golang-determining-whethe

```

Wednesday January 16, 2019

./pkg/fio/templatefiles.go

```

Dec 17, 18 18:35      templatefiles.go      Page

r-file-points-to-file-or-directory
switch mode := file.Mode(); {
case mode.IsDir():
    return &fuse.Attr{
        Mode: fuse.S_IFDIR | 0550,
    }, fuse.OK
case mode.IsRegular():
    return &fuse.Attr{
        Mode: fuse.S_IFREG | 0550,
        Size: uint64(len(name)),
    }, fuse.OK
}

return nil, fuse.EINVAL
}

func (t *FIOTemplatefiles) OpenDir(name string, context *fuse.Context) ([]fuse.DirEntry, fuse.Status) {
    Log.Debug.Printf("ops=OpenDir name=\"%v\"\\n", name)

    // get filepath to templates
    filepath := getCorrectPath(name)

    // check whether filepath exists
    file, err := os.Stat(filepath)
    if err != nil {
        Log.Error.Println(err)
        return nil, fuse.ENOENT
    }

    // check whether filepath is a directory
    // https://stackoverflow.com/questions/8824571/golang-determining-whethe
    if !file.Mode().IsDir() {
        Log.Error.Printf("op=OpenDir msg=\"%not a directory\" filepath=%s\"\\n", filepath)
        return nil, fuse.ENOTDIR
    }

    entries, err := ioutil.ReadDir(filepath)
    if err != nil {
        Log.Error.Print(err)
        return nil, fuse.EBUSY
    }

    dirs := []fuse.DirEntry{}
    for _, e := range entries {
        d := fuse.DirEntry{
            Name: e.Name(),
            Mode: uint32(e.Mode()),
        }
        dirs = append(dirs, d)
    }
    return dirs, fuse.OK
}

func (t *FIOTemplatefiles) Open(name string, flags uint32, context *fuse.Context) (nodefs.File, fuse.Status) {
    Log.Debug.Printf("ops=Open name=\"%v\"\\n", name)

    // get filepath to templates
    filepath := getCorrectPath(name)

    // check whether filepath exists

```

Dec 17, 18 18:35

templatefiles.go

Page 3/4

```

    file, err := os.Stat(filepath)
    if err != nil {
        Log.Error.Println(err)
        return nil, fuse.ENOENT
    }

    // check whether filepath is a file
    // https://stackoverflow.com/questions/8824571/golang-determining-whether-file-points-to-file-or-directory
    if !file.Mode().IsRegular() {
        Log.Error.Printf("op=Open msg=\"not a directory\" filepath=\"%s\\n\",filepath)
        return nil, fuse.EISDIR
    }

    filename := path.Base(filepath)
    parser, err := template.New(filename).ParseFiles(filepath)
    // error handling
    if err != nil {
        errs := err.Error()
        Log.Error.Println(errs)
        return nil, fuse.EREMOTEIO
    }

    // https://gowalker.org/text/template#Template_Execute
    // https://yourbasic.org/golang/io-writer-interface-explained/
    // https://gowalker.org/bytes#Buffer_Bytes
    // https://stackoverflow.com/questions/23454940/getting-bytes-buffer-does-not-implement-io-writer-error-message
    var buf bytes.Buffer
    secret := secret{
        flags: flags,
        context: context,
        t: t,
    }

    err = parser.Execute(&buf, secret)
    if err != nil {
        Log.Error.Println(err)
        switch {
        case strings.Contains(err.Error(), fmt.Sprintf(fuse.EACCES)):
            return nil, fuse.EACCES
        default:
            return nil, fuse.EREMOTEIO
        }
    }

    return nodefs.NewDataFile(buf.Bytes()), fuse.OK
}

func (t *FIOTemplatefiles) FIOPath() string {
    return "templatefiles"
}

// getCorrectPath returns the corrected Path for reading the file from local
// filesystem
func getCorrectPath(name string) string {
    filepath := viper.GetString("PATH_TO_TEMPLATES")+name
    Log.Debug.Printf("op=getCorrectPath variable=filepath value=\"%s\\n\",filepath)
    return filepath
}

```

Wednesday January 16, 2019

./pkg/fio/templatefiles.go

Dec 17, 18 18:35

templatefiles.go

Page

```

// Get is the function that will be called from inside of the templatefile.
// You need to use following scheme to get secrets substituted:
// {{ .Get "path/to/secret" }}
func (s secret) Get(filepath string) (string, error) {
    sto := store.GetStore()
    content, status := sto.Open(filepath, s.flags, s.context)
    if status != fuse.OK {
        Log.Error.Printf("op=Get msg=\"There was an error while loading secret from store\" fuse.Status=\"%s\\n\",status)
        //return "", errors.New("There was an error while loading secret from store, fuse.Status="+fmt.Sprintf(status))
        return "", errors.New(fmt.Sprintf(status))
    }
    return content, nil
}

func init() {
    fioprov := FIOTemplatefiles{
        templpath: viper.GetString("PATH_TO_TEMPLATES"),
    }
    fm := FIOMap{
        Provider: &fioprov,
    }
    RegisterProvider(&fm)
}

```



Dec 17, 18 18:35

types.go

Page 1/1

```
package fio

import (
    "github.com/hanwen/go-fuse/fuse"
    "github.com/hanwen/go-fuse/fuse/nodefs"
)

// FIOProvider interface provides all necessary calls used by FUSE.
// FIOProvider implementations will be called by the secretsfs high-top filesystem.
//
// If you want to program your own FIO plugin, please implement this interface
// and register your provider with the RegisterProvider(*FIOMap) function.
// The FUSE-calls are adopted from https://godoc.org/github.com/hanwen/go-fuse/fuse/pathfs#FileSystem
type FIOProvider interface {
    GetAttr(string, *fuse.Context) (*fuse.Attr, fuse.Status)
    Open(string, uint32, *fuse.Context) (nodefs.File, fuse.Status)
    OpenDir(string, *fuse.Context) ([]fuse.DirEntry, fuse.Status)

    // FIOPath returns a string containing the name and path of the FIO plugin.
    // It decides, on which subdirectory it will be available after mounting.
    // FIOPath was done as a function rather than as an attribute, because it is not
    // possible to define attributes for interfaces in golang.
    // Also see https://github.com/golang/go/issues/23796
    FIOPath() string
}

// FIOMap maps the FIOProvider to a MountPath.
// Used for registering FIOProviders.
type FIOMap struct {
    Provider FIOProvider
    Enabled bool
}
```

```

Jan 14, 19 9:53      secretsfs.go      Page 1/3

// Secretsfs
//
// Secretsfs contains the high-top filesystem, that controls top-paths and
// correctly redirects calls to the correct FUSE Input/Output (FIO) plugin
//
// If the tool secretsfs is mounted at /mnt/secretsfs, then for /mnt/secretsfs/x
// x will be the paths of the registered FIO Plugins.
// like e.g.:
//      /mnt/secretsfs/secretsfiles
//      /mnt/secretsfs/templatefiles
//
// FUSE Calls will be redirected 1:1 to those plugins, only changes made are:
// 1. calls directly on the top layer, like `ls -ld /mnt/secretsfs/secretsf
//
// that call will be answered by secretsfs itself
// 2. the called path will be shortened, so it matches more accurately
// that means, when a user calls `ls -la /mnt/secretsfs/secretsfiles/foo
//
// instead of passing the values /mnt/secretsfs/secretsfiles/foo/bar or
// secretsfiles/foo/bar
// the value foo/bar will be returned
// inspired by this example: https://github.com/hanwen/go-fuse/blob/master/examp
// le/hello/main.go
package secretsfs

import (
    "errors"
    "strings"
    "path/filepath"
    "os/user"
    "strconv"

    "github.com/hanwen/go-fuse/fuse"
    "github.com/hanwen/go-fuse/fuse/nodefs"
    "github.com/hanwen/go-fuse/fuse/pathfs"

    "github.com/muryoutaisuu/secretsfs/pkg/fio"
    "github.com/muryoutaisuu/secretsfs/pkg/store"
    "github.com/muryoutaisuu/secretsfs/pkg/sfslog"
)

// Log is used for shared logging properties
var Log *sfslog.Log = sfslog.Logger()

// SecretsFS is the high-top filesystem.
// It contains references to FIOMap (mapping mountpath to a plugin) and the
// currently used store.
//
type SecretsFS struct {
    pathfs.FileSystem
    fms map[string]*fio.FIOMap
    store store.Store
}

// NewSecretsFS return a fully configured SecretsFS, that is ready to be mounted
//
// Also does a pre-check whether a store was defined. Returns an error if that
// is not the case.
func NewSecretsFS(fs pathfs.FileSystem, fms map[string]*fio.FIOMap, s store.Stor
e) (*SecretsFS, error) {
    if s == nil {

```

Wednesday January 16, 2019

./pkg/secretsfs/secretsfs.go

```

Jan 14, 19 9:53      secretsfs.go      Page

return nil, errors.New("could not initialize store, store is

!")
    }
    sfs := SecretsFS{
        FileSystem: fs,
        fms: fms,
        store: s,
    }
    return &sfs, nil
}

func (sfs *SecretsFS) GetAttr(name string, context *fuse.Context) (*fuse.Attr,
fuse.Status) {
    root, subpath := rootName(name)
    Log.Debug.Printf("ops=GetAttr name=\"%v\" root=\"%v\" subpath=\"%v\"
name,root,subpath)
    u,e := getUser(context)
    if e != nil {
        return nil, fuse.EPERM
    }
    Log.Info.Printf("ops=GetAttr name=\"%v\" root=\"%v\" subpath=\"%v\"
erid=\"%v\" username=\"%v\"",name,root,subpath,u.Uid,u.Username)

    if root == "" && subpath == "" {
        return &fuse.Attr{Mode: fuse.S_IFDIR | 0755}, fuse.OK
    }
    if _,ok := sfs.fms[root]; ok && sfs.fms[root].Enabled {
        return sfs.fms[root].Provider.GetAttr(subpath, context)
    }
    return &fuse.Attr{}, fuse.ENOENT
}

func (sfs *SecretsFS) OpenDir(name string, context *fuse.Context) (c []fuse.En
try, code fuse.Status) {
    root, subpath := rootName(name)
    Log.Debug.Printf("ops=GetAttr name=\"%v\" root=\"%v\" subpath=\"%v\"
name,root,subpath)
    u,e := getUser(context)
    if e != nil {
        return nil, fuse.EPERM
    }
    Log.Info.Printf("ops=GetAttr name=\"%v\" root=\"%v\" subpath=\"%v\"
erid=\"%v\" username=\"%v\"",name,root,subpath,u.Uid,u.Username)

    if name == "" {
        c = []fuse.DirEntry{}
        for k := range sfs.fms {
            c = append(c, fuse.DirEntry{Name: k, Mode: fuse.S_IF
        })
    }
    return c, fuse.OK
}

if _,ok := sfs.fms[root]; ok && sfs.fms[root].Enabled {
    return sfs.fms[root].Provider.OpenDir(subpath, context)
}
return nil, fuse.ENOENT
}

func (sfs *SecretsFS) Open(name string, flags uint32, context *fuse.Context)
le nodefs.File, code fuse.Status) {
    root, subpath := rootName(name)
    Log.Debug.Printf("ops=GetAttr name=\"%v\" root=\"%v\" subpath=\"%v\"

```

Jan 14, 19 9:53

secretsfs.go

Page 3/3

```
name,root,subpath)
    u,e := getUser(context)
    if e != nil {
        return nil, fuse.EPERM
    }
    Log.Info.Printf("ops=GetAttr name=\"%v\" root=\"%v\" subpath=\"%v\" \n userid=\"%v\" username=\"%v\"",name,root,subpath,u.Uid,u.Username)

    if name == "" {
        return nil, fuse.EINVAL
    }
    if _,ok := sfs.fms[root]; ok && sfs.fms[root].Enabled {
        return sfs.fms[root].Provider.Open(subpath, flags, context)
    }
    return nil, fuse.EPERM
}

// rootName calculates, which FIO the call came from and what the subpath for
// the FIO is
func rootName(path string) (root, subpath string) {
    list := strings.Split(path, string(filepath.Separator))
    root = list[0]
    subpath = filepath.Join(list[1:]...)
    return
}

// getUser returns user element
// used for getting userinfo for logging
func getUser(context *fuse.Context) (*user.User, error) {
    return user.LookupId(strconv.Itoa(int(context.Owner.Uid)))
}
```

Dec 17, 18 18:35

store.go

Page 1/1

```
package store

import (
    "github.com/muryoutaisuu/secretsfs/pkg/sfslog"
    "github.com/spf13/viper"
    //"github.com/muryoutaisuu/secretsfs/cmd/secretsfs/config"
)

// store contains the registered Store
var store Store

// available stores
var stores []string

// Log contains all the needed Loggers
var Log *sfslog.Log = sfslog.Logger()

// Store returns currently active Store Implementation
func GetStore() Store {
    return store
}

// RegisterStore registers available stores
// if a store is also set to be the backend store it will be set here
func RegisterStore(s Store) {
    stores = append(stores, s.String())
    if viper.GetString("CURRENT_STORE") == s.String() {
        store = s
    }
}

// GetStores returns all registered stores.
// Registered stores are all available stores that a user may configure as a
// store of secretsfs.
func GetStores() []string {
    return stores
}

func init() {
    stores = []string{}
}
```

Dec 17, 18 18:35

types.go

Page 1/1

```
package store

import (
    "github.com/hanwen/go-fuse/fuse"
    //"github.com/hanwen/go-fuse/fuse/nodefs"
)

// Store interface provides all necessary calls to backend store. Store will be
// called by the FIOProviders.
//
// If you want to program your own Store plugin, please implement this interface
// and register your store with the RegisterStore(Store) function. The FUSE-calls
// are adopted from https://godoc.org/github.com/hanwen/go-fuse/fuse/pathfs#File
// System
type Store interface {
    GetAttr(name string, context *fuse.Context) (*fuse.Attr, fuse.Status)
    Open(name string, flags uint32, context *fuse.Context) (string, fuse.Status)
    OpenDir(name string, context *fuse.Context) ([]fuse.DirEntry, fuse.Status)

    // String returns a string containing the name of the store plugin.
    // It decides, how to react on FUSE-calls.
    // String was done as a function rather than as an attribute, because it is not
    // possible to define attributes for interfaces in golang.
    // Also see https://github.com/golang/go/issues/23796
    String() (string)
}
```

```

Dec 17, 18 18:35      vault.go      Page 1/8
package store

import (
    "errors"
    "fmt"
    "strconv"
    "io/ioutil"
    "path/filepath"
    "os/user"
    "strings"
    "path"

    "github.com/hashicorp/vault/api"
    "github.com/hanwen/go-fuse/fuse"
    "github.com/spf13/viper"

    "github.com/muryoutaisuu/secretsfs/pkg/sfshelpers"
)

// Path internals of vault made configurable with viper
// taken from https://www.vaultproject.io/api/secret/kv/kv-v2.html
//var (
//    MTDATA string
//    DTDATA string
//)
var MTDATA string
var DTDATA string

// Filetype define the type of the returned value element of vault
type Filetype byte
const (
    CTrueDir    Filetype = 0 // exists in Vault as a directory
    CFile       Filetype = 1 // Key of a key=value pair, emulated as a direct
ory
    CValue      Filetype = 2 // Value of a key=value pair
    CNull       Filetype = 3 // not a valid vault element
)

// Vault struct implements the calls called by fuse and returns accordingly
// requested resources.
// It's a store and may be coupled with multiple fio structs
type Vault struct {
    client *api.Client
}

func (v *Vault) GetAttr(name string, context *fuse.Context) (*fuse.Attr, fuse.St
atus) {
    Log.Debug.Printf("ops=GetAttr name=\"%v\"\\n", name)
    Log.Debug.Printf("ops=GetAttr MTDATA=%s", viper.GetString("MTDATA"))
    Log.Debug.Printf("ops=GetAttr Token=%s", v.client.Token())

    // opening directory (aka secretsfiles/)
    if name == "" {
        return &fuse.Attr{
            Mode: fuse.S_IFDIR | 0550,
        }, fuse.OK
    }

    if err := v.setToken(context); err != nil {
        Log.Error.Print(err)
        return nil, fuse.EACCES
    }
}

```

```

Dec 17, 18 18:35      vault.go      Page
}
defer Log.Debug.Printf("op=GetAttr msg=\"successfully cleared token
ken=%s\"\\n", v.client.Token())
defer v.client.ClearToken()
defer Log.Debug.Printf("op=GetAttr msg=\"successfully cleared token
ken=%s\"\\n", v.client.Token())

// get type
Log.Debug.Printf("name=\"%v\"\\n", name)
_, t := v.getType(name)
Log.Debug.Printf("op=GetAttr t=\"%v\"\\n", t)

// act according to type
switch t {
case CTrueDir:
    return &fuse.Attr{
        Mode: fuse.S_IFDIR | 0550,
    }, fuse.OK
case CFile:
    Log.Debug.Printf("op=GetAttr t=CFile\\n")
    return &fuse.Attr{
        Mode: fuse.S_IFDIR | 0550,
    }, fuse.OK
case CValue:
    return &fuse.Attr{
        Mode: fuse.S_IFREG | 0550,
        Size: uint64(len(name)),
    }, fuse.OK
default:
    return nil, fuse.ENOENT
}
}

func (v *Vault) OpenDir(name string, context *fuse.Context) ([]fuse.DirEntry,
se.Status) {
    Log.Debug.Printf("GetAttr name=\"%v\"\\n", name)

    if err := v.setToken(context); err != nil {
        Log.Error.Print(err)
        return nil, fuse.EACCES
    }
    defer Log.Debug.Printf("op=OpenDir msg=\"successfully cleared token
ken=%s\"\\n", v.client.Token())
    defer v.client.ClearToken()

    _, t := v.getType(name)
    Log.Debug.Printf("ops=OpenDir t=\"%v\"\\n", t)

    switch t {
case CTrueDir:
    dirs, err := v.listDir(name)
    if err != nil {
        Log.Error.Print(err)
        return *dirs, fuse.EIO
    }
    Log.Debug.Printf("op=OpenDir name=\"%v%v\" dirs=\"%v\" err=
\\n", MTDATA, name, dirs, err)
    return *dirs, fuse.OK
case CFile:
    dirs, err := v.listFiles(name)
    Log.Debug.Printf("op=OpenDir dirs=\"%v\" err=\"%v\"\\n", dirs
    if err != nil {

```

```

Dec 17, 18 18:35      vault.go      Page 3/8

        Log.Error.Print(err)
        return nil, fuse.EIO
    }
    Log.Debug.Printf("op=OpenDir ctype=CFile secretType=\"%T\" secre
t=\"%v\"\\n",dirs,dirs)
    return *dirs, fuse.OK
    case CValue:
        return nil, fuse.ENOTDIR
    }
    return nil, fuse.ENOENT
}

func (v *Vault) Open(name string, flags uint32, context *fuse.Context) (string,
fuse.Status) {
    Log.Debug.Printf("op=Open name=\"%v\"\\n",name)

    if err := v.setToken(context); err != nil {
        Log.Error.Print(err)
        return "", fuse.EACCES
    }
    defer Log.Debug.Printf("op=Open msg=\"successfully cleared token\" token
=\"%s\"\\n",v.client.Token())
    defer v.client.ClearToken()

    s,t := v.getType(name)
    Log.Debug.Printf("op=Open t=\"%v\"\\n",t)

    switch t {
    case CTrueDir:
        return "", fuse.EISDIR
    case CFile:
        return "", fuse.EISDIR
    case CValue:
        // get substituted value (if substitution must be done, else kee
p original)
        Log.Debug.Printf("op=Open msg=\"before substituting name\" varia
ble=name value=%v\\n",name)
        name,_, err := v.getCorrectName(name, true)
        if err != nil {
            Log.Error.Print(err)
            return "", fuse.EIO
        }
        Log.Debug.Printf("op=Open msg=\"after substituting name\" variab
le=name value=%v\\n",name)

        Log.Debug.Printf("op=Open s=\"%v\" name=\"%v\"\\n",s,name)
        data,ok := s.Data["data"].(map[string]interface{})
        if ok != true {
            return "", fuse.EIO
        }
        entry,ok := data[name].(string)
        if ok != true {
            return "", fuse.EIO
        }
        return entry, fuse.OK
    }
    return "", fuse.ENOENT
}

func (v *Vault) String() (string) {
    return "Vault"
}

```

Wednesday January 16, 2019

.pkg/store/vault.go

```

Dec 17, 18 18:35      vault.go      Page

// setToken is called within the fuse interaction calls and sets a working
// accesstoken depending on the calling user
// usually should be used in conjunction to a deferred clear call:
// if err := v.setToken(context); err != nil {
//     Log.Error.Print(err)
//     return nil, fuse.EACCES
// }
// defer v.client.ClearToken()
func (v *Vault) setToken(context *fuse.Context) error {
    u,err := user.LookupId(strconv.Itoa(int(context.Owner.Uid)))
    if err != nil {
        return err
    }
    a,err := v.getAccessToken(u)
    if err != nil {
        return err
    }
    v.client.SetToken(a.Auth.ClientToken)
    // TODO: Remove this debug line, not secure!!
    Log.Debug.Printf("op=setToken msg=\"successfully set token\" token=
n\",v.client.Token())
    return nil
}

// getAccessToken reads the currently set authentication token inside of the
// users home and authenticates with it and returns afterwards the secret
// containing the accesstoken
func (v *Vault) getAccessToken(u *user.User) (*api.Secret, error) {
    auth,err := v.readAuthToken(u)
    if err != nil {
        Log.Error.Print(err)
        return &api.Secret{}, err
    }
    // https://groups.google.com/forum/#!topic/vault-tool/-4F2RLnGrSE
    postdata := map[string]interface{}{
        "role_id": auth,
    }
    Log.Debug.Printf("login_payload=%v\\n",postdata)
    resp,err := v.client.Logical().Write("auth/approle/login", postdata)
    if err != nil {
        Log.Error.Printf("op=getAccessToken msg=\"Got an error while
henticating\"\\n")
        return nil,err
    }
    Log.Debug.Printf("resp=%v Data=%v\\n ClientToken=\"%v\"\\n",resp,resp
,resp.Auth.ClientToken)
    if err != nil {
        Log.Error.Print(err)
        return &api.Secret{}, err
    }
    if resp.Auth == nil {
        return resp, fmt.Errorf("no auth info returned")
    }
    return resp,err
}

// readAuthToken opens the file containing the authenticationtoken and trim
func (v *Vault) readAuthToken(u *user.User) (string, error) {

```

```

Dec 17, 18 18:35      vault.go      Page 5/8

    // path := filepath.Join(u.HomeDir, os.Getenv("SECRETSFS_FILE_ROLEID"))
    path := filepath.Join(u.HomeDir, viper.GetString("FILE_ROLEID"))
    Log.Debug.Printf("msg=\reading authToken\ path=\"%v\"\\n",path)
    o,err := ioutil.ReadFile(path)
    if err != nil {
        Log.Error.Print(err)
        return "",err
    }
    authToken := strings.TrimSuffix(string(o), "\\n")
    Log.Debug.Printf("msg=\authToken successfully read\ path=\"%v\"\\n",pat
h)
    return authToken,nil
}

// listDir lists all entries inside a vault directory type=CTrueDir
func (v *Vault) listDir(name string) ([]fuse.DirEntry, error) {
    Log.Debug.Printf("op=listDir MTDATA=\"%v\" name=\"%v\"\\n",MTDATA,name)
    s,err := v.client.Logical().List(MTDATA + name)
    Log.Debug.Printf("secret=\"%v\"\\n",s)

    // can't list in vault
    if err != nil || s == nil {
        if err == nil {
            err = errors.New("cant list path "+MTDATA+name+" in vault")
        }
        Log.Error.Print(err)
        return nil, err
    }

    Log.Debug.Printf("GetAttr name=\"%v\" secret=\"%v\" secret.Data=\"%v\"\\n",name,s.Data)
    dirs := []fuse.DirEntry{}
    // https://github.com/asteris-llc/vaultfs/blob/master/fs/root.go
    // TODO: add Error Handling
    Log.Debug.Printf("op=listDir dirs=\"%v\"\\n",dirs)
    for i := 0; i < len(s.Data["keys"].([]interface{})); i++ {
        d := fuse.DirEntry{
            Name: path.Base(s.Data["keys"].([]interface{})[i].(string)),
            Mode: fuse.S_IFREG,
        }
        dirs = append(dirs, d)
        Log.Debug.Printf("op=listDir dirs=\"%v\"\\n",dirs)
    }
    return &dirs,nil
}

// listFile lists the contents of a virtual directory in secretsfs
// (aka a file in vault) type=CFile
// returns a Slice containing all valid entries
// valid means no entries containing a / in their names
func (v *Vault) listFile(name string) ([]fuse.DirEntry, error) {
    data,err := v.listFilesNames(name)
    if err != nil {
        return nil, err
    }

    dirs := []fuse.DirEntry{}
    for k := range data {
        key := data[k]
        // special treatment for entries containing the substitution cha

```

Wednesday January 16, 2019

./pkg/store/vault.go

```

Dec 17, 18 18:35      vault.go      Page

racter
    if strings.Contains(key, "/") { // viper.GetString("subst_ch
) { // strings.Contains(k, "/") {
    key = strings.Replace(key, "/", string(viper.GetString("subst_char"))[0]), -1)
    }

    d := fuse.DirEntry{
        Name: key,
        Mode: fuse.S_IFREG,
    }
    dirs = append(dirs, d)
}
Log.Debug.Printf("op=listFile dirs=\"%v\"\\n",dirs)
return &dirs,nil
}

// listFileNames is very similar to listFile, but instead of returning fully
// finished fuse.DirEntry types, it only returns []string containing the key
func (v *Vault) listFileNames(name string) ([]string, error) {
    Log.Debug.Printf("op=listFileNames msg=\going to read data\ path=
\\n",DTDATA + name)
    s,err := v.client.Logical().Read(DTDATA + name)
    if err != nil || s == nil {
        if err == nil {
            errors.New("cant read")
        }
        return nil,err
    }
    Log.Debug.Printf("op=listFile secret=\"%v\"\\n",s)
    Log.Debug.Printf("op=listFile secret.Data=\"%v\" secret.DataType=\\n",s.Data,s.DataType)
    data,ok := s.Data["data"].(map[string]interface{})
    if !ok {
        return nil, errors.New("s.Data[\\data\\] resulted in a error")
    }
    Log.Debug.Printf("op=listFileNames data=\"%v\" dataType=\\%T\"\\n",data,
ata)

    filenames := []string{}
    for k := range data {
        filenames = append(filenames, k)
    }
    return filenames, nil
}

// getType returns type of the requested resource
// used by most fuse actions for simplifying reasons
// types may be the defined FileType byte constants on top of this file
func (v *Vault) getType(name string) (*api.Secret, filetype){
    Log.Debug.Printf("op=getType name=\"%v\"\\n",name)
    s,err := v.client.Logical().List(MTDATA + name)
    Log.Debug.Printf("op=getType MTDATA=%s",MTDATA)
    Log.Debug.Printf("op=getType s=\"%v\" err=\"%v\"\\n",s,err)
    if err == nil && s != nil {
        return s, CTrueDir
    }

    s,err = v.client.Logical().Read(DTDATA + name)
    if err == nil && s != nil {
        return s, CFile
    }
}

```



Dec 17, 18 18:35

vault.go

Page 7/8

```

        name = path.Dir(name) // clip last element
        s,err = v.client.Logical().Read(DTDATA + name)
        if err == nil && s!=nil {
            return s, CValue
        }

        return nil, CNull
    }

// getCorrectName checks whether a path contains any maybe substituted character
s.
// If yes, it checks in Vault whether there is a substituted key available and
// returns it incl. whole path.
// if only the Name itself is wished, set nameonly=true
// if no value found, then throws an error and returns ""
//
// Why is there a nameonly=true ?
// Problem being the fact, that with the original value in Vault the correct
// path for getting the Secret from Vault may be quite tricky
// e.g. the substituted value:  GET secret/my_bad_key
// would become:              GET secret/my/bad/key
// where a simple path.Base(path) won't return the secret's name anymore
func (v *Vault) getCorrectName(pathname string, nameonly bool) (string, bool, error) {
    value := pathname

    // split if nameonly is true
    if nameonly {
        value = path.Base(pathname)
    }

    // check whether name contains any characters, that may be substituted
    if !strings.Contains(value, viper.GetString("subst_char")) {
        Log.Debug.Printf("op=getCorrectName msg=\"contains no characters
that may be substituted\" variable=value value=\"%v\"\\n\",value)
        return value, false, nil
    }

    dir := path.Dir(pathname)
    Log.Debug.Printf("op=getCorrectName msg=\"do a listFileNames with specif
ic dir\" variable=dir value=\"%v\"\\n\",dir)
    filenames,err := v.listFilesNames(dir)
    if err != nil {
        return "", false, err
    }
    Log.Debug.Printf("op=getCorrectName msg=\"got actual contents of vault s
ecret\" variable=filenames value=\"%v\"\\n\",filenames)

    possibilities := sfsHelpers.SubstitutionPossibilities(value, viper.GetSt
ring("subst_char"), "/")
    Log.Debug.Printf("op=getCorrectName msg=\"got all possible key names\" v
ariable=possibilities value=\"%v\"\\n\",possibilities)
    for _,f := range filenames {
        for _,p := range possibilities {
            if f == p {
                Log.Debug.Printf("op=getCorrectName msg=\"found
correct substituted name\" variable=filename value=\"%v\"\\n\",f)
                if nameonly {
                    return f, true, nil
                }
                return dir+"/"+f, true, nil
            }
        }
    }
}

```

Dec 17, 18 18:35

vault.go

Page

```

    }
}

return "", false, errors.New("can't find any substituted possibilities")
}

func init() {
    c,err := api.NewClient(&api.Config{
        // Address: os.Getenv("VAULT_ADDR"),
        Address: viper.GetString("VAULT_ADDR"),
    })
    if err != nil {
        Log.Error.Fatal(err)
    }
    v := Vault{
        client: c,
    }
    v.client.ClearToken()
    RegisterStore(&v) //https://stackoverflow.com/questions/40823315/x-c
not-implement-y-method-has-a-pointer-receiver
    if viper.GetString("CURRENT_STORE") == v.String() {
        Log.Debug.Printf("op=init MTDATA=%s",viper.GetString("MTDATA"))
        MTDATA = viper.GetString("MTDATA")
        DTDATA = viper.GetString("DTDATA")
    }
}

```

Dec 17, 18 18:35

sfslog.go

Page 1/1

```
// sfslog package provides all needed logging features for a consistent logging
// format through all provided plugins.
// Code taken from example:
// https://www.ardanlabs.com/blog/2013/11/using-log-package-in-go.html
package sfslog

import (
    "log"
    "os"
    "io"
    //"io/ioutil"
    // TODO: make debug configurable
)

// Log will contain four different logging levels, which themselves can be
// called like any other default go logger (because they are default go logger
// in reality).
type Log struct {
    Debug    *log.Logger
    Info     *log.Logger
    Warn     *log.Logger
    Error    *log.Logger
}

// Logger return a struct of type Log, which contains for different logging leve
l
// default go loggers.
func Logger() *Log {
    var l Log
    // log setup
    logInit(&l, os.Stdout, os.Stdout, os.Stdout, os.Stderr)
    //logInit(&l, ioutil.Discard, os.Stdout, os.Stdout, os.Stderr)
    return &l
}

func logInit(
    l *Log,
    debugHandle io.Writer,
    infoHandle io.Writer,
    warnHandle io.Writer,
    errorHandle io.Writer) {

    l.Debug = log.New(debugHandle,
        "DEBUG: ",
        log.Ldate|log.Ltime|log.Lshortfile)

    l.Info = log.New(infoHandle,
        "INFO: ",
        log.Ldate|log.Ltime|log.Lshortfile)

    l.Warn = log.New(warnHandle,
        "WARN: ",
        log.Ldate|log.Ltime|log.Lshortfile)

    l.Error = log.New(errorHandle,
        "ERROR: ",
        log.Ldate|log.Ltime|log.Lshortfile)
}
```

Dec 17, 18 18:35

**sfshelpers.go**

Page 1/1

```
// sfshelpers package just contains some really general helpers that were used
// throughout the development of secretsfs
package sfshelpers
```

Dec 17, 18 18:35

string\_substitution.go

Page 1/2

```

package sfshelpers

// SubstitutionPossibilities calculates all different possibilities that can be
// be achieved by substituting a string one way or another.
// Takes string as arg s, then a bad string as b, and a new string as n.
// Also look at this example:
//
//      func main() {
//          orig := "a_b_c_d"
//          mytest := SubstitutionPossibilities(orig, "_", "/")
//          fmt.Printf("orig: %s\nvariants: %s", orig, mytest)
//      }
//
// output will be:
// orig: a_b_c_d
// variants: [a_b_c_d a_b_c/d a_b/c_d a_b/c/d a/b_c_d a/b_c/d a/b/c_d a/b/c
// /d]
func SubstitutionPossibilities(s, b, n string) []string {
    l := len(s)
    var mys []string

    if l > 1 {
        mys = SubstitutionPossibilities(s[l:1], b, n)
    } else {
        mys = []string{s}
        if needInv(s, b) {
            mys = append(mys, inv(s,b,n))
        }
        return mys
    }

    lm := len(mys)
    for i:=0; i<lm; i++ {
        mys[i] = string(s[0])+mys[i]
        if needInv(s,b) {
            mys = append(mys, inv(s,b,n)+mys[i][1:])
        }
    }
    return mys
}

func inv(s, b, n string) string {
    if len(s) < 1 {
        return ""
    }

    c := string(s[0])
    if c == b {
        return n
    }
    return c
}

func needInv(s, b string) bool {
    if len(s) < 1 {
        return false
    }

    if string(s[0]) == b {
        return true
    }
    return false
}

```

Dec 17, 18 18:35

string\_substitution.go

Page

}