Branch: **master**

Find file | Copy path

bytom-1 / accesstoken / **accesstoken.go**

shenao78 replace goleveldb with tendermint db (Bytom#1660)

5f62f8f Mar 29, 2019

4 contributors

Raw | Blame | History

144 lines (118 sloc) | 3.52 KB

```go
// Package accesstoken provides storage and validation of Chain Core
// credentials.
package accesstoken

import (
	"crypto/rand"
	"encoding/json"
	"fmt"
	"regexp"
	"strings"
	"time"

	"github.com/bytom/crypto/sha3pool"
	"github.com/bytom/errors"
	dbm "github.com/bytom/database/leveldb"
)

const tokenSize = 32

var (
	// ErrBadID is returned when Create is called on an invalid id string.
	ErrBadID = errors.New("invalid id")
	// ErrDuplicateID is returned when Create is called on an existing ID.
	ErrDuplicateID = errors.New("duplicate access token ID")
	// ErrBadType is returned when Create is called with a bad type.
	ErrBadType = errors.New("type must be client or network")
	// ErrNoMatchID is returned when Delete is called on nonexisting ID.
	ErrNoMatchID = errors.New("nonexisting access token ID")
	// ErrInvalidToken is returned when Check is called on invalid token
	ErrInvalidToken = errors.New("invalid token")

	// validIDRegexp checks that all characters are alphumeric, _ or -.
	// It also must have a length of at least 1.
	validIDRegexp = regexp.MustCompile(`^[\w-]+$`)
)

// Token describe the access token.
type Token struct {
	ID      string    `json:"id"`
	Token   string    `json:"token,omitempty"`
	Type    string    `json:"type,omitempty"`
	Created time.Time `json:"created_at"`
```

```go
}

// CredentialStore store user access credential.
type CredentialStore struct {
	DB dbm.DB
}

// NewStore creates and returns a new Store object.
func NewStore(db dbm.DB) *CredentialStore {
	return &CredentialStore{
		DB: db,
	}
}

// Create generates a new access token with the given ID.
func (cs *CredentialStore) Create(id, typ string) (*Token, error) {
	if !validIDRegexp.MatchString(id) {
		return nil, errors.WithDetailf(ErrBadID, "invalid id %q", id)
	}

	key := []byte(id)
	if cs.DB.Get(key) != nil {
		return nil, errors.WithDetailf(ErrDuplicateID, "id %q already in use", id)
	}

	secret := make([]byte, tokenSize)
	if _, err := rand.Read(secret); err != nil {
		return nil, err
	}

	hashedSecret := make([]byte, tokenSize)
	sha3pool.Sum256(hashedSecret, secret)

	token := &Token{
		ID:      id,
		Token:   fmt.Sprintf("%s:%x", id, hashedSecret),
		Type:    typ,
		Created: time.Now(),
	}

	value, err := json.Marshal(token)
	if err != nil {
		return nil, err
	}
	cs.DB.Set(key, value)

	return token, nil
}

// Check returns whether or not an id-secret pair is a valid access token.
func (cs *CredentialStore) Check(id string, secret string) error {
	if !validIDRegexp.MatchString(id) {
		return errors.WithDetailf(ErrBadID, "invalid id %q", id)
	}

	var value []byte
	token := &Token{}

	if value = cs.DB.Get([]byte(id)); value == nil {
		return errors.WithDetailf(ErrNoMatchID, "check id %q nonexisting", id)
	}
	if err := json.Unmarshal(value, token); err != nil {
		return err
	}

	if strings.Split(token.Token, ":")[1] == secret {
		return nil
	}
```

```go

	return ErrInvalidToken
}

// List lists all access tokens.
func (cs *CredentialStore) List() ([]*Token, error) {
	tokens := make([]*Token, 0)
	iter := cs.DB.Iterator()
	defer iter.Release()

	for iter.Next() {
		token := &Token{}
		if err := json.Unmarshal(iter.Value(), token); err != nil {
			return nil, err
		}
		tokens = append(tokens, token)
	}
	return tokens, nil
}

// Delete deletes an access token by id.
func (cs *CredentialStore) Delete(id string) error {
	if !validIDRegexp.MatchString(id) {
		return errors.WithDetailf(ErrBadID, "invalid id %q", id)
	}

	if value := cs.DB.Get([]byte(id)); value == nil {
		return errors.WithDetailf(ErrNoMatchID, "check id %q", id)
	}

	cs.DB.Delete([]byte(id))
	return nil
}
```