go-ntlm / ntlm / ntlmv1.go

Etienne Stalmans SEAL and UNSEAL (basically same thing, should... 0cbbff5 · 8 years ago History

```go
//Copyright 2013 Thomson Reuters Global Resources. BSD License please see License file

package ntlm

import (
	"bytes"
	rc4P "crypto/rc4"
	"errors"
	"log"
	"strings"
)

/*******************************
 Shared Session Data and Methods
*******************************/

type V1Session struct {
	SessionData
}

func (n *V1Session) SetUserInfo(username string, password string, domain string) {
	n.user = username
	n.password = password
	n.userDomain = domain
}

func (n *V1Session) GetUserInfo() (string, string, string) {
	return n.user, n.password, n.userDomain
}

func (n *V1Session) SetMode(mode Mode) {
	n.mode = mode
}

func (n *V1Session) SetTarget(target string) {
	n.target = target
}

func (n *V1Session) Version() int {
	return 1
}

func (n *V1Session) SetNTHash(nthash []byte) {
	//fmt.Printf("Passed: %x\n", nthash)
	n.responseKeyNT = nthash
}

func (n *V1Session) fetchResponseKeys() (err error) {
	//check if we have set the LM from the command line (pass the hash)
	if len(n.responseKeyNT) > 0 {
		return
	}
	n.responseKeyLM, err = lmowfv1(n.password)
	if err != nil {
		return err
	}
	n.responseKeyNT = ntowfv1(n.password)
	return
}

func (n *V1Session) computeExpectedResponses() (err error) {

	if NTLMSSP_NEGOTIATE_EXTENDED_SESSIONSECURITY.IsSet(n.NegotiateFlags) {
		n.ntChallengeResponse, err = desL(n.responseKeyNT, md5(concat(n.serverC
		if err != nil {
			return err
		}
		n.lmChallengeResponse = concat(n.clientChallenge, make([]byte, 16))
	} else {
		n.ntChallengeResponse, err = desL(n.responseKeyNT, n.serverChallenge)
		if err != nil {
			return err
		}
		// NoLMResponseNTLMv1: A Boolean setting that controls using the NTLM r
		// response to the server challenge when NTLMv1 authentication is used.
```

```go
**************/

type V1ClientSession struct {
        V1Session
}

/*
NTLMSSP_NEGOTIATE_56: true
NTLMSSP_NEGOTIATE_KEY_EXCH: true
NTLMSSP_NEGOTIATE_128: true
NTLMSSP_NEGOTIATE_VERSION: true
NTLMSSP_NEGOTIATE_EXTENDED_SESSIONSECURITY: true
NTLMSSP_NEGOTIATE_ALWAYS_SIGN: true
NTLMSSP_NEGOTIATE_NTLM: true
NTLMSSP_REQUEST_TARGET: true
NTLM_NEGOTIATE_OEM: true
NTLMSSP_NEGOTIATE_UNICODE: true
*/

func (n *V1ClientSession) GenerateNegotiateMessage() (nm *NegotiateMessage, err error)
        flags := uint32(0)
        flags = NTLMSSP_NEGOTIATE_KEY_EXCH.Set(flags)
        flags = NTLMSSP_NEGOTIATE_56.Set(flags)
        flags = NTLMSSP_NEGOTIATE_128.Set(flags)
        flags = NTLMSSP_NEGOTIATE_VERSION.Set(flags)
        flags = NTLMSSP_NEGOTIATE_EXTENDED_SESSIONSECURITY.Set(flags)
        flags = NTLMSSP_NEGOTIATE_ALWAYS_SIGN.Set(flags)
        flags = NTLMSSP_NEGOTIATE_NTLM.Set(flags)
        flags = NTLMSSP_REQUEST_TARGET.Set(flags)
        flags = NTLM_NEGOTIATE_OEM.Set(flags)
        flags = NTLMSSP_NEGOTIATE_UNICODE.Set(flags)

        neg := new(NegotiateMessage)
        neg.Signature = []byte("NTLMSSP\x00")
        neg.MessageType = 1
        neg.NegotiateFlags = flags

        //if NTLMSSP_NEGOTIATE_OEM_DOMAIN_SUPPLIED
        neg.DomainNameFields = new(PayloadStruct)
        //if NTLMSSP_NEGOTIATE_OEM_WORKSTATION_SUPPLIED
        neg.WorkstationFields = new(PayloadStruct)

        neg.Version = &VersionStruct{ProductMajorVersion: uint8(5), ProductMinorVersion

        return neg, nil
}

func (n *V1ClientSession) ProcessChallengeMessage(cm *ChallengeMessage) (err error) {
        n.challengeMessage = cm
        n.serverChallenge = cm.ServerChallenge
        n.clientChallenge = randomBytes(8)

        // Set up the default flags for processing the response. These are the flags th
        // in the authenticate message
        flags := uint32(0)
        flags = NTLMSSP_NEGOTIATE_KEY_EXCH.Set(flags)
        // NOTE: Unsetting this flag in order to get the server to generate the signatu
        flags = NTLMSSP_NEGOTIATE_VERSION.Set(flags)
        flags = NTLMSSP_NEGOTIATE_EXTENDED_SESSIONSECURITY.Set(flags)
        flags = NTLMSSP_NEGOTIATE_TARGET_INFO.Set(flags)
        flags = NTLMSSP_NEGOTIATE_IDENTIFY.Set(flags)
        flags = NTLMSSP_NEGOTIATE_ALWAYS_SIGN.Set(flags)
        flags = NTLMSSP_NEGOTIATE_NTLM.Set(flags)
        flags = NTLMSSP_NEGOTIATE_DATAGRAM.Set(flags)
        flags = NTLMSSP_NEGOTIATE_SIGN.Set(flags)
        flags = NTLMSSP_REQUEST_TARGET.Set(flags)
        flags = NTLMSSP_NEGOTIATE_UNICODE.Set(flags)

        n.NegotiateFlags = flags

        err = n.fetchResponseKeys()
        if err != nil {
                return err
        }

        err = n.computeExpectedResponses()
        if err != nil {
                return err
        }

        err = n.computeSessionBaseKey()
        if err != nil {
                return err
        }

        err = n.computeKeyExchangeKey()
        if err != nil {
                return err
        }

        err = n.computeEncryptedSessionKey()
        if err != nil {
                return err
        }

        err = n.calculateKeys(cm.Version.NTLMRevisionCurrent)
        if err != nil {
                return err
        }

        n.clientHandle, err = rc4Init(n.ClientSealingKey)
```

Files

cd032d4

Go to file

- ntlm
  - md4
  - av_pairs.go
  - challenge_responses.go
  - crypto.go
  - crypto_test.go
  - helpers.go
  - helpers_test.go
  - keys.go

go-ntlm / ntlm / ntlmv1.go   Top

Code Blame 473 lines (397 loc) · 14.5 KB Raw

```go
        if err != nil {
                return err
        }

        return nil
}

func (n *V1ClientSession) GenerateAuthenticateMessage() (am *AuthenticateMessage, err e
        am = new(AuthenticateMessage)
        am.Signature = []byte("NTLMSSP\x00")
        am.MessageType = uint32(3)
        am.LmChallengeResponse, _ = CreateBytePayload(n.lmChallengeResponse)
        am.NtChallengeResponseFields, _ = CreateBytePayload(n.ntChallengeResponse)
        am.DomainName, _ = CreateStringPayload(n.userDomain)
        am.UserName, _ = CreateStringPayload(n.user)
        am.Workstation, _ = CreateStringPayload("RULER")
        am.EncryptedRandomSessionKey, _ = CreateBytePayload(n.encryptedRandomSessionKey
```

- message_authenticate.go
- message_authenticate_test.go
- message_challenge.go
- message_challenge_test.go
- message_negotiate.go
- negotiate_flags.go
- negotiate_flags_test.go
- ntlm.go
- ntlmv1.go
- ntlmv1_test.go
- ntlmv2.go
- ntlmv2_test.go
- payload.go
- signature.go
- signature_test.go
- version.go
- .gitignore
- License
- README.md
- go.mod

```go
        am.NegotiateFlags = n.NegotiateFlags
        am.Version = &VersionStruct{ProductMajorVersion: uint8(5), ProductMinorVersion:
        return am, nil
}
func (n *V1ClientSession) GenerateAuthenticateMessageAV() (am *AuthenticateMessage, err
        return nil, nil
}

func (n *V1ClientSession) computeEncryptedSessionKey() (err error) {
        if NTLMSSP_NEGOTIATE_KEY_EXCH.IsSet(n.NegotiateFlags) {
                n.exportedSessionKey = randomBytes(16)
                n.encryptedRandomSessionKey, err = rc4K(n.keyExchangeKey, n.exportedSes
                if err != nil {
                        return err
                }
        } else {
                n.encryptedRandomSessionKey = n.keyExchangeKey
        }
        return nil
}

/********************************
 NTLM V1 Password hash functions
********************************/

func ntowfv1(passwd string) []byte {
        return md4(utf16FromString(passwd))
}

//      ConcatenationOf( DES( UpperCase( Passwd)[0..6],"KGS!@#$%"), DES( UpperCase( Pas
func lmowfv1(passwd string) ([]byte, error) {
        asciiPassword := []byte(strings.ToUpper(passwd))
        keyBytes := zeroPaddedBytes(asciiPassword, 0, 14)

        first, err := des(keyBytes[0:7], []byte("KGS!@#$%"))
        if err != nil {
                return nil, err
        }
        second, err := des(keyBytes[7:14], []byte("KGS!@#$%"))
        if err != nil {
                return nil, err
        }

        return append(first, second...), nil
}
```