**Problem**

<https://github.com/rclone/rclone/issues/4783>

<https://github.com/rclone/rclone/commit/193c30d57038017370594d5bc8ee9bc32580ddf2>

random.Password was factored out into lib/random.

At that time the library crypto/rand was accidentally replaced with math/rand leading to the pseudo random number generator being used instead of the crypto strong random number generator.

Graphical user interface, application

Description automatically generated

Consequences:

Callers of random.Password will have been getting a password based on math/rand instead of crypto/rand which reduces the amount of entropy for passwords enormously.

fs/config/config.go: Password = random.Password

* This is choosing random passwords for users in the config generator.
* This is a problem since users may have used these to configure services.

fs/rc/rcserver/rcserver.go: randomPass, err := random.Password(128)

* This is choosing short lived random passwords for use with the web ui.
* This is a minor problem since these passwords are regenerated every time rclone is run.

lib/oauthutil/oauthutil.go: state, err := random.Password(128)

* This is making some random state for the oauth callback.
* This isn't a security problem

Rclone initialised the seed of math/rand in cmd/cmd.go Main with rand.Seed(time.Now().Unix())

However, time.Now().Unix() only changes every second, meaning **passwords generated only change every second**. The passwords generated by random.Password are therefore completely determinstic based on the unix second that rclone was started.

If you run multiple copies of it at once which start at the same second, you can see that with a vulnerable rclone all the passwords generated are the same. Pass it an rclone binary to test (or leave off to use the one on the path)

$ ./test-rclone-password.sh rclone-v1.53.2 & ./test-rclone-password.sh rclone-v1.53.2 & ./test-rclone-password.sh rclone-v1.53.2

*Bits> Your password is: eULvaUR9A\_A*

*Bits> Your password is: eULvaUR9A\_A*

*Bits> Your password is: eULvaUR9A\_A*

**math/rand (problem for cryptography, not safe for concurrent use)**

<https://cs.opensource.google/go/go/+/refs/tags/go1.19.3:src/math/rand/rng.go>

// Seed uses the provided seed value to initialize the generator to a deterministic state.

// Seed should not be called concurrently with any other Rand method.

func (r \*Rand) Seed(seed int64) {

if lk, ok := r.src.(\*lockedSource); ok {

lk.seedPos(seed, &r.readPos)

return

}

r.src.Seed(seed)

r.readPos = 0

}

// Read generates len(p) random bytes and writes them into p. It always returns len(p) and a nil error.

// **Read should not be called concurrently with any other Rand method.**

func (r \*Rand) Read(p []byte) (n int, err error) {

if lk, ok := r.src.(\*lockedSource); ok {

return lk.read(p, &r.readVal, &r.readPos)

}

return read(p, r.src, &r.readVal, &r.readPos)

}

func read(p []byte, src Source, readVal \*int64, readPos \*int8) (n int, err error) {

pos := \*readPos

val := \*readVal

rng, \_ := src.(\*rngSource)

for n = 0; n < len(p); n++ {

if pos == 0 {

if rng != nil {

val = rng.Int63()

} else {

val = src.Int63()

}

pos = 7

}

p[n] = byte(val)

val >>= 8

pos--

}

\*readPos = pos

\*readVal = val

return

}

**crypto/rand (good for cryptography, safe for concurrent use)**

<https://cs.opensource.google/go/go/+/refs/tags/go1.19.3:src/crypto/rand/rand_getentropy.go>

// Read is a helper function that calls Reader.Read using io.ReadFull.

// On return, n == len(b) if and only if err == nil.

func Read(b []byte) (n int, err error) {

return io.ReadFull(Reader, b)

}

// ReadFull reads exactly len(buf) bytes from r into buf.

// It returns the number of bytes copied and an error if fewer bytes were read.

// The error is EOF only if no bytes were read.

// If an EOF happens after reading some but not all the bytes, ReadFull returns ErrUnexpectedEOF.

// On return, n == len(buf) if and only if err == nil.

// If r returns an error having read at least len(buf) bytes, the error is dropped.

func ReadFull(r Reader, buf []byte) (n int, err error) {

return ReadAtLeast(r, buf, len(buf))

}

// ReadAtLeast reads from r into buf until it has read at least min bytes.

// It returns the number of bytes copied and an error if fewer bytes were read.

// The error is EOF only if no bytes were read.

// If an EOF happens after reading fewer than min bytes,

// ReadAtLeast returns ErrUnexpectedEOF.

// If min is greater than the length of buf, ReadAtLeast returns ErrShortBuffer.

// On return, n >= min if and only if err == nil.

// If r returns an error having read at least min bytes, the error is dropped.

func ReadAtLeast(r Reader, buf []byte, min int) (n int, err error) {

if len(buf) < min {

return 0, ErrShortBuffer

}

for n < min && err == nil {

var nn int

nn, err = r.Read(buf[n:])

n += nn

}

if n >= min {

err = nil

} else if n > 0 && err == EOF {

err = ErrUnexpectedEOF

}

return

}

**random.go**

import (

"encoding/base64"

**"math/rand"**

func Password(bits int) (password string, err error) {

bytes := bits / 8

if bits%8 != 0 {

bytes++

}

var pw = make([]byte, bytes)

n, err := **rand.Read**(pw)

if err != nil {

return "", errors.Wrap(err, "password read failed")

}

**Fix**

<https://github.com/rclone/rclone/commit/7985df37681f54d013816a4641da4f9b085b3aa5>

**random.go**

import (

**cryptorand "crypto/rand"**

"encoding/base64"

**mathrand "math/rand"**

func Password(bits int) (password string, err error) {

bytes := bits / 8

if bits%8 != 0 {

bytes++

}

var pw = make([]byte, bytes)

n, err := **cryptorand.Read**(pw)

if err != nil {

return "", errors.Wrap(err, "password read failed")

}

**Super fix**

**Not a real fix but provides more entropy to math/rand**

**(Useful only if someone mistakenly uses math/rand)**

<https://github.com/rclone/rclone/commit/f0905499e340f9e73e2552cf0c8b79cbf14ecbc4>

// Seed the global math/rand with crypto strong data

// This doesn't make it OK to use math/rand in crypto sensitive

// environments - don't do that! However, it does help to mitigate the

// problem if that happens accidentally. This would have helped with CVE-2020-28924 - #4783

**func Seed() error {**

**var seed int64**

**err := binary.Read(cryptorand.Reader, binary.LittleEndian, &seed)**

**if err != nil {**

**return errors.Wrap(err, "failed to read random seed")**

**}**

**mathrand.Seed(seed)**

**return nil**

**}**

Graphical user interface, text

Description automatically generated