NetSec - Exercise 03

**Che-Hao Kang, Aman Azim**

May 24, 2016

**Task 3.1 (theoretical): Authentication Beyond Passwords**

**Part (a)**

***1. Biometrics:***

In this technique completely private information of a person is used to act as authentication media such as finger print, voice, heartbeat, eye, etc.

**Advantage:** Biometric information is unique. That is, two people can never have the same finger print but people can have same passwords.

**Disadvantage:** We can change a password thousand times but we cannot change our heartbeat or finger print. If somehow a hacker steals someone’s finger print then the person is as good as dead, he has to get new finger.

**Use:** Samsung Galaxy S6 edge uses this way to unlock the phone.

***2.Token:***

In this technique a user is provided with a unique piece of data such as a picture, a sound clip, etc. to act as authentication media.

**Advantage:** It can provide a second layer of authentication when used along with password which is easy and low cost and gives more security than using only password. The user can forget password but with token there is no chance for that.

**Disadvantage:** A digital media is required to carry the token.

**Use:** FlixBus (a travel bus company) provides its travelers with barcode tokens and authenticates them by reading the token with mobile phone camera. I know it as I use this bus service to travel every time.

***3.Temporary and single password:***

In this technology when a user wants to login to a service every time the service provider sends a temporary and new password directly to the user phone by SMS. This technology was proposed by Yahoo! in 2015.

**Advantage:** Users don’t need to remember their passwords and as the password is renewed every time a hacker cannot trace it.

**Disadvantage:** The phone which receives the password could be lost or stolen, then who ever have the phone can login as a real user.

**Use:** To access Yahoo! mail from 3rd parties by apps like iOS mail, Android mail or Outlook, Yahoo! requires users to use this technology.

**Part (b)**

***Two Factors:***

In this technique along with password a second media is used to provide strong authentication.

**Advantage:** Hackers may get my password but he/she cannot get the code number which is directly sent to my phone by the website or server of a particular organization as a second factor of my authentication.

**Disadvantage:** If the media that carries the second factor after password is somehow not accessible then a user cannot login to the particular service which can cause a big issue. However, in pure password based authentication the users carry their passwords in their mind which they can access any time if they are alive.

**Use:** While transferring money online, a user needs to use both of his/her bank pin code and TAN.

**Task 3.2 (theoretical): Reconnaissance in the SecLab**

⚫ We use **ifconfig** to find out ip addresses and corresponding subnet masks of the current SecLab computer. Based on the above information, we employ **sudo nmap -O ip\_address/subnet\_mask** to discover all computers under certain subnets. The information of computers consists of ip addresses, running services with ports and operating systems plus versions.

⚫ The script we use is as following ([Download](https://www.dropbox.com/s/k28hs0mcqrhj26t/exercise3_2.sh?dl=1)):

**+++++START+++++**

#!/bin/bash

rm -rf scanSummary.txt

**# use ifconfig to gather ip addresses and subnet masks of the current computer of SecLab**

addrOfThisHost=`ifconfig | grep "inet addr" | sed -r 's/^.\*inet addr:([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+).\*Mask:([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+).\*$/\1 \2/g' | grep -v "127.0.0.1"`

i=-1

for addr in $addrOfThisHost

do

i=$((i+1))

addrMaskInfo[$i]=$addr

done

index=0

while [ $index -lt $i ]

do

**# arrange the document output**

case "${addrMaskInfo[$((index+1))]}" in

"255.0.0.0") mask=8 addrMaskInfo[$index]=`echo ${addrMaskInfo[$index]} | sed -r 's/([0-9]+).\*/\1\.0\.0\.0/g'`

;;

"255.255.0.0") mask=16 addrMaskInfo[$index]=`echo ${addrMaskInfo[$index]} | sed -r 's/([0-9]+\.[0-9]+).\*/\1\.0\.0/g'`

;;

"255.255.255.0") mask=24 addrMaskInfo[$index]=`echo ${addrMaskInfo[$index]} | sed -r 's/([0-9]+\.[0-9]+\.[0-9]+).\*/\1\.0/g'`

;;

esac

**# output the ip address and subnet mask to scanSummary.txt**

echo -e "===${addrMaskInfo[$index]}/$mask===" >> scanSummary.txt

echo "+++=======START nmap=============+++"

**# scan computers under certain subnets and save the result of "nmap -O" to nmap.txt**

sudo nmap -O "${addrMaskInfo[$index]}/$mask" > nmap.txt

echo "---=======END nmap=============---"

serviceStart="n"

**# read nmap.txt line by line**

cat nmap.txt | while read line

do

echo $line | grep "Nmap scan report for"

**# discovered computers with ip addresses**

if [ "$?" == "0" ]; then

nowIP=`echo $line | sed -r 's/.\*Nmap scan report for ([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+).\*/\1/g'`

echo -e "$nowIP:" >> scanSummary.txt

echo -e "----------------" >> scanSummary.txt

continue

fi

echo $line | grep -E "PORT\s+STATE\s+SERVICE"

**# discovered computers with running services and ports**

if [ "$?" == "0" ]; then

serviceStart="y"

echo -e $line | sed -r 's/(.\*)\s+(.\*)\s+(.\*)/\1\t\t\2\t\3/g' >> scanSummary.txt

continue

fi

if [ "$serviceStart" == "y" ]; then

echo $line | grep "/"

if [ "$?" != "0" ]; then

serviceStart="n"

echo -e "" >> scanSummary.txt

continue

fi

port=`echo $line | sed -r 's/([0-9]+)\/.\*/\1/g'`

if [ "$((port/1000))" -gt "0" ]

then

echo -e $line | sed -r 's/(.\*)\s+(.\*)\s+(.\*)/\1\t\2\t\3/g' >> scanSummary.txt

else

echo -e $line | sed -r 's/(.\*)\s+(.\*)\s+(.\*)/\1\t\t\2\t\3/g' >> scanSummary.txt

fi

fi

echo $line | grep -E "OS\s+(CPE)\:"

**# discovered computers with operating systems and versions**

if [ "$?" == "0" ]; then

echo -e $line >> scanSummary.txt

continue

fi

echo $line | grep -E "OS\s+(details)\:"

if [ "$?" == "0" ]; then

echo -e $line >> scanSummary.txt

echo -e "" >> scanSummary.txt

continue

fi

done

echo -e "" >> scanSummary.txt

index=$((index+2))

done

**-----END-----**

⚫ All gathered information is as following:

**+++++START+++**

===10.0.0.0/24===

10.0.0.5:

----------------

PORT STATE SERVICE

22/tcp open ssh

OS CPE: cpe:/o:linux:linux\_kernel:3 cpe:/o:linux:linux\_kernel:4

OS details: Linux 3.2 - 4.4

10.0.0.10:

----------------

PORT STATE SERVICE

22/tcp open ssh

53/tcp open domain

OS CPE: cpe:/o:linux:linux\_kernel:3 cpe:/o:linux:linux\_kernel:4

OS details: Linux 3.2 - 4.4

10.0.0.11:

----------------

PORT STATE SERVICE

22/tcp open ssh

5432/tcp open postgresql

OS CPE: cpe:/o:freebsd:freebsd:7 cpe:/o:freebsd:freebsd:8 cpe:/o:freebsd:freebsd:9 cpe:/o:freebsd:freebsd:10

OS details: FreeBSD 7.0-RELEASE-p1 - 10.0-CURRENT

10.0.0.12:

----------------

PORT STATE SERVICE

22/tcp open ssh

80/tcp open http

4242/tcp open vrml-multi-use

OS CPE: cpe:/o:linux:linux\_kernel:3 cpe:/o:linux:linux\_kernel:4

OS details: Linux 3.2 - 4.4

10.0.0.13:

----------------

PORT STATE SERVICE

22/tcp open ssh

2049/tcp open nfs

OS CPE: cpe:/o:linux:linux\_kernel:2.6.32 cpe:/o:linux:linux\_kernel:3

OS details: Linux 2.6.32, Linux 2.6.32 - 3.10, Linux 2.6.32 - 3.13

10.0.0.42:

----------------

PORT STATE SERVICE

21/tcp open ftp

135/tcp open msrpc

139/tcp open netbios-ssn

445/tcp open microsoft-ds

OS CPE: cpe:/o:microsoft:windows\_xp::sp2 cpe:/o:microsoft:windows\_xp::sp3

OS details: Microsoft Windows XP SP2 or SP3

10.0.0.99:

----------------

PORT STATE SERVICE

22/tcp open ssh

2222/tcp open EtherNetIP-1

OS CPE: cpe:/o:linux:linux\_kernel:3 cpe:/o:linux:linux\_kernel:4

OS details: Linux 3.2 - 4.4

10.0.0.1:

----------------

PORT STATE SERVICE

22/tcp open ssh

111/tcp open rpcbind

139/tcp open netbios-ssn

445/tcp open microsoft-ds

OS CPE: cpe:/o:linux:linux\_kernel:3 cpe:/o:linux:linux\_kernel:4

OS details: Linux 3.8 - 4.4

===10.1.1.0/24===

10.1.1.1:

----------------

PORT STATE SERVICE

22/tcp open ssh

OS CPE: cpe:/o:linux:linux\_kernel:3 cpe:/o:linux:linux\_kernel:4

OS details: Linux 3.2 - 4.4

10.1.1.2:

----------------

PORT STATE SERVICE

22/tcp open ssh

111/tcp open rpcbind

139/tcp open netbios-ssn

445/tcp open microsoft-ds

OS CPE: cpe:/o:linux:linux\_kernel:3.19 cpe:/o:linux:linux\_kernel:4

OS details: Linux 3.19, Linux 3.8 - 4.4

**-----END-----**

From the above, we can see **“OS CPE: cpe:/o:linux:linux\_kernel:3 cpe:/o:linux:linux\_kernel:4”**. This represents the operating system of the computer is Linux and the kernel version is 3-4 (similar to **uname -r**).

**Bonus:**

Based on the hints, we tried to connect to every service of each computer by using **nc ip\_address port**. When communicating with **10.0.0.12** by port **4242 (nc 10.0.0.12 4242)**, we got

**HELO**

**201 OK**

**This is a beautiful red-yellow-green-white-black-hat bonbon!**

**Task 3.3 (practical): DNS sniffing**

⚫ In order to spoof a DNS response, we need to intercept DNS packets to know which computers sent out DNS queries. Based on this, we are able to mimic real DNS responses, insert malicious material inside and send fake DNS responses back to those computers.

There are two types of DNS queries, **IPv4** and **IPv6:**

In **IPv4**, the DNS query is like

**12:08:09.371817 IP 10.0.0.5.33487 > 10.0.0.10.domain: 56519+ A? net.cs.uni-bonn.de. (36)**

**12:08:09.372753 IP 10.0.0.10.domain > 10.0.0.5.33487: 56519\*- 1/0/0 A 131.220.242.41 (52)**

The computer **10.0.0.5** sent out an IPv4 DNS query about **net.cs.uni-bonn.de** to the DNS server **10.0.0.10**. Then, the DNS server **10.0.0.10** replied the ip address of **net.cs.uni-bonn.de** to the computer **10.0.0.5**.

In **IPv6**, the DNS query is like

**12:12:31.442316 IP eduroam-203-130.wlan.uni-bonn.de.58590 > nic.rhrz.uni-bonn.de.domain: 54311+ AAAA? plus.google.com. (33)**

**12:12:31.443523 IP nic.rhrz.uni-bonn.de.domain > eduroam-203-130.wlan.uni-bonn.de.58590: 54311 1/4/4 AAAA 2a00:1450:4001:810::200e (197)**

The computer **eduroam-203-130.wlan.uni-bonn.de** sent out an IPv6 DNS query about **plus.google.com** to the DNS server **nic.rhrz.uni-bonn.de.domain**. Then, the DNS server **nic.rhrz.uni-bonn.de.domain** replied the ip address of **plus.google.com** to the computer **eduroam-203-130.wlan.uni-bonn.de**.

⚫ We use **"sudo tcpdump -l udp”** to gather DNS packets and use regular expression to filter out the information we want. The source code is as following ([Download](https://www.dropbox.com/s/bwd17v67gwqjsnp/exercise3.3.py?dl=1)):

**+++++START+++++**

#!/usr/bin/env python

import re

import os

import sys

from subprocess import Popen, PIPE, STDOUT

if \_\_name\_\_ == '\_\_main\_\_':

try:

**# filter out IPv4 DNS query**

reStringIPv4Req = '^.\*?IP\s+([0-9a-zA-Z\.]+)\s+>\s+([0-9a-zA-Z\.]+).\*?\s+A\?\s+([0-9a-zA-Z\.]+)\..\*?'

**# filter out IPv6 DNS query**

reStringIPv6Req = '^.\*?IP\s+([0-9a-zA-Z\.]+)\s+>\s+([0-9a-zA-Z\.]+).\*?AAAA\?\s+([0-9a-zA-Z\.]+)\..\*?'

**# filter out IPv4 DNS response**

reStringIPv4Resp = '^.\*?IP\s+([0-9a-zA-Z\.]+)\s+>\s+([0-9a-zA-Z\.]+).\*?\s+A\s+([0-9a-zA-Z\.]+)\s+.\*?'

**# filter out IPv6 DNS response**

reStringIPv6Resp = '^.\*?IP\s+([0-9a-zA-Z\.]+)\s+>\s+([0-9a-zA-Z\.]+).\*?\s+AAAA\s+([0-9a-zA-Z\:]+)\s+.\*?'

**# use "sudo tcpdump -l udp" to gather UDP packets**

p = Popen(["sudo", "tcpdump", "-l", "udp"], stdout=PIPE, stderr=STDOUT)

**# use regular expressions to filter out related information**

for line in iter(p.stdout.readline, b''):

reobj = re.compile(reStringIPv4Req, re.IGNORECASE)

m = reobj.finditer(line)

for i in m:

print ("++reStringIPv4Req+")

print i.group(1), i.group(2), i.group(3)

print ("---")

reobj = re.compile(reStringIPv4Resp, re.IGNORECASE)

m = reobj.finditer(line)

for i in m:

print ("++reStringIPv4Resp+")

print i.group(1), i.group(2), i.group(3)

print ("---")

reobj = re.compile(reStringIPv6Req, re.IGNORECASE)

m = reobj.finditer(line)

for i in m:

print ("++reStringIPv6Req+")

print i.group(1), i.group(2), i.group(3)

print ("---")

reobj = re.compile(reStringIPv6Resp, re.IGNORECASE)

m = reobj.finditer(line)

for i in m:

print ("++reStringIPv6Resp+")

print i.group(1), i.group(2), i.group(3)

print ("---")

print line, "\n"

p.wait() **# wait for the subprocess to exit**

except:

print "Unexpected error:", sys.exc\_info()[0]

**-----END-----**

⚫ The sample output is as following:

**+++++START++++**

**++reStringIPv4Req+**

10.0.0.5.53239 10.0.0.10.domain cve.mitre.org

**---**

**++reStringIPv4Resp+**

10.0.0.10.domain 10.0.0.5.53239 192.52.194.135

**---**

**++reStringIPv4Req+**

10.0.0.5.44604 10.0.0.10.domain packetstormsecurity.com

---

**++reStringIPv4Resp+**

10.0.0.10.domain 10.0.0.5.44604 198.84.60.198

**---**

**-----END-----**

The computer **10.0.0.5** sent DNS query about **cve.mitre.org** to **10.0.0.10** and then got the response **cve.mitre.org is 192.52.194.135** from **10.0.0.10**.

The computer **10.0.0.5** sent DNS query about **packetstormsecurity.com** to **10.0.0.10** and then got the response **packetstormsecurity.com is 198.84.60.198** from **10.0.0.10**.

**Task 3.4 (practical): Hash Collisions**

⚫ The source code is as following ([Download](https://www.dropbox.com/s/prd32foe5r1bfrr/exercise3.4.py?dl=1)):

**+++++START++++**

#!/usr/bin/env python

import re

import os

import sys

import random

import hashlib

from subprocess import Popen, PIPE, STDOUT

FLAG\_1 = 0b1111

FLAG\_2 = 0b11111111

FLAG\_3 = 0b111111111111

**# use SHA256 to check bit coincidence. "numOfBits" identifies how many bits must be the same.**

def checkSame(seq1, seq2, numOfBits):

for i in xrange(numOfBits/4):

if (int(hashlib.sha256(seq1).hexdigest()[i], 16) & FLAG\_1) != \

(int(hashlib.sha256(seq2).hexdigest()[i], 16) & FLAG\_1):

return False

return True

if \_\_name\_\_ == '\_\_main\_\_':

**# Open a file for saving all used sequences**

dataFile = open("exercise3\_4\_data.txt", "wb")

**# a dictionary for storing counters for 4, 8, 12, 16 and 20 bits**

allCounter = dict()

allCounter[4] = []

allCounter[8] = []

allCounter[12] = []

allCounter[16] = []

allCounter[20] = []

**# we run this collision programming 10 times for 4, 8, 12, 16 and 20 bits**

for times in xrange(10):

counter = dict()

counter[4] = 0

counter[8] = 0

counter[12] = 0

counter[16] = 0

counter[20] = 0

**# this list is used to record which one doesn't find the collision yet**

bitsList = [4, 8, 12, 16, 20]

**# This counter is used to record how many times for a prefix to find a collision**

counterRun = 0

going = True

while going:

counterRun += 1

**# Generate random sequences**

randomSequence1 = open("/dev/urandom", "rb").read(64)

randomSequence2 = open("/dev/urandom", "rb").read(64)

dataFile.write("===Sequence 1===\n")

dataFile.write(randomSequence1 + "\n")

dataFile.write("===Sequence 2===\n")

dataFile.write(randomSequence2 + "\n\n")

deletedBits = []

**# Iterate through each prefix**

for i in bitsList:

**# Check if certain prefixes have collisions**

if counter[i] == 0 and checkSame(randomSequence1, randomSequence2, i):

print "Collision of", i, "bits:", counterRun

counter[i] = counterRun

deletedBits.append(i)

print (hashlib.sha256(randomSequence1).hexdigest())

print (hashlib.sha256(randomSequence2).hexdigest() + "\n")

**# Remove prefixes which alreay got collisions**

for bits in deletedBits:

bitsList.remove(bits)

if not bitsList:

going = False

print counter

**# Print out all collision information**

for bits in xrange(4, 24, 4):

allCounter[bits].append(counter[bits])

print allCounter

dataFile.close()

**-----END-----**

**⚫** The following is information about collision:

**+++++START+++++**

**Collision of 4 bits: 49**

**e**52c052fb76b26886b42c5d2e89ad156ed4dd44cf967fe5ff38a234c093375c9

**e**ee06336ec252c85e9d227e449b331554b16626bd4bd4c650b29e9ce79b1fec6

**Collision of 8 bits: 295**

**6c**6a7419920d42016402c4916cf3dca8659688db91f4310d0ba3b8f035d471fd

**6c**f060b9abdf9bf0cc80288d8aecde33efbecbcf07ebb8a1be177fb6aaf89702

**Collision of 12 bits: 2653**

**f5f**484205339231e503a27857a3a9f44eca7c768491f7e723a06941f208518bd

**f5f**9865102cf6122b892f0fc20ae34bcb35bfd10f4fb26cd0f73fbb636050d64

**Collision of 16 bits: 52310**

**8b3f**28393912ba1e7565f5f3657aa2603695b2eee43c5caf18f5eef0e5b8cca2

**8b3f**adc2e63fa9cb7cdbbac85a037678a32b1ae8cb6fe36136d611bc074cce18

**Collision of 20 bits: 185200**

**7ab94**908953eaf811f9813e2f867e0e0c8218b8e4e4016547492b89a4ce74023

**7ab94**15c963b2ab503286ed789e65afeccf4c7397ee0c2b09eb88daef1aae256

**{4: 49, 8: 295, 12: 2653, 16: 52310, 20: 185200}**

**-----END-----**

From the above, we know that in order to find a collision of **the 4-bit prefix**, we need to use **49 times** and vice versa for other prefixes.

Also, we display some random sequence pairs which are not correlated to each other:

**===Sequence 1===**

^\v^[聶JZV?繭;<84>Q?@PD<9d>#^YK竅S<9c>g<9f>?0?礙?

繭?繭\-{?<95>^MF簿mh羸/<84>H繭^P1<97>c繒穡!?<82><98>羹簿穢?癒

**===Sequence 2===**

)繞?i^Wq\<8f>?~瞿.^Vs瞿i1<96>J2<95>簽R?<9a>^^繭<82>4jJ^Ri=籀dN竄穢]u6<86>簣y?瓣R

!a^BhF?穡NhI%瞿H繩G

**===Sequence 1===**

m^W^\_^F?<9a>U+?簣Z2)<97>`簧a繞簞穠<8e>j~04sR\*R^LE^C穡^XP^\_^NR疇^\?????簸jF穠^\_<86>S\?G??~S(C-<96>^\_

**===Sequence 2===**

?x繞v^M^<91>?;穠z瞿簣?繡e<95>^A^E<96>#?職2<9f><93>?繭^Lv^N^\_繡簞5\*<9a>?簷<8e><82>簣臘:?藩A疆^P

簷CT^P@繡糧y^C?^[[OK

⚫ In the end, we ran this program **ten times** to generate ten different counters for each prefix to encounter a collision. The result is as following:

{

**4: [49, 28, 40, 30, 5, 12, 26, 9, 11, 1],**

**8: [295, 168, 569, 113, 198, 318, 987, 32, 91, 38],**

**12: [2653, 6663, 5051, 1801, 4939, 3379, 11191, 1667, 1535, 3331],**

**16: [52310, 27417, 61383, 50101, 20019, 65600, 161608, 14938, 72832, 67439],**

**20: [185200, 476022, 309090, 4899040, 212291, 640868, 2397083, 1484002, 923254, 183082]**

}

So, the averages for each prefix to have a collision are:

**bits times**

**4 21.1**

**8 280.9**

**12 4221**

**16 59364.7**

**20 1170993.2**

The plotting script is the below ([Download](https://www.dropbox.com/s/ksnn5b0b2kh4ht1/exercise3_4.R?dl=1)):

**+++START+++**

library(package = "lattice");

**# Import the data**

data <- read.table("E:/Dropbox/University\_Bonn/Summer\_Semester\_2016/Network\ Security/Exercise\_03/exercise3\_4\_counters.txt", header = TRUE);

attach(data);

**# draw the line based on the data**

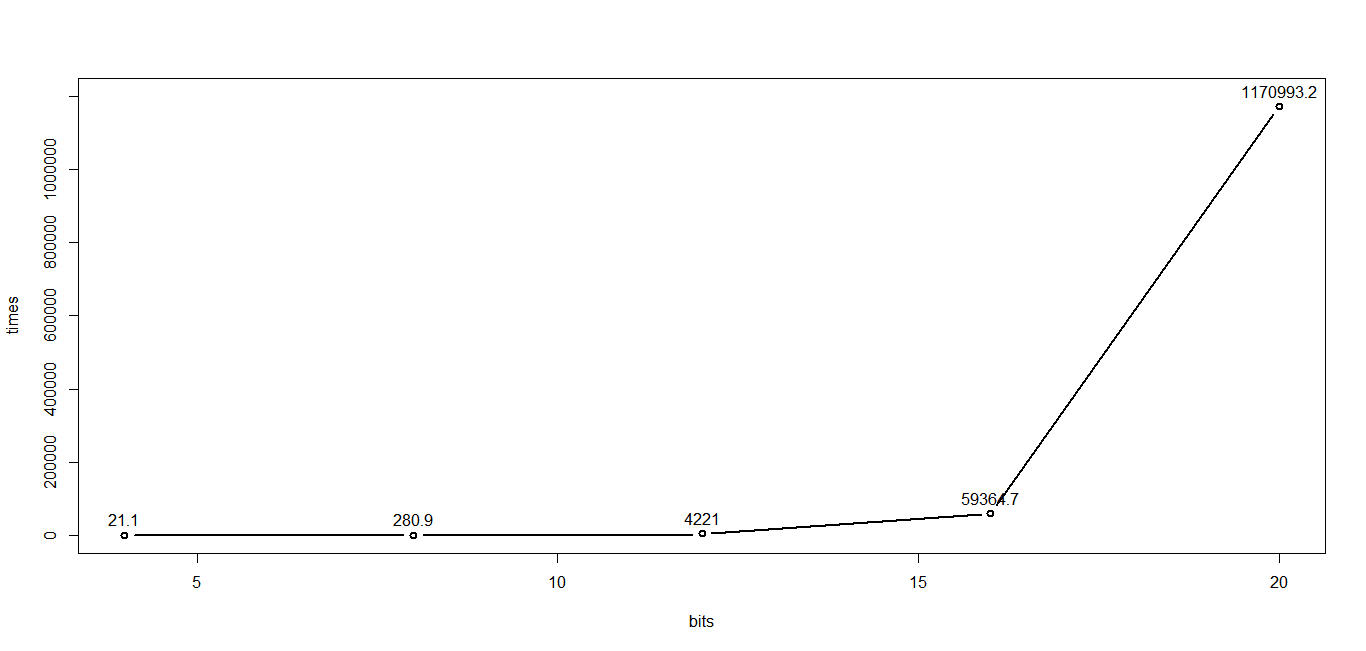
plot(bits, times, ylim = c(0, 1200000), type="b", lwd="2")

**# Show the value directly beside the point**

text(bits, times, labels=round(times,2),pos=3)

**-----END-----**

⚫ The output of the plotting script:

We can see that in the **20-bit prefix**, the number of trials to have a collision rocketed to **1,170,993.2** which means it will get harder and harder to find a collision when the number of bits get bigger.

**Task 3.5 (theoretical): Designing Asymmetric Encryption Schemes**

**Part (a):**

***Is this method secure?***

Yes this method seems secure to me because every time when the secret box is traveling, it is protected with at-least one padlock.

***Does it also work with cryptographic means?***

It works with cryptographic means.

***Which problems could arise?***

From my point of view, the main problem is time that it takes to complete the whole process.

Also, to transfer the box between the sender and receiver two times will include more cost.

**Part (b):**

***Does it work?***

Yes.

Example:

Assume, Digital data = 11 00 00 01

Bob’s key = 11 11 00 00

Alice’s key = 00 00 11 11

**Step 1:** Bob sends the data by doing bitwise XORs with his key to Alice

Digital data ^ Bob’s key = 11 00 00 01 ^ 11 11 00 00 = 00 11 00 01= Encrypted data.

**Step 2:** Alice receives the encrypted data and performs again bitwise XORs on it with her key

Encrypted data ^ Alice’s key=00 11 00 01 ^ 00 00 11 11= 00 11 11 10 = Encrypted data with both keys.

**Step 3:** Now Alice sends the double-key-encrypted data to Bob again so that, he can remove his lock by doing bitwise XORs on it again with his key.

Encrypted data with both keys ^ Bob’s key = 00 11 11 10 ^ 11 11 00 00 = 11 00 11 10 =Encrypted data with only Alice’s key.

**Step 4:** Now Bob sends the data back to Alice again where she performs bitwise XORs on it with her key and gets the actual data.

Encrypted data with only Alice’s key ^ Alice’s key = 11 00 11 10 ^ 00 00 11 11 =

11 00 00 01 = Actual Digital Data.

***Can confidentiality be assured?***

From my point of view, it is not completely assured. Because a hacker can pose as Alice and perform her role and Bob has no way to identify Alice because Bob doesn’t have any information about Alice’s key.

***Can integrity be assured?***

Yes. but, if during transmission any bit of the data is lost. Then while performing XORs with a key, the decrypted data will not match the right data.

***Would choosing different random keys for each message have an impact?***

Choosing random key for each message will increase the security because then the hacker cannot trace them. Hackers can only know the key length as it is at least the data size.