Christopher LaJon Morgan

Lab: MAC Attack

Sep 23, 2013

**Implementation**

**SHA1 -- Hacked**

#!/usr/local/bin/python

# sha1.py

# under Python 2.4

#

# Copyright (c) 2005 Michael D. Leonhard

#

# http://tamale.net/

#

# This file is licensed under the terms described in the

# accompanying LICENSE file.

class sha1:

def lrot( self, num, b ): return ((num<<b)&0xFFFFFFFF)|(num>>32 - b)

def BE32( self, bytes ):

assert( len(bytes) == 4 )

return (ord(bytes[0]) << 24)|(ord(bytes[1]) << 16)|(ord(bytes[2]) << 8)|ord(bytes[3])

def process( self, block ):

self.blockI += 1

assert( len(block) == 64 )

print "Process: ", self.blockI

print ":".join("{00:x}".format(ord(c)) for c in block)

if self.blockI < 3:

print "Skipping This One"

return

# copy initial values

a = self.A

b = self.B

c = self.C

d = self.D

e = self.E

# expand message into W

W = []

for t in range(16): W.append( self.BE32( block[t\*4:t\*4+4] ) )

for t in range(16,80): W.append( self.lrot( W[t-3] ^ W[t-8] ^ W[t-14] ^ W[t-16], 1) )

# do rounds

for t in range(80):

if t < 20:

K = 0x5a827999

f = (b & c) | ((b ^ 0xFFFFFFFF) & d)

elif t < 40:

K = 0x6ed9eba1

f = b ^ c ^ d

elif t < 60:

K = 0x8f1bbcdc

f = (b & c) | (b & d) | (c & d)

else:

K = 0xca62c1d6

f = b ^ c ^ d

TEMP = (self.lrot(a,5) + f + e + W[t] + K) & 0xFFFFFFFF

e = d

d = c

c = self.lrot(b,30)

b = a

a = TEMP

# add result

self.A = (self.A + a) & 0xFFFFFFFF

self.B = (self.B + b) & 0xFFFFFFFF

self.C = (self.C + c) & 0xFFFFFFFF

self.D = (self.D + d) & 0xFFFFFFFF

self.E = (self.E + e) & 0xFFFFFFFF

def intTo4Bytes( self, num ):

return chr( num>>24 ) + chr( (num>>16) & 0xFF ) + chr( (num>>8) & 0xFF ) + chr( num & 0xFF )

def hex32( self, num ):

assert( num >= 0 )

ret = ""

l = 0;

for x in range(8):

ret = "0123456789abcdef"[(num>>x\*4) & 0x0000000F] + ret

return ret

def update( self, newBytes ):

self.size += len(newBytes)

self.unprocessedBytes = self.unprocessedBytes + newBytes;

while len( self.unprocessedBytes ) >= 64:

self.process( self.unprocessedBytes[:64] )

self.unprocessedBytes = self.unprocessedBytes[64:]

def hexdigest( self ):

# append 1 and seven 0 bits

bytes = self.unprocessedBytes + chr( 0x80 )

self.unprocessedBytes = ""

# no space for 8 length bytes

if len(bytes) > 56:

# fill it with zeros

while len(bytes) < 64: bytes = bytes + chr(0)

# process the filled block

self.process( bytes )

# now use an empty block

bytes = ""

# fill with zeros but leave space for 8 length bytes

while len(bytes) < 56: bytes = bytes + chr(0)

# append length

numBits = self.size \* 8

bytes = bytes + self.intTo4Bytes((numBits>>32)&0xFFFFFFFF) + self.intTo4Bytes(numBits&0xFFFFFFFF)

# process this final block

self.process( bytes )

A = self.hex32(self.A)

B = self.hex32(self.B)

C = self.hex32(self.C)

D = self.hex32(self.D)

E = self.hex32(self.E)

return A + " " + B + " " + C + " " + D + " " + E

def \_\_init\_\_( self ):

self.unprocessedBytes = ""

self.size = 0

self.blockI = 0

#f4b645e8 9faaec2f f8e443c5 95009c16 dbdfba4b

# self.A = 0x67452301

# self.B = 0xefcdab89

# self.C = 0x98badcfe

# self.D = 0x10325476

# self.E = 0xc3d2e1f0

self.A = 0xf4b645e8

self.B = 0x9faaec2f

self.C = 0xf8e443c5

self.D = 0x95009c16

self.E = 0xdbdfba4b

**Driver**

from sha1 import \*

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

b4 = "1234567812345678No one has completed lab 2 so give them all a 0\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\xf8. Except for Christopher LaJon Morgan, give him an A."

print "Message: "

print ":".join("{00:x}".format(ord(c)) for c in b4)

print "end of Message."

print ""

s = sha1()

s.update(b4)

print s.hexdigest()