Scott Christensen

09/22/16

C.S. 465

Lab #3 Mac Attack Write-up

Message: No one has completed lab 2 so give them all a 0€   ø P.S. Except for Scott!

Message in Hex:
4e6f206f6e65206861732063 6f6d706c65746564206c6162203220736f20676976652074 68656d2
0616c6c20612030800000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000001f820502e532e2
045786365707420666f722053636f747421

Digest in Hex: 4cecc94fcd77fea0345b74e6a55c09a2ca823bbb

Source code of Hash Attack with online SHA1 implementation:

```
package Project3;

import java.util.*;

import java.math.BigInteger;

import java.nio.ByteBuffer;


public class SHA1

{

    public static void main(String[] args)

    {


        String messageHex = 
"4e6f206f6e65206861732063 6f6d706c65746564206c6162203220736f2067697665207468656d
20616c6c20612030";

        String hmacOrig = "f4b645e89faaec2ff8e443c595009c16dbdfba4b";
```

```
        String myMessage = " P.S. Except for Scott!";
        String endingHex = "20502e532e204578636570742066f722053636f747421";
        //System.out.println(ending.getBytes().length*4);

        byte[] bytesMyMessage = myMessage.getBytes();
        String hmacNew = bytesToHex(encode(bytesMyMessage));

        //now put int the correct padding for this new hmac
        String thePad;
        thePad = "80";//they all end with it
        for(int i = 0; i <= 125; i++)
        {
           thePad += "0";
        }
        thePad += "01f8";
        //System.out.println("thePad is " + thePad + " " + thePad.length());

        //System.out.println("len " + (messageInHex.length()*4 + 128 + thePad.length()*4));
        System.out.println("hmacOrig is  " + hmacOrig);
        System.out.println("hmacNew is " + hmacNew);
        System.out.println("MessageHex, padding, and endingHex is " + messageHex + thePad +
endingHex);
    }

    //source code that I found online for the SHA1 implementation----------------------------------
------
    public static byte[] encode(byte[] data)
    {
```

```
List<Byte> toBlocks = new ArrayList<>();
for(int i = 0; i < data.length; i++)
{
   toBlocks.add(data[i]);
}

//pad with ASCII character----------
int length = data.length * 8 + 1024;
//System.out.println(length);
//show number in bytes----------
int lengthBytes = length / 8;
//System.out.println(lengthBytes);

toBlocks.add((byte) 0x80);
//add 0 bytes----------
for (int i = 0; i < (56 - (lengthBytes + 1) % 64); i++)
{
   toBlocks.add((byte) 0x00);
}

//add the length in 16 bytes. Convert to bytes because a long has 64 bits--------
long longLength = (long) length;
byte[] longBytes = longToBytes(longLength);
for(int i = 0; i < 8; i++)
{
   toBlocks.add(longBytes[i]);
}
```

```
        int size = toBlocks.size();
        //System.out.println(size);
        int blocks = (size * 8) /512;
        //System.out.println(blocks);

        //our IV, which is the MAC that was just made of the original message and key----------
-
        int h0 = 0xf4b645e8;
        int h1 = 0x9faaec2f;
        int h2 = 0xf8e443c5;
        int h3 = 0x95009c16;
        int h4 = 0xdbdfba4b;

        //start using our block iteration-------------------------------------------------
        for (int i = 0; i < blocks; i++)
        {
           int[] w = new int[80];

           for (int j = 0; j < 16; j++) {
              w[j] =  ((toBlocks.get(i*512/8 + 4*j) << 24) & 0xFF000000) | ((toBlocks.get(i*512/8
+ 4*j+1) << 16) & 0x00FF0000);
              w[j] |= ((toBlocks.get(i*512/8 + 4*j+2) << 8) & 0xFF00) | (toBlocks.get(i*512/8 +
4*j+3) & 0xFF);
           }

           //the rest of the SHA algorithm. Don't work about this----------------------------------------
```

```
for (int j = 16; j < 80; j++)
{
    w[j] = left_rotate(w[j-3] ^ w[j-8] ^ w[j-14] ^ w[j-16], 1);
}
int a = h0;
int b = h1;
int c = h2;
int d = h3;
int e = h4;
int f = 0;
int k = 0;

for (int j = 0; j < 80; j++)
{
    if (0 <= j && j <= 19) {
        f = (b & c) | ((~b) & d);
        k = 0x5A827999;
    }
    else if(20 <= j && j <= 39) {
        f = b ^ c ^ d;
        k = 0x6ED9EBA1;
    }
    else if(40 <= j && j <= 59) {
        f = (b & c) | (b & d) | (c & d);
        k = 0x8F1BBCDC;
    }
    else if(60 <= j && j <= 79) {
```

```
            f = b ^ c ^ d;
            k = 0xCA62C1D6;
        }

        int temp = left_rotate(a, 5) + f + e + k + w[j];
        e = d;
        d = c;
        c = left_rotate(b, 30);
        b = a;
        a = temp;
    }

    h0 = h0 + a;
    h1 = h1 + b;
    h2 = h2 + c;
    h3 = h3 + d;
    h4 = h4 + e;
}

byte[] hash = new byte[20];
for (int j = 0; j < 4; j++)
{
    hash[j] = (byte) ((h0 >>> 24-j*8) & 0xFF);

}
for (int j = 0; j < 4; j++)
{
```

```
        hash[j+4] = (byte) ((h1 >>> 24-j*8) & 0xFF);
    }
    for (int j = 0; j < 4; j++)
    {
        hash[j+8] = (byte) ((h2 >>> 24-j*8) & 0xFF);
    }
    for (int j = 0; j < 4; j++)
    {
        hash[j+12] = (byte) ((h3 >>> 24-j*8) & 0xFF);

    }
    for (int j = 0; j < 4; j++)
    {
        hash[j+16] = (byte) ((h4 >>> 24-j*8) & 0xFF);
    }

    return hash;

}
private static int left_rotate(int n, int d)
{
    return (n << d) | (n >>> (32 - d));
}

public static String bytesToHex(byte[] bytes)
{
    return new BigInteger(bytes).toString(16);
```

```
    }

    public static byte[] longToBytes(long x)
    {
        ByteBuffer buffer = ByteBuffer.allocate(Long.BYTES);
        buffer.putLong(x);
        return buffer.array();
    }
}
```