UNIVERSITY OF MARYLAND
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ENEE 459C
Computer Security
Instructor: Charalampos Papamanthou

Homework 1

Out: 09/11/14 Due: 09/19/14 11:59pm

Instructions

1. Strictly adhere to the University of Maryland Code of Academic Integrity.
2. Submit your solutions as a pdf document at Canvas. Include your full name in the solutions document. Name the solutions document as x-hw1.pdf, where x is your last name.

Problem 1 (50 points) As studied in class, a one-time pad can be used to encrypt one message only. Encrypting more than one message with the same key will start to leak information. In this problem, your goal will be to decrypt multiple messages encrypted with the same one-time pad, but without the key.

The following are hexadecimal representations of the encrypted messages. The length of all messages and the key is 52 bytes each. All the original plaintext messages are valid English text.

Cipher Text of Message 1:
737f68797e691a10020c1d6e75100c7009651d1d6561060c6d0b
6f046116650c031e1c001c09130a001b0163016569637a626f72

Cipher Text of Message 2:
6e6f7c6f78001d00180172701400167700e720b1d656f06690e17
7900150b671f0e020d6917680a0b791c6e0a0b006f676b636673

Cipher Text of Message 3:
6378737a7e6f091717151a69167311650268010714751111a6d17
651c18646f036f1f04741c0d0c0f74060d10656d65797e66730e

You are given that the following four words are expected to be in the plain text of some of the messages (note that all plaintext messages are already in UPPER CASE):

CRYPTOGRAPHIC - PASSWORDS - DECIPHER - MATHEMATICS

1. Argue with an example (that you devise yourself) why using one-time pad multiple times can leak information, and how this can generally be useful for an attacker.

2. Describe how knowing certain words in the plaintext may even enable an attacker to recover some other portions that could be important. Think about small examples. Imagine two simple expressions in some war context:

   LOSS: 50 SOLDIERS
   WITHDRAW TOMORROW

   If these two messages are encrypted with the same one-time pad, and an adversary knows that the words ”SOLDIERS” and ”WITHDRAW” may appear in the actual plaintext, how can this be useful for the adversary to infer the rest of the plaintext?

3. Given your thoughts about point 2, decrypt as much as possible from the encrypted messages listed earlier. You will need to write some code in a programming language of your choice to help you decrypt the messages. The code is not required to automatically decrypt the messages directly (this is possible, but it can be challenging). It can be used as an intermediate stage to help you get to the right solution.

Your submission should include the code you used, its output, and a complete description in the report for how it was used to decrypt the messages. Providing the decrypted messages directly will result in no points for this item. It is required to illustrate your approach and effort.

Problem 2 (50 points) A brief refresher on the OpenSSL Heartbleed bug: The bug is caused by improper handling of the Heartbeat Extension packets in OpenSSL that allows an attacker to send malformed heartbeat packets to a server. The intended purpose of exchanging such packets is for servers and clients to see if connections are alive. However, the bug lets malformed packets trigger a buffer over-read through which an attacker can download normally inaccessibe process memory. This has the unfortunate effect of potentially leaking a server's private encryption keys and other highly sensitive data. Check out this XKCD comic for a more visual reference of how the bug works.

1. Go to https://filippo.io/Heartbleed/ to test a server for Heartbleed. For an initial example, paste in “heartbleed.insign.ch” and make sure to tick the “ignore certificates” option. Copy the outputted 80 bytes of leaked data into your solution document. Also test a server of your own choosing, your bank server perhaps, and hope it's secure. You don't have to tick the “ignore certificates” option in this case. Include your choice of server in your answers and report whether or not it is vulnerable to Heartbleed.
2. Let's take a look at the actual code that introduced the bug. Read the code carefully (elipses represent lines of code skipped for this exercise) and explain what you think is the vulnerability. Where exactly (give a line number) and how does the code allow a buffer over-read attack? Also provide a suggestion on how to fix the code. Give your own idea first, including code, then find the official bug fix online and explain how it eliminates the vulnerability. You may want to include the official fixed code in your explanation as well.

```
int
dtls1_process_heartbeat(SSL *s)
    {
    unsigned char *p = &s->s3->rrec.data[0], *pl;
    unsigned short hbtype;
    unsigned int payload;
    unsigned int padding = 16; /* Use minimum padding */

...


/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;

if (s->msg_callback)
    s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
         &s->s3->rrec.data[0], s->s3->rrec.length,
         s, s->msg_callback_arg);

if (hbtype == TLS1_HB_REQUEST)
    {
    unsigned char *buffer, *bp;
    int r;

    /* Allocate memory for the response, size is 1 byte
     * message type, plus 2 bytes payload length, plus
     * payload, plus padding
     */
    buffer = OPENSSL_malloc(1 + 2 + payload + padding);
    bp = buffer;

    /* Enter response type, length and copy payload */
    *bp++ = TLS1_HB_RESPONSE;
    s2n(payload, bp);
    memcpy(bp, pl, payload);

    r = ssl3_write_bytes(s, TLS1_RT_HEARTBEAT, buffer, 3 + payload +
        padding);

...
```