UNIVERSITY OF MARYLAND
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ENEE 459C
Computer Security
Instructor: Charalampos Papamanthou

Homework 1

Out: 09/11/15 Due: 09/18/15 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 (20 points) In class we talked about two different techniques for ensuring the *recoverability* of n blocks $b_1, b_2, \ldots, b_n$. One that does simple replication and one that uses error-correcting codes. Both techniques store $2n$ blocks, instead of n. In the first technique (simple replication), what is the minimum number of blocks that the adversary can delete in order to make some b_i irretrievable? How about in the second technique? Explain.

Problem 2 (40 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:
6e6f7c6f78001d00180172701400167 70e720b1d656f06690e17
7900150b671f0e020d6917680a0b791c6e0a0b006f676b636673

Cipher Text of Message 3:
6378737a7e6f091717151a69167311650268010714751 11a6d17
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 3 (20 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 visual reference of how the bug works.

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.

```
1 int
2 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);

...
```

Problem 4 (20 points—*taken from SEED labs* `http://www.cis.syr.edu/~wedu/seed/`) On September 24, 2014, a severe vulnerability in `Bash` (`http://en.wikipedia.org/wiki/Bash_(Unix_shell)`) was identified. Nicknamed Shellshock, this vulnerability can exploit many systems and be launched either remotely or from a local machine. For this problem, you will launch an attack that exploits the Shellshock vulnerability on a remote web server. First you need to download and install the latest Virtual Machine Software provided from the following website: `http://www.cis.syr.edu/~wedu/seed/lab_env.html`

We now summarize the Shellshock vulnerability. Many web servers enable CGI, which is a standard method used to generate dynamic content on Web pages and Web applications. Many CGI programs are written using shell script. Therefore, before a CGI program is executed, the shell program must be invoked first, and such an invocation is triggered by a user from a remote computer.

Step 1: Set up the CGI Program. You can write a very simple CGI program (called `myprog.cgi`) like the following. It simply prints out `"Hello World"` using shell script.

```
\#!/bin/bash

echo "Content-type: text/plain"
echo
echo
echo "Hello World"
```

Please place the above CGI program in the `/usr/lib/cgi-bin` directory of the virtual machine that you downloaded and set its permission to 755 using Unix `chmod` utility (so it is executable). You need to use the root privilege to do these (using `sudo`), as the folder is only writable by the root. change this setting, you can modify `/etc/apache2/sites-available/default`, which is the Apache configuration file.

To access this CGI program from the Web, you can either use a browser by typing the following URL: `http://localhost/cgi-bin/myprog.cgi`, or use the following command line program `curl` to do the same thing (documentation on `curl` can be found here: http://curl.haxx.se/docs/manpage.html):

```
\$ curl http://localhost/cgi-bin/myprog.cgi
```

In our setup, we run the Web server and the attack from the same computer, and that is why we use `localhost`. In real attacks, the server is running on a remote machine, and instead of using `localhost`, we use the hostname or the IP address of the server.

Step 2: Launch the Attack. After the above CGI program is set up, you can launch the Shellshock attack. The attack does not depend on what is in the CGI program, as it targets the `Bash` program, which is invoked first for the execution of CGI script. Your goal is to launch the attack through the URL `http://localhost/cgi-bin/myprog.cgi`, such that you can achieve something malicious. For example, you can delete some file on the server, or fetch some file (that is not accessible to the attacker) from the server.

Please describe how your attack works.

Hint 1: In order for the attack to be executed, the web server needs to accept an http request header. For example, the following command configures an empty header:

```
\$ curl -A "() { :; };" http://localhost/cgi-bin/myprog.cgi
```

You can take advantage of this command in order to release your attack.

Hint 2: Outline an attack that displays some private information. To do so, first create a new file at the `/usr/lib/cgi-bin` directory. Then, show that using Shellshock attack you are able to get the content of the file, which is not directly accessible from a remote user. Take screenshots of all the outputs.