UNIVERSITY OF WATERLOO
David R. Cheriton School of Computer Science

CS 458/658 **Computer Security and Privacy** **Winter 2015**
Urs Hengartner

ASSIGNMENT 3

Assignment due date: **Monday, April 6th, 2015 3:00 pm**

Total marks: 51

Written Response Questions TA: Nabiha Asghar
Programming Questions TA: Andrew Tinits

Please use Piazza for all communication. Ask a private question if necessary. The TAs office hours are also posted to Piazza.

Written Response Questions [29 marks]

Note: For written questions, please be sure to use complete, grammatically correct sentences where appropriate. You will be marked on the presentation and clarity of your answers as well as the content.

1. [8 marks total] **Anonymous communications systems.**
   Peer-to-peer systems often rely on the existence of many independent (i.e., non-colluding) remote entities to mitigate threats from hostile peers. The "Sybil" attack (`http://www.cs.rice.edu/Conferences/IPTPS02/101.pdf` — not a required reading) is a general attack on such systems wherein the attacker accumulates sufficiently many distinct "identities" in the system to violate the necessary conditions for security. Tor and other "low-latency" anonymous communication systems are inherently susceptible to Sybil attacks: An adversary who controls some fraction f of the Tor network's capacity can deanonymize about a fraction f^2 of all connections through Tor.

   (a) [2 marks] Explain how an adversary that controls the first and last hop in a Tor circuit might be able to deanonymize that circuit.

   (b) [2 marks] Anyone (including you!) can run a Tor relay. Tor lets you configure your own relay as an "exit relay", which can act as the last hop in a circuit and relay traffic to the public Internet, or as a "non-exit relay" that only routes traffic between other relays

and Tor clients. Explain the reason for this configuration option; that is, why might a relay operator prefer to run a non-exiting relay?

(c) [2 marks] In part (b), we learned that there may be good reasons to not run an exit relay. As it turns out, running an exit node can actually be of interest to nefarious entities (or to researchers). Why?

(d) [2 marks] So far, we have considered low-latency anonymous communications systems; for the last part of this question we switch our focus to high-latency anonymous communications systems. Consider an anonymous remailer network in which mix nodes wait until they have received a batch of N messages, at which point they randomly permute the order of the messages, decrypt one layer of encryption from each message, and then forward each message to its next hop. How might an adversary who is capable of observing the (encrypted) packets flowing to and from a particular mix node correlate incoming/outgoing messages to/from that mix node?

2. [8 marks total] **GnuPG.**
A GnuPG public key for `nasghar@uwaterloo.ca` is provided along with the assignment on the course website (nasghar.asc). Perform the following tasks. You can install GnuPG on your own computer, or use the version we have installed on the ugster machines.

(a) [2 marks] Generate a GnuPG key pair for yourself. Use the RSA and RSA algorithm option, your real name, and an email address of your-userid@uwaterloo.ca. Export this key using ASCII armor into a file called **key.asc**. [Note: older versions of GnuPG might not have the RSA and RSA algorithm option, so check that the version you are using has this option. The ugster machines have a new enough version, but the student.cs machine may not.]

(b) [2 marks] Use this key to sign (not local-sign) the nasghar@uwaterloo.ca key. Its true fingerprint is: `EF44 351A 4139 45D4 509B FAD6 5528 0053 B481 AF37`. Export your signed version of the nasghar key into a file called **nasghar-signed.asc**; be sure to use ASCII armor. [Note: signing a key is not the same operation as signing a message.]

(c) [2 marks] Create a message containing your userid and name. Sign it using the key you generated, and encrypt it to the nasghar key. You should do both the encryption and signature in a single operation. Make sure to use ASCII armor, and save the output in a file called **message.asc**.

(d) [2 marks] Briefly explain the importance of fingerprints in GnuPG. In particular, explain how users should check fingerprints and what type of attacks are possible if users do not follow this procedure properly.

3. [5 marks total] **Database Security.**
SQL uses the GRANT operation to grant a user the right to execute an SQL operation on a table or a view, maybe only for a specific set of columns. For example, the operation

```
GRANT SELECT, UPDATE (Day, Flight)
  ON TABLE Diary
  TO Arthur, Zoe
```

grants Arthur and Zoe the right to select any field of table Diary and to update the fields Day and Flight in table Diary. It is also possible to grant access rights on a view (here, the keyword VIEW would be used instead of TABLE).

   (a) [3 marks] Consider a table Accounts with records (CustomerName, AccountNumber, Balance, CreditRating). Define an access structure, e.g., through views, so that:
   - Customer Alice can read (only) her own account;
   - A clerk can read all fields other than CreditRating and update Balance for all accounts;
   - A manager can create new records, read all fields, and update CreditRating for all accounts.

   You can assume that access rights can be granted to groups of people (e.g., Manager).

   (b) [2 marks] Explain how blind writes can happen in a view, that is, a user sees some data in her view of a table, modifies this data (assuming she has the required UPDATE privilege), and then the modified data disappears from the user's view.

4. [8 marks total] **Legal and Ethical Issues**
A breach of privacy occurred at a hospital a few years ago, involving computerized health records. Study the Findings at
`http://www.ipc.on.ca/images/Findings/up-HO_002.pdf` — in particular the sections "Nature of the Complaint" (pages 2–3) and "Did the Hospital Comply with Section 12(1) of the Act" (pages 11–18) — and answer the following questions:

   (a) [2 mark] The affected subject in this case is particularly at risk in part because her ex-husband's girlfriend is a nurse at the same hospital. Give two other types of patients who may be significantly at risk of a loss in privacy, and briefly explain why.

   (b) [2 mark] The subject did not give her explicit consent for her records to be viewed by the nurse. Some privacy advocates have argued that a patient's explicit consent should be necessary before any doctor or nurse is able to access their health records. Give two reasons why this may pose a problem.

   (c) [2 mark] Many people fear that the use of computerized health records rather than paper records may endanger a patient's privacy. Give two reasons why such worries may be justified, with references to this case.

   (d) [2 mark] Give and explain two problems in the hospital's response to the privacy breach that allowed the nurse to access the patient's data no less than ten times.

Programming Questions [22 marks]

Background

In this section we will study padding oracle attacks. Block ciphers operate on a fixed number of bits, such as 64 (DES) or 128 (AES). To encrypt longer messages, a mode of operation such as CBC can be used. However, this still requires that the messages be a multiple of the block size. To accommodate arbitrary-length messages, a padding scheme is used. The decryption routine will need to check that the message padding is valid. Information about the validity of this padding can easily be leaked to an attacker. This "padding oracle" can allow the attacker to decrypt (and sometimes encrypt) messages without knowing the encryption key.

The padding oracle attack was originally described in a 2002 paper by Serge Vaudenay. This paper is available at `https://www.iacr.org/archive/eurocrypt2002/23320530/cbc02_e02d.pdf` and will serve as your reference for this attack. Note that the author frequently uses the term "word" where we would usually say "byte". Padding oracle attacks have continued to be relevant since their inception, with new attacks being discovered regularly. One recent example is the POODLE attack on SSL 3.0 and TLS, discovered by Google security engineers in 2014 (not a required reading).

Application Description

A simple web application is running at `http://ugsterXX.student.cs.uwaterloo.ca:4555` for each ugster machine. You can test this URL in your web browser, but in your programs please use `http://localhost:4555`. This will ensure that your programs run quickly no matter which ugster machine we test them on. When you visit this application, it will set a cookie named `user` in your browser. In other words, the HTTP response will contain a `Set-Cookie` header like the following:

```
Set-Cookie: user="K/wLMO+V8Qbo5B4spv6/PP98W1mofu7vQT83dbascyLcyc9mlFi4qzLyYjsyTLWF"
```

The value of this cookie is encrypted with AES using a key known only to the web server. AES is used in CBC mode, and the IV is prepended to the ciphertext. The result is then Base64 encoded to form the cookie value. Base64 is a standard way of encoding binary data into printable ASCII characters. It is implemented in many programming languages and the `base64` Unix utility, and is specified in RFC 3548. The padding scheme used is that of IP ESP referred to in Section 4 of the padding oracle paper. The n bytes of padding have incrementing values from 1 to n, so 1, 12, and 123 are valid examples. There is always at least one byte of padding.

When you visit the site again, your web browser will send the cookie back to the server. This is done with a `Cookie` header with the same form as the `Set-Cookie` header above. This behaviour can of course be emulated programmatically. The web server will attempt to decrypt any cookie value sent to it using its encryption key. If the decryption is successful the HTTP response will contain a 200 (OK) or 404 (Not Found) response code. However, if the decryption fails a 500 (Internal Server Error) response code will be returned. See the section below titled "HTTP with `curl`" for more information on working with HTTP.

Questions

5. [2 marks] What is the average case and worst case complexity of this attack in terms of the number of calls to the padding oracle?

6. [2 marks] How would you fix this vulnerability? Assume that the web server must return a different response for users with valid vs invalid cookies.

7. [3 marks] How would you modify the attack described in the paper to account for the different padding scheme used by the web server? List the lines in Sections 3.1 and 3.2 that require changes and show how they should be changed.

8. [10 marks] Write a program called `decrypt` that implements the padding oracle attack on the web application described above. Your program will be called with a single command line argument containing a Base64-encoded cookie value to be decrypted. Your program should generate the appropriate cookie values, send them to the web server using HTTP, and observe its response codes in order to decrypt the given cookie value. It should print the resulting plaintext to `stdout`, which will consist of only printable ASCII characters. You can print debug output to `stderr` if you like. Your program should run in a maximum of 10 minutes.

   You may use any programming or scripting language available on the ugster machines. If your preferred language is not available, we may be able to accommodate requests. Most common programming languages have built-in libraries for making HTTP requests to web servers. Alternatively, you can use the `curl` program as described in the section below titled "HTTP with `curl`". You will submit a single file named `src.tar`. For evaluation, this file will be extracted and we will attempt to run an executable at the top level with `./decrypt <cookie>`. This executable could be your program itself, or it could be a script that compiles and then runs your program. For the example cookie shown above, the invocation would be:

   ```
   ./decrypt K/wLMO+V8Qbo5B4spv6/PP98W1mofu7vQT83dbascyLcyc9mlFi4qzLyYjsyTLWF
   ```

9. [5 marks] Write a program called `encrypt` that encrypts arbitrary cookie values such that they will be correctly decrypted by the web application. Your program does not need to know the web application's encryption key. It will be called with a single command line argument

containing a printable ASCII plaintext to be encrypted. Your program should generate the appropriate cookie values, send them to the web server using HTTP, and observe its response codes in order to encrypt the given value. It should print the resulting Base64-encoded ciphertext to `stdout`. You can print debug output to `stderr` if you like. Your program should run in a maximum of 10 minutes.

Submission is similar to the previous question. Your program source files should be included in the same `src.tar`. For this question, we will attempt to run an executable at the top level with `./encrypt <plaintext>`.

Hint: In CBC mode there are three stages a block goes through in decryption. It starts as a ciphertext, is decrypted by the block cipher to an intermediary value, and then is XORed with the previous ciphertext block (or the IV for the first block) to form the final plaintext value. If we know the intermediary value for a given ciphertext block, we can manipulate the IV to gain complete control over what that block will be decrypted to. If we want to produce a plaintext x, then the IV should be x XORed with the intermediary value. This idea can easily be extended to multi-block messages. Start from the last block and move backwards. Pick any value to be the ciphertext for the last block and decrypt it (using your method from the previous question) to find the corresponding intermediary value. Then calculate the IV required to produce the desired plaintext. This IV will be the ciphertext for the second-last block, and so on.

What to Hand In

It is very important that you follow the rules outlined in the Assignments section of the LEARN course site for submitting your assignment. Otherwise we may not be able to mark your assignment and you may lose partial or all marks. By the **a3 deadline**, you are required to hand in:

a3.pdf: A PDF file containing your answers for the written-response and relevant programming questions.

key.asc: Your GnuPG public key from question 2.

nasghar-signed.asc: Your signed version of Nabhiha's public key from question 2.

message.asc: Your encrypted and signed message from question 2

src.tar: A tar archive containing your decryption and encryption program source files. To create the tarball, `cd` to the directory containing your code and run the command

```
tar cvf src.tar .
```

(including the `.`).

HTTP with `curl`

HTTP (Hypertext Transfer Protocol) is a simple protocol for transferring text over a network. If you've used the Internet, you've used HTTP. In this protocol, a client makes a request to a server and the server replies with a response. A request generally asks for a resource located at a particular URL, and the response provides that resource. These resources are often HTML web pages, but here we'll be using mostly plain textual content. Let's see an example of a request and a response:

```
$ curl -v http://localhost:4555
> GET / HTTP/1.1
> User-Agent: curl/7.19.7
> Host: localhost:4555
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Length: 37
< Set-Cookie: user="K/wLMO+V8Qbo5B4spv6/PP98W1mofu7vQT83dbascyLcyc9mlFi4qzLyYjsyTLWF"
< Server: TornadoServer/4.1
< Etag: "4b44c375286c4a668502645e4da51dd29441e4e9"
< Date: Wed, 11 Mar 2015 14:28:20 GMT
< Content-Type: text/html; charset=UTF-8
<
Hello! I'll remember when I saw you.
```

Note that some extra debug output from `curl` has been omitted. The lines starting with a > are the client talking to the server, and lines starting with a < show communication in the other direction. The HTTP request starts with a method; we'll only be using GET. It then specifies the requested path and the protocol version. The following lines contain headers which specify additional information. These are present in both requests and responses. The response starts by confirming the protocol version. It then presents the status code, which is essential for us as it is the source of our padding oracle. The main response header of interest to us is the `Set-Cookie` header. We can use the `base64` utility to decode it, although it will likely result in unprintable characters, so we'll display the binary data as a hex dump with `xxd`:

```
$ echo "K/wLMO+V8Qbo5B4spv6/PP98W1mofu7vQT83dbascyLcyc9mlFi4qzLyYjsyTLWF"
    | base64 -d | xxd
0000000: 2bfc 0b30 ef95 f106 e8e4 1e2c a6fe bf3c  +..0.......,...<
0000010: ff7c 5b59 a87e eeef 413f 3775 b6ac 7322  .|[Y.~..A?7u..s"
0000020: dcc9 cf66 9458 b8ab 32f2 623b 324c b585  ...f.X..2.b;2L..
```

To send the cookie back to the server, we can do the following:

```
$ curl -v -b user=K/wLMO+V8Qbo5B4spv6/PP98W1mofu7vQT83dbascyLcyc9mlFi4qzLyYjsyTLWF
    http://localhost:4555
> GET / HTTP/1.1
> User-Agent: curl/7.19.7
> Host: localhost:4555
> Accept: */*
> Cookie: user=K/wLMO+V8Qbo5B4spv6/PP98W1mofu7vQT83dbascyLcyc9mlFi4qzLyYjsyTLWF
>
< HTTP/1.1 200 OK
< Date: Wed, 11 Mar 2015 15:22:39 GMT
< Content-Length: 47
< Etag: "187a53481dad5afce82d5c79e2fcc946d341a405"
< Content-Type: text/html; charset=UTF-8
< Server: TornadoServer/4.1
<
Hello! I first saw you at: 1969-12-31 19:00:00
```

You're probably better off using an HTTP library for your programming language when writing your solutions, but you can call the `curl` program directly as a last resort. `curl` is also very useful for manually debugging web applications.