UNIVERSITY OF WATERLOO
Cheriton School of Computer Science

CS 458/658 **Computer Security and Privacy** **Spring 2016**

Erinn Atwater and Doug Stinson

ASSIGNMENT 3

Assignment due date: **Tuesday, July 26, 2016 3:00 pm**

Total Marks: 54
Written Response Questions TA: William Herring (Office hours: Thursdays 2–3:00 pm in DC 3332)
Programming Questions TA: Navid Nasr Esfahani (Office hours: Thursdays 12:30–1:30 pm in DC 3332)

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

Written Response Questions [27 marks]

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

1. (10 marks) **Firewalls**

   Suppose you are establishing a movie streaming service that has distributed the work load on multiple servers across the globe. The service must have the following properties:

   - Movies and customer accounts are divided into "regular" and "premium" tiers.
   - Only customers with premium accounts can access premium movies. (All accounts can access regular movies.)
   - A customer can log in to a website with their credentials, and browse all movies available for their tier.
   - When a customer chooses a movie to watch, it should start streaming from the closest server to them, geographically.
   - No (malicious) customer can alter the available movies, their assigned tier, or the tier of any customer account (including their own).

   As long as these requirements are met, you are free to design the service as you wish. Keeping in mind the above scenario, answer the following:

(a) Draw a **network diagram** outlining how you would design the service. Your diagram may include web servers, file servers, databases, routers/switches, and internet links. Briefly explain the key elements of your design, including what the purpose of each server is. You may choose to have some components be replicated and other components be unique; indicate this somehow. Your diagram should contain sufficient detail to answer the following two questions.

(b) Your security budget only includes enough money to purchase a single firewall. Where in your network diagram will you choose to place this one firewall, and why? Of the different types of firewall discussed in class, which would you choose, and why?

(c) Suppose you are then given budget for more firewalls; you can now place one at every geographic location, as long as they all have the same type and configuration. Where do you place them? Which type do you choose, and why?

2. (6 marks) **Inference Attacks**
The game *World of Warcraft* has a table of size N, `Players`, in its database. Below is an excerpt (not the entire table):

Username	Server	Class	Level	Gold
thrall91	PvP	Shaman	100	83,559
XdarksephirothX	PvE	Rogue	98	58,044
leg0las	PvE	Hunter	100	45,791
zerocool	PvP	Paladin	100	25,882
thibbledorf	PvE	Druid	99	94,250
AidenStormbringer	PvP	Warrior	100	284,962
doomlaser	PvP	Demon Hunter	20	37,543
potterfan1	PvE	Mage	58	74,641
noheals4u	PvP	Priest	100	391,831

Table 1: Part of `Players` Table

A public website exists that allows players to view basic information about each other. The database is set up to suppress the `Gold` field in the output of queries. However, users can execute queries of the form `SELECT SUM(Gold) FROM Players WHERE ...` where queries that match fewer than k or more than $N - k$ (but not all N) records are rejected. We will use $k = \frac{N}{8}$.

(a) Use a tracker attack, as defined in class, to design a tracker and a set of three queries based on this tracker that will let you infer XdarksephirothX's wealth. Both the tracker and the three queries need to be of the form `SELECT SUM(Gold) FROM Players ...` Assume that players' names are unique and not known to the attacker (apart from XdarksephirothX's) and that the attacker has no additional information about XdarksephirothX. For the tracker attack to succeed, the attacker does need to make an assumption about the distribution of (some of) the values stored in the database. This

assumption should be realistic so that your tracker attack (likely) would also work on a different set of records, not only on the set shown above. In your solution, you should give 1) your assumption, 2) your tracker, and 3) the set of three queries.

(b) The game developer becomes aware of the tracker attack and forbids `SUM(Gold)` queries. Instead the website allows only queries of the form `SELECT COUNT(*) FROM Players WHERE Q`. Note that `Q` may include testing the value of the Gold field. (Again, queries that match fewer than k or more than $N-k$, with the exception of N, records are ruled out.) How would you use queries of this type to learn doomlaser's gold? You may assume that no one in the database has more than 20 billion gold and that all values are non-negative if that simplifies your attack.

3. (11 marks) **Rule 41**

On May 18, CNN published an article titled "Privacy hawks in Congress prep last stand against Justice hacking rule". The article can be found here:

http://www.cnn.com/2016/05/18/politics/justice-department-hacking-rule-41-ron-wyden/

The purpose of this question is to explore several issues raised by the article. Please give your answers in clear, complete sentences. As with previous assignment questions of this nature, you may wish to do some research on the internet to help answer questions. This is acceptable provided you cite your sources appropriately.

(a) The changes to Rule 41 are intended to solve two major issues with the existing process for United States federal law enforcement officers to get warrants for accessing computers. State the two issues succinctly, as well as how the changes attempt to solve them.

(b) What does Peter Carr likely mean when he says "sophisticated anonymizing technologies to conceal their identity while they engage in crime over the Internet"? Why would this technology make it difficult to get a warrant under existing laws?

(c) Give **three** examples of technologies whose users might be affected by this change.

(d) What are two possible negative consequences of passing the proposed changes to Rule 41? Could any of them affect Canadians?

(e) What is "judge shopping"? Give another example of judge shopping (or venue shopping).

Programming Questions [27 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 (see Figure 1). However, this still requires that the message length 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. Unfortunately, information about the validity of the 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. Some recent examples are the POODLE attack discovered in 2014 and the DROWN attack discovered in March 2016, both attacking SSL and TLS (not required reading).

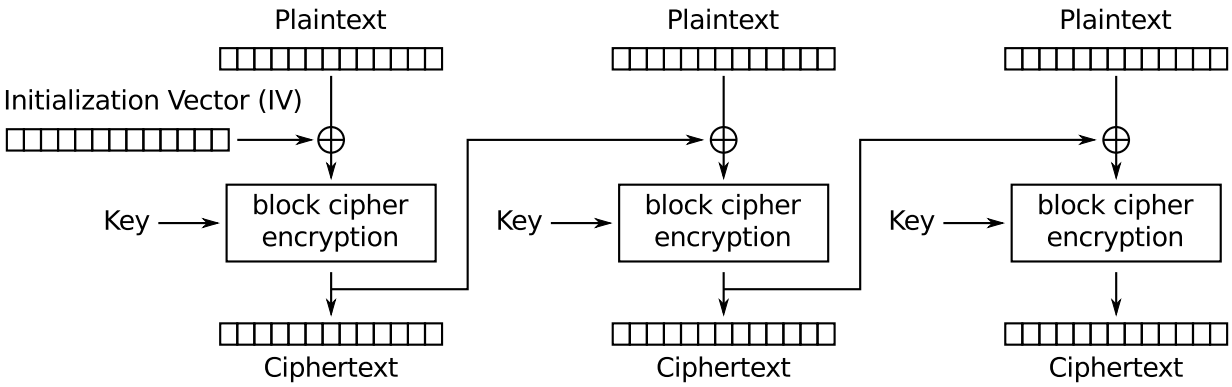

Figure 1: CBC Mode Encryption

Application Description

A simple web application is running at `http://localhost:4555` on each ugster machine. To view this application in your web browser, you can use `http://ugsterNN.student.cs.uwaterloo.ca:4555` (instructions for working from home are the same as with previous

assignments). However, please use `http://localhost:4555` in your programs to ensure that they run quickly no matter which ugster machine we test them on.

When you visit the web 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="k0UtzN9gwc1qfe2CMYn3rESGL6mJwyG8Z+ztkAJJBVM="
```

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 randomly chosen 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 by the web application is as follows: for a padding of length n, the first $n - 1$ bytes of the padding have decrementing values from $n - 1$ to 1, and the last byte of the padding is the value n. There is always at least one byte of padding, so 1, 12, 213, and 3214 are valid examples. There is always at least one byte of padding.

When you visit the web application again, your 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 use its encryption key to attempt to decrypt any cookie sent to it. If the decryption is successful the HTTP response will contain a 200 (OK) 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

1. [2 marks] What are the average case and worst case number of padding oracle calls required to perform this attack?

2. [3 marks] What cryptographic tool could be used to fix this vulnerability? Which of the CIA properties {Confidentiality, Integrity, Authenticity} does it provide? How does it fix the vulnerability? Assume that the web server must return a different response for users with valid versus invalid cookies.

3. [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.

4. [12 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. These cookies will have at least two 16-byte blocks (the IV and at least one ciphertext block), but may not have the same number as those returned by the server. 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 (without padding) 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 for inputs up to 10 blocks long.

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. Many programming languages have built-in libraries for making HTTP requests and encoding and decoding Base64. You can also use third party libraries for these tasks if necessary. Another option is to call the `curl` and `base64` programs 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 a script at the top level with `./decrypt <cookie>`. This script could be your program itself, or it could be a script that compiles and then runs your program. In either case it should start with an appropriate shebang line. Your submission should not contain any compiled executables. For the example cookie shown above, the invocation would be:

```
./decrypt k0UtzN9gwc1qfe2CMYn3rESGL6mJwyG8Z+ztkAJJBVM=
```

5. [7 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 for inputs up to 10 blocks long.

   Submission is similar to the previous question. Your program source files should be included in the same `src.tar` file. For this question, we will attempt to run a script 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, the ciphertext is decrypted by the block cipher to an intermediary value, and then the intermediary value 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

All assignment submission takes place on the `student.cs` machines (not the ugster or the virtual environments), using the `submit` facility. In particular, log in to the Linux student environment (`linux.student.cs.uwaterloo.ca`), go to the directory that contains your solution, and submit using the following command: `submit cs458 3 .` (dot included). CS 658 students should also use this command and ignore the warning message.

By the **A3 deadline**, you are required to hand in:

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

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 `.`).

a3.pdf must contain, at the top of the first page, your name, UW userid, and student number. If it does not, **a 5 mark penalty will be assessed**.

Be sure to "embed all fonts" into your PDF files. Some students' files were unreadable in the past. If we can't read it, we can't mark it.

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 ">" are the client talking to the server, and lines starting with "<" 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..
```

You can see that this cookie contains three 16-byte blocks. The first is the IV, and the next two are ciphertext blocks. 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
```

It would be best to use 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.