EECS 388: Lab 4

Review: Project 1 Part 1
Review: Project 1 Part 2
Introduce Project 2 (Web Security)
SQL Introduction
SQL Injection

Current Assignments

- Lab 2: Available Now (Thursday, Sep. 21st)!
  - **Due Thursday, Sep. 28th at 6 PM**
- Project 2: Web Security
  - Released: Thursday, Sep. 21st
  - **Due Thursday, Oct. 5th at 6 PM**
  - Coverage:
    - SQL Injection
    - CSRF Attack
    - XSS Attack

Partners are optional!

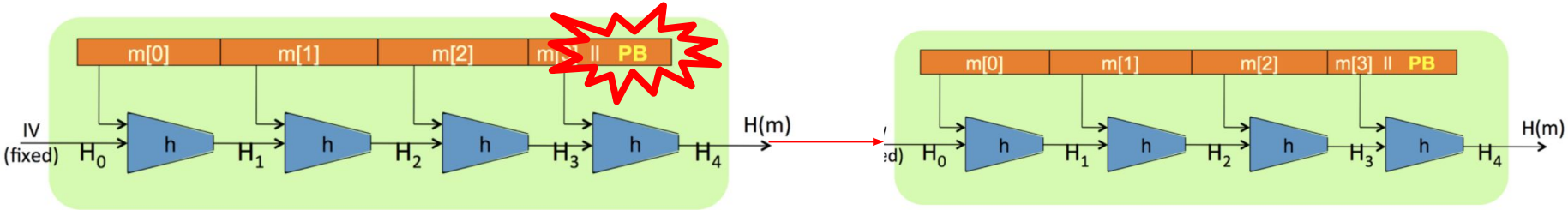
Reminder: Weekly Canvas Quizzes

Review:
Project 1, Part 1

Key Takeaways

Length Extension

- Padding from original message must remain in the extended message!



Hash Collisions

- Differentiate blobs using their SHA-256 hashes!

```
sha256(blob.encode("latin-1")).hexdigest()
```

```
print("Use SHA-256 instead!")
```

```
print("MD5 is perfectly secure!")
```

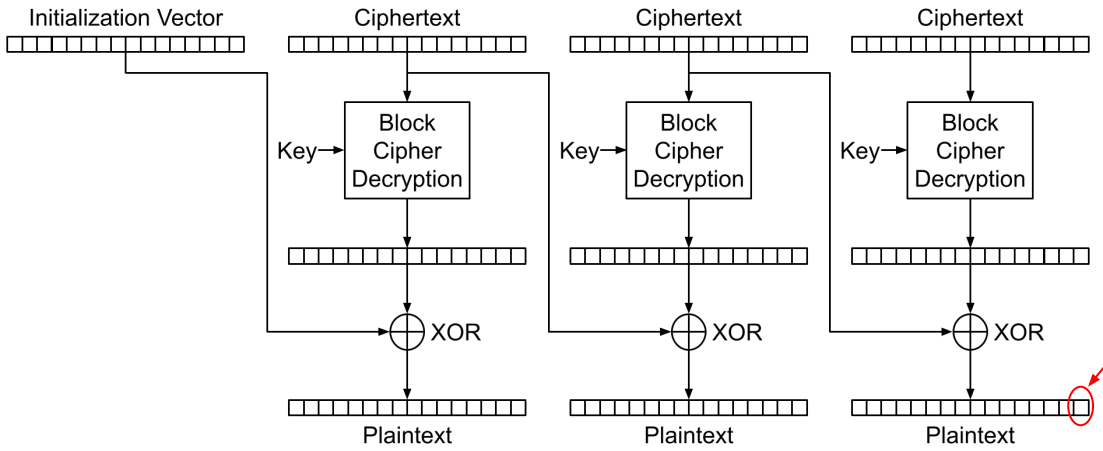
Review:
Project 1, part 2

Padding Oracle Attack

- General Idea
  - There exists some “oracle” that allows the attacker to differentiate between failure and success
- Key idea
  - If an attacker can tell whether an integrity check fails due to incorrect padding or an incorrect MAC, they can figure out the padding scheme.
  - Then an attacker can spoof message on the system
- Protection
  - **ALWAYS** check the MAC ***before*** decrypting and checking the padding scheme
    - Cryptographic Doom Principle!

Padding Oracle Attack

- Why is it dangerous for a server to send an error for incorrect padding?
- Explicitly explain how you could exploit this to solve for this byte



Hint: you can manipulate whatever bytes of ciphertext you'd like, send the ciphertext to Bob, and see whether he gives you a padding error or not

Cryptographic doom principle: If you need to do any work before checking the MAC, then you are doomed!

Padding Oracle Attack

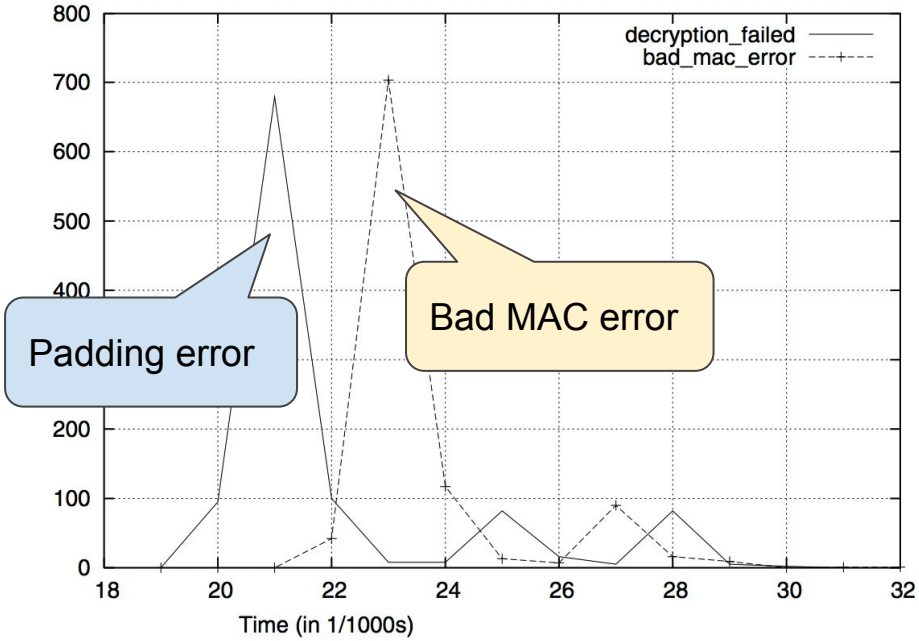
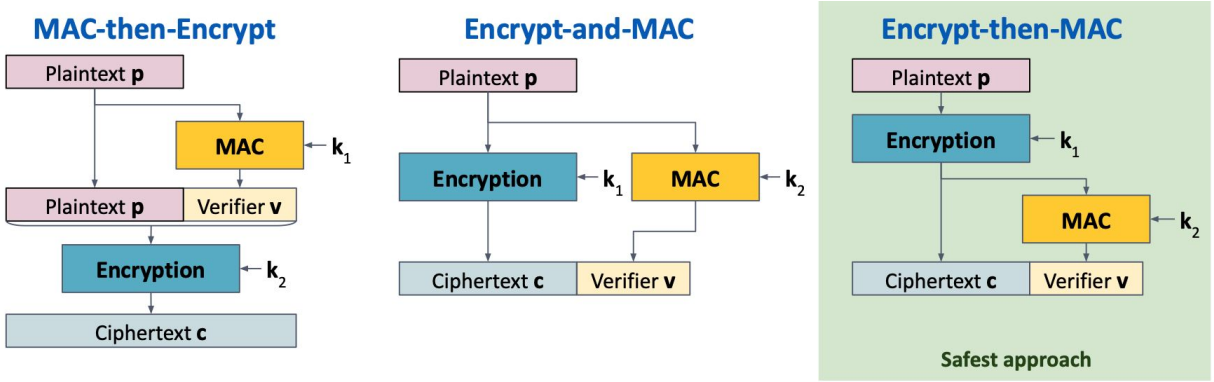


Image credit: https://link.springer.com/content/pdf/10.1007/b11817.pdf#page=596

- What if we don't get a "padding error" or "MAC error"?
  - What if the server just returns "error"?
  - Is there still the potential for a padding oracle?

Approaches to AEAD

1. MAC-then-Encrypt
   - Never!
2. Encrypt-and-MAC
   - Still nope
3. Encrypt-then-MAC
   - The only safe way



Cryptographic doom principle: If you need to do any work before checking the MAC, then you are doomed!

Bleichenbacher Attack

- *Vulnerability*: Small public key and improper verification means we can forge a signature
- *Attack*: Construct proper padding, take cube root, do “magic” math. Why?
  - When it was cubed, the signature was floored
  - Adding 1 only messes with arbitrary bytes and the signature will be valid (even though it shouldn’t be)

$$\texttt{00 01 FF 00}\ \underbrace{\texttt{30 31 30 0d 06 09 60 86 48 01 65 03 04 02 01 05 00 04 20}}_{\text{ASN.1 “magic” bytes denoting type of hash algorithm}}\ \underbrace{\texttt{XX XX XX} \cdots \texttt{XX}}_{\substack{\text{SHA-256 digest}\\ \text{(32 bytes)}}}\ \underbrace{\texttt{YY YY YY} \cdots \texttt{YY}}_{\substack{k/8-55\\ \text{arbitrary bytes}}}$$

Intro to Web Project

Project 2 Preview

- Part 1: SQL Injection Attacks
  - Lecture 8, today's lab!
- Part 2: XSS Attacks
  - Lecture 8, next week's lab
- Part 3: CSRF Attacks
  - Lecture 8, next week's lab

- Skills you'll need
  - Basic JavaScript, HTML, knowledge of DOM, basic SQL (Lecture 7)
  - Using basic webdev tools (Lab Assignment 2)
  - Running a basic Python webserver (Project 2 spec)
  - Ability to Google basic questions as needed

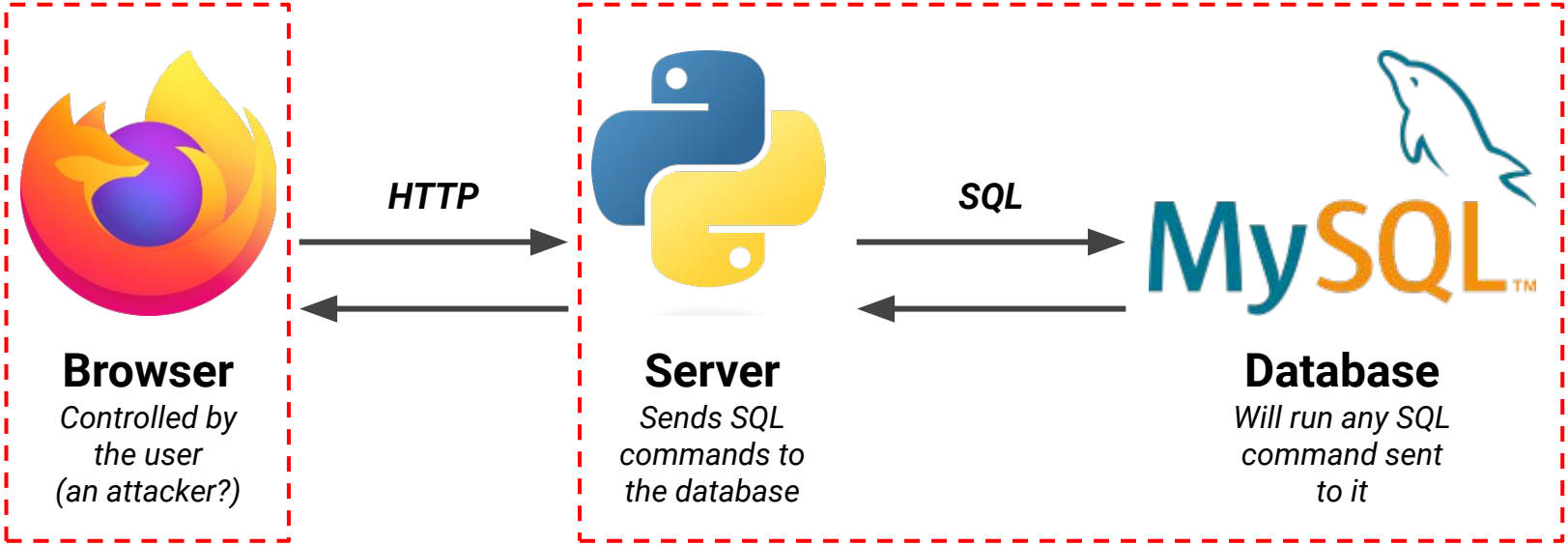
But Wait!

NEVER use an important password for an account on an insecure site!

(Also, never reuse a password at all)

SQL 101

Databases



Very important that users (attackers) can't send any command that they want to the database!

Structured Query Language (SQL)

- Allows you to interact with a database
- Can store and retrieve data
- Built to look kinda (???????) like English

users

username	password	birthday
adamau	12345	04-23-2001
hoffcar	password	06-11-2000
danlliu	x1w41dd5$	05-10-2000

```
SELECT * FROM users WHERE username = "danlliu"
```

(Get all data rows from the table “users” where the “username” column contains exactly “danlliu”)

Good SQL tutorial by W3:: https://www.w3schools.com/sql/

Example Code - SQL in Python using SQLite3

```
import sqlite3
...
con = sqlite3.connect('user.db')
sql_statement =
  "SELECT birthday FROM users WHERE username = ? AND password = ?"

result = con.execute(sql_statement, (user_input_username, user_input_password))
if result is not empty:
  # print the user’s birthday
```

- SQL = Blue
- SELECT = choose the column
- birthday = the attribute to be selected
- FROM = choose the table
- WHERE = specify logical conditions
- AND = logical operator that requires both conditions to be true

Example Code - SQL in Python

```
...

sql_statement =
  "SELECT birthday FROM users WHERE username = ? AND password = ?"
result = sql_execute(sql_statement, (user_input_username, user_input_password))
if result is not empty:
  # print the user's birthday
```

users

username	**password**	**birthday**
adamau	123456	04-23-2001
hoffcar	password	06-11-2000
danlliu	x1w41dd5$	05-10-2000

{leslie, haxor}

{danlliu,x1w41dd5$ }

no user found

05-10-2000

SQL Injections

A Main Vulnerability: Data vs. Code

- What is the difference?
  - The context in which they are used
    - You can treat a file of source code as just a long string, and vice versa
  - If we can confuse the context, we can get data to be executed as code
    - SQL Injection
    - XSS

SQL Injections - Example Code

Not using prepared statements 😱

```
...
sql_statement =
  "SELECT * FROM users WHERE username = '" + user_input_username +
  "' AND password = '" + user_input_password + "'"
result = con.execute(sql_statement)
if result is not empty:
  # let the user log in to the site
```

- * means to select all
- Double quotes - surround the SQL code snippet
- Single quotes - surround the user input within the SQL code

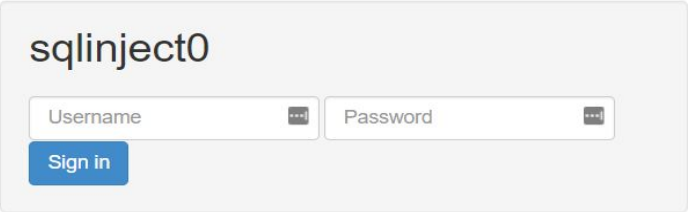
- Where is the vulnerability? How could an attacker exploit this?

SQL Injections: Exercise

- Navigate to https://www.hacksplaining.com/exercises/sql-injection
- Complete the demo
- Let us know if you have questions!

Project 2: SQL Injections

- Manipulate the SQL by getting your string input to be interpreted as SQL
- Do not need to be in Docker's Firefox browser for this part
  - https://project2.eecs388.org/sqlinject/0
  - Can change defense level with number at end
- Situation
  - You are breaking into the account of username victim
  - Come up with a password that will allow you to manipulate the SQL behind the scenes



Real Examples of SQL Injections

- SQL Injection Attack on Heartland Payment Systems, 7-Eleven
  - https://www.wired.com/2010/03/heartland-sentencing/
- GhostShell SQL Attacks
  - https://www.darkreading.com/attacks-breaches/ghostshell-haunts-websites-with-sql-injection
- 2016 Russian-government Attacks Against Voter Registration Systems
  - https://rollcall.com/2019/04/22/mueller-report-russia-hacked-state-databases-and-voting-machine-companies/

Lab Assignment 2

Useful for Project 2!

Lab 2 Information and Spec Walkthrough

- Lab 2 seeks to gently introduce you to:
  - The Firefox GUI, you'll need this for Project 2
  - Dev Tools, this will assist you in completing the XSS / CSRF attacks
  - JavaScript

- Quick walkthrough of the Lab 2 Spec!

See you next week!