COMP 307

Principles of Web Development

Lecture 21

Unit 6 – Security

Security 2

Announcements

- Last lecture!
  - Don’t forget McGill’s course evaluations
- Project questions?
- About Nov 23 midterm
  - Sample test ?
- Class time will turn into office hours in ENGMC 323

About the final

- Lectures 1 to 21
- The midterm is a myCourses Quiz
  - I will be present in class to answer questions
  - Use Email to post private questions if you are not in class
    - Randomized questions from a pool, so screenshot your question
  - 60-minute test
  - Closed book
- No programming questions, cumulative test
- Question types
  - Protocols, communication methods, and packets
  - Run-time environments (server, browser, languages)
  - Storyboard, elements and shape choices, case studies, wireframes, directory structure
  - Performance and load
  - What is: static, responsive, interactive, dynamic content, dynamic page generation, single-page, multi-page. Languages that support
  - 3-tierd applications (when Mongo, SQL?)
  - Security attacks and defenses

About the final

- This test is generated randomly by the computer selecting questions from a pool. The questions have been balanced to make it fair. Each student will receive their own test.
- Midterm test format
  - Password question
  - I have not cheated question (especially important for students who will not be in class)
  - 10 multiple-choice, multi-select, ordering, etc. questions
  - 2 long answer questions

Contents

Practical Security

Class Outline

- The central issue in website security
- Example simple security
- Improvements

Readings

- Course Textbook
  - N/A
- Online resources
  - https://en.wikipedia.org/wiki/JSON_Web_Token
  - https://www.bezkoder.com/jwt-json-web-token/
  - https://auth0.com/docs/secure/tokens/json-web-tokens

Contents

Practical Security

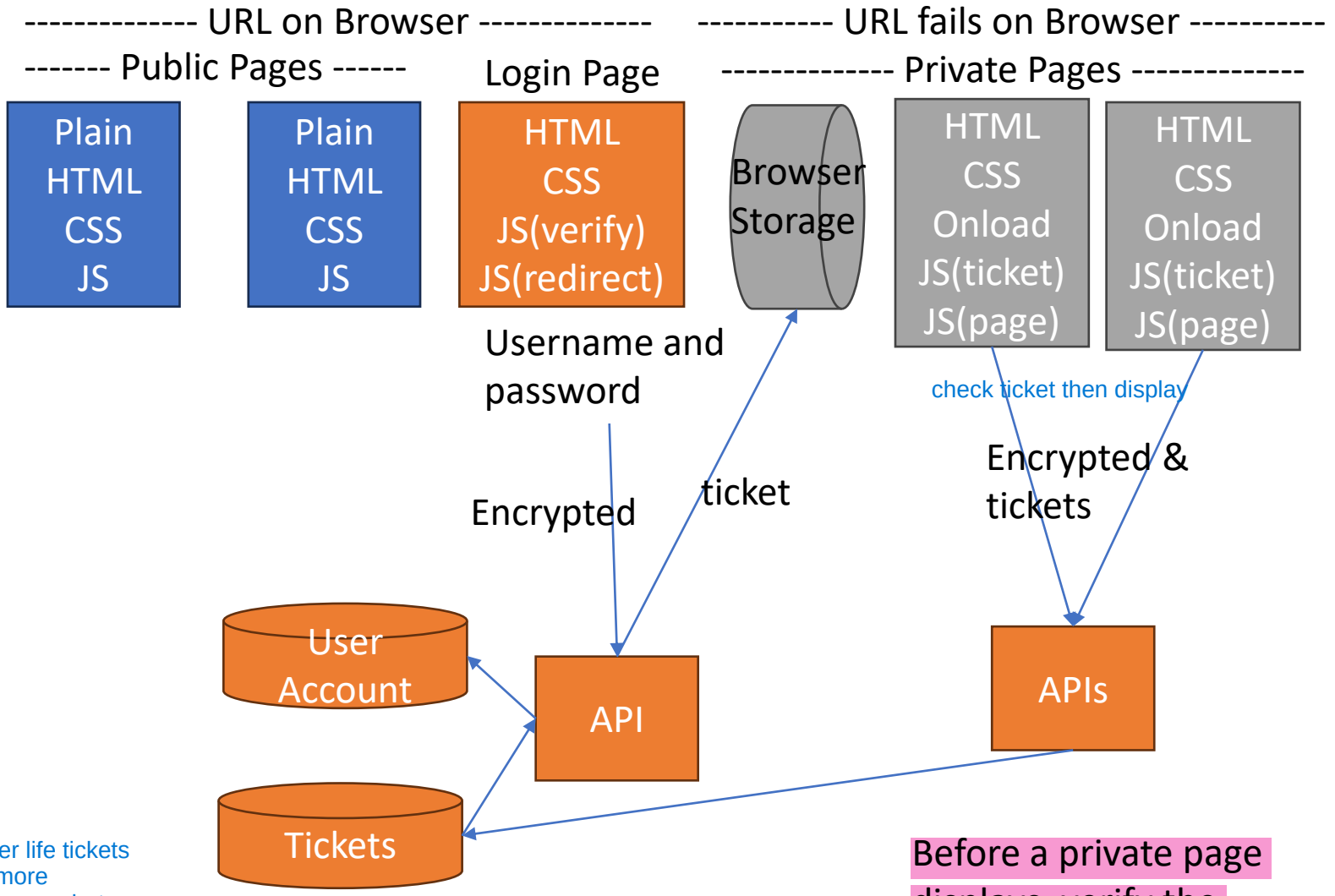
The Basic Problem

- Notice on my SOCS website that you can access any of the pages by simply typing in its URL.
  - Demo
- This should not be permitted for private pages. Only public pages should permit users to type their URL and get access.
  - You must programmatically protect these pages

Contents

Practical Security

The Basic Solution



Flowchart

- Version 1
  - Assumes bad-guy found out URL to secure page and types it in on the browser
  - Algorithm
    - The page at URL loads a JS program that checks browser storage for ticket field.
      - If not found, display 403 Forbidden error code.
      - If found, JS makes AJAX call to API to validate ticket ID
        - Validation fails, display 403 Forbidden error code.
        - Validation success, redirect to valid page or generate the valid page (preferably)
- Version 2
  - Assumes the URL incorporates the ticket number. Example: http://URL/command/{ticket_number}
  - Bad-guy will need to type a ticket number
  - Algorithm
    - API extracts ticket_number from the URL
    - If no ticket number present, then display 403 Forbidden error code.
    - If ticket present, then SQL to check if ticket is valid
      - Validation fails, then 403 error, otherwise generate the webpage

Contents

Practical Security

Note

- Every secure webpage has this code.
- Best practices:
  - Make a common program that handles logistics
    - PHP – like in my website, the one program handles everything (routing & security)
    - Node – "server" script handles everything
  - Ticket should not be in URL but in the POST or Header data
    - This is the Version 2 solution from previous slide but removing the ticket from the URL for more privacy, but not for more security since you can Wireshark that info from POST or Header data.

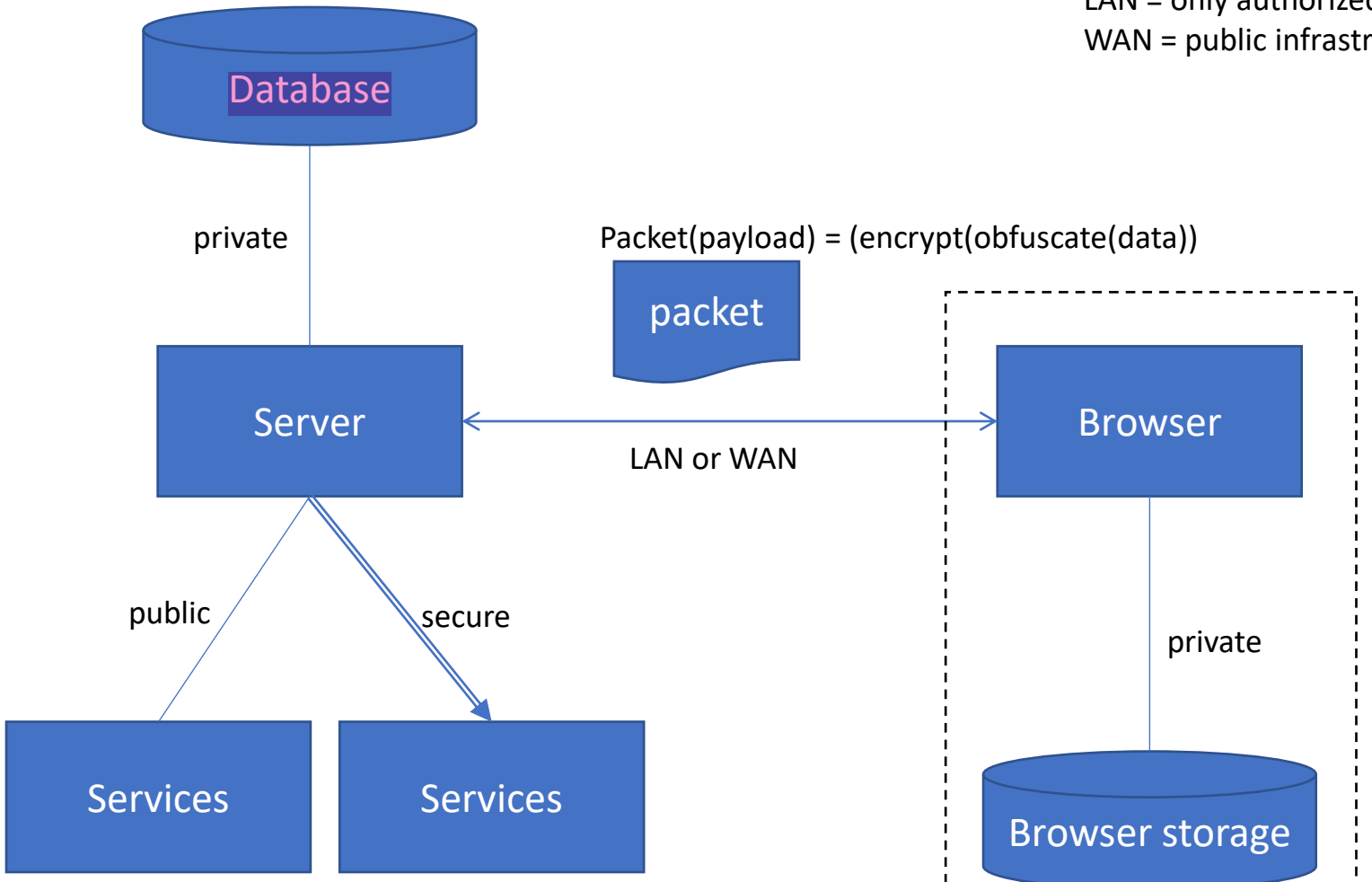
Contents

Practical Security

Architecture

- Stored outside of the website's directory tree.

Private = never seen by public
Secure = only authorized users
LAN = only authorized users
WAN = public infrastructure



- In public_html
- No authentication checking on anything.

- Not in public_html
- Generated pages.
- Authentication checking on every page.

- Everything runs on client's local device.

What is going on here?

Databases

SERVER SIDE

BROWSER SIDE

User ID

Or any key

UID	UserName	Encryted(password)	Public Key

URL	Public Key	Expiry Date	Fields to remember
			Ticket, pass

Device public key

Device private key

UID	Ticket (ID)	Resource ID	Permissions (s,o,r,w,x)

S = super
O = owner
R = read
W = write
X = execute

Server public key

Server private key

- Password privacy issues
- Encryption key information
- Resource access via Tickets (which includes login-success ticket)
- Each resource requires its own ticket
- Certificates and databases stored outside of website directory tree

Login Security

Security 3

Contents

Practical Security

Before we do anything

url attacks

// **index.html**

handling incorrect urls

```
<html>
        <body>
        <script>
                window.location.replace("http://www.abc.com/login.html");
        </script>
        </body>
</html>
```

Servers normally default to **index.html** or **default.html** when the user enters an incorrect URL. Best to create your own version of this HTML file in all your subdirectories to handle exploration attacks and typos.

The code above redirects the person to some other webpage that you would like them to see. In this case, I ask them the login, but you could redirect to landing page.

Contents

Practical Security

Before we do anything

encryption attacks

Use encryption to help keep information secret.

For example, even though a bad-guy can copy and paste your encrypted password and use it to login, the encrypted password has the property of keeping the original password a secret.

This is called balancing risk. If a bad-guy can compromise your website, you are still protecting your customer's information to some degree.

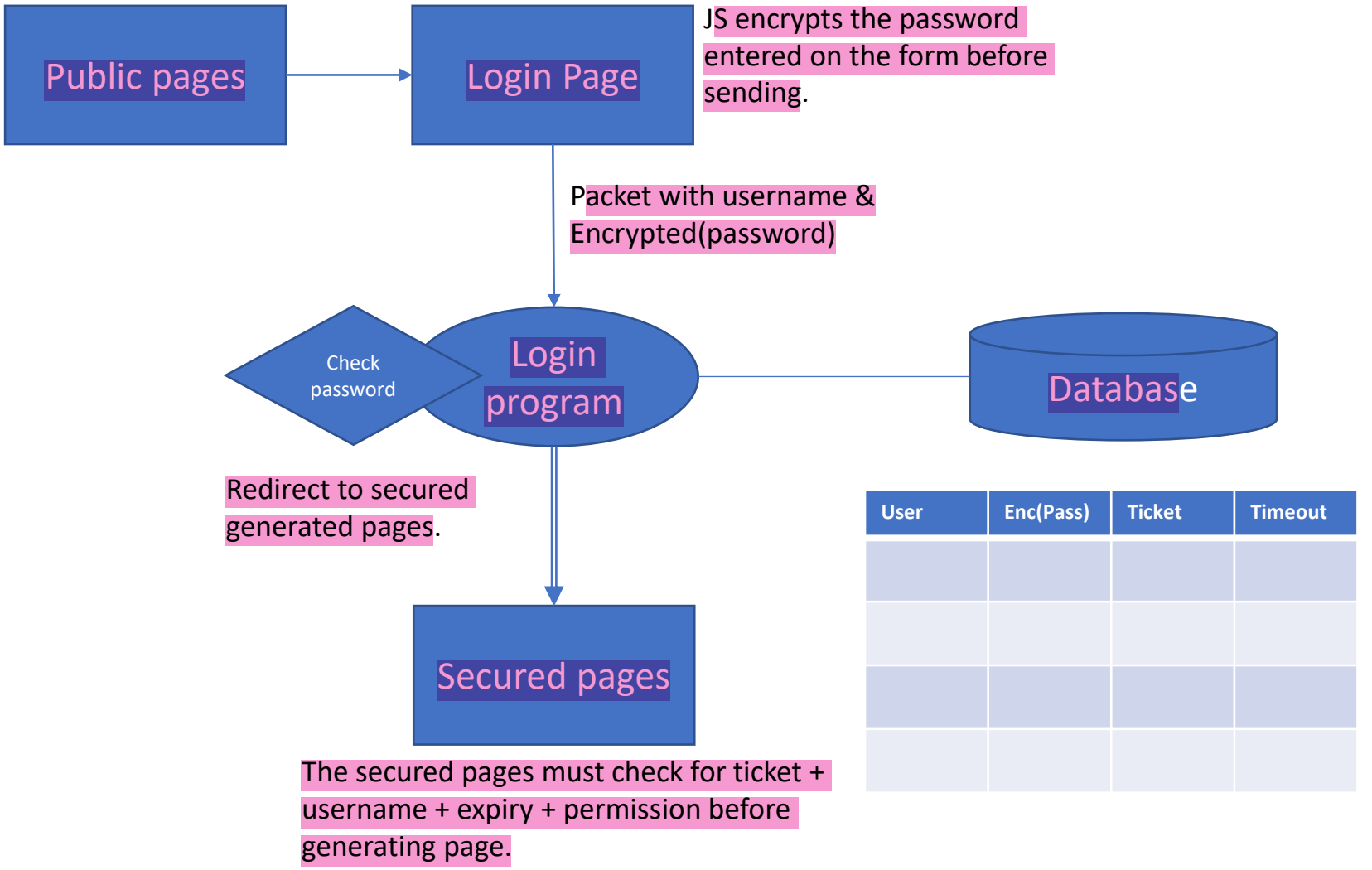
Good or Bad?

- Benefits
  - Easy to implement
    - A ticket is obtained by validating your identity
      - Provide a valid username/password
      - Provide a valid biometric image
      - Provide a valid physically scanned key-card (like metro pass)
- Drawbacks
  - Tickets can be stolen
    - It is important to change tickets from time-to-time
      - At each login and/or after a fixed amount of time (every 2 hours)
    - Alternatively, use different tickets for different permissions
      - Implement an expiry date per permission
    - Alternatively, do a combination of the two.

Contents

Practical Security

Basic Implementation



The Login Page

```
<html>
  <script>

     function encrypt(message, key) {
        … encryption code …
        return cipher_message;
     }

     function sendLoginRequest() {
        username = document.getElementById('user').value;
        password = document.getElementById('pass').value;
        encrypted_password = encrypt(password, Storage.key);

        … Ajax synchronous call to login.php with …
        … username & encrypted_password …
        … if login successful, store ticket in local storage …
     }
  </script>

  <body>
     <form>
        Username <input id="user" type="text" name="username"><br/>
        Password <input id="pass" type="password" name="pass"><br/>
        <button type="button" onclick="sendLoginRequest()">Login</button>
    </form>
  </body>
</html>
```

The Login Program

1. Username & encrypted_pass from Packet
2. Query DB with username & get DB_pass
3. If not found, then return error
4. If both encrypted passwords are equal, then
    1. Compute a randomized new ticket ID number (or chars+digits)
    2. Save ticket in Server DB with expiration date/time
    3. Return ticket to browser
        - Version 1: login.html has JS to store ticket and redirection code to dashboard.php
        - Version 2: login.php generates dashboard.php with the ticket in an onload JS program to store ticket into browser storage, if not already there.
5. Else
    - Return error message

Contents

Practical Security

Login Password Checking

- Example, returning the ticket to existing webpage
  - User types: www.website.com/login.html at browser
  - User inputs username & password in form and presses submit
  - Server finds form's action program: login.php

```
<?php
  :
  sql = "select * from valid_users where username=".$_POST[].
            " and password=".$_POST[]"
  :
?>
```

  - Executes PHP/PY/C
  - Loads packet with output
  - Version 1, Adds: <input type="hidden" name="ticket" value="ID#">
  - Version 2, returns JSON {'ticket':ID#}
  - Send to browser
  - Browser's JS saves to browser's storage

Contents

Practical Security

Login Password Checking

- Example, not returning ticket, instead generating a private webpage:

```
<?php
        // Get the username and password from $_POST
        …

        if (SQL with $_POST found in server database)
        {
                generate ticket & save in server database
                display the website, with JS to save ticket on client
        }
        else
        {
                display a 400, or 403, 404 error
        }
?>
```

Contents

Practical Security

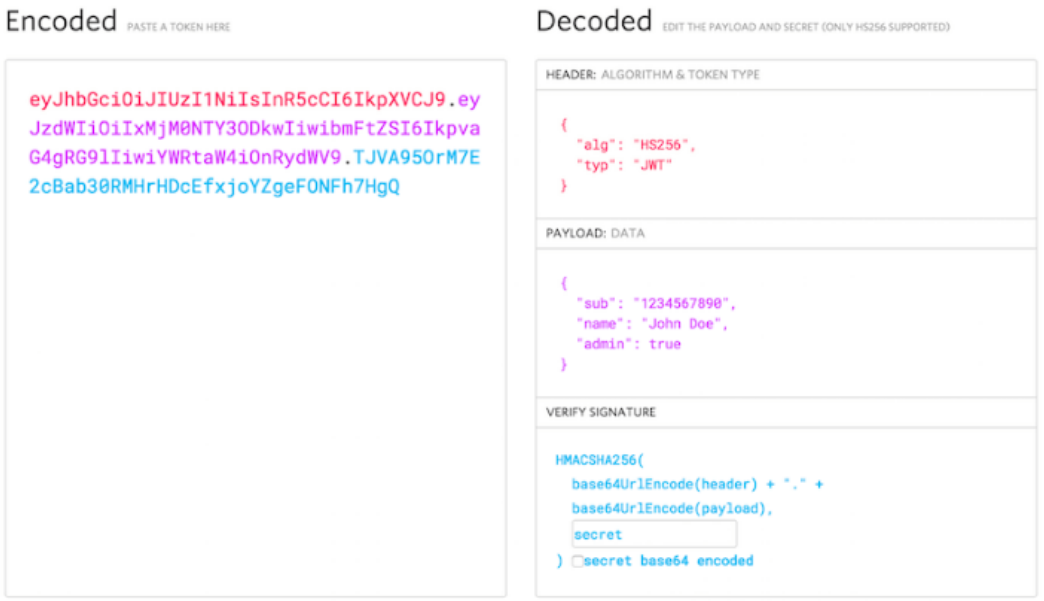
generates tickets for you

JWT Web tokens

https://www.bezkoder.com/jwt-json-web-token/

- JSON web token (JWT), pronounced "jot", is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.

Uses hash to skip DB search



For example, a server could generate a token that has the claim "logged in as administrator" and provide that to a client. The client could then use that token to prove that it is logged in as admin. The tokens can be signed by one party's private key (usually the server's) so that any party can subsequently verify the token is legitimate.

Webpage security

Security 3

What is webpage security

- After login, all other pages are secure
  - Typing the URL of a secure webpage directly results in an error message
  - Two forms:
    - Regular webpages
      - A regular webpage is an actual **page.html** file
    - Generated webpages
      - A generated webpage is a **page.php** program
      - The program can be in any language (not only php)

Contents

Practical Security

Regular webpage security

```
<script>

  const xhttp = new XMLHttpRequest();

  xhttp.onload = function() {

    if (this.responseText == "notValid")

          window.location.replace("http://www.abc.com/My404Error.html");

  }

  xhttp.open("GET", "validateUser.php");

  xhttp.send("ticket="+localStorage.getItem("Ticket")+
             "user="+localStorage.getItem("Username"));

</script>
```

Note: This is not in a function. We want this to execute as soon as the page loads.

validateUser.php:

SQL username → ticket number & expiry date
if (ticket number == $_GET["ticket"] && date() < expiry date) print("Valid")
else print("notValid");

Generated webpage security

Invoking a webpage generator from a webpage interaction

```
<script>
  function userClick() {
     window.location.replace("http://www.abc.com/dashboard.php?"+
         "ticket="+encrypt(localStorage.getItem("Ticket"))+
         "user="+encrypt(localStorage.getItem("Username")));
  }

</script>
```

The generator

```
  <?php
     // must be the very first thing in script
     sql = "select * from users where username=".decrypt($_POST["Username"])

     // check for the following
     // 1. Only one record was found (username is unique)
     // 2. Check that $_POST["ticket"] == DB ticket
     // 3. Check date() < DB expiry date
  if (recordcount==1 && decrypt($_POST["Ticket"]==Dbticket && date()<Dbexpire)
  { print the webpage }
  else { print 404 error message }
 ?>
```

Encryption

Security 3

Securing communication

- We want to secure information when:
  - Sending payloads with private data
    - Passwords, ticket IDs, credit card numbers, etc.
  - After we have successfully logged in all information needs to be restricted
- We have already seen how to restrict access to pages after logging in
- We want to look at encrypting payloads
  - All payloads need to be encrypted

Contents

Practical Security

Types on encryption

- Modern encryption rules:
  - Knowing the algorithm should not give any advantage to the bad-guy.
  - **Kerckhoff's Principle**: a cryptosystem should be secure, even when everything about the system, except the key, is public knowledge.
- Modern encryption algorithms:
  - Single key examples
    - DES ~ not so good these days
    - AES
  - Two key example
    - RSA Rivest–Shamir–Adleman Public – Private Key System – used in SSH
  - Hashing example
    - DSA for hashing

Contents

Practical Security

Single key encryption

```
<script>
        function encrypt(message, key) {
                … encryption algorithm …
                return cipher;
        }

        function decryption(cipher, key) {
                … decryption algorithm …
                return message;
        }

        function ajaxCall(publicMessage) {
                … do ajax call without encryption …
        }

        function secureAjaxCall(privateMessage) {
                cipher = encrypt(privateMessage, storage.key);
                … do ajax call with cipher in payload
        }
</script>
```

In a library this is the API

Contents

Practical Security

Two key encryption

```
<script>
        function encrypt(message, key_server) {
                … encryption algorithm …
                return cipher;
        }

        function decryption(cipher, key_user) {
                … decryption algorithm …
                return message;
        }

        function ajaxCall(publicMessage) {
                … do ajax call without encryption …
        }

        function secureAjaxCall(privateMessage) {
                cipher = encrypt(privateMessage,storage.pub_s);
                … do ajax call with cipher in payload
        }
</script>
```

Contents

Practical Security

Crypto Libraries in JS

https://gist.github.com/jo/8619441

Prepare for Next Class

- Assignments
  - Mini 6 (extended – no late penalty)
  - Project handout – pick your teams (3 people)
- Lab this week
  - Lab E
- Do on your own
  - Further study:
    - COMP 547 Cryptography and Data Security (4 credits)
    - COMP 647 Advanced Cryptography (4 credits)
  - Implement the login strategy described here

Contents

Practical Security