0x03 Access Control Exercise Session

Recap

Authentication

*Authentication is the process of **verifying** someone's or something's **identity***

Downside of naive password-based authentication

Authentication Factors

- Something you **know**: passwords, pin codes
- Something you **own**: paper card, smartphone, certificate, electronic token
- Something you **are**: biometrics

Recall authentication mechanisms listed in Lecture, what type of authentication factor are they? What are their Pros and Cons?

Access Control

Access control *defines and enforces* ***operations*** *that* ***subjects*** *can do on* ***objects***

Principle of Least Privilege: subjects only have the minimum rights required for their job

What are examples, pros and cons of the following access control schemes?

- Role-based Access Control (RBAC)
- Discretionary Access Control (DAC)
- Mandatory Access Control (MAC)

Authentication Protocol

Downsides of naive password authentication, and how does **challenge-response** authentication solve them?

Key ideas of Kerberos: **delegated authentication** and **separation of concerns**

What are roles of **AS** and **TGS** in Kerberos?

Oauth2: bringing the idea of delegated authentication to the internet.

Recall Oauth2 workflow and the role of user, client, authentication server, and resource server.

Demo: Cookie Tampering

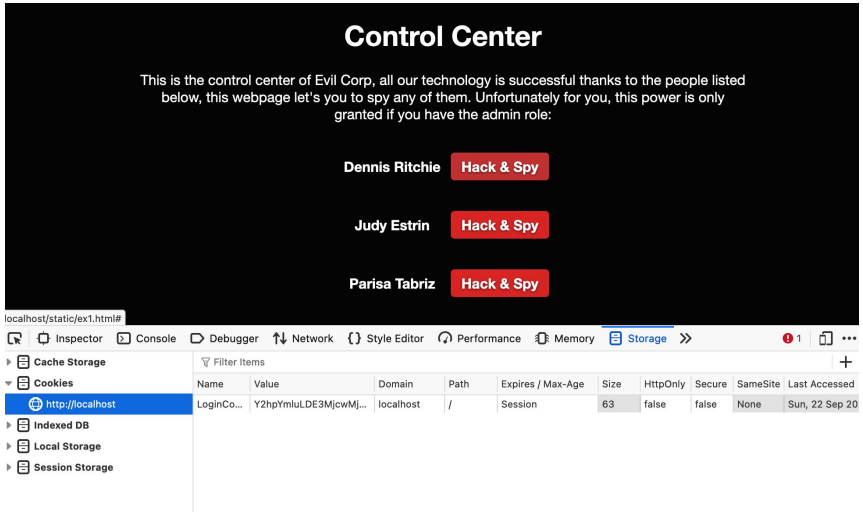
Cookie Tampering

Websites usually store user session (login) information in cookies…

What if the cookies are **unauthenticated?**

Docker image on DockerHub

Run `docker run -it -p 80:80 com402/hw3ex13`



Sucess!: Cookie tampering is not that hard

Cookie Tampering: `server.py`

```python
# @cross_origin()
@ex_attack.route("/api/ex1/auth",methods=['POST'])
def ex1_list():
    if not cookie_name in request.cookies:
        return "", unauthorized

    cookie = request.cookies[cookie_name]
    if is_admin_cookie(cookie):
        return "Cookie tampering is not that hard", success

    return "", unauthorized

def is_admin_cookie(cookie):
    infos = base64.b64decode(cookie).decode('utf-8').split(",")
    return len(infos) == 6 and infos[5] == "admin"
```

Key issue: server does not check whether the cookie is the original one it sent to the client

HMAC for Cookies

Now that we know **unauthenticated** cookies are bad, how do we fix it?

Solution: HMAC!

- More specifically, store a hash checksum (with a secret key) of the credentials and store it alongside in the cookie.
- When reading the cookie provided by the client, verify its content is intact by recomputing and comparing the checksum (HMAC).
- The client can not compute checksum because it does not know the secret key.
- Let's implement it in `server.py`!

HMAC for Cookies

```
def create_cookie(username, password):
    ...
    # create the "base" cookie, that will be
    base_cookie = ",".join([username,t,domair

    #create the HMAC
    my_hmac = hmac.new(SECRET_KEY_YOU_WILL_NE
                       base_cookie.encode(), #mes
                       sha1)
    final_cookie = base_cookie + "," + my_hma
    # base64encode the cookie (with .encode()
    # then back to utf-8 for a readable cook
    return base64.b64encode(final_cookie.enc
```

```
def cookie_validate(cookie):
    try: # optional: try to decode, but don't trust too much
        decoded = base64.b64decode(cookie.encode()).decode("utf-8")
    except:
        # The cookie isn't even a base64 encoding anymore,
        # it definitely has been tampered with...
        return False
    # separate base cookie from hmac
    base_cookie, cookie_hmac = decoded.rsplit(",", 1)
    # re-compute the hmac from the base cookie
    reference_hmac = hmac.new(SECRET_KEY_YOU_WILL_NEVER_FIND,
                              base_cookie.encode(),
                              sha1).hexdigest()

    # return whether or not they are equal
    return cookie_hmac == reference_hmac
```

HMAC for Cookies

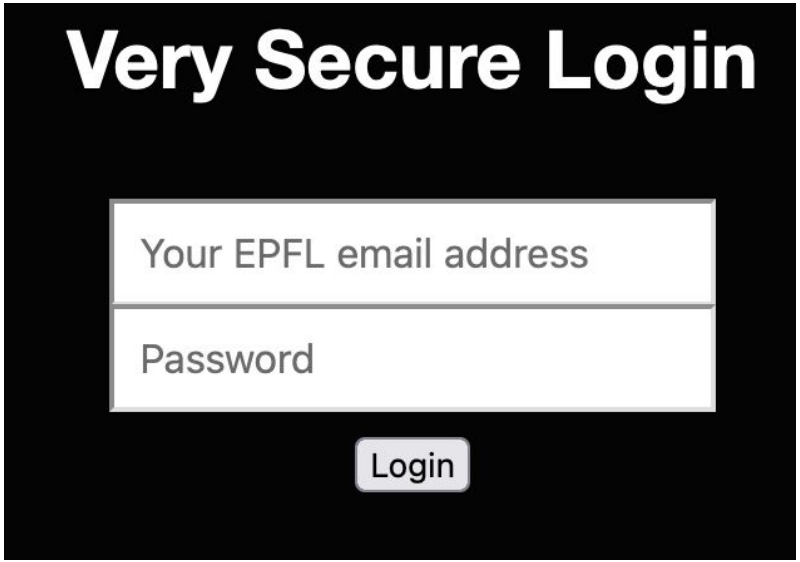
```
$ curl --cookie-jar /tmp/cookie -d 'username=u&password=p' -X POST http://localhost:5000/login
Logged in⏎
$ cat /tmp/cookie
# Netscape HTTP Cookie File
# https://curl.se/docs/http-cookies.html
# This file was generated by libcurl! Edit at your own risk.
localhost   FALSE   /   FALSE   0   LoginCookie
dSwxNzI3MDM0MTEwLGNvbTQwMixodzMsZXgyLHVzZXIsODg5NGU2N2RmZDUxYmYwMDZiMTA1YTM2NDJiN2I3YmJhZjA3NzM2MA==
$ echo dSwxNzI3MDM0MTEwLGNvbTQwMixodzMsZXgyLHVzZXIsODg5NGU2N2RmZDUxYmYwMDZiMTA1YTM2NDJiN2I3YmJhZjA3NzM2MA==
| base64 -d
u,1727034067,com402,hw3,ex2,user,bad06fe5551955cf94e2767df35cf0f7153242fa⏎
$ curl http://localhost:5001/auth
Naughty, naughty, naughty...⏎
$ curl -b /tmp/cookie http://localhost:5000/auth
User⏎
```

Client Side Authentication

What if authentication is implemented, but at client side in JavaScript?

Go to `localhost/static/ex3.html`

Right click and View Page Source…



Client Side Authentication

```
function hash(msg,key) {
   if (key.length < msg.length) {
       var diff = msg.length - key.length;
       key += key.substring(0,diff);
   }
   var amsg = msg.split("").map(ascii);
   var akey = key.substring(0,msg.length).split("").map(ascii);
   return btoa(amsg.map(function(v,i) {
       return v ^ akey[i];
   }).map(toChar).join(""));
}
// …
   if (password != hash(username,mySecureOneTimePad)) {
       alert("I didn't say it would be easy, Neo. I just said it would be the truth.");
       return;
   }
```

Key issue: client can always compute password, or bypass JS altogether!