6.857 Final Project: JWT Web Security

Fu, Jamie
jamiefu@mit.edu

Liu, Katherine
liukat@mit.edu

Liu, Richard
liuri@mit.edu

Wong, Anna
annawong@mit.edu

May 2022

Contents

Abstract

We examine JSON web token (JWT) use cases and common attacks and vulnerabilities present in the JWT scheme. In particular, we focus on two MIT applications, COVID Pass and Spectacle, and their usage of JWT. We attempt some common JWT attacks on COVID Pass, and although these attacks were not successful, we find that the storage of JWT in both applications is not secure.

1 Introduction

There are several different types of tokens that web applications utilize to verify a user's identification, check their access credentials, re-authenticate a user, and send user information to third parties. A JSON web token, or JWT, is an open standard, compact, URL-safe method of securely transferring information between two parties, typically between a client and a server on a web application [1]. JWTs provide users the ability to transfer information that is signed and can be verified using either a shared secret (HMAC) or a public and private key pair algorithm (RSA, ECDSA). Typically, JWTs are encoded in Base64, and occasionally applications will add an extra layer of security over the JWT by encrypting it [2].

This paper will focus on the signature and authentication benefits that JWTs provide web applications. We focused our research on two web applications in the MIT sphere: MIT COVID Pass, an application that tracks all student information regarding COVID including vaccine status, attestation (deprecated), test results, etc, and Spectacle, a HackMIT platform where hackers can upload and view projects from the hackathon. These two platforms utilize JWTs for user authentication but have differing strategies on how to store and handle them. This paper presents several several common vulnerabilities with JWTs, our investigation into these two platforms and several attacks we attempted, and closes with some best practices and methods to enhance the security of JWTs.

2 JWT Scheme

To understand the attacks and vulnerabilities of JWTs enumerated in this paper, it is important to understand the general construction and structure of a JWT and where they are most frequently used.

2.1 Construction

The structure of a JWT looks like: `header.payload.signature`, where the header, payload, and signature are the three sections of the token, each separated by a period. Figure 1 shows an example JWT where each section is a different color. The header (red) contains information about the token itself, such as the token type (JWT) and the algorithm used to sign the JWT. The payload (purple/pink) contains the relevant data the sender wishes to convey to the receiver. The signature (blue) takes the header and the payload, encodes them in base64, then applies the signing algorithm specified in the header to produce a valid signature with the agreed upon secret key or a public/private key pair. When the receiver receives the JWT, they will decode the text, and before accepting the payload, they will authenticate the signature with either the shared secret key or a public/private key pair.

If the JWT was signed by a malicious party, the receiver will (ideally) be unable to authenticate the signature and thus ignore the rest of the JWT. As you may imagine, much of the security of the JWT relies on the security of the signing algorithm the platform decides to utilize, however, attackers can still exploit the construction and general JWT use practices to glean information.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKx
wRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

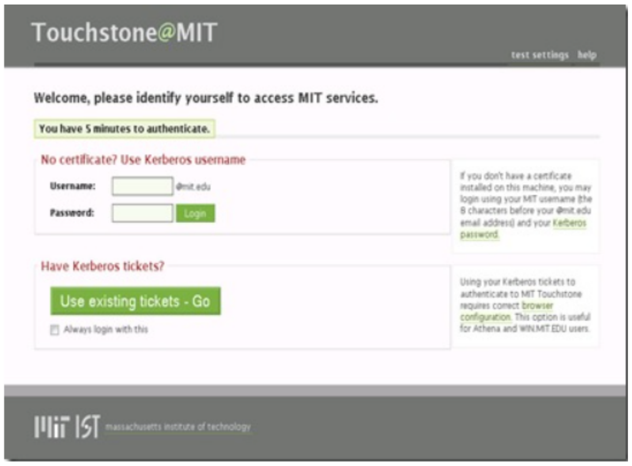
Figure 1: Sample JSON Web Token

2.2 Use Cases

Most commonly, JWTs are used for user authentication in Single Sign On (SSO) for web applications. After logging on to a platform, users are typically not expected to log in again while navigating the same web page for the duration of their session. Any activity within that session such as routing and access to services or resources within the platform will trigger the transfer of a JWT. When navigating from one endpoint in the site to another, the JWT will be sent along with the request, and it will be authenticated by the receiving endpoint before the user who sent the token is allowed to navigate to that endpoint. This process is typically very quick and produces no discernible lag in access times.

An example of a platform that utilizes JWTs to authenticate user activity on the web is Touchstone@MIT. Every MIT student or faculty has likely encountered the Touchstone login prompt shown in Figure 2. Once logged in, Touchstone will generate a JWT to pass to the application the user is attempting to access, and the user's access is granted just as described in the previous paragraph.

JWTs can also be used outside the world of user authentication for information sharing. As you might imagine, the ability to sign and verify the information being passed without having to produce an extraordinarily long string is quite useful. However, this paper will focus on examining the security of JWTs used for user authentication, since that is the primary use case of JWTs.



Welcome, please identify yourself to access MIT services.

You have 5 minutes to authenticate.

If you don't have a certificate installed on this machine, you may login using your MIT username (the 8 characters before your @mit.edu email address) and your Kerberos password.

Using your Kerberos tickets to authenticate to MIT Touchstone requires correct browser configuration. This option is useful for Athena and WIN.MIT.EDU users.

Figure 2: SSO Example, Touchstone Authentication at MIT

3 Common JWT Vulnerabilities

3.1 Token Capture

Token capture is when an adversary is able to get a hold of a user's token, perhaps by intercepting a message containing the token. Once a token has been captured, it is easy for an adversary to reveal private information, as an adversary can simply use the `base64UrlDecode` function on the payload section of the token in order to reveal the payload's information. Furthermore, an adversary with this token can reuse the token in order to use the application as the user.

One way to avoid a token being captured is to use a secure connection when transferring the token. We can also avoid sending private or sensitive information in tokens, so that if a token is captured, this information will not be revealed. We can also set an expiration date for the access token and use refresh tokens to give the user a new access token when the old one expires. [3]

3.2 None Algorithm Attacks

The signing algorithm for a token, specified in the header of the token, is usually some form of RSA or HMAC. However, for debugging purposes, the algorithm type in the JWT header is allowed to be specified as "none". When the algorithm is "none", that means the token is not signed, and with no signature, we cannot check that the token is authentic. Thus, when the algorithm type is "none", the application does not check for a signature since one does not exist. As a result, an adversary could change crucial information in a payload and still get the application to accept a token by changing the algorithm field to "none". One way to prevent such an attack is to have the application keep a list of what algorithms are authorized, and reject any tokens that are not signed using an algorithm on the list. [4]

3.3 `kid` Attacks

Another parameter for the header section of the token is the "kid" parameter (key ID). There's no strict definition for how this parameter should be formatted. As a result, attacks such as SQL injections or path traversal attacks can be inserted into this field. To protect against such attacks, a developer can check the input from the token to make sure it is not a malicious command. [3]

3.4 JKU and JWK Header Parameter Attacks

Similar to the `kid` attack, there are other header parameters that can be exploited by an adversary. The JKU (short for "JWK Set URL") parameter is an optional field that be used to pass in a URL that gives the keys used for verification. If precautions are not taken to protect this field, an adversary could pass in a URL for their own keys and thus have the application verify tokens using the adversary's key. Similarly, the optional JWK (short for "JSON Web Key") parameter can be used to put the verification key directly inside the token, so an adversary can pass in its own key. [5]

3.5 Signature Algorithm Attacks

JWT allows for a variety of different signing algorithms, some of which use public/private keys (RSA), and some of which use shared secret keys (HMAC). When it comes to verifying a signature, a symmetric algorithm like HMAC will use the secret signing key for verification, whereas an asymmetric algorithm like RSA will use the public key. However, an adversary can send a token signed with the HMAC secret key to an application expecting a token signed with the RSA secret key, and this will cause the application to mistakenly believe that the RSA public key is an HMAC secret key. The adversary can then create and sign tokens that will be accepted by the application because they can sign a payload using the RSA public key as a fake HMAC secret key. [4]

3.6 Brute Force Attacks

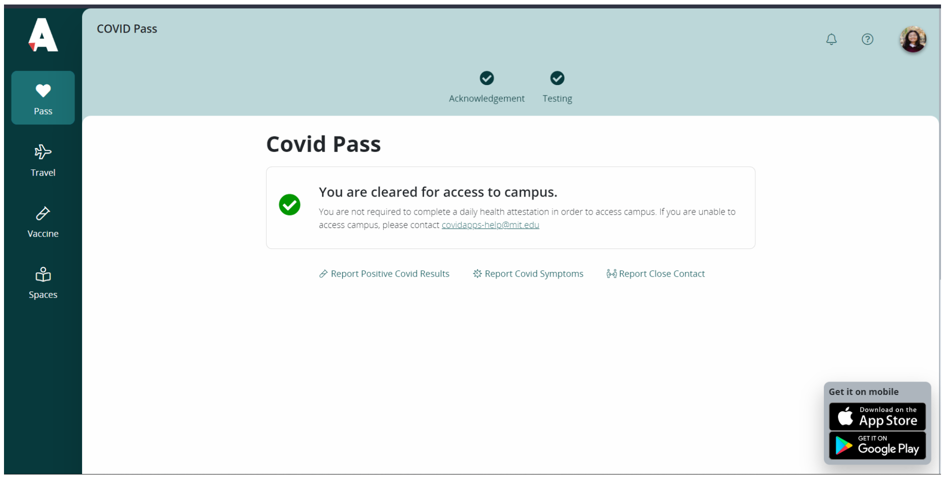
JWT is also susceptible to brute force attacks on the secrets for their signing algorithms. In the HS256 signing algorithm, if a short and weak secret key is used, it is possible to brute force the key. An adversary can then use the secret key to sign other tokens. These attacks can be prevented by following general guidelines for secret keys in cryptography, such as making the key long with many different characters, and occasionally changing the secret key. [3]

4 COVID Pass Investigation

COVID Pass is MIT's contact-tracing, testing, and access system. It has been used on campus since Fall 2020 due to the COVID-19 pandemic in order to keep the community safe. Up until this March 2022, users were required to submit a daily attestation, leading to many students attempting to automate the process in some way. In a previous semester some students were successful in automating the process through a browser interaction script. In our investigation we look specifically at COVID Pass's usage of JWT and look for vulnerabilities and exposed information. Furthermore, we also examine the possibility of a headless browser automation for attestation.

4.1 Token Capture

One of the major vulnerabilities of JWT is the dangers of token capture. Because JWT itself does not enforce how tokens are actually stored on the client-side, many applications misuse it and inadvertently expose their tokens to both users and adversaries. In the case of COVID Pass, both an access token and ID token can be found in the Local Storage, among other relevant items.

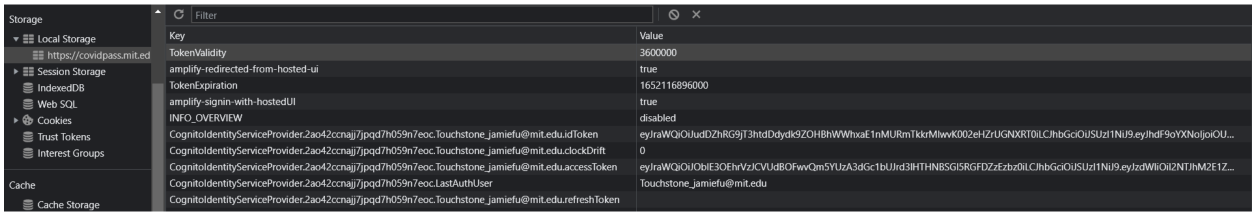


Covid Pass

You are cleared for access to campus.

You are not required to complete a daily health attestation in order to access campus. If you are unable to access campus, please contact covidapps-help@mit.edu

Figure 3: Covid Pass Web Application [6]



Key	Value
TokenValidity	3600000
amplify-redirected-from-hosted-ui	true
TokenExpiration	1652116896000
amplify-signin-with-hostedUI	true
INFO_OVERVIEW	disabled
CognitoIdentityServiceProvider.2ao42ccnajj7jpqd7h059n7eoc.Touchstone_jamiefu@mit.edu.idToken	eyJraWQiOiJudDZhRG9jT3htdDdydk9ZOHBhWWhxaE1nMURmTkkrMlwvK002eHZrUGNXRT0iLCJhbGciOiJSUzI1NiJ9.eyJhdF9oYXNoIjoiOU...
CognitoIdentityServiceProvider.2ao42ccnajj7jpqd7h059n7eoc.Touchstone_jamiefu@mit.edu.clockDrift	0
CognitoIdentityServiceProvider.2ao42ccnajj7jpqd7h059n7eoc.Touchstone_jamiefu@mit.edu.accessToken	eyJraWQiOiJObIE3OEhrVzJCVUdBOFwvQm5YUzA3dGc1bUJrd3lHTHNBSGI5RGFDZzEzbz0iLCJhbGciOiJSUzI1NiJ9.eyJzdWIiOiI2NTJhM2E1Z...
CognitoIdentityServiceProvider.2ao42ccnajj7jpqd7h059n7eoc.LastAuthUser	Touchstone_jamiefu@mit.edu
CognitoIdentityServiceProvider.2ao42ccnajj7jpqd7h059n7eoc.Touchstone_jamiefu@mit.edu.refreshToken	

Figure 4: Covid Pass Local Storage Sample, Jamie Fu

There are a few noteworthy fields in Local Storage:

1. `TokenValidity`
   In epoch milliseconds, `TokenValidity` represents the time that each token is valid for. In the case of COVID Pass, 3600000 ms equals to exactly 1 hour.

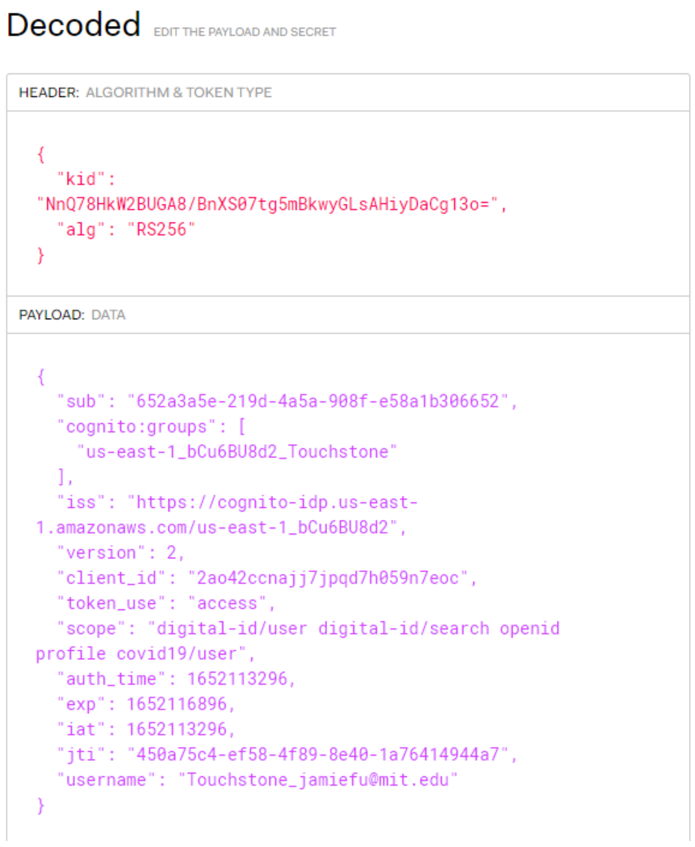
2. `TokenExpiration`
   Also in epoch milliseconds, `TokenExpiration` refers to the time of expiration of the currently issued tokens. In Figure 4, 1652116896000 translates to May 9, 2022 13:21:36.

3. `CognitoIdentityServiceProvider...idToken`
   A quick lookup of the `CognitoIdentityServiceProvider` prefix points to Amazon Cognito, an Amazon service that "handles user authentication and authorization for your web and mobile apps" [7]. The format of the token itself indicates base64 encoding, which implies JWT roots. A decoded header and payload can be found in Figure 5.

   There are several items in the payload, but as the information is only base64-encoded, it is entirely exposed to anybody who sees this token. For example, despite my name and email most likely being linked information publicly, my `employeeID` (blacked out) reveals my student ID number, which I may not want others to know. On campus, my student ID number allows me to tap into dining halls, check out library books, and much more.

4. `CognitoIdentityServiceProvider...accessToken`
   The access token, decoded in Figure 5, contains less personally-identifying information. However, this access token is passed in every single personally-information API request. Holding this token essentially means holding the ability to obtain personal health information about the associated user. This is discussed further below.

5. `CognitoIdentityServiceProvider...refreshToken`
   With JWT, users can expire old tokens and use a refresh token to generate new access tokens. In proper usage, the refresh token should always stay on the server-side, and the client should never have access to it. Having access to the refresh token is almost like having access to the signing key of the token itself. Despite having a field for the refresh token in Local Storage, the actual value of this field is empty. This indicates that it's possible that the authors of Covid Pass may have initially wanted to place the refresh token on the client side, then realized that doing so would expose more than just a one-time token, it would give adversaries the ability to generate their own tokens.



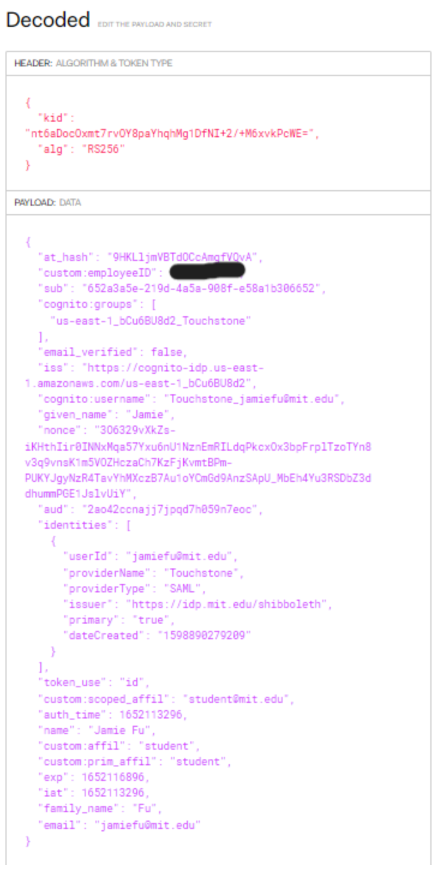
Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "kid":
"NnQ78HkW2BUGA8/BnXS07tg5mBkwyGLsAHiyDaCg13o=",
  "alg": "RS256"
}
```

PAYLOAD: DATA

```
{
  "sub": "652a3a5e-219d-4a5a-908f-e58a1b306652",
  "cognito:groups": [
    "us-east-1_bCu6BU8d2_Touchstone"
  ],
  "iss": "https://cognito-idp.us-east-
1.amazonaws.com/us-east-1_bCu6BU8d2",
  "version": 2,
  "client_id": "2ao42ccnajj7jpqd7h059n7eoc",
  "token_use": "access",
  "scope": "digital-id/user digital-id/search openid
profile covid19/user",
  "auth_time": 1652113296,
  "exp": 1652116896,
  "iat": 1652113296,
  "jti": "450a75c4-ef58-4f89-8e40-1a76414944a7",
  "username": "Touchstone_jamiefu@mit.edu"
}
```



Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "kid":
"nt6aDocOxmt7rvOY8paYhqhMg1DfNI+2/+M6xvkPcWE=",
  "alg": "RS256"
}
```

PAYLOAD: DATA

```
{
  "at_hash": "9HKL1jmVBTdOCcAmgfVQvA",
  "custom:employeeID": [redacted]
  "sub": "652a3a5e-219d-4a5a-908f-e58a1b306652",
  "cognito:groups": [
    "us-east-1_bCu6BU8d2_Touchstone"
  ],
  "email_verified": false,
  "iss": "https://cognito-idp.us-east-
1.amazonaws.com/us-east-1_bCu6BU8d2",
  "cognito:username": "Touchstone_jamiefu@mit.edu",
  "given_name": "Jamie",
  "nonce": "306329vXkZs-
iKHthIir0INNxMqa57Yxu6nU1NznEmRILdqPkcxOx3bpFrplTzoTYn8
v3q9vnsK1m5VOZHczaCh7KzFjKvmtBPm-
PUKYJgyNzR4TavYhMXczB7Au1oYCmGd9AnzSApU_MbEh4Yu3RSDbZ3d
dhummPGE1JslvUiY",
  "aud": "2ao42ccnajj7jpqd7h059n7eoc",
  "identities": [
    {
      "userId": "jamiefu@mit.edu",
      "providerName": "Touchstone",
      "providerType": "SAML",
      "issuer": "https://idp.mit.edu/shibboleth",
      "primary": "true",
      "dateCreated": "1598890279209"
    }
  ],
  "token_use": "id",
  "custom:scoped_affil": "student@mit.edu",
  "auth_time": 1652113296,
  "name": "Jamie Fu",
  "custom:affil": "student",
  "custom:prim_affil": "student",
  "exp": 1652116896,
  "iat": 1652113296,
  "family_name": "Fu",
  "email": "jamiefu@mit.edu"
}
```

Figure 5: Covid Pass Access Token (left) and ID Token (right) Sample, Jamie Fu

Placing tokens in Local Storage is particularly dangerous because these they can be grabbed using XSS attacks. Furthermore, there are many fields where users are asked to for textual or photo input, such as COVID test barcodes or vaccine photo uploads. If these inputs are not properly sanitized, JavaScript can also be executed within these inputs.

In addition, the access token is sent as part of the Request Headers in many major API requests. This means that tokens can also be grabbed from intercepted requests. We discuss this further in the following section, Section 4.2.

4.2 API Endpoint Access

Exposing the access token reveals a whole host of personal information accessible to an adversary. We have mapped a set of exposed API endpoints that can be hit by attaching the proper JWT access token to request headers. For brevity and security we do not always explicitly attach the response, but will instead examine the implications of the fields that are available to the user and adversary.

Note that these are endpoints that can *only* be accessed by attaching the proper access token, but they are also *other* endpoints *not* listed here that are also accessible with the proper access token.

1. `GET https://api.mit.edu/pass-v1/pass/access_status`
   This endpoint returns a large JSON body response indicating the access ability of the user on the site itself. This includes health acknowledgement, vaccine, and attestation status. Due to changes in MIT policy, many of the fields are actually deprecated (notably attestation), but the response values still persist.

2. `PUT https://api.mit.edu/pass-v1/pass/access_status`
   Payload: {`"status_id":6`}
   This endpoint allows users to update their own access status on the COVID Pass website. This endpoint is only hit when a user mistakenly reports symptoms, then clicks the "I made a mistake" button. We believe that this endpoint should have higher priority in being secured than other `GET` requests because it allows an adversary to actually write to the COVID Pass database. The `status_id` field in the payload with value 6 indicates a normal access status. However, further testing revealed that other surrounding numbers (0-5, 7-10) are considered "invalid" responses for some reason. This may indicate that certain statuses have been phased out, as 6 is an oddly-selected number.

3. `GET https://api.mit.edu/pass-v1/pass/building_access`
   This endpoint reveals the "special" buildings that a user has access to. For on-campus users, at least one of these buildings is what dorm the user actually lives in. On initialization, both this endpoint and the `access_status` endpoints are called.

4. `GET https://api.mit.edu/medical-v1/unobserved/status`
   This endpoint concerns the unobserved testing that was implemented starting Fall 2021. A sample set of values returned are:

```
{
    "canSubmit": true,
    "nextMedicalTestDate": null,
    "barcode": null,
    "isOpen": false,
    "requireUserInfoUpdate": false,
    "canCancelSubmittedTest": false
}
```

5. `POST https://api.mit.edu/pass-v1/pass/report_symptoms`
   Payload: {`"has_symptoms":true`}
   We also believe that this endpoint is dangerous because it allows an adversary to attest to the victim having symptoms when they may not actually be exhibiting. This is almost a silent effect–no notifications are sent to the victim to confirm their symptoms after the original `POST` request. After an adversary posts to this endpoint, the victim's campus access is restricted, and a cascade of consequences occurs. MIT Medical will contact this individual, especially if they continue to try to access campus during this period when they do not realize their access has expired.

6. `GET https://api.mit.edu/vaccine-v1/vaccine/status`
   This endpoint returns many pieces of information about the user's vaccines. Not only does it return the number of vaccines this user has received, it also indicates the vaccine type, administration date, location, and lot numbers. An image URL is attached linking to an Amazon S3 bucket named "vaccine-master" with a direct link and security token in the parameters to access the photo proof.

7. `POST https://api.mit.edu/pass-v1/pass/attestations`
   See Figure 6 for payload.
   The payload reveals that the COVID Pass authors did not fully phase out old questions, as the IDs are interspersed and do not start from index 0. Furthermore, the server does not verify the types of the payload before processing the response. Therefore, if the boolean values are accidentally sent as

string values, they will all be intepreted as true, indicating that the user is attesting to symptoms.

In recent months the daily attestation requirement was fully disassembled, and elements from the COVID Pass web UI were removed. However, if a user selects "Check in at a Testing Center", they can still access this form (which no longer has any impact on their access). Prior to the removal of the requirement, if an adversary had access to this endpoint, they could build a false payload and make the user attest to having COVID symptoms, which like the `report_symptoms` endpoint, would put a hold on campus access and alert MIT Medical.

8. `GET https://api.mit.edu/digital-id-v1/pin`
   This is a relatively new endpoint added to COVID Pass that generates a six-digit pin code that supposedly "can be used to authenticate you in the COVID-19 IVR system or disarm alarm panels you have been granted access to" [6]. However, this pin is permanent and never changes on re-request or token refreshes. This is concerning because an adversary merely needs to pull the access pin once and can use it forever. As of now, our team has no use for the access pin in our daily lives as students on campus, but it does leave us wondering whether future versions of campus access will leverage this pin to access alarm panels.

4.3 Attestation Automation

As indicated by the previous section, attestation was one of the major requirements that every user on the COVID Pass system needed to fulfill on a daily basis to access campus. Previous projects have examined how one might use Selenium to simulate browser interaction and attest automatically every day. Due to our research with JWT and API endpoint access, we believed that we could write a script to automatically attest on a regular schedule. Recall that we can build a request with the payload described in Figure 6.

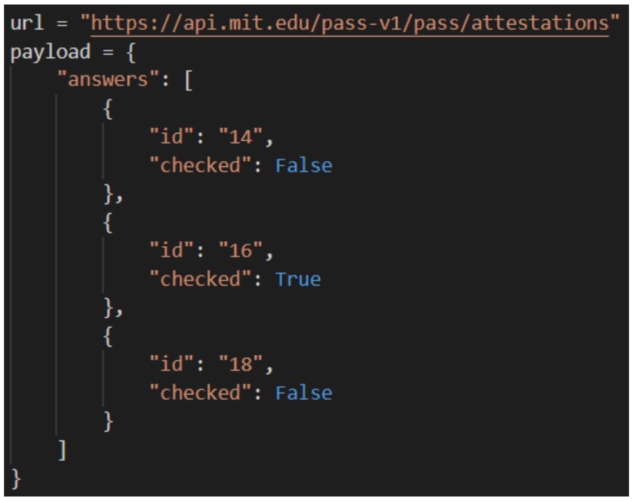

```
url = "https://api.mit.edu/pass-v1/pass/attestations"
payload = {
    "answers": [
        {
            "id": "14",
            "checked": False
        },
        {
            "id": "16",
            "checked": True
        },
        {
            "id": "18",
            "checked": False
        }
    ]
}
```

Figure 6: Covid Pass Attestation Endpoint and Payload

We constructed the following simple script:

```
url = "https://api.mit.edu/pass-v1/pass/attestations"
payload = {
    "answers": [
        {
            "id": "14",
            "checked": False
        },
        {
            "id": "16",
            "checked": True
        },
        {
            "id": "18",
            "checked": False
        }
    ]
}

headers = {'authorization': 'Bearer
    eyJraWQiOiJOblE3OEhrVzJCVUdBOFwvQm5YUzA3dGc1bUJrd3lHTHNBSGl5RGFDZzEzbzOiLCJhbGciOiJSUzI1NiJ9.
    eyJzdWIiOiI2NTJhM2E1ZS0yMTlkLTRhNWEtOTA4Zi1lNThhMWIzMDY2NTIiLCJjb2du...
    FlyI1wKlZD9ApcUHsCXiXn71ibMXMsKoYaJM1my40zhRcNtx9vL-8JUm-K79x6sin25ubB6QJo8cnQP-26
    SzpeZCVwhEKW4DAs4-ToUiKg9Y21rGY3MnrZ-S8BscYw8dvBQcH-PV6QtI8w'}

def attest():
    r = requests.post(url, json=payload, headers=headers)
    print(r.json())

if __name__ == '__main__':
    scheduler = BlockingScheduler()
    scheduler.add_job(attest, 'interval', seconds=30)
    print('Press Ctrl+{0} to exit'.format('Break' if os.name == 'nt' else 'C'))

    try:
        scheduler.start()
    except (KeyboardInterrupt, SystemExit):
        pass
    attest()
```

Using a scheduler, we can make the attestation `POST` request a job that runs daily at the same time. However, recall that we saw that `refreshToken` was not available in Local Storage from Section 4.1. We also discovered that a new token was generated whenever a browser login occurred and the user's prior access token had expired. Those tokens were expired an hour after issuance.

This implies that the process is not fully automated with access to the user's browser, but it does leave a window of time for adversaries to use grabbed tokens to authenticate requests.

4.4 None Algorithm Attack

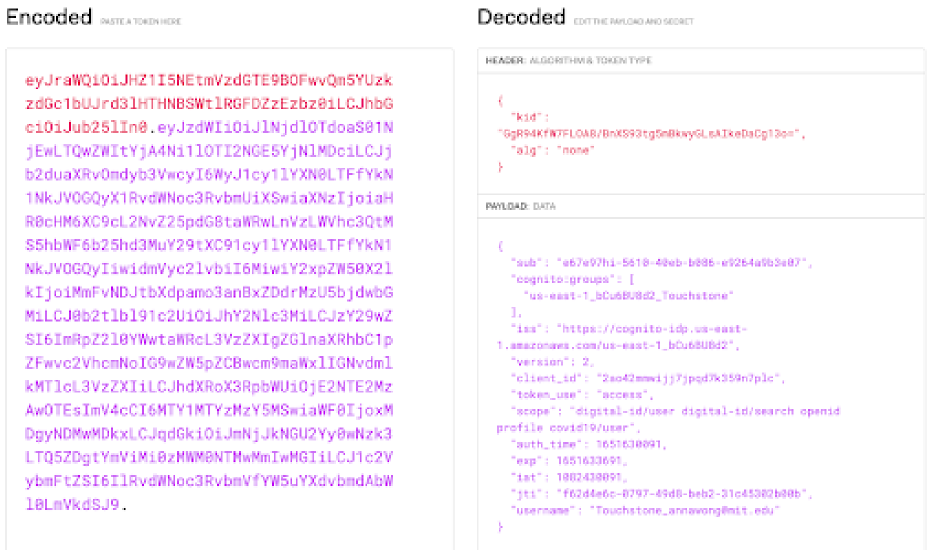
We attempted the none algorithm attack on COVID Pass. We took the token stored in local storage and modified the header to use the "none" algorithm, and removed the signature (see Figure 7). However, passing this in as the token did not work as the website did not accept the modified token. This suggests that the developers of COVID Pass took precautions to defend against this attack, perhaps by making sure the algorithm specified in the header is not "none".

4.5 `kid` Attack

We attempted to attack the COVID Pass site through a directory path traversal attack. To do this, we started a server on our local machine with the python `SimpleHTTPServer` command. Then, we generated new public and private RSA keys using the following command:

```
ssh-keygen -t rsa -b 4096 -m PEM -f jwtRS256.key
```

We modified the original ID Token JWT from a recent COVID Pass session. To modify the token, we replaced the `kid` field with the file path through the local server to the public key stored on the local machine. The modified header and `kid` value is shown in Figure 8. We then used the public and private keys generated to sign the encoded message according to the algorithm expected by the COVID Pass website (RS256).



Encoded PASTE A TOKEN HERE

```
eyJraWQiOiJHZ1I5NEtmVzdGTE9BOFwvQm5YUzk
zdGc1bUJrd3lHTHNBSWtlRGFDZzEzbz0iLCJhbG
ciOiJub25lIn0.eyJzdWIiOiJlNjdlOTdoaS01N
jEwLTQwZWItYjA4Ni1lOTI2NGE5YjNlMDciLCJj
b2duaXRvOmdyb3VwcyI6WyJ1cy1lYXN0LTFfYkN
1NkJVOGQyX1RvdWNoc3RvbmUiXSwiaXNzIjoiaH
R0cHM6XC9cL2NvZ25pdG8taWRwLnVzLWVhc3QtM
S5hbWF6b25hd3MuY29tXC91cy1lYXN0LTFfYkN1
NkJVOGQyIiwidmVyc2lvbiI6MiwiY2xpZW50X2l
kIjoiMmFvNDJtbXdpamo3anBxZDdrMzU5bjdwbG
MiLCJ0b2tlbl91c2UiOiJhY2Nlc3MiLCJzY29wZ
SI6ImRpZ2l0YWwtaWRcL3VzZXIgZGlnaXRhbC1p
ZFwvc2VhcmNoIG9wZW5pZCBwcm9maWxlIGNvdml
kMTlcL3VzZXIiLCJhdXRoX3RpbWUiOjE2NTE2Mz
AwOTEsImV4cCI6MTY1MTYzMzY5MSwiaWF0IjoxM
DgyNDMwMDkxLCJqdGkiOiJmNjJkNGU2Yy0wNzk3
LTQ5ZDgtYmViMi0zMWM0NTMwMmIwMGIiLCJ1c2V
ybmFtZSI6IlRvdWNoc3RvbmVfYW5uYXdvbmdAbW
l0LmVkdSJ9.
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "kid":
"GgR94KfW7FLOA8/BnXS93tg5mBkwyGLsAIkeDaCg13o=",
  "alg": "none"
}
```

PAYLOAD: DATA

```
{
  "sub": "e67e97hi-5610-40eb-b086-e9264a9b3e07",
  "cognito:groups": [
    "us-east-1_bCu6BU8d2_Touchstone"
  ],
  "iss": "https://cognito-idp.us-east-
1.amazonaws.com/us-east-1_bCu6BU8d2",
  "version": 2,
  "client_id": "2ao42mmwijj7jpqd7k359n7plc",
  "token_use": "access",
  "scope": "digital-id/user digital-id/search openid
profile covid19/user",
  "auth_time": 1651630091,
  "exp": 1651633691,
  "iat": 1082430091,
  "jti": "f62d4e6c-0797-49d8-beb2-31c45302b00b",
  "username": "Touchstone_annawong@mit.edu"
}
```

Figure 7: Modified Token with None Algorithm

```
{
  "kid":
"nt6aDocOxmt7rvOY8paYhqhMg1DfNI+2/+M6xvkPcWE=",
  "alg": "RS256"
}
```

```
{
  "kid": "http://10.10.10.10:8000/jwtRS256.key",
  "alg": "RS256"
}
```

Figure 8: The original header + `kid` (left) and the modified header + `kid` (right)

Unfortunately, after changing the private and public key values, the token was no longer accepted by the COVID Pass website and its subsequent API endpoints. We theorize that this could be due to the fact that the COVID Pass `kid` value in the JWT is not technically a file path, or the file path is further encrypted using a method that is not public to us. Another important thing to note is that the COVID Pass website is hosted by a CDN (Amazon CloudFront) which prevented us from accessing the true IP address of the site and, thus, attacking the `kid` by exploiting similar file paths.

4.6 Brute Force Attack

We attempted a brief brute force attack on the secret key/public key pair for the RS256 algorithm.

Earlier, we described a brute force attack in which weak secret keys compromise the security of HS256 signed JWT tokens. The type of brute force attack we described earlier is one in which a weak key such as ”covid-pass-22” might be used, therefore being vulnerable to dictionary attacks and other password-cracking methods. Covid Pass instead operates using RS256 signed tokens. Since this key pair is a RSA key pair, it is exceedingly difficult to recover the actual secret key by following the process of key generation, since it could vary so widely in the type of randomness used and the process used to generate the primes. This is in contrast to unlike in HS256 where a weak key might be hardcoded rather than generated.

In our brute force attempt to recover the secret key, we instead focus on recovering the secret key from the public key instead of trying to generate a correct secret key. We do so by factoring – getting the prime factors of the modulus, which if successful would lead to the secret key. We obtain the following JWK string formatted public key and convert it to PEM.

The public key in JWK string format:

```
{
  "e": "AQAB",
  "kty": "RSA",
  "n": "o7D0z3Px22Q7DfKDoM4fqn_5Xg8Us7Ic7lsglTBrl29xjz_0vWKqCMvahbagpW4_
  SuvZhf7BZXM-Ne4e9E4kjoxBpXXcybUwRa2v73WtCLQatrPUb-yGAFw1yV9z6XA49pHS8q
  9nhf2VuOZZaAdgzXXi3FkmFNJxogJQGEDu0WSXgqqfHICVkAruZgILvUuES_Wgl0rGoXD_
  BHNU07yVEyXDkIhoGez2SiBerghRCX7wqqT8eoRWScue28fpCLJEuFaj4WTcYMr__mlInU
  Fe_TbCfD5tELqlCiBjpUKZDNmi4zXPb6ZTFDLP8eTcJajlrR3JrjclvBkKGvPN908rUw"
}
```

The public key in PEM format:

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAo7D0z3Px22Q7DfKDoM4f
qn/5Xg8Us7Ic7lsglTBrl29xjz/0vWKqCMvahbagpW4/SuvZhf7BZXM+Ne4e9E4k
joxBpXXcybUwRa2v73WtCLQatrPUb+yGAFw1yV9z6XA49pHS8q9nhf2VuOZZaAdg
zXXi3FkmFNJxogJQGEDu0WSXgqqfHICVkAruZgILvUuES/Wgl0rGoXD/BHNU07yV
EyXDkIhoGez2SiBerghRCX7wqqT8eoRWScue28fpCLJEuFaj4WTcYMr//mlInUFe
/TbCfD5tELqlCiBjpUKZDNmi4zXPb6ZTFDLP8eTcJajlrR3JrjclvBkKGvPN908r
UwIDAQAB
-----END PUBLIC KEY-----
```

Using the pyjwt library, we can verify the correctness of the public key (it should verify the certificate of the signed JWT).

```
import jwt
encoded_jwt = "eyJraWQiOiJu..."
key = b"-----BEGIN PUBLIC KEY-----\nMIIBI..."
decoded = jwt.decode(encoded_jwt, key, algorithms=["RS256"], options={"verify_exp": False, "
    verify_aud":False})
```

We can then obtain the modulus using the following command:

```
$openssl rsa -in pubkey.pem -pubin -modulus -noout
Modulus=A3B0F4CF73F1DB643B0DF283A0CE1FAA7FF95E0F14B3B21CEE5B2095
306B976F718F3FF4BD62AA08CBDA85B6A0A56E3F4AEBD985FEC165733E35EE1E
F44E248E8C41A575DCC9B53045ADAFEF75AD08B41AB6B3D46FEC86005C35C95F
73E97038F691D2F2AF6785FD95B8E659680760CD75E2DC592614D271A2025018
40EED1649782AA9F1C8095900AEE66020BBD4B844BF5A0974AC6A170FF047354
D3BC951325C390886819ECF64A205EAE0851097EF0AAA4FC7A845649CB9EDBC7
E908B244B856A3E164DC60CAFFFE69489D415EFD36C27C3E6D10BAA50A2063A5
42990CD9A2E335CF6FA6531432CFF1E4DC25A8E5AD1DC9AE3725BC190A1AF3CD
F74F2B53
```

And then use the following script to brute force the factorization process:

```
from tqdm import tqdm

max_test = 10_000_000_000
m = int('A3B0F4CF73F...', 16)
for i in tqdm(range(3,max_test,2)):
    if m % i == 0:
        print(i)
        break
```

We tested possible prime factors up to 10 billion (10 digits) and found no prime factors of the modulus in the public key. This testing was carried out on a consumer-level laptop (16 inch Macbook Pro), with a poorly optimized script, and only for the duration of 2 hours. Real adversaries should be assumed to have much more compute at their disposal and also invest much more time. As such, although we cannot definitively conclude that their key generation process is not vulnerable to this type of attack, primes were probably chosen with enough care that this type of attack is not practical or feasible.

4.7 Suggestions to Secure COVID Pass

Currently, Covid Pass uses JWT tokens to authenticate the user actions on the Covid Pass website. We have noted previously that the storage of JWT tokens with important information (e.g. student ID) makes such information vulnerable to user side attacks (e.g. XSS). We have 2 suggestions that would work to provide better security.

1. Restrict the data within the JWT Token, optimally to just a single obscure user identifier (e.g. hash of student ID), and allow recipients of the JWT Token (e.g. API Endpoints) to query the Token Provider with additional information. The Token Provider can then restrict themselves to only responding to verified API endpoints, for example through the use of firewalls. Such a design reduces the information an attacker can get by just obtaining the JWT Token. The attacker would additionally have to successfully pose as the API Endpoint in order to obtain user-specific information.
2. Encrypt the data within the JWT Token. Since Covid Pass uses the JWT tokens for only a limited amount of recipients (notably for API Endpoint Access), public-key cryptography could be used to encrypt the contents of the JWT for those recipients (who are able to decrypt with their own private key). This would means that the browser would be able to access the API Endpoints as desired, while adversaries who are able to compromise the browser and obtain the JWT tokens would be presented with no additional important information about the user.

More care can also be placed in determining where the JWT tokens are stored. To increase the difficulty for attackers to access the JWT tokens, the tokens should be stored differently. Moving the tokens to Session Storage would make sure they are cleared when the browser session ends and are not accessible form other tabs. Moving the tokens to in-memory storage would be even better (e.g. inside a JS closure, where language semantics make XSS extremely difficult) [8].

API endpoints can also be modified to reduce the amount of damage an attacker can cause to the system, should they obtain an access token. For example, the report_symptoms endpoint might require more from the user in the post request (e.g. a new photo of the user) to make it harder for the attacker to generate a correct request.

5 Spectacle Investigation

After our COVID Pass investigation, we also decided to compare our findings with the usage of JWT in Spectacle, a project submission platform developed by members of the HackMIT team [9]. The header and payload of the access tokens we discovered on Spectacle are in Figure 10. The payload contains significantly less personally-identifying information, only including mandatory fields and an identity field. The identity field is the UUID associated with the particular user, but bears no relation to the actual user's name itself.

5.1 Client Storage Method

One of the design decisions that the authors of Spectacle made that protects their access tokens better is by placing them in HTTPS-protected cookies on the client-side. This means that the access token is only attached to requests when the connection is secured, mitigating the risk of spying adversaries. Furthermore, cookies are also not vulnerable to XSS as keeping tokens in Local Storage is.

We noted that cookies were not always wiped between sessions, however. That is, when a user logged out, the cookies persisted. If an adversary had direct access to a browser, even if the user had logged out, the

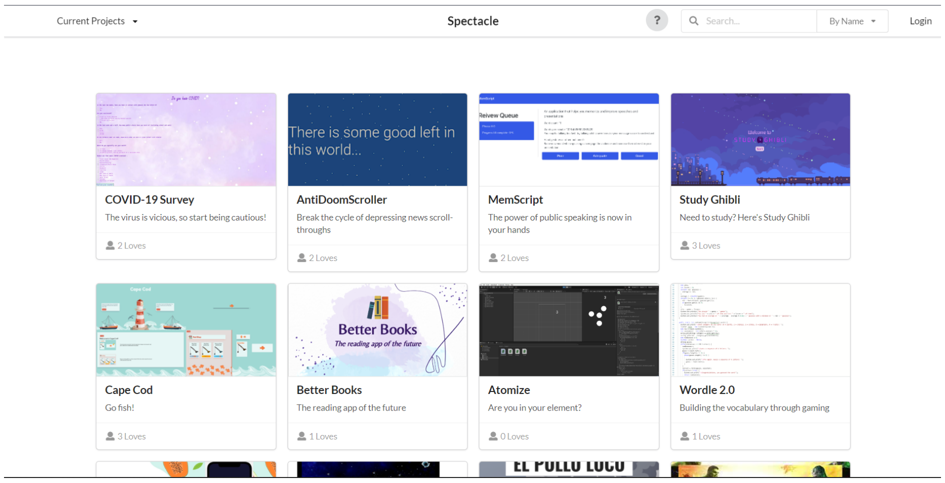

Current Projects
Spectacle
Search...
By Name
Login
There is some good left in this world...
COVID-19 Survey
The virus is vicious, so start being cautious!
2 Loves
AntiDoomScroller
Break the cycle of depressing news scroll-throughs
2 Loves
MemScript
The power of public speaking is now in your hands
2 Loves
Study Ghibli
Need to study? Here's Study Ghibli
3 Loves
Better Books
The reading app of the future
Cape Cod
Go fish!
3 Loves
Better Books
The reading app of the future
1 Loves
Atomize
Are you in your element?
0 Loves
Wordle 2.0
Building the vocabulary through gaming
1 Loves
EL POLLO LOCO

Figure 9: Spectacle Web Application [9]

adversary could pull the cookie and send it to themselves. Therefore, we believe that while storing access tokens in cookies is the right design decision, it can still be vulnerable to attacks.

5.2 API Endpoint Access

One of the major discoveries we made on Spectacle is that the server did not differentiate between each user's access tokens on several important endpoints. In particular, the following endpoint asks for information about teammates with `team_name`.

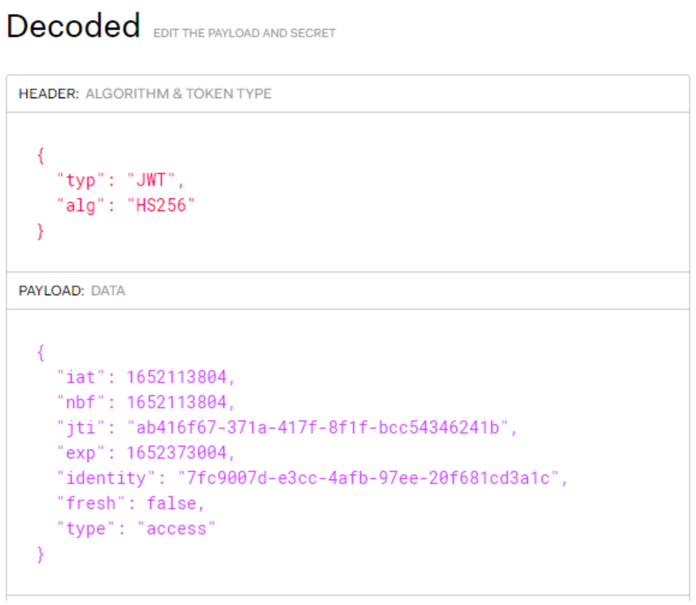
```
curl --location --request GET 'https://spectacle.hackmit.org/api/user/team/teammates?team_name=Hack
     Team' \
--header 'cookie: access_token_cookie=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
    eyJpYXQiOjE2NTIxNTQ3MTAsIm5iZiI6MTY1MjE1NDcxMCwianRpIjoiYWMzM2MwMGQtMTI4Ni00YzAyLWE5YzQtMDg2OD
    ...XNoIjpmYWxzZSwidHlwZSI6ImFjY2VzcyJ9.NRRrldr3COYIKg62OLdZ5jlQzqB1UWXOAKIRzuhczYO'
```

The response body of this endpoint returns far more information than is publicly available on the website itself. This implies that the client only uses a portion of the response it receives and "discards" the other information it receives. Because the information returned only depends on the `team_name` parameter, a user can access the personal information about anybody on any team. Information revealed includes users' emails, the projects they liked, and their notifications. This is information that every user needs for themselves, but they should not be able to access other users' information with their own access token.

We found that admin endpoints were protected, so regular users all had the same privileges as each other. The problem was that their privileges were not limited and specific to the user itself. The server merely checked for the presence of a valid access token rather than verifying the identity of the user itself.

5.3 Expiration and Refreshing

Although the access tokens on Spectacle reveal less information about the user, they also fail to expire in a timely manner. As one can calculate from Figure 10, each access token expires after 3 full days. Most HackMIT events last for less than 3 days. This means that if an adversary were to obtain an access token at the beginning of the event, they could use it for the duration of the entire event. Paired with the previous vulnerability, they could obtain information about all of the participants of the event.



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

```
{
  "iat": 1652113804,
  "nbf": 1652113804,
  "jti": "ab416f67-371a-417f-8f1f-bcc54346241b",
  "exp": 1652373004,
  "identity": "7fc9007d-e3cc-4afb-97ee-20f681cd3a1c",
  "fresh": false,
  "type": "access"
}
```

Figure 10: Spectacle Access Token Sample, HackMIT member

6 JWT Best Practices

To protect against attacks mentioned in previous sections, the following actions, recommended by IETF, should be taken to mitigate the ability of the attacker to compromise the security of JWTs[10]:

- None Algorithm Attacks can be mitigated through careful consideration of appropriate signing algorithms. Algorithms need to still secure at the time of use to protect JWTs from attackers. Although the "none" algorithm is fine in certain contexts (e.g. when protected by TLS), they should only be allowed after careful consideration

- Brute force attacks can be mitigated by ensuring that cryptographic keys are sufficiently random and should not be a human-memorizable password.

- KID/JKU attacks can be mitigated by treating received claims within the JWT as untrustworthy. Specifically, the "kid" field should be santized to prevent SQL or other database injection attacks. Similarly, the "jku" field should be checked and treated with suspicion (e.g. check against an up-to-date whitelist, and making sure as little information as possible is transmitted through the GET request for the key).

- Token capture attacks can be mitigated in scope through the use of optional fields such as the Issuer, Subject, and Audience fields ("iss", "sub", "aud" respectively).

  The issuer claim allows the application to validate that the keys used for the operation belong to the issuer (so that attacks that just sign the JWT with another key do not work).

  The subject claim allows the application to validate that the subject is valid before proceeding to validate the token, making sure that the correct party is using the token.

  The audience claim allows the JWT to be targeted towards a specific final recipient. For example, in the Covid Pass Investigation, splitting up tokens to be used for different API endpoints may allow for better security against the accidental leakage of singular tokens.

IETF recommends other best practices to be carried out to ensure the cryptographic security of JWT. For example, they recommend not compression data before encryption to avoid leaking plaintext information. The full report by the IETF for JWT best practices can be found here.

7 Conclusion

While we were not necessarily able to utilize the attacks we theorized about to exploit COVID Pass's usage of JWTs, we found that their JWT storage technique was likely not entirely secure. Much of the security of a JWT depends on the security of the signature algorithm itself. However, because JWTs are typically not encrypted and easily decoded, we were able to gain information from both the COVID Pass site and Spectacle that could be used maliciously by an attacker. The structure of a JWT and typical JWT use patterns leave them vulnerable to certain attacks such as the None algorithm attack, `kid` attacks, brute force attacks, and even attacks upon the signature algorithm. We found that COVID Pass had much stronger security measures than we originally anticipated, likely because their JWTs are produced and handled by a service provided by Amazon, which promises extended security against extended attacks on JWTs. In our investigation into HackMIT's platform, Spectacle, we found that the application was using inadequate security measures to protect information users were not supposed to access. Because they did not actually determine if the user was allowed to access some endpoints and instead only checked that the signature was valid, they unintentionally exposed a lot of private information that any single user could access.

We believe all web applications should be taking extra precautions when it comes to the use of JWTs in their service. Third party tools such as Amazon Cognito help applications achieve extra security without having to dedicate extraordinary time and effort into building the layers of web security themselves.

We would like to extend a special thanks to the 6.857 Spring 2022 course staff for giving us the background and the time we needed to successfully complete our project. We would also like to thank the MIT COVID Pass and HackMIT teams for permitting us to investigate their applications.

References

[1] “JWT IETF.” https://datatracker.ietf.org/doc/html/rfc7519. Accessed 2022-05-09.

[2] “JWT.” https://jwt.io/. Accessed 2022-05-09.

[3] “Security of JSON Web Tokens (JWT).” https://cyberpolygon.com/materials/security-of-json-web-tokens-jwt/. Accessed 2022-05-09.

[4] “Critical vulnerabilities in JSON Web Token libraries.” https://auth0.com/blog/critical-vulnerabilities-in-json-web-token-libraries/. Accessed 2022-05-09.

[5] “Hacking JSON Web Tokens (JWTs).” https://medium.com/swlh/hacking-json-web-tokens-jwts-9122efe91e4a. Accessed 2022-05-09.

[6] “COVID Pass.” https://covidpass.mit.edu/#. Accessed 2022-05-09.

[7] “Amazon Cognito Documentation.” https://docs.aws.amazon.com/cognito/index.html. Accessed 2022-05-09.

[8] “Secure Browser Storage: The Facts.” https://auth0.com/blog/secure-browser-storage-the-facts/. Accessed 2022-05-09.

[9] “Spectacle.” https://spectacle.hackmit.org/. Accessed 2022-05-09.

[10] “RFC8725 - JSON Web Token Best Current Practices.” https://datatracker.ietf.org/doc/rfc8725/. Accessed 2022-05-09.