



HACKTHEBOX



WayWitch

20th October 2024 / Document No. D24.102.212

Prepared By: `Xc1ow3n`

Challenge Author: `Xc1ow3n`

Difficulty: **Very Easy**

Classification: Official

Synopsis

- WayWitch is an easy web challenge where players exploit a client-side JWT signing.

Skills Required

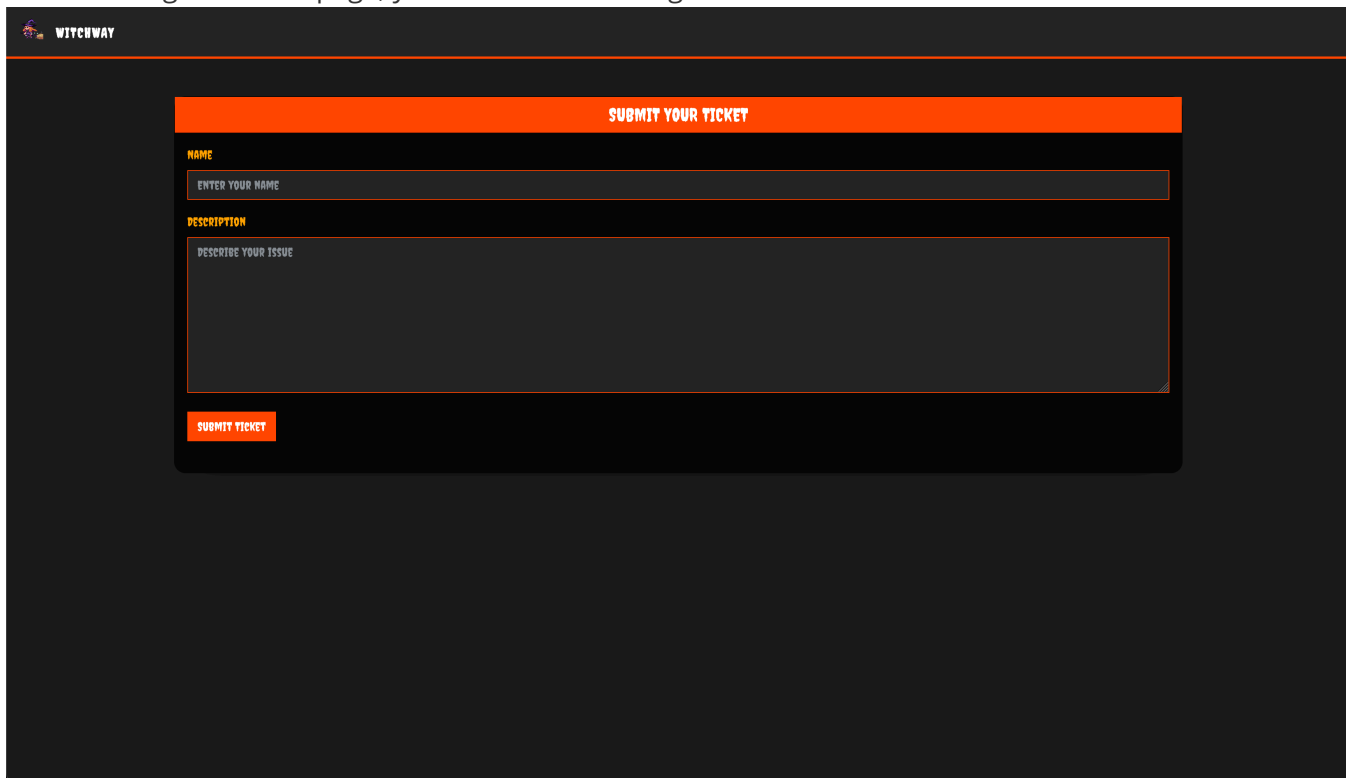
- Basic knowledge of JavaScript

Skills Learned

- Understanding how sensitive tokens like signing keys can be exposed client-side

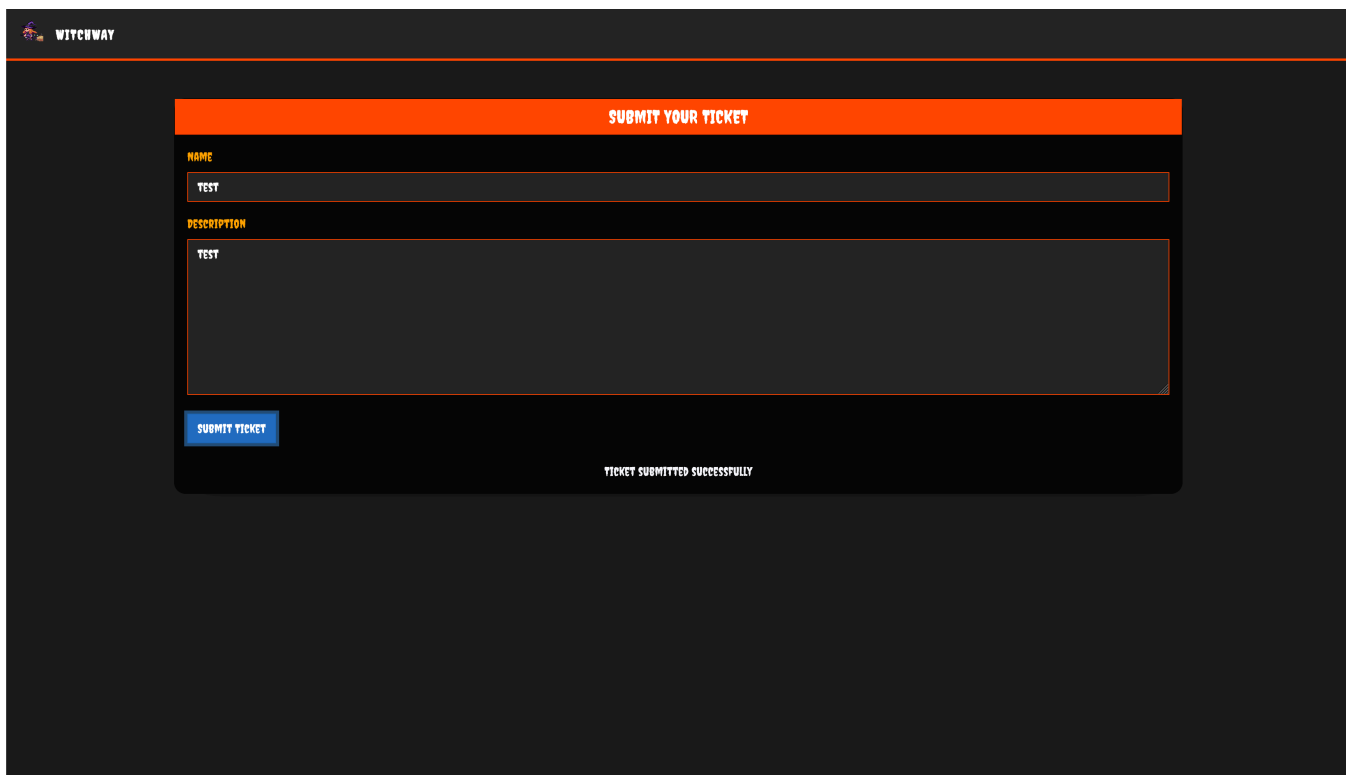
Solution

When visiting the home page, you'll see the following screen:



The screenshot shows a dark-themed web interface for 'WITCHWAY'. At the top left is a logo with a witch hat and the text 'WITCHWAY'. Below it is a large orange header bar with the text 'SUBMIT YOUR TICKET' in white. Underneath the header is a form with two sections: 'NAME' and 'DESCRIPTION'. The 'NAME' section has a text input field with the placeholder 'ENTER YOUR NAME'. The 'DESCRIPTION' section has a large text area with the placeholder 'DESCRIBE YOUR ISSUE'. At the bottom of the form is an orange button labeled 'SUBMIT TICKET'.

You can submit tickets here:



This screenshot shows the same 'SUBMIT YOUR TICKET' form as the previous one, but with test data entered. The 'NAME' field contains 'TEST' and the 'DESCRIPTION' field contains 'TEST'. The orange 'SUBMIT TICKET' button has been replaced by a blue button with the same text. Below the form, the text 'TICKET SUBMITTED SUCCESSFULLY' is displayed in white. The 'WITCHWAY' logo and orange header bar remain at the top.

That's most of the functionality. However, if we inspect the source code, we find the `generateJWT` function:

```

async function generateJWT() {
  const existingToken = getCookie("session_token");

  if (existingToken) {
    console.log("Session token already exists:", existingToken);
    return;
  }

  const randomNumber = Math.floor(Math.random() * 10000);
  const guestUsername = "guest_" + randomNumber;

  const header = {
    alg: "HS256",
    typ: "JWT",
  };

  const payload = {
    username: guestUsername,
    iat: Math.floor(Date.now() / 1000),
  };

  const secretKey = await crypto.subtle.importKey(
    "raw",
    new TextEncoder().encode("halloween-secret"),
    { name: "HMAC", hash: "SHA-256" },
    false,
    ["sign"],
  );

  const headerBase64 = btoa(JSON.stringify(header))
    .replace(/\+/g, "-")
    .replace(/\//g, "_")
    .replace(/>=+$/g, "");
  const payloadBase64 = btoa(JSON.stringify(payload))
    .replace(/\+/g, "-")
    .replace(/\//g, "_")
    .replace(/>=+$/g, "");

  const dataToSign = `${headerBase64}.${payloadBase64}`;
  const signatureArrayBuffer = await crypto.subtle.sign(
    { name: "HMAC" },
    secretKey,
    new TextEncoder().encode(dataToSign),
  );

  const signatureBase64 = btoa(
    String.fromCharCode.apply(
      null,
      new Uint8Array(signatureArrayBuffer),
    ),
  )
    .replace(/\+/g, "-")

```

```

        .replace(/\\/g, "\\")
        .replace(/=+$/, "");

    const token = `${dataToSign}.${signatureBase64}`;

    document.cookie = `session_token=${token}; path=/; max-age=${60 * 60 * 24}; Secure`;

    console.log("Generated JWT Session Token:", token);
}

```

This function creates a JWT with a random username and uses `halloween-secret` as the signing key.

Next, in `src/routes/index.js`, we find the following route:

```

router.get("/tickets", async (req, res) => {
    const sessionToken = req.cookies.session_token;

    if (!sessionToken) {
        return res.status(401).json(response("No session token provided"));
    }

    try {
        const username = getUsernameFromToken(sessionToken);


        if (username === "admin") {
            try {
                const tickets = await db.get_tickets();
                return res.status(200).json({ tickets });
            } catch (err) {
                return res
                    .status(500)
                    .json(response("Error fetching tickets: " + err.message));
            }
        } else {
            return res
                .status(403)
                .json(response("Access denied. Admin privileges required."));
        }
    } catch (err) {
        return res.status(400).json(response(err.message));
    }
});

```


It extracts the username from the token and checks if it is `admin`. If it is, you can view the tickets, which include the flag. Since we know the secret key (`halloween-secret`), we can sign our own token as `admin` to get the flag.

For this, we can use jwt.io to sign the token.

Missed DevDay24? Register for the Best of DevDay →

JWT

DebuggerLibrariesIntroductionAsk

Crafted by  Auth0 by Okta

Algorithm

HS256

Encoded

PASTE A TOKEN HERE

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImFkbWluIiwiaWF0IjoxNzY2fQ.X4pLGMCH1e0FKwqjjypzRED5UcJefBsSV950-27MCys

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

{

"alg": "HS256",

"typ": "JWT"

}

PAYLOAD: DATA

{

"username": "admin",

"iat": 1729661766

}

VERIFY SIGNATURE

HMACSHA256(

base64UrlEncode(header) + "." +

base64UrlEncode(payload),

halloween-secret

)

☐ secret base64 encoded

Signature Verified

SHARE JWT

Once we use the modified token as our session cookie, we can access the tickets and retrieve the flag:

JSONRaw DataHeaders

SaveCopyCollapse AllExpand AllFilter JSON

tickets:

0:

id:

1

name:

"John Doe"

username:

"guest_1234"

content:

"I need help with my account."

1:

id:

2

name:

"Jane Smith"

username:

"guest_5678"

content:

"There is an issue with my subscription."

2:

id:

3

name:

"Admin"

username:

"admin"

content:

"Top secret: The Halloween party is at the haunted mansion this year. Use this code to enter HTB([REDACTED])"

3:

id:

4

name:

"Paul Blake"

username:

"guest_9812"

content:

"Can someone assist with resetting my password?"

4:

id:

5

name:

"Alice Cooper"

username:

"guest_3456"

content:

"The app crashes every time I try to upload a picture."

5:

id:

6

name:

"test"

username:

"guest_7355"

content:

"test"

This concludes the challenge :)