Example Premium Report

Security Assessment Report

PREPARED BY
ABC

REPORTING PERIOD
Q1 · 1 Jan – 31 Mar

Contents

Executive Summary

SECURITY SCORE

0

/ 100

Wet Paper Bag

RISK LEVEL

High Risk

Poor security posture. Critical vulnerabilities present.

TOTAL ISSUES

41

11C 4H 21M 5L

CATASTROPHIC SECURITY BREACH DETECTED

IMMEDIATE ACTION REQUIRED - PRODUCTION ENVIRONMENT COMPROMISED

• /.env exposed with 2 secrets (1 Possible AWS Access Key ID, 1 Possible Generic Secret Key) - immediate production takeover risk

This exposure enables complete infrastructure takeover. Attackers can access databases, cloud accounts, payment systems, and customer data. Revenue loss, regulatory fines (GDPR/CCPA), customer lawsuits, and permanent reputation damage are imminent.

Findings Breakdown

CRITICAL	HIGH	MEDIUM	LOW
11	4	21	5

This example premium report identified a total of 41 potential issues across multiple categories. The most critical areas requiring attention are highlighted in the sections below.

Findings Overview

Exposed Configuration Files (WITH SECRETS)	1	CRITICAL
Insecure Cookies	2	HIGH
Storage Security Issues	2	HIGH
Known Vulnerabilities (2 technologies)	5	MEDIUM

Detailed Findings & Remediation

Exposed Configuration Files

Business Impact

This security finding represents a risk to business operations and data security.

Financial Risk:

Potential for unauthorized access or data exposure.

A security gap that reduces overall defense posture.

What This Means

Configuration files like .env, .git/config, and cloud credentials are publicly accessible on your web server. These files often contain database passwords, API keys, and other sensitive credentials.

Attackers routinely scan for these files using automated tools. Once found, credentials can be used to access databases, cloud accounts, and internal systems within minutes.

.env files are particularly dangerous as they often contain complete application configuration including all API keys, database credentials, and secret tokens.

Potential Risks

- Complete infrastructure takeover
- Database access and data theft
- Cloud account compromise
- Ransomware deployment
- Cryptocurrency mining on your infrastructure
- Total business compromise

Identified Instances (1)

1. /.env

Type: Environment File | Severity: CRITICAL

CATASTROPHIC EXPOSURE

This file contains 2 exposed secrets: 1 Possible AWS Access Key ID, 1 Possible Generic Secret Key

IMMEDIATE ACTION REQUIRED: This exposure enables complete production infrastructure takeover. Attackers can access databases, cloud accounts, payment systems, and customer data within minutes.

Impact: IMMEDIATE BUSINESS RISK: This file typically contains database passwords, API keys, payment gateway credentials (Stripe/PayPal), and authentication secrets. Attackers gain complete access to your infrastructure, customer data, payment systems, and third-party services. Expected impact: complete data breach, unauthorized charges, regulatory fines (GDPR/CCPA), customer lawsuits, and permanent reputation damage. Average breach cost: $4.45M (IBM 2023).

Remediation: Remove .env files from web root. Use server-side environment variables. Add .env to .gitignore. Block access via .htaccess or nginx config.

How to Fix:

Immediate Actions:

1. Block access to sensitive files immediately
2. Rotate ALL credentials found in exposed files
3. Check access logs for previous downloads
4. Audit systems for unauthorized access

Blocking Files:

Nginx:

```
location ~ /\. {
deny all;
}
```

Apache (.htaccess):

```
<FilesMatch "^\.(env|git|htaccess|htpasswd)">
Order allow,deny
Deny from all
</FilesMatch>
```

Prevention:

1. Never store .env files in web root
2. Add sensitive files to .gitignore
3. Use environment variables on server
4. Regular security scans

Code Examples:

```
// Nginx - Block all dot files:
location ~ /\. {
  deny all;
  return 404;
}

// Apache - Block sensitive files:
<FilesMatch "\.(env|config|sql|bak|log)$">
  Order allow,deny
  Deny from all
</FilesMatch>

// .gitignore - Prevent accidental commits:
.env
.env.local
.env.production
*.pem
*.key
```

Insecure Cookies

Business Impact

This security finding represents a risk to business operations and data security.

Financial Risk:

Potential for unauthorized access or data exposure.

A security gap that reduces overall defense posture.

What This Means

Cookies containing sensitive data (session tokens, authentication credentials) are missing important security flags that protect them from theft and manipulation.

The HttpOnly flag prevents JavaScript from accessing the cookie, protecting against XSS attacks. The Secure flag ensures cookies are only sent over HTTPS. The SameSite flag helps prevent CSRF attacks.

Missing these flags allows attackers to steal session cookies via XSS attacks, intercept them over unencrypted connections, or use them in cross-site request forgery attacks.

Potential Risks

- Session hijacking via XSS attacks
- Cookie theft through network interception
- Cross-site request forgery (CSRF)
- Account takeover and impersonation
- Data theft and unauthorized access
- Compliance violations (PCI-DSS, GDPR)

Identified Instances (2)

1. Possible not-httponly

2. Possible not-httponly

How to Fix:

Essential Cookie Security Flags:

1. HttpOnly: Prevents JavaScript access

```
Set-Cookie: session=abc123; HttpOnly
```

2. Secure: Only send over HTTPS

```
Set-Cookie: session=abc123; Secure
```

3. SameSite: Prevent CSRF attacks

```
Set-Cookie: session=abc123; SameSite=Strict
```

Complete Example:

```
Set-Cookie: session=abc123; HttpOnly; Secure; SameSite=Strict; Path=/; Max-Age=3600
```

Implementation:

1. Audit all cookies set by your application
2. Add HttpOnly to ALL session/auth cookies
3. Add Secure flag (requires HTTPS)
4. Add SameSite=Strict or Lax
5. Set appropriate expiration

Code Examples:

```
// Express.js example:
res.cookie("session", token, {
  httpOnly: true,
  secure: process.env.NODE_ENV === "production",
  sameSite: "strict",
  maxAge: 3600000 // 1 hour
});

// PHP example:
setcookie("session", $token, [
  "expires" => time() + 3600,
  "path" => "/",
  "secure" => true,
  "httponly" => true,
  "samesite" => "Strict"
]);

// Nginx header:
add_header Set-Cookie "session=$token; HttpOnly; Secure; SameSite=Strict";
```

Storage Security Issues

Business Impact

This security finding represents a risk to business operations and data security.

Financial Risk:

Potential for unauthorized access or data exposure.

A security gap that reduces overall defense posture.

What This Means

Sensitive data stored in browser localStorage or sessionStorage is vulnerable to XSS attacks. Unlike cookies with HttpOnly flag, JavaScript storage is always accessible to scripts.

localStorage persists indefinitely and survives browser restarts, while sessionStorage lasts until the tab is closed. Both are vulnerable if any XSS vulnerability exists on the page.

Storing authentication tokens, API keys, personal data, or other sensitive information in browser storage creates significant security risks.

Potential Risks

- Token theft via XSS attacks
- Persistent data exposure
- Session hijacking
- Personal data leakage
- API key exposure
- Privacy violations

Identified Instances (2)

1. Sensitive LocalStorage Item (JWT Token)

localStorage: user_token

```
localStorage.getItem('user_token'):
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkFjbWUgVXNlciIsImlhdCI6MTU...
```

2. Sensitive SessionStorage Item (Sensitive Data)

sessionStorage: api_key

```
sessionStorage.getItem('api_key'):
some_session_key_123
```

How to Fix:

Best Practices:

1. **Don't store sensitive data: Never put tokens, passwords, or PII in localStorage**
2. **Use httpOnly cookies: For authentication tokens when possible**
3. **Short-lived tokens: If you must use storage, use short expiration**
4. **Encrypt data: Encrypt before storing (adds complexity)**
5. **Implement CSP: Strong Content Security Policy prevents XSS**

Migration Steps:

1. Identify sensitive data in storage
2. Move authentication to httpOnly cookies
3. Clear existing storage on login
4. Update token refresh logic

Code Examples:

```
// BAD - Token in localStorage:
localStorage.setItem("authToken", token);
const token = localStorage.getItem("authToken");

// GOOD - Use httpOnly cookie instead:
// Server sets cookie on login:
res.cookie("authToken", token, {
  httpOnly: true,
  secure: true,
  sameSite: "strict"
});

// Client makes authenticated requests:
fetch("/api/data", {
  credentials: "include" // Sends cookies automatically
});

// If storage is unavoidable, use sessionStorage:
sessionStorage.setItem("tempData", data); // Cleared on tab close
```

Known Vulnerabilities (CVEs)

The following Common Vulnerabilities and Exposures (CVEs) were identified in the software components used by this application.

jQuery 1.8.3

CVE IDENTIFIER	SEVERITY	CVSS	DESCRIPTION
CVE-2020-11022	MEDIUM	6.9	In jQuery versions greater than or equal to 1.2 and before 3.5.0, passing HTML from untrusted sources - even after sanitizing it - to one of jQuery's ...
CVE-2020-11023	MEDIUM	6.9	In jQuery versions greater than or equal to 1.0.3 and before 3.5.0, passing HTML containing <option> elements from untrusted sources - even after sani...

WordPress 5.8.2

CVE IDENTIFIER	SEVERITY	CVSS	DESCRIPTION
CVE-2022-43497	MEDIUM	6.1	Cross-site scripting vulnerability in WordPress versions prior to 6.0.3 allows a remote unauthenticated attacker to inject an arbitrary script. The de...
CVE-2022-43500	MEDIUM	6.1	Cross-site scripting vulnerability in WordPress versions prior to 6.0.3 allows a remote unauthenticated attacker to inject an arbitrary script. The de...
CVE-2022-43504	MEDIUM	5.3	Improper authentication vulnerability in WordPress versions prior to 6.0.3 allows a remote unauthenticated attacker to obtain the email address of the...

Appendix

Technologies Detected

- Apache HTTP Server (v2.4.52)
- cdnjs
- core-js (v3.19.0)
- Google Tag Manager
- jQuery (v1.8.3)
- jQuery CDN
- Ubuntu
- WordPress (v5.8.2)
- PHP
- MySQL

Scan Information

Scanner Version	1.5.12
Scan Date	1/30/2026, 11:48:17 AM
Scan Duration	5.91s
Target URL	http://virtualfishingcoach.com/

Disclaimer