Other Web Application Vulnerabilities

Connor Corcoran

Lab Setup

Environment: Azure VM running Windows Server with IIS hosting + PfSense running as the network firewall

Files Deployed: addalbum.aspx, listalbums.aspx,upload.apsx,login.aspx,firectiorylist.aspx, plus helper files (getcookies.htm, logit.aspx,hair.svg).

Permissions & config: wwwroot/uploads writable by IIS_IURS; web.config modified to allow HTML input

Web Application Threats

Cross Site Scripting

What: Occurs when an application displays user input on a page allowing injected Javascript or HTML to run in a users browser
Impact: Can lead to theft of session cookies, user impersonation, or unauthorized actions
Fix: Encode output
Detection: Review logs for missing or invalid CSRF tokens and unexpected state changes

Cross Site Request Forgery

What: Tricks and authenticated users browsers into sending unintended requests to a web application
Impact: May cause unauthorized changes such as role updates or data modifications
Fix: implement anti-CSRF tokens, use POST for sensitive actions, and set cookies with SameSite protection
Detection: Review logs for missing or invalid CSRF tokens and unexpected state changes.

Session Hijacking

What: attackers obtain a valid session token and refuses it to impersonate a user
Impact: Account takeover - access/modify sensitive data, perform actions.
Fix: Use HttpOnly Secure, SameSite cookies; enforce HTTPS/HSTS; regenerate session id on login/privilege change; short timeouts
Detection: monitor same session ID from multiple IPs/UA; check cookies in browser DevTools

Unrestricted File Upload

What: App accepts and serves uploaded files without validating type/contents or blocking execution.
Impact: Upload webshells or HTML/JS pages -> remote code execution, data exposure, persistent backdoors.
Fix: Store uploads outside webroot or block script handlers in upload folder.
Detection: Alert on executable/disallowed uploads and access to uploaded files

Cross-Site Scripting (XSS): Attacker Impact

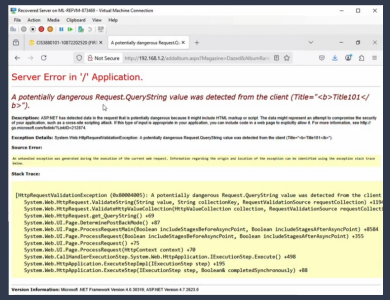
Steal or reuse session cookies (if not HttpOnly), carry out admin actions.

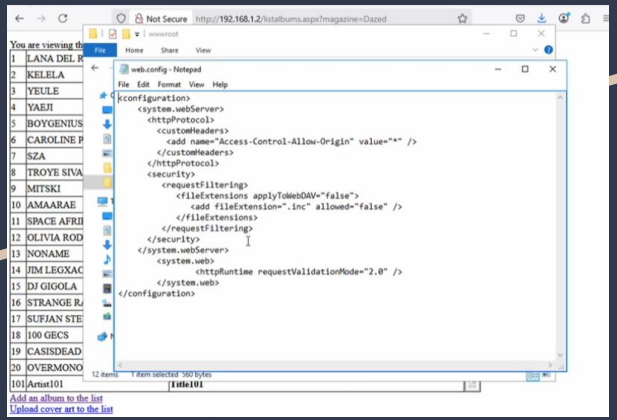
Deface site or insert malicious iframes/phishing pages.

Run actions in context of user.

Data exfiltration from pages the user can access.

XSS: where it arises in your code





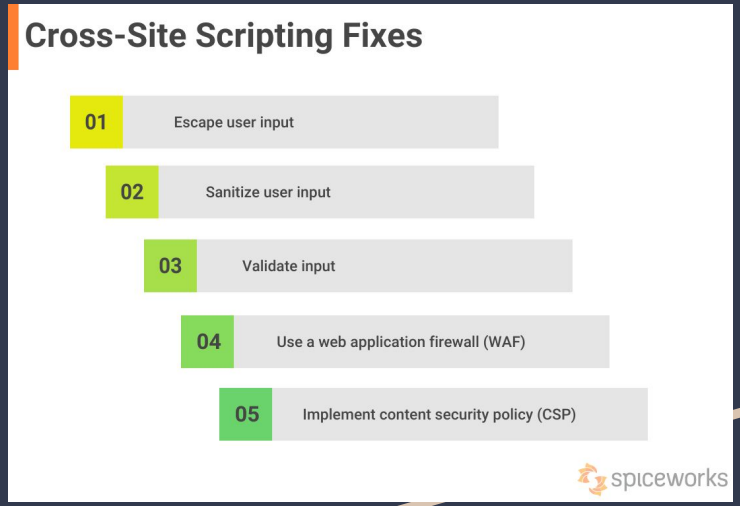
Where it arises

- Origin: add album.aspx (user enters album title/artist/description) → saved to database or storage.
- Enabling config: web.config set to allow HTML (e.g. pages validateRequest= “false” or requestValidationMode= “2.0”) which lets <script>/<iframe> to pass through

Why the code is vulnerable

- The app renders untrusted input directly into HTML
- The web.config change removed an input level filter so malicious markup can be saved

XSS: Secure Fixes



Implement a multi-layered approach using input validation (sanitizing and filtering user input) and output encoding (correctly escaping data before displaying it)

Use Content Security Policy headers to instruct browser on what resources are safe to load.

Set the HttpOnly flag for cookies

Unrestricted File Upload: Attacker Impact

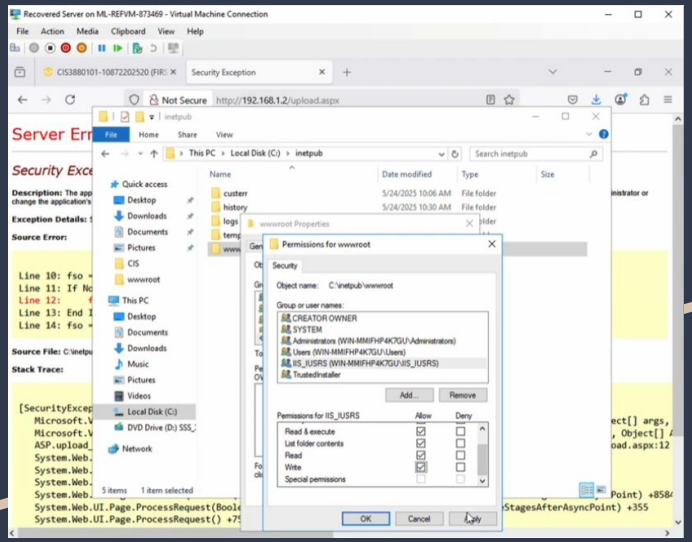
Upload and run a webshell/backdoor on the server.

Read or steal sensitive files (configs, source, DB creds).

Maintain persistent access (hidden/backdoor files).

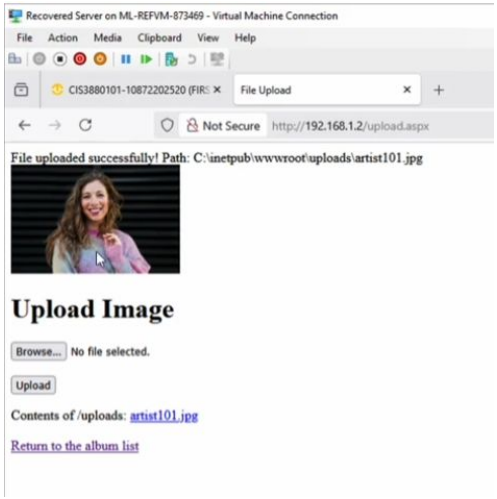
Deploy malware or disrupt service (ransomware/defacement).

Unrestricted File Upload: Where it Arises



Where it arises:

- Upload.aspx – file input saved to ~/uploads under wwwroot using use supplied filename (e.g. SaveAS(Server.MapPath("~/uploads/" + fileName)))
- Also granted IIS_IUSRS write permission to uploads folder so uploads succeed



Unrestricted File Upload: Fixes

1. Whitelist file extension and check content by magic bytes
2. Rename files to same names and store outside wwwroot:
3. Block execution in /uploads if inside web root by policing a web.config that denies .aspx
4. Tighten ACLs: minimal write permission for app account only; deny execute where possible.
5. Scan uploads with AV/malware scanner and limit file sizes

Cross-Site Request Forgery (CSRF): Attacker Impact

Attackers crafts a page that triggers the admins browser to call a “promote” URL while admin is authenticated → silent privilege escalation

Consequence: attacker can make an admin change other employees to admin, pivot, steal PII, etc.

CSRF: where it arises

Where it arises:

- Endpoint: login.aspx or whatever page performs the promote action.
- The promote/promotion logic executes from a GET request rather than requiring a POST and token

Why the code is vulnerable:

- State-changing actions are allowed via GET and without any proof that the request originated from the app user interface
- Browser automatically attaches cookies to cross-origin resource loads, so the server cannot tell the request was forged

CSRF Fixes

Require POST for state changes.

Add anti-CSRF token: The server generates a unique unpredictable token for each user session.

Use SameSite cookies: Controls when cookies are sent with cross-site requests.

Session Hijacking: Attacker Impact

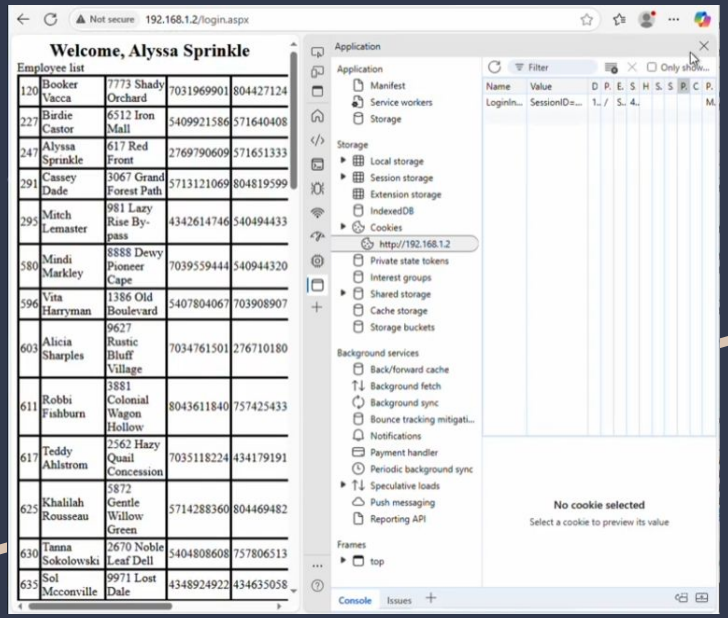
Immediately impersonate the victim (account takeover).

Access privileged functions if the victim is an admin.

Exfiltrate confidential data accessible to the user.

Perform fraudulent actions that appear legitimate.

Session Hijacking: where it arises



- Auth cookie creation (in login.aspx / auth code) — if cookie is created without HttpOnly/Secure flags.
- Upload vector (upload.aspx → wwwroot\uploads) — attacker can host JS (getcookies.htm / hair.svg) that runs in victim's browser.
- XSS/vector rendering (addalbum.aspx / listalbums.aspx) — stored XSS also allows JS to run in admin's browser.

Session Hijacking: Fixes

1. Set cookie flags when creating authentication cookie.
2. Regenerate session ID on login and on privilege changes
3. Enforce TLS/HSTS across site; set short timeout for session; re-authenticate for critical actions.
4. Removes JS-executable files from uploads and fix XSS vectors to prevent theft

Key Findings

Key Findings:

- Multiple critical web vulnerabilities identified: XSS, CSRF, Session Hijacking, Unrestricted File Upload.
- Exploits demonstrated real-world impact: account takeover, cookie theft, privilege escalation, and potential code execution.
- Root causes include: weak session management, lack of input/output validation, permissive file uploads, and missing anti-CSRF protections.
- Business & operational risks: data breaches, regulatory violations, loss of user trust, and potential financial losses.

Takeaways

Takeaways

- Always validate and encode all user input/output.
- Enforce strong session management (HttpOnly, Secure, SameSite cookies).
- Restrict and sanitize file uploads, and disable execution in upload directories.
- Implement anti-CSRF tokens and convert sensitive GET requests to POST.
- Monitor and alert on suspicious activity (unexpected uploads, sudden privilege changes, or external requests).
- Test in isolated environments before deploying changes to production.