## #1 SQL Injection in ConverterResponse.aspx (btcAmount JSON parameter)

**Category:** Injection / Input Validation

**Severity:** Critical

**CVSS:** 9.8 (approx)

CWE-89: SQL Injection

**CURRENT STATUS:** Active

## AFFECTED ENDPOINT(S) :

http://aspnet.testsparker.com/ConverterResponse.aspx

## DESCRIPTION -

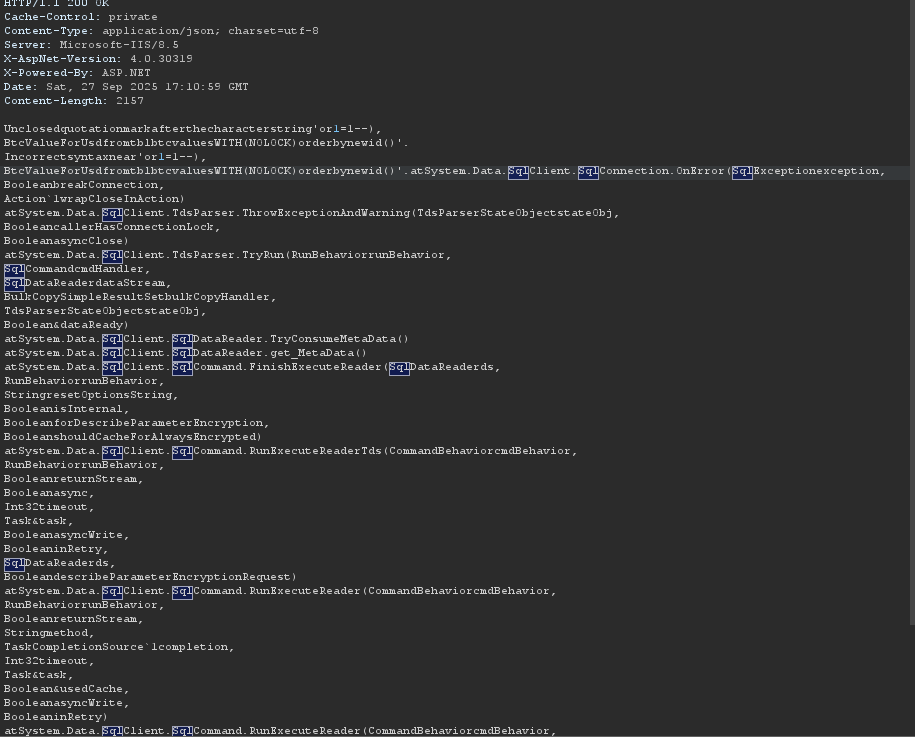
SQL injection occurs when a malicious user can interfere with an application's communication with its underlying database and can directly communicate with the database through the application using the implicit trust of the application on user input. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given le present on the DBMS le system and in some cases issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to affect the execution of predefined SQL commands.

In this application, it is identified that a SQL injection in the btcAmount parameter, a crafted payload (' OR 1=1--) sent in a POST request triggered a database error and server stack trace, confirming that unsensitized input is concatenated into SQL on the server side. This vulnerability allows arbitrary SQL execution — enabling attackers to read, modify, or delete database contents, bypass authentication or takeover accounts if credentials are stored, exfiltrate sensitive PII and secrets, and create conditions for privilege escalation or chained server compromise — with clear business impacts including data breach, service disruption, and reputational or legal exposure.

## STEPS TO REPRODUCE

1. Intercept a normal Converter request in Burp (or test platform).  
2. Send the following POST to ConverterResponse.aspx with malicious payload in JSON body.  
Example vulnerable payload (error/boolean-based):  
{"btcAmount":"' OR 1=1--"}  
3. Observe server response containing SQL error and stack trace (evidence shows SQL syntax error and application file path). The code file and line shown (converterResponse-AjaxJsonSqlInjection.ascx.cs:line59) point to the exact vulnerable handler.  
  
Blind/time-based confirmation (MSSQL):  
Payload:  
{"btcAmount":"1'; WAITFOR DELAY '00:00:05'--"}  
  
A reproducible ~5 second delay confirms blind/time-based injection.REQUESTS & RESPONSES

## 



## #2 Improper Error Handling

**CATEGORY -** Input Validation / File Access (Authentication area observed during authenticated session)

**SEVERITY -** High

**CVSS -** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

**CWE -** CWE-98: Improper Control of Filename for include/require statement (Local File Inclusion)

## AFFECTED ENDPOINT(S)

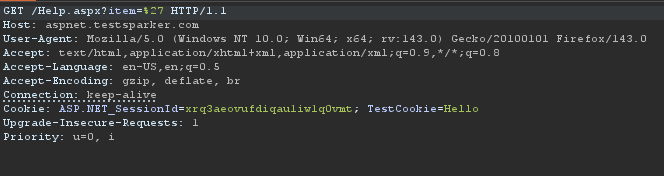
http://aspnet.testsparker.com/Help.aspx?item=’

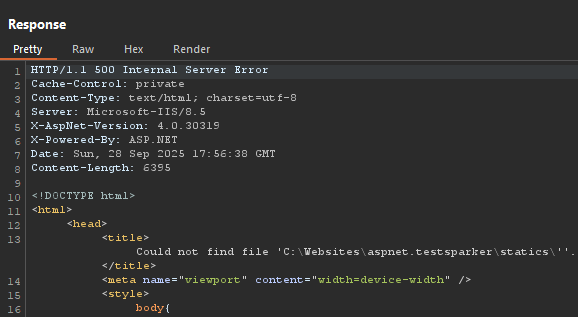
## DESCRIPTION

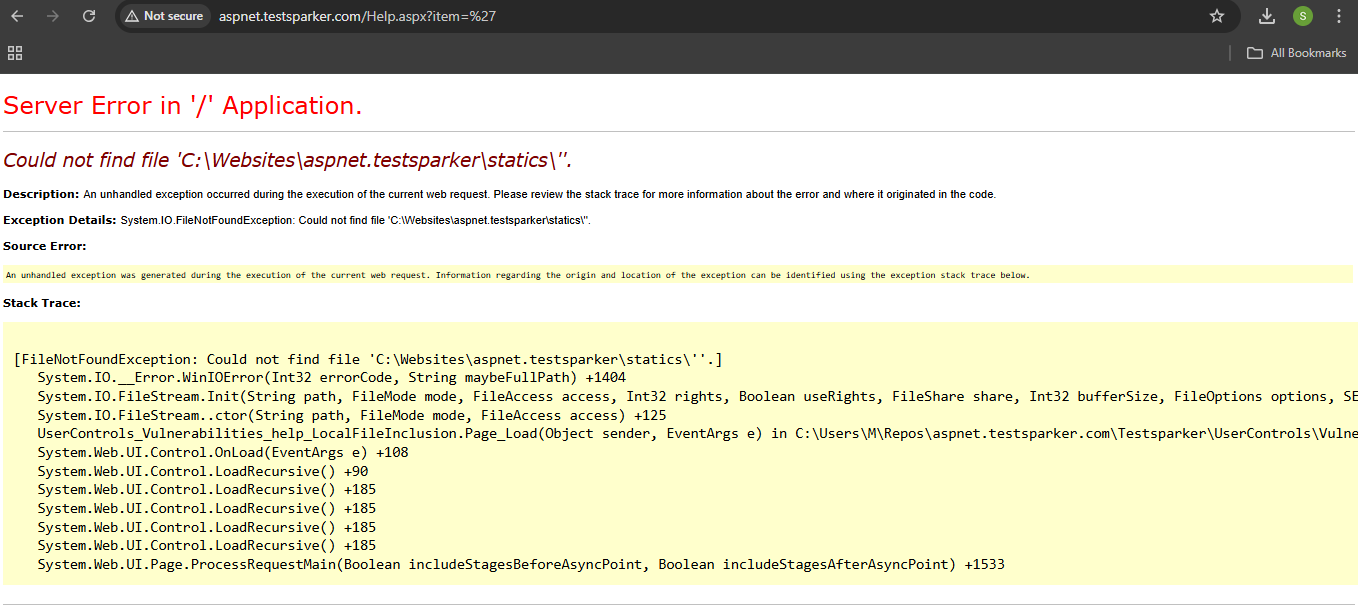
The application directly uses a user-supplied `item` parameter in a file access routine without proper validation or sanitization. Supplying a single-quote (') as the `item` parameter caused the application to attempt to access a file path containing the unescaped quote and resulted in an unhandled System.IO.FileNotFoundException. The application returned a server error page with a full .NET stack trace and an absolute file path (C:\Websites\aspnet.testsparker\statics\') — disclosing sensitive server filesystem information.  
  
**STEPS TO REPRODUCE**

1. Open the application and authenticate using the provided test credentials:  
 - Email: alan@turing.com  
 - Password: theturingtest  
2. In a browser, request the vulnerable endpoint with a single quote as `item` parameter:  
 GET /Help.aspx?item=%27 HTTP/1.1  
 Host: aspnet.testsparker.com  
3. Observe server response: an HTTP 500 page is returned containing the .NET exception and stack trace. Example excerpt observed shows FileNotFoundException and full paths including source file and line number.  
4. Confirm that error reveals absolute paths, source file reference and line number (help-LocalFileInclusion.ascx.cs:43).

## REQUESTS & RESPONSES



RESPONSE (excerpt):  




## MITIGATION

1. Input Validation / Whitelist: Do not accept arbitrary file names/paths from user input. Implement an allowlist of permitted help topics/IDs and map those to server-side filenames.  
     
   2. Sanitize & Normalize Input: Strictly validate and normalize values (no '.', no path separators, no quoting characters). Reject anything that doesn’t match strict pattern (e.g., ^[A-Za-z0-9\_-]{1,64}$).

**#3 Server Version Disclosure (Site-wide Information Exposure)**

**Category -** Information Disclosure / Configuration

**Severity -** Medium

**CVSS -** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N (Approx)

CWE-200: Information Exposure

## AFFECTED ENDPOINT(S)

Site-wide — all responses that include the following headers (observed during requests to /Guestbook.aspx and other pages):

- Server: Microsoft-IIS/8.5

- X-AspNet-Version: 4.0.30319

- X-Powered-By: ASP.NET

## DESCRIPTION

The application and web server disclose detailed server and framework version information in HTTP response headers. During testing a POST to /Guestbook.aspx returned headers including `Server: Microsoft-IIS/8.5`, `X-AspNet-Version: 4.0.30319` and `X-Powered-By: ASP.NET`. Exposing such versions allows attackers to accurately fingerprint the server stack and search for publicly known, version-specific vulnerabilities or exploit chains. This is an information disclosure issue caused by default server/application configuration and can be remediated by suppressing or removing version-identifying headers.

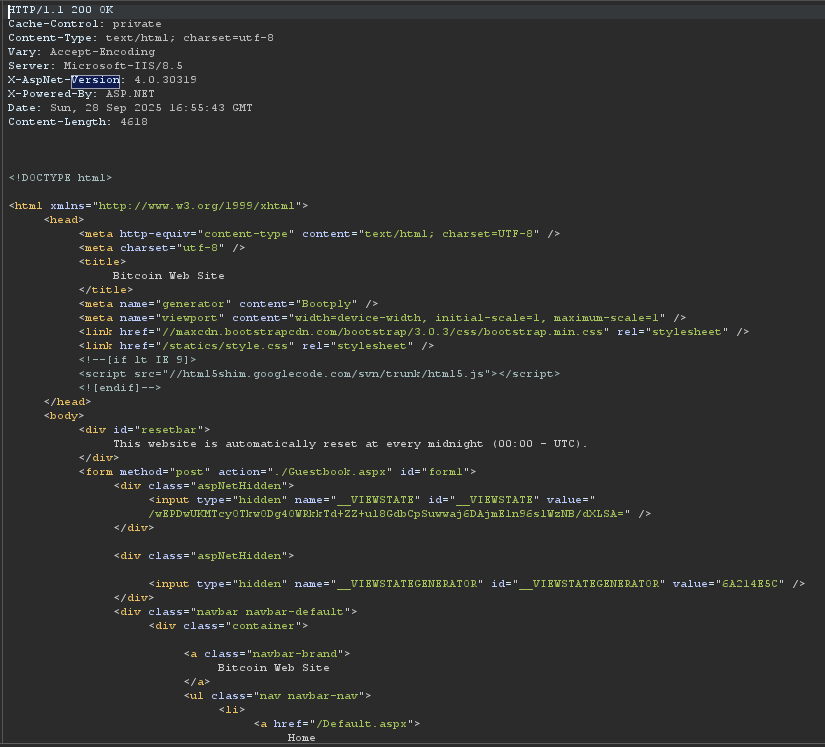
## PROOF OF CONCEPT (STEPS TO REPRODUCE)

1. Send an HTTP request (GET or POST) to any application endpoint (e.g., /Guestbook.aspx, /Default.aspx).  
   2. Inspect the HTTP response headers returned by the server.  
   3. Observe the presence of version-identifying headers such as `Server: Microsoft-IIS/8.5`, `X-AspNet-Version: 4.0.30319`, and `X-Powered-By: ASP.NET`.

REQUEST:



RESPONSE:

1. 

## MITIGATION:

1. Remove or suppress server and framework version headers in responses:  
 - For ASP.NET, set `httpRuntime` to disable the version header in web.config:  
 <system.web>  
 <httpRuntime enableVersionHeader="false" />  
 </system.web>  
  
 - For IIS, consider removing or rewriting the `Server` header via URL Rewrite outbound rules or using a fronting reverse proxy that strips/overwrites headers.  
  
 - Remove `X-Powered-By` by clearing the header in web.config or via custom headers removal:  
 <system.webServer>  
 <security>  
 <requestFiltering removeServerHeader="true" />  
 </security>  
 <httpProtocol>  
 <customHeaders>  
 <remove name="X-Powered-By" />  
 </customHeaders>  
 </httpProtocol>  
 </system.webServer>