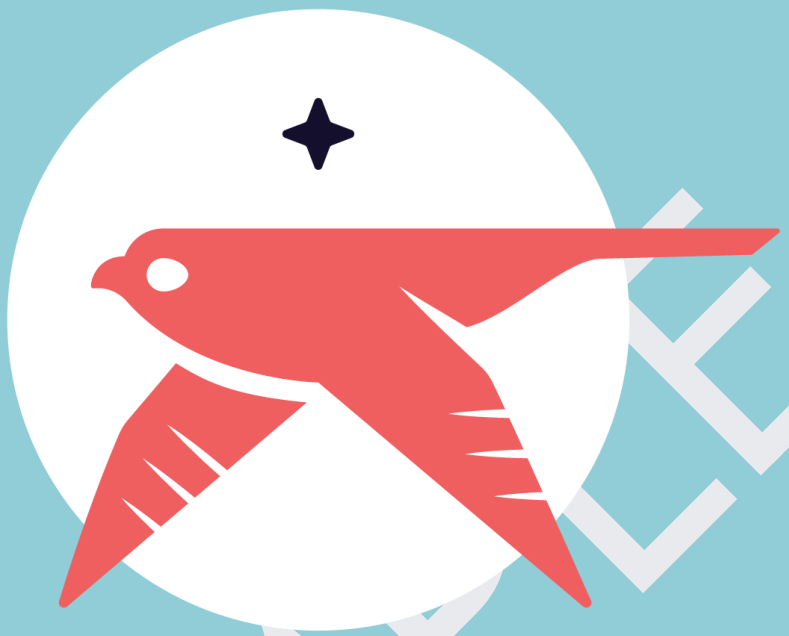


adversis

Web App Penetration Test Sample App

Prepared for Acme, Inc

John Doe
jdoe@example.com

January 1, 2025

Adversis, LLC

PO Box 2953
Kalispell, MT 59901
+1 (877) 353-1337
hello@adversis.io

Contents

Executive Summary

App	ACME App
Type	Web Application Penetration Test
Scope	acmetest.example.com
Dates	January 1 to January 14, 2024
Outcome	Robust Controls

Overview

Adversis conducted a black-box assessment of Acme Inc's ACME Application above and beyond OWASP ASVS Level 1 methodology.

Testing evaluated the web application and REST API powering ACME's App solution using SDK-based and direct REST approaches.

Aspects of regulatory compliance pertaining to key data elements were also considered.

Key Findings

The ACME application uses a robust web framework and configuration, inherently providing strong defenses against common vulnerabilities. Specifically, the authentication and authorization framework is industry standard and well-designed with secure features and defaults. Unlike using raw SQL queries, the app also uses safe, language-integrated database queries in the code to prevent vulnerabilities that can lead to data loss.

Critically, a reliance on insufficient built-in input filters and missing output encoding capabilities introduces stored cross-site scripting vulnerabilities, which can lead to account compromise and eventual data leakage.

Additionally, the absence of role authorization checks during report execution enables some users to access information they should not be able to, although highly sensitive information remains restricted through appropriate database checks.

Positive Controls

The ACME application demonstrates strong security measures and robust configuration, enhancing its resilience against common cyber threats. These include:

- A robust web framework and secure configuration
- Secure account authentication library usage and strong data handling measures

In particular, the team identified several items worth championing.

- Strong authentication: The OWIN authentication middleware automatically implements essential security features, which help mitigate vulnerabilities in account takeovers. In addition, the application enforces an automatic session timeout with a judiciously set duration, minimizing the risk of unauthorized access during periods of inactivity.
- Secure File Handling: Files cannot be uploaded or made accessible without proper authentication. Additionally, files are neither hosted nor executed on the application server; instead, Azure services and unique identifiers are used, providing an extra layer of security.
- Validation and Encryption: Both server and client-side validations are rigorously applied, including measures against Cross-Site Request Forgery (CSRF) and encryption of ViewState data. These controls are essential for maintaining the integrity and confidentiality of user interactions.
- SQL Injection Protection: Using Linq to Entities Framework for database interactions effectively prevents SQL injection, a prevalent threat that can lead to significant data loss.

- Effective Authorization Checks: The application exhibits a well-implemented system for user roles and authorization verification, ensuring access rights are strictly enforced according to user privileges.

Overall, the ACME web application's security posture is commendable. It reflects a thoughtful and proactive approach to protecting data and user interactions against a range of threats.

Strategic Recommendations

To build upon this strong foundation, several approaches should be considered.

1. Ensure all user-provided information is encoded before displaying it to protect against Cross-Site Scripting (XSS) attacks. This practice ensures that any special characters are converted into HTML entities, preventing them from being interpreted as code.

   Implementing HTML encoding systematically across the application will fortify your defenses against one of the most common web security threats. Fortunately, the .NET `HtmlEncode` method can be used to accomplish this easily.

2. Authorization controls for data and report execution should be bolstered. Given the sensitivity of the information in reports, implementing additional authorization requirements will help ensure that only qualified users can access or generate reports.

   Fortunately, data element access is still restricted by the current user role, preventing sensitive information such as names from being produced in unauthorized report data.

Conclusion

Adversis has highlighted the ACME application's strong security and effective protective measures through this penetration test against a variety of threats.

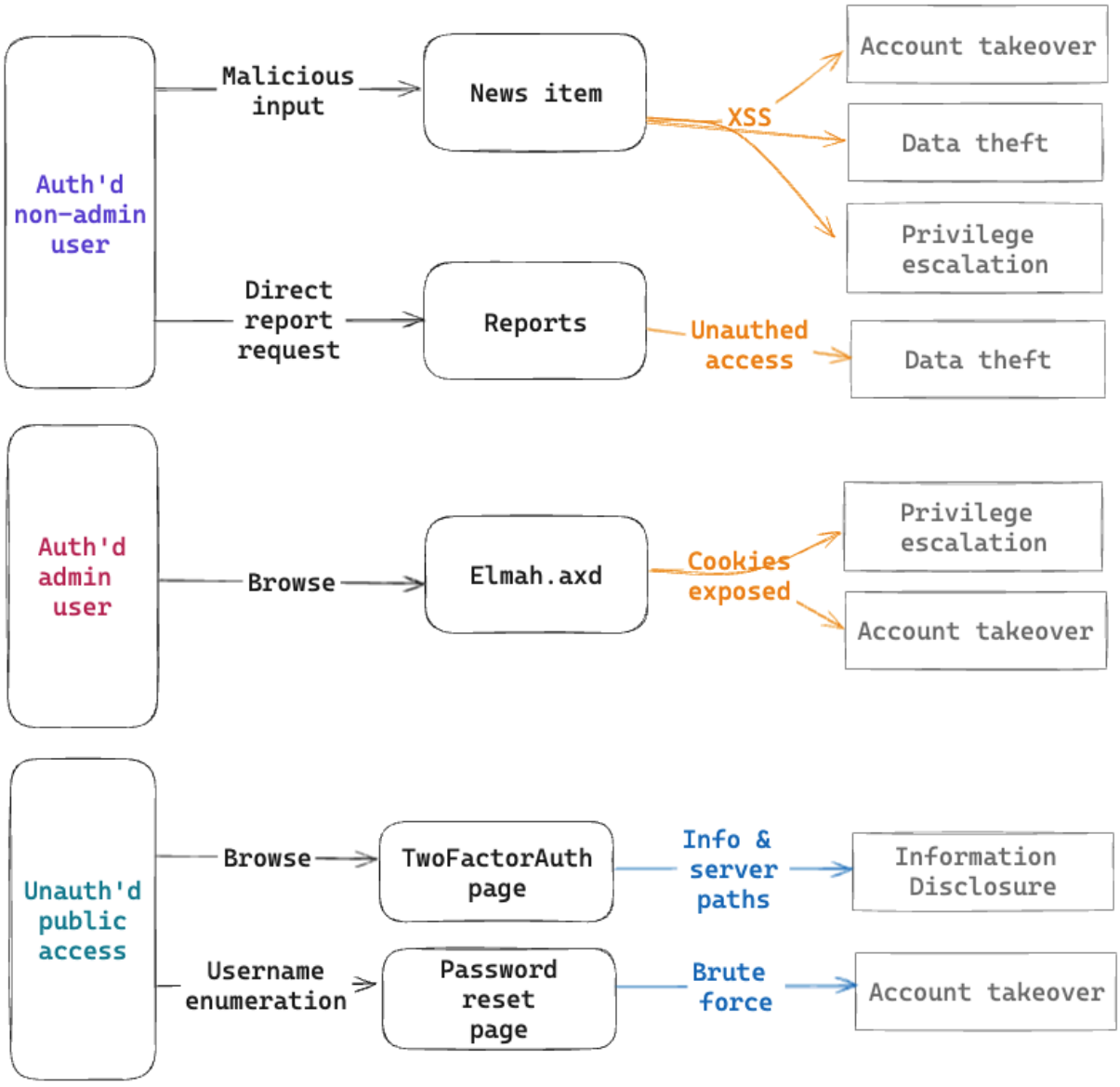
By utilizing secure coding practices, robust session management, and advanced security mechanisms, the application maintains a high level of protection for both data and user interactions.

To build upon this strong foundation, we recommend implementing HTML encoding for all outputs and enhancing authorization protocols for report access.

These enhancements will secure the application against emerging threats and ensure it remains resilient in a dynamic security environment.

Attack Summary Diagram

Following is a basic visual description of issues identified in the ACME application.



Findings Overview

ID	Description	Risk
1	**Stored Cross-Site Scripting (XSS) Vulnerability in Contact Form** Filter bypass and a lack of output encoding allow attackers to introduce arbitrary code that affects a user's browsing, typically leading to account takeover.	Moderate
2	**Error Logging Modules and Handlers (ELMAH) Exposed to Admins** All admin users can view sensitive authentication cookies for other users, which could allow privilege escalation and unauthorized data access.	Moderate
3	**Unauthorized Report Execution by Low-Privilege Users** Low-privileged users can run reports and obtain data from reports typically not accessible to them by specifying the report to run.	Moderate
4	**Username Enumeration Vulnerability in Forgot Password Flow** Login messages vary their responses, allowing an attacker to determine usernames to facilitate account takeover and brute force attempts.	Low
5	**Missing Content-Security-Policy Security Header** Missing security headers is a defense in depth configuration to decrease the impact of cross-site scripting and other client-side vulnerabilities.	Low
6	**Information Disclosure in Detailed Error Messages** An unhandled error on an unauthenticated login page returns sensitive information about the application, including software version info, server paths, and file names.	Informational

Several issues were still noted but not documented due to limited security impact. These include:

- Missing HttpOnly flag on a non-sensitive cookie
- Cross-domain source file inclusion for a content distribution network
- Missing Strict-Transport-Security Headers

Findings Details

1. Stored Cross-Site Scripting (XSS) Vulnerability in Contact Form

Description

Stored Cross-Site Scripting (XSS) occurs when an application does not encode special characters before displaying them to the user's browser. If an attacker injects a malicious script into a web page, which is then viewed by another user, the attacker can influence the code run in the victim's browser. In the reported case, the application's Contact form is vulnerable to an XSS attack via encoded characters that bypass the built-in Request Validation feature. Specifically, encoding a dotless 'i' allows an attacker to evade filters and inject harmful scripts, such as an "iframe" element, which is then stored and executed on the client's browser. Critically, the harmful script is not encoded (or neutralized) before being displayed to the user's browser.

Risk

Moderate

Impact

An attacker can execute arbitrary HTML and JavaScript code in a user's browser to perform malicious activities like keystroke logging, user redirection, or data exfiltration, which can compromise user privacy and data integrity.

Location

Component: Contact submission form

- https://acmetest.example.com/Contact.aspx?Key=16
- https://acmetest.example.com/User.aspx?Key=127

Recommendations

- Use Context-Sensitive Encoding such as the HttpUtility.HtmlEncode to encode all user-supplied input before rendering it on any page, ensuring that any malicious scripts are neutralized. The XSS doesn't render on the home page since HtmlEncode is used. Microsoft also states not to rely on ASP.NET's ValidateRequest filter.[1]
- Implement a Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that may still occur, including restricting the loading of remote attacker-supplied resources. (See *Missing Content-Security-Policy Header* finding).

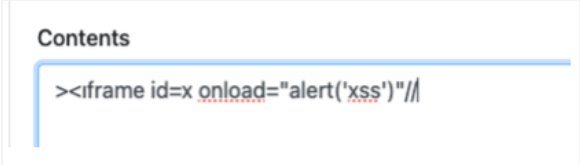
[1] https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ff647397(v=pandp.10)

Details

Within the Contact page, create a new item. Enter the following text.

><ıframe id=x onload="alert('xss')"//

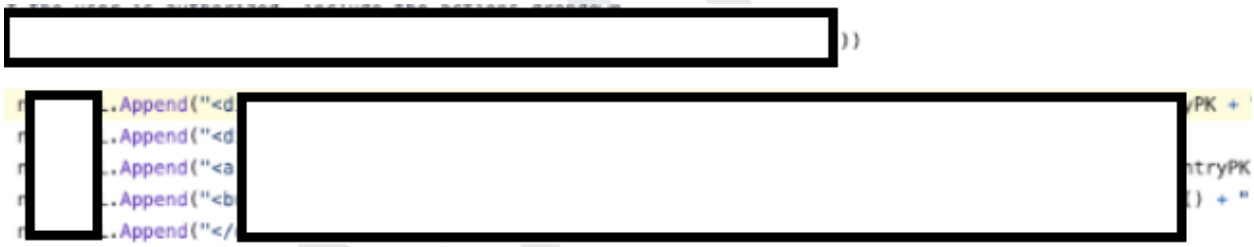
Note that a dotless 'ı' character is used in place of the normal "i" character. Typically the ValidateRequest filter will prevent this input but the Unicode letter bypasses the filter. Once the Contact item is saved, it is converted to a standard "i", spelling "iframe". The code executes once the Contact item is viewed.



A malicious payload about to be saved in the Contact item

The encoded payload will execute if stored and rendered without proper sanitization.

As seen below in the code, the Contact entry is simply appended to the HTML output without being safely encoded.



Insecure code does not safely encode user-supplied input

2. Error Logging Modules and Handlers (ELMAH) Exposed to Admins

Description

Error Logging Modules and Handlers (ELMAH) is an error logging utility that captures unhandled exceptions in ASP.NET applications and offers a built-in web interface for viewing these errors. The identified issue is that the ELMAH interface allows access from all administrator roles. This configuration could lead to privilege escalation and unauthorized actions because authentication cookies are accessible to anyone accessing the ELMAH page.

Risk

Medium

Impact

While the ELMAH page is restricted to the admin role, any user with the admin role can impersonate any other user by visiting the Elmah.axd page and obtaining another user's application cookie. These include Application Admin and Super Admin roles. If a Super Admin causes an error to occur, other admin roles could compromise the account of a Super Admin in this manner.

Location

- https://acmetest.example.com/elmah.axd
- web.config's elmah section sets allowRemoteAccess to true

Recommendations

While the Elmah configuration appears to be correct in that only Admins can access the file, consider further restricting access to only Super Admins if possible. For example, remove the Elmah handler in web.config and implement the controller in the application, only allowing Super Admins to access it.

Resources

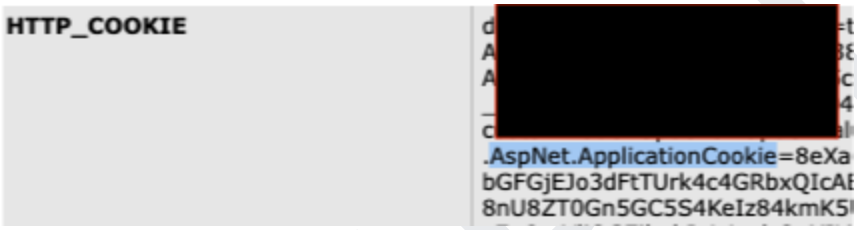
- http://beletsky.net/2011/03/integrating-elmah-to-aspnet-mvc-in.html

Details

After browsing to the /elmah.axd page with an admin role returns all application errors, as seen below.

Application error logs as seen in elmah.axd

After selecting one of the errors, debug information for the user's request is returned, including the .AspNet.ApplicationCookie value, which allows anyone to impersonate that user's session.



Session cookie value exposed in the elmah.axd log

3. Unauthorized Report Execution by Low Privilege Users

Description

Low-privileged users can execute reports they do not have permission to access through the web application interface itself. While these users are restricted from viewing unauthorized data within these reports, the ability to execute and interact with these reports represents a breach of access control.

Risk

Moderate

Impact

Although sensitive information within the reports, such as names and birthdays, remains protected and inaccessible to roles that do not have explicit access, the ability for unauthorized users to execute reports could lead to information disclosure.

Location

- https://acmetest.example.com/Reports/Reports.aspx

Recommendations

Implement strict access control checks to ensure that users can only execute reports for which they have explicit permissions.

Details

Log in with a low-privileged user, navigate to Reports, and select the "Log Analysis Export" report. Intercept the request and modify the report form values as seen below by specifying the specific report to run.

```
# As a low-privileged user
POST /Reports/Reports.aspx HTTP/1.1
Host: acmetest.example.com
...[snip]...
RunReport=3
RunName=Log+Analysis+Export
...[snip]...

# Response
HTTP/1.1 200 OK
Content-Type: application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
Content-Disposition: attachment; filename=LogAnalysisExport.xlsx

Key5Ÿ¨AçE@¬Ç/xl/worksheets/sheet1.xml
...
```

HTTP request and response from a user requesting an unauthorized report

The web application returns a user data dump with user emails and phone numbers.

4. Username Enumeration Vulnerability in Forgot Password Flow

Description

The application exhibits a username enumeration vulnerability within its forgot password functionality. This issue arises when different responses are provided based on whether the submitted username exists in the system.

In the forgot password section of the application, when someone enters a username, the system checks if that username exists. If it does not exist, the application explicitly states so. However, if the username is valid, the application indicates an email has been sent. This difference in responses can let an attacker deduce which usernames are valid on the platform simply by observing the messages returned by the system.

Risk

Low

Impact

Username enumeration vulnerabilities allow malicious actors to gather valid usernames. This can lead to targeted attacks, such as phishing or brute force attacks, where these valid usernames are exploited. If an attacker successfully identifies valid usernames, they can target specific users with phishing attempts and perform brute-force attacks on these accounts to guess passwords. Importantly, the application does not disclose the user's email address, which limits spear phishing attempts.

Recommendations

To mitigate this vulnerability, the application should provide consistent responses for Forgot Password attempts, regardless of whether the username exists. For example:

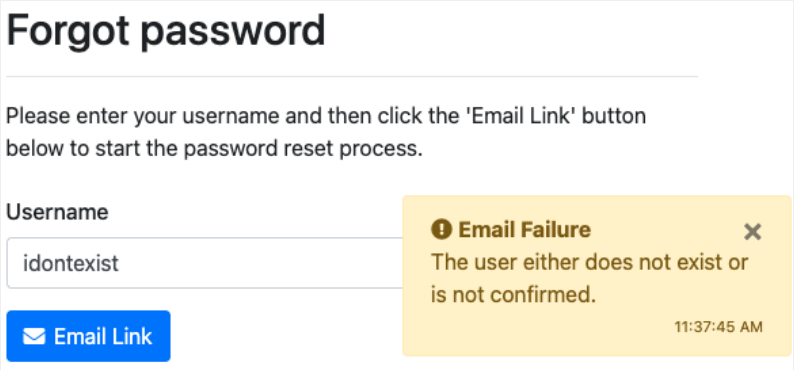
- Uniform Error Messaging: Modify the forgot password response to be generic. For example: "If your username is recognized, a password reset email will be sent."

Location

- https://acmetest.example.com/Account/Forgot.aspx

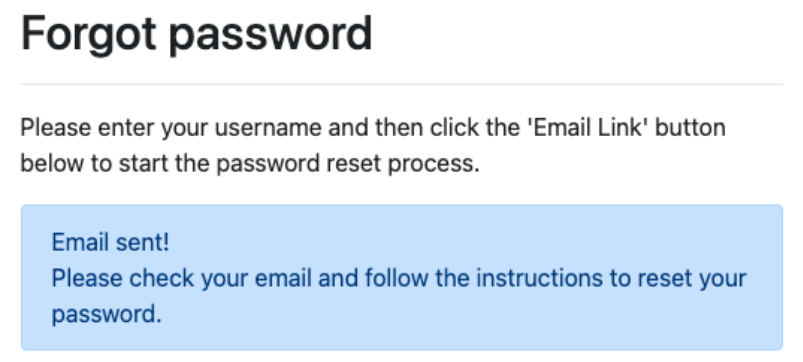
Details

Access the Forgot Password Page. Enter a username that you know does not exist. Note the specific error message stating that the user doesn't exist.



Application informing the user that the user does not exist

Enter a known valid username and observe that the response changes to indicate an email has been sent.



Application informing the user that the provided username exists

The difference in error messages allows attackers to build a list of usernames that can be used in slow password attacks over time.

5. Missing Content-Security-Policy Security Header

Description

A Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks, which could allow attackers to inject malicious scripts into web pages viewed by users. No CSP policy was identified.

Risk

Low

Impact

Without a CSP, the application is susceptible to attacks where unauthorized scripts can be run in the context of a user's session, potentially leading to unauthorized access, data theft, or malicious redirection. The absence of a Content-Security-Policy can lead to data breaches and account takeovers as malicious scripts executed through XSS can access session tokens, user personal information, and other sensitive data.

Recommendations

Implement a Content Security Policy that specifies which sources can load resources like scripts, stylesheets, and images. For example, implement a configuration in web.config or Startup.cs with the following config:

```
<system.webServer>
  <httpProtocol>
    <customHeaders>
      <add name="Content-Security-Policy"
        value="default-src 'self';
        img-src data: https: http:;
        script-src 'self' example.cloudfront.net 'unsafe-inline';
        style-src 'self' 'unsafe-inline';
        connect-src 'self';
        worker-src 'self' blob:;
        frame-src 'self' blob:;/>
    </customHeaders>
  </httpProtocol>
</system.webServer>
```

- Regularly check and update the CSP as needed, and monitor for any attempted breaches that could signal a need for CSP adjustment.
    - https://report-uri.com/products/content_security_policy
    - https://report-uri.com/home/analyse

Resources

- https://content-security-policy.com/
- https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html#csp-sample-policies
- https://docs.devexpress.com/GeneralInformation/404541/security/content-security-policy

6. Information Disclosure in Detailed Error Messages

Description

Detailed error messages can provide significant information about the application's internal workings. This typically includes database errors, server paths, or software versions. While this information is useful for development and debugging, exposing it to end users or the public can lead to security vulnerabilities by giving potential attackers clues on exploiting the system. In this case, an uncaught application exception in the TwoFactorAuthentication page discloses web application file paths and version information.

Risk

Informational

Impact

Exposure of system details through error messages can help an attacker formulate more targeted attacks, potentially leading to unauthorized access or data exposure. Displaying detailed error messages can lead to information leakage, revealing sensitive information about the backend systems, such as software versions and server paths, which can be used to find and build exploits in less time.

Location

- https://acmetest.example.com/Account/TwoFactorAuthenticationSignIn

Recommendations

Configure the application to remove detailed stack trace information and application version information. Consider replacing the "RemoteOnly" config with "DetailedLocalOnly" and rely on ELMAH.

A sample web.config section follows:

```
<system.web>
  <httpRuntime enableVersionHeader="false"/>
</system.web>
<system.webServer>
  <security>
    <requestFiltering removeServerHeader="true" />
    <httpErrors errorMode="DetailedLocalOnly">
</system.webServer>
```

Resources

- https://cheatsheetseries.owasp.org/cheatsheets/DotNet_Security_Cheat_Sheet.html#encryption
- https://learn.microsoft.com/en-us/previous-versions/dotnet/netframework-4.0/h0hfz6fc(v=vs.100)#example

Details

Browsing to the page /Account/TwoFactorAuthentication when a user is unauthenticated and not logged in, the web application displays an error message that includes a detailed error message disclosing the folder and path of the application, including the developer name and source file name. The file path:

C:\vendor\client\TwoFactorAuthentication.aspx.cs:12

can be in the below stack trace.

```
[InvalidOperationException: UserId not found.]
   Microsoft.AspNet.Identity.<GetValidTwoFactorProvidersAsync>d__129.MoveNext() +1002
   System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task) +138
   System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task
   Microsoft.AspNet.Identity.AsyncHelper.RunSync(Func`1 func) +266
   Microsoft.AspNet.Identity.UserManagerExtensions.GetValidTwoFactorProviders(UserManager`2
```

Detailed stack trace exposed along with source code path

Appendix - Web Application Security Assessment Methodology

Adversis brings extensive experience in assessing web applications. Adversis consultants hold CVE's across various web applications and common libraries and have experience building and breaking modern web application technologies such as React and GraphQL.

Comprehensive Assessment: Adversis conducts a thorough assessment leveraging the OWASP Application Security Verification Standard (ASVS) while identifying additional security issues, particularly business logic flaws. Our examination ensures comprehensive coverage of potential vulnerabilities to a minimum of OWASP ASV Level 1 standards and higher as appropriate.

Manual Testing: Adversis's approach to manual testing is exhaustive. We review and test action-performing (CRUD) requests within the application, ensuring a thorough evaluation of potential vulnerabilities, emphasizing, but not limited to, the OWASP Top 10.

Tool Utilization: Our primary tool, Burp Suite Pro, is integral to our testing process. It is selected for its capabilities in advanced vulnerability identification and manual testing features, which enable our team to pinpoint and address security weaknesses efficiently.

OWASP Top 10

Adversis always reviews the following categories for the OWASP Top Ten vulnerabilities and believes that Level One controls of the OWASP Application Security Verification Standard (ASVS)[2] are critical for any publicly facing application.

1. Broken Access Control
2. Cryptographic Failures
3. Injection
4. Insecure Design
5. Security Misconfiguration
6. Vulnerable and Outdated Components
7. Identification and Authentication Failures
8. Software and Data Integrity Failures
9. Security Logging and Monitoring Failures
10. Server Side Request Forgery (SSRF)

In addition, the team considers your application's purpose, functionality, and data requirements, considering how data flows may be abused or functionality impacted.

[2] https://owasp.org/www-project-application-security-verification-standard/

Appendix - Risk Framework

Adversis leverages an industry-standard NIST (National Institute of Standards and Technology) threat matrix outlined in SP 800-30[3]. This matrix is a framework for analyzing risks and threats modified with input from the Factor Analysis in Information Risk (FAIR)[4] ontology. The matrix provides a structured and standardized approach to identifying, assessing, and prioritizing cybersecurity risks.

Likelihood of Threat Event	Risk Rating				
Very High	High	High	Very High	Very High	Very High
High	Medium	High	High	Very High	Very High
Significant	Medium	Medium	High	High	Very High
Moderate	Low	Medium	Medium	High	High
Low	Low	Low	Medium	Medium	Medium
Very Low	Low	Low	Medium	Medium	Medium
Loss Magnitude	**Neglible**	**Minor**	**Moderate**	**Major**	**Severe**

Event Frequency	Probability Range	Calibration Anchor
Very High	>76%	Likely exploitation within 12 months
High	51-75%	More likely than not to see exploitation at some point
Moderate	26-50%	It could go either way
Low	5-25%	Unlikely but possible
Very Low	<5%	All the stars would have to align

Loss Magnitude	Loss Factors	Business Context
Severe	>$1 Million	Financial impact, regulatory action, executive changes
Major	$100k - $1 million	Significant operational disruption, compliance violations
Moderate	$10k - $100k	Noticeable service degradation, limited data exposure
Minor	$1k - $10k	Minimal disruption, no sensitive data
Neglible	$0 - $1k	Best practice

Recommended Response Times

Critical	Remediation within 7 days
High	Remediation within 60 days
Medium	Remediation within 180 days
Low	Accept or remediate within 365 days

[3] https://csrc.nist.gov/pubs/sp/800/30/r1/final
[4] https://www.fairinstitute.org/what-is-fair