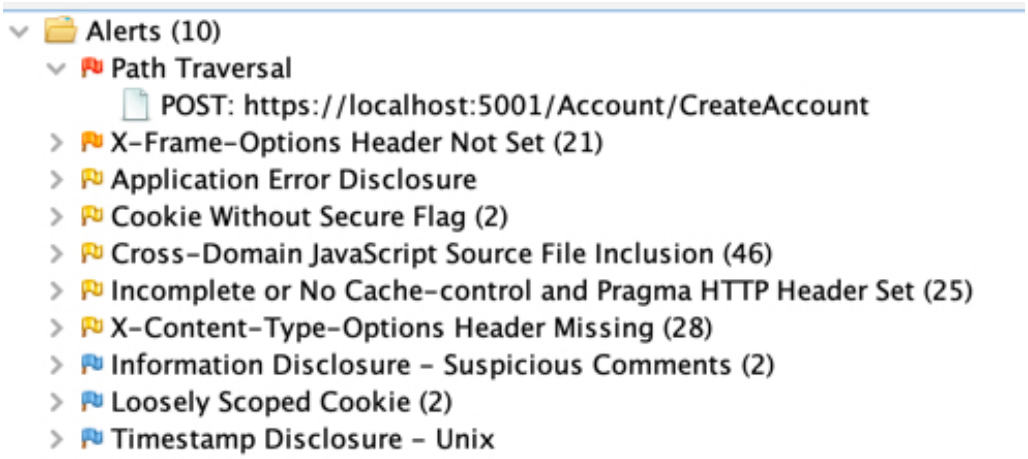
Jecca Arthur
CS296N
Term Project
Web Security

Security Alerts:



High Risk: Path Traversal

Path Traversal	
URL:	https://localhost:5001/Account/CreateAccount
Risk:	High
Confidence:	Medium
Parameter:	Username
Attack:	/CreateAccount
Evidence:	
CWE ID:	22
WASC ID:	33
Source:	Active (6 - Path Traversal)

Description

The Path Traversal attack technique allows an attacker access to files, directories, and commands that potentially reside outside the web document root directory. An attacker may manipulate a URL in such a way that the web site will execute or reveal the contents of arbitrary files anywhere on the web server. Any device that exposes an HTTP-based interface is potentially vulnerable to Path Traversal.

Solution

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use an allow list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely

exclusively on looking for malicious or malformed inputs (i.e., do not rely on a deny list). However, deny lists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

Reference
http://projects.webappsec.org/Path-Traversal
http://cwe.mitre.org/data/definitions/22.html
https://docs.microsoft.com/en-us/aspnet/mvc/overview/security/preventing-open-redirection-attacks

Mitigation
Placed additional validation on Username property of CreateAccountVM to only allow alphanumeric characters and underscores:

> *[RegularExpression(@"^[a-zA-Z0-9_]{1,60}$", ErrorMessage = "Username can contain alphanumeric characters and underscores only")]*
> *public string Username { get; set; }*

Medium Risk: X-Frame-Options Header Not Set

X-Frame-Options Header Not Set	
URL:	https://localhost:5001/
Risk:	Medium
Confidence:	Medium
Parameter:	X-Frame-Options
Attack:	
Evidence:	
CWE ID:	16
WASC ID:	15
Source:	Passive (10020 - X-Frame-Options Header)

Description
X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks.

Solution
Most modern Web browsers support the X-Frame-Options HTTP header. Ensure it's set on all web pages returned by your site (if you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you

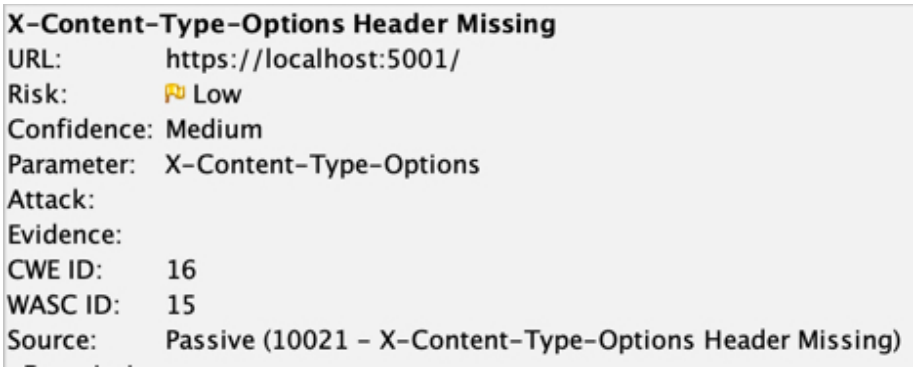
never expect the page to be framed, you should use DENY. ALLOW-FROM allows specific websites to frame the web page in supported web browsers).

Reference
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options

Mitigation
Added to Configure method in Startup.cs:

```
app.Use(async (ctx, next) =>
        {
           ctx.Response.Headers.Add("X-Frame-Options", "SAMEORIGIN");
           ctx.Response.Headers.Add("X-Content-Type-Options", "nosniff");
           await next();
        });
```

Low Risk: X-Content-Type-Options Header Missing



X-Content-Type-Options Header Missing	
URL:	https://localhost:5001/
Risk:	Low
Confidence:	Medium
Parameter:	X-Content-Type-Options
Attack:	
Evidence:	
CWE ID:	16
WASC ID:	15
Source:	Passive (10021 - X-Content-Type-Options Header Missing)

Description
The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.

Other Info
This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type.
At "High" threshold this scan rule will not alert on client or server error responses.

Solution
Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages.
If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing.

Reference
http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspx
https://owasp.org/www-community/Security_Headers

Mitigation
Added to Configure method in Startup.cs:

```
app.Use(async (ctx, next) =>
        {
           ctx.Response.Headers.Add("X-Frame-Options", "SAMEORIGIN");
           ctx.Response.Headers.Add("X-Content-Type-Options", "nosniff");
           await next();
        });
```

Low Risk: Cookie Without Secure Flag

Cookie Without Secure Flag	
URL:	https://localhost:5001/Account/Login
Risk:	Low
Confidence:	Medium
Parameter:	.AspNetCore.Antiforgery.6KUxEwJbJWw
Attack:	
Evidence:	Set-Cookie: .AspNetCore.Antiforgery.6KUxEwJbJWw
CWE ID:	614
WASC ID:	13
Source:	Passive (10011 - Cookie Without Secure Flag)

Description
A cookie has been set without the secure flag, which means that the cookie can be accessed via unencrypted connections.

Solution

Whenever a cookie contains sensitive information or is a session token, then it should always be passed using an encrypted channel. Ensure that the secure flag is set for cookies containing such sensitive information.

Reference
https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/06-Session_Management_Testing/02-Testing_for_Cookies_Attributes.html

Mitigation
Added to Configure method in Startup.cs:

> *app.UseCookiePolicy(new CookiePolicyOptions { HttpOnly = HttpOnlyPolicy.Always, Secure = CookieSecurePolicy.Always });*