CCSE Assessment 2

Student ID: 2136685

Table of Contents

Table of Figures

1. Section 2A

1.1. Testing Evidence

For the testing of my web application, I did Static Application Security Testing (SAST) to test the source code for vulnerabilities and Dynamic Application Security Testing (DAST) to test the web interface for vulnerabilities.

1.2. SAST Vulnerabilities Found

For SAST testing, I used snyk.io to scan my GitHub repository directly. The following three vulnerabilities were found in my code.

1.2.1. SQL Injection Vulnerability

The SQL Injection vulnerability, seen in Figure 1, shows up on multiple pages, with three being of high severity and four being of medium severity. The problem comes from the fact that unsanitised input is placed directly in the SQL queries, then calling the query, making it vulnerable to SQL injection (Snyk, 2022c).

To solve this, the SQL query needs to be parameterised. If the query is parameterised, then instead of an SQL injection statement being passed to the query, such as "' OR 1=1; --", the query will look, for example, for a username that matches "' OR 1=1; --" exactly, avoiding SQL injection as a result (Snyk, 2022c).

1.2.2. Path Traversal Vulnerability

The path traversal vulnerability, seen in Figure 2, appears twice in the same page, "finance-processing.php", both findings being of high severity. This vulnerability occurs when certain procedures use paths to do something related to files. For example, in the "finance-processing.php" page, an applicant uploads a supporting document for their financing application. The user can upload an arbitrary file and comprimise the server (Mitre, 2022; Snyk, 2022b).

To fix this, one needs to validate the input before inputting it into the path. To validate it, the requrest can be parameterised, as explained in the SQL injection section above (Mitre, 2022; Snyk, 2022b).

1.2.3. Cross-Site Scripting (XSS) Vulnerability

As seen in Figure 3, two high-severity cross-site scripting vulnerabilities can be found in the “finance-processing.php” page, again. This occurs when unsanitised input, for example from a $_POST variable in PHP, is echoed, allowing a user to insert HTML code where it is shouldn’t happen (Snyk, 2022a).

To fix this, the input must be sanitised, or as in the SQL injection vulnerability above, it must display the input exactly, e.g. instead of echoing “<div> hello </div>”, it will echo “ ’<div> hello </div>’ ”. Or one could just not echo $_POST variables (Snyk, 2022a).

1.3. DAST Vulnerabilities Found

For DAST testing, I used OWASP ZAP. In total, after a quick scan, there were 16 alerts, with a maximum priority of medium, and 5 were only informational alerts, as seen in Figure 4.

1.3.1. Absence of Anti-CSRF Tokens Vulnerability

CSRF stands for cross-site request forgery, which my website is vulnerable to, as seen in Figure 5. CSRF allows a threat actor to exploit an authenticated user on a website to perform an unwanted action, because browser requests include all cookies, regardless if they are forged authenticated requests or real authorised requests (OWASP, 2012; OWASP ZAP, 2021).

To fix this vulnerability, I can introduce a synchroniser token pattern which generate CSRF tokens on the server-side only once per session, or for each request. These tokens can be verified and validated so that the request can be carried out (OWASP, 2012; OWASP ZAP, 2021).

1.3.2. Content Security Policy (CSP) Header Not Set Vulnerability

As seen in Figure 6, the website is vulnerable to CSP header not set vulnerability. CSP headers are used to detect and deter attacks such as Cross-Site Scripting and data injection attacks (MacLeod, 2023; OWASP, 2022).

To fix this vulnerability, one must create the CSP firstly by writing the policies required for the website, such as “Content-Security-Policy:frame-src 'none’” to prevent iframes from being loaded. Then one must enable the header in the NGINX - which Azure uses - website’s config file, in the “sites-enabled” folder, by writing each policy as such, “add_header Content-Security-Policy:frame-src 'none’” (MacLeod, 2023; OWASP, 2022; Webdock, 2022).

1.3.3. Missing Anti-clickjacking Header Vulnerability

As seen in Figure 7, my website is vulnerable to clickjacking, as it is missing an anti-clickjacking header. A clickjacking attack occurs when a user is tricked into clicking an invisible element such as an iframe (OWASP ZAP, 2023d).

The way to fix this vulnerability is by setting a CSP header to restrict frames, as explained in the section above.

1.3.4. Cookie No HttpOnly Flag Vulnerability

Figure 8 shows that my website's cookie has no HttpOnly flag, which makes the cookie accessible by JavaScript. To fix this, I need to ensure that the HttpOnly flag is set for all cookies by adding "add_header Set-Cookie "Path=/; HttpOnly; Secure";" in the nginx.conf file under the http block (Kumar, 2017; OWASP ZAP, 2023a).

1.3.5. Cookie Without Secure Flag Vulnerability

Figure 9 shows that the cookies used in my website don't have the secure flag set, allowing the cookies to be accessed via unencrypted connections. To fix this, I need to add the line shown in the section above to the nginx.conf file, specifically the "Secure" part (Kumar, 2017; OWASP ZAP, 2023c).

1.3.6. Cookie without SameSite Attribute Vulnerability

As seen in Figure 10, my website does not have the "SameSite" attribute set for the cookies, allowing cookies to be sent through cross-site request. Setting the SameSite attribute is also a good counter measure for CSRF, which was discussed in a section above. To fix this, I need to add "SameSite=none" to the end of the line discussed in section 1.3.4. (NGINX, 2023; OWASP ZAP, 2023b).

1.3.7. Cross-Domain JavaScript Source File Inclusion Vulnerability

Figure 11 warns me that my website uses third-party JavaScript files. This is true, as I use some icons and Bootstrap JavaScript files, however this is fine as long as the source is trustworthy. To fix this, I can download the JavaScript file to ensure I have a local copy which I control.

1.3.8. Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s) Vulnerability

As seen in Figure 12, my website leaks my PHP version information through "X-Powered-By" HTTP response headers. This can help threat actors find vulnerabilities in the software the website server uses. To fix this PHP leak, I need to set "expose_php = off" in the php.ini config file or I can set "server_tokens off;" in the nginx.conf config file (r00t4bl3, 2018).

1.3.9. Server Leaks Version Information via "Server" HTTP Response Header Field Vulnerability

Figure 13 shows the same problem as in the section above, however it exposes the nginx version instead. To fix this, I need to set "server_tokens off;", as explained in the section above.

1.3.10. Strict-Transport-Security Header Not Set Vulnerability

Seen in Figure 14, my website doesn't have the Strict-Transport-Security Header set. This header ensures that browsers can only interact with my website using HTTPS connections. To fix this, I need to add the line "add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;" to the website's config file's server block in my "sites-enabled" folder (NGINX, 2016).

2. Section 2B

2.1. GitHub Repository

The GitHub repository, seen in Figure 15, has been set up since the first part of this project. The Azure DevOps pipeline authorised for this repository to ensure that for every update, the package is delivered to the website's server (Continuous Integration) and deployed (Continuous Deployment), therefore achieving automation with a CI/CD pipeline and parallelism.

2.2. Azure DevOps Setup

The DevOps Setup, as seen in Figures 16 and 17, is a automatically configured file by Azure DevOps. The web application was set up in the normal Azure Portal, with a MySQL database server for the data stored and the actual web application service. Azure DevOps, during setup prompted me to select the type of application and I chose a PHP Web Application with Azure. This choice generated the "azure-pipelines.yml", which specified the PHP version the app will use and the Azure Portal Web App, so that the GitHub repository package can be deployed to the web app. The .yml file assumes the app will require composer, however this threw an error, as I have no files related to composer, since I don't require it for the app. Therefore, I commented the line out of the .yml file and the website works as intended.

2.3. CI/CD Pipeline

The CI/CD pipeline, an example seen in Figure 18, automatically responds to any GitHub repository updates and instantly packages them to be deployed to the Azure web app. The CI/CD pipeline is executed following the "azure-pipelines.yml" script I discussed in the section above.

2.4. Web Application

The Azure Web App, seen in Figure 19, is running at "https://ccse-cw2-razvan.azurewebsites.net/". The application has some errors since I had to change the way the app connects to the database, as the Azure MySQL database requires a certificate to connect to it, and this caused my SQL queries to mostly not work. However, this was not the scope of this assignment, the scope was successfully achieved, because I was able to test for vulnerabilities with SAST and DAST scans and to place the app on the Azure Cloud and integrate a CI/CD pipeline, all successfully.

3. References

Kumar, C. (2017). How to Implement HTTPOnly and Secure Cookie in Nginx? [online] Geekflare. Available at: https://geekflare.com/httponly-secure-cookie-nginx/ [Accessed 18 May 2023].

MacLeod, R. (2023). *How to Set Up a Content Security Policy (CSP) in 3 Steps*. [online] Sucuri Blog. Available at: https://blog.sucuri.net/2023/04/how-to-set-up-a-content-security-policy-csp-in-3-steps.html [Accessed 18 May 2023].

Mitre (2022). *CWE - CWE-23: Relative Path Traversal (4.5)*. [online] cwe.mitre.org. Available at: https://cwe.mitre.org/data/definitions/23.html [Accessed 18 May 2023].

NGINX (2016). *HTTP Strict Transport Security (HSTS) and NGINX*. [online] NGINX. Available at: https://www.nginx.com/blog/http-strict-transport-security-hsts-and-nginx/ [Accessed 18 May 2023].

NGINX (2023). *Cookie-Flag*. [online] NGINX. Available at: https://www.nginx.com/products/nginx/modules/cookie-flag/ [Accessed 18 May 2023].

OWASP (2012). *Cross-Site Request Forgery Prevention · OWASP Cheat Sheet Series*. [online] Owasp.org. Available at: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html [Accessed 18 May 2023].

OWASP (2021). *SQL Injection Prevention · OWASP Cheat Sheet Series*. [online] Owasp.org. Available at: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html [Accessed 12 May 2023].

OWASP (2022). *Content Security Policy · OWASP Cheat Sheet Series*. [online] cheatsheetseries.owasp.org. Available at: https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html [Accessed 18 May 2023].

OWASP ZAP (2021). *Absence of Anti-CSRF Tokens*. [online] Zaproxy.org. Available at: https://www.zaproxy.org/docs/alerts/10202/ [Accessed 18 May 2023].

OWASP ZAP (2023a). *OWASP ZAP – Cookie No HttpOnly Flag*. [online] www.zaproxy.org. Available at: https://www.zaproxy.org/docs/alerts/10010/ [Accessed 18 May 2023].

OWASP ZAP (2023b). *OWASP ZAP – Cookie without SameSite Attribute*. [online] www.zaproxy.org. Available at: https://www.zaproxy.org/docs/alerts/10054/ [Accessed 18 May 2023].

OWASP ZAP (2023c). *OWASP ZAP – Cookie Without Secure Flag*. [online] www.zaproxy.org. Available at: https://www.zaproxy.org/docs/alerts/10011/ [Accessed 18 May 2023].

OWASP ZAP (2023d). *OWASP ZAP – Missing Anti-clickjacking Header*. [online] www.zaproxy.org. Available at: https://www.zaproxy.org/docs/alerts/10020-1/ [Accessed 18 May 2023].

r00t4bl3 (2018). *How to Hide nginx Web Server Version Signature and Remove X-Powered-By PHP Header on CentOS - r00t4bl3.com*. [online] r00t4bl3. Available at: https://r00t4bl3.com/post/how-to-hide-nginx-web-server-version-signature-and-remove-x-powered-by-php-header-on-centos [Accessed 18 May 2023].

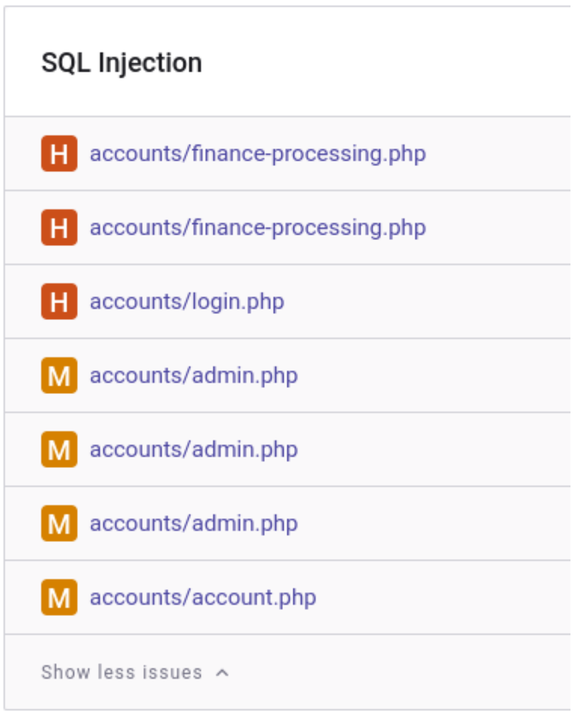
Snyk (2022a). *What is cross-site scripting (XSS)? | Tutorial & examples*. [online] Snyk Learn. Available at: https://learn.snyk.io/lessons/xss/php/ [Accessed 18 May 2023].

Snyk (2022b). *What is directory traversal? | Tutorial & examples*. [online] Snyk Learn. Available at: https://learn.snyk.io/lessons/directory-traversal/php/ [Accessed 18 May 2023].

Snyk (2022c). *What is SQL injection (SQLi)? | Tutorial & examples*. [online] Snyk Learn. Available at: https://learn.snyk.io/lessons/sql-injection/php/ [Accessed 18 May 2023].

Webdock (2022). *Configure Security Headers in Nginx and Apache» Webdock.io*. [online] webdock.io. Available at: https://webdock.io/en/docs/how-guides/security-guides/how-to-configure-security-headers-in-nginx-and-apache [Accessed 18 May 2023].

4. Appendix



SQL Injection

H accounts/finance-processing.php

H accounts/finance-processing.php

H accounts/login.php

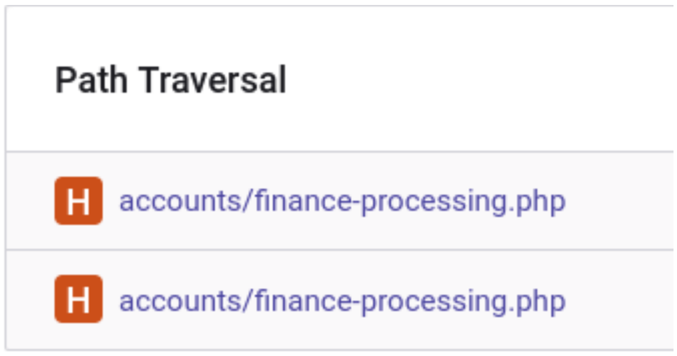
M accounts/admin.php

M accounts/admin.php

M accounts/admin.php

M accounts/account.php

Figure 1: SAST SQL Injection Vulnerability

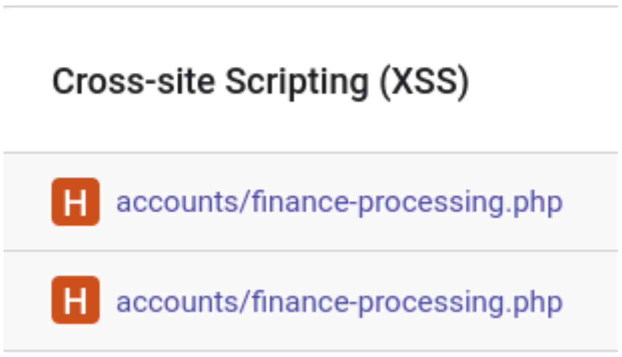


Path Traversal

H accounts/finance-processing.php

H accounts/finance-processing.php

Figure 2: Path Traversal Vulnerability


Cross-site Scripting (XSS)

H accounts/finance-processing.php

H accounts/finance-processing.php

Figure 3: Cross-Site Scripting (XSS) Vulnerability

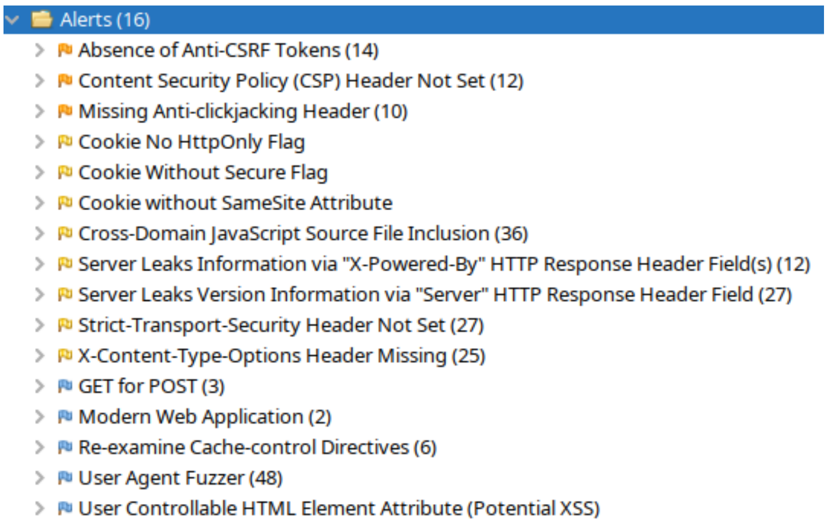
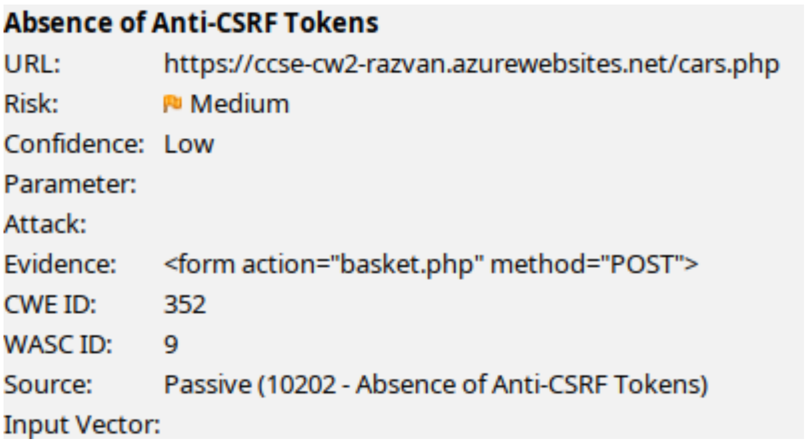
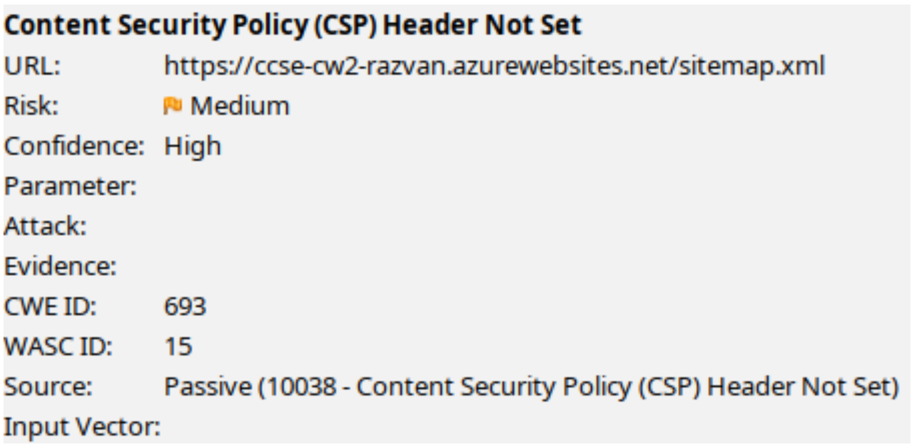

```
Alerts (16)
  Absence of Anti-CSRF Tokens (14)
  Content Security Policy (CSP) Header Not Set (12)
  Missing Anti-clickjacking Header (10)
  Cookie No HttpOnly Flag
  Cookie Without Secure Flag
  Cookie without SameSite Attribute
  Cross-Domain JavaScript Source File Inclusion (36)
  Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s) (12)
  Server Leaks Version Information via "Server" HTTP Response Header Field (27)
  Strict-Transport-Security Header Not Set (27)
  X-Content-Type-Options Header Missing (25)
  GET for POST (3)
  Modern Web Application (2)
  Re-examine Cache-control Directives (6)
  User Agent Fuzzer (48)
  User Controllable HTML Element Attribute (Potential XSS)
```

Figure 4: All DAST Vulnerabilities


Absence of Anti-CSRF Tokens

URL:	https://ccse-cw2-razvan.azurewebsites.net/cars.php
Risk:	Medium
Confidence:	Low
Parameter:	
Attack:	
Evidence:	<form action="basket.php" method="POST">
CWE ID:	352
WASC ID:	9
Source:	Passive (10202 - Absence of Anti-CSRF Tokens)
Input Vector:	

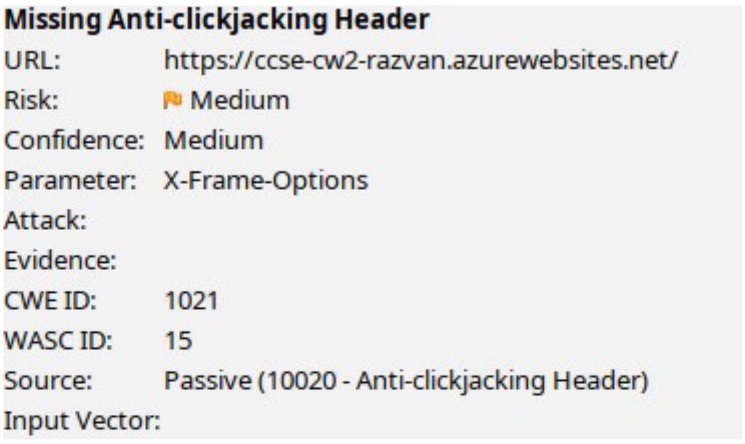
Figure 5: Absence of Anti-CSRF Tokens Vulnerability



Content Security Policy (CSP) Header Not Set

URL:	https://ccse-cw2-razvan.azurewebsites.net/sitemap.xml
Risk:	Medium
Confidence:	High
Parameter:	
Attack:	
Evidence:	
CWE ID:	693
WASC ID:	15
Source:	Passive (10038 - Content Security Policy (CSP) Header Not Set)
Input Vector:	

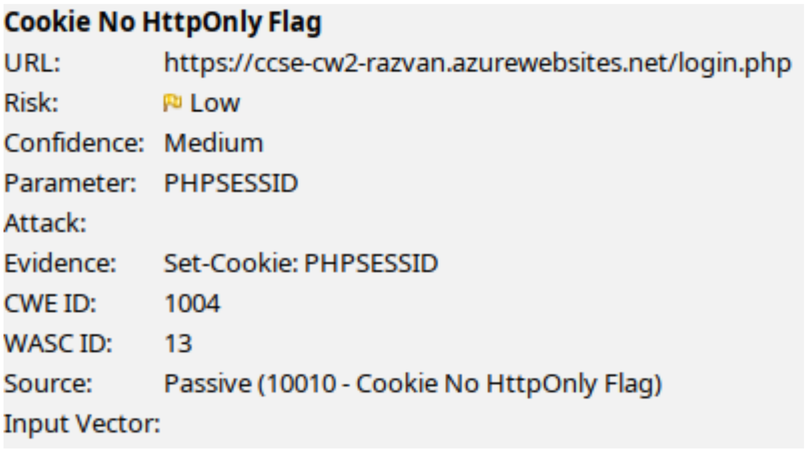
Figure 6: Content Security Policy (CSP) Header Not Set Vulnerability



Missing Anti-clickjacking Header

URL:	https://ccse-cw2-razvan.azurewebsites.net/
Risk:	Medium
Confidence:	Medium
Parameter:	X-Frame-Options
Attack:	
Evidence:	
CWE ID:	1021
WASC ID:	15
Source:	Passive (10020 - Anti-clickjacking Header)
Input Vector:	

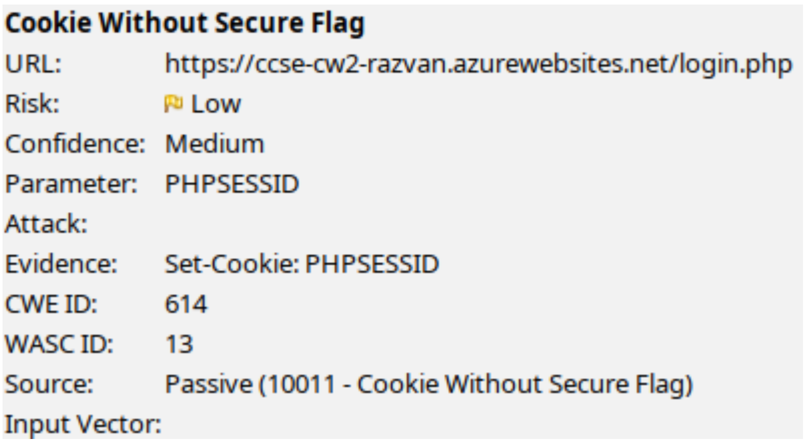
Figure 7: Missing Anti-clickjacking Header Vulnerability



Cookie No HttpOnly Flag

URL:	https://ccse-cw2-razvan.azurewebsites.net/login.php
Risk:	Low
Confidence:	Medium
Parameter:	PHPSESSID
Attack:	
Evidence:	Set-Cookie: PHPSESSID
CWE ID:	1004
WASC ID:	13
Source:	Passive (10010 - Cookie No HttpOnly Flag)
Input Vector:	

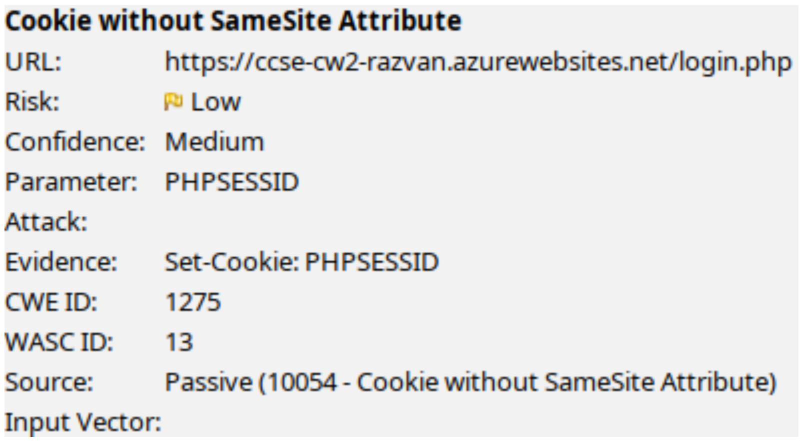
Figure 8: Cookie No HttpOnly Flag Vulnerability



Cookie Without Secure Flag

URL:	https://ccse-cw2-razvan.azurewebsites.net/login.php
Risk:	Low
Confidence:	Medium
Parameter:	PHPSESSID
Attack:	
Evidence:	Set-Cookie: PHPSESSID
CWE ID:	614
WASC ID:	13
Source:	Passive (10011 - Cookie Without Secure Flag)
Input Vector:	

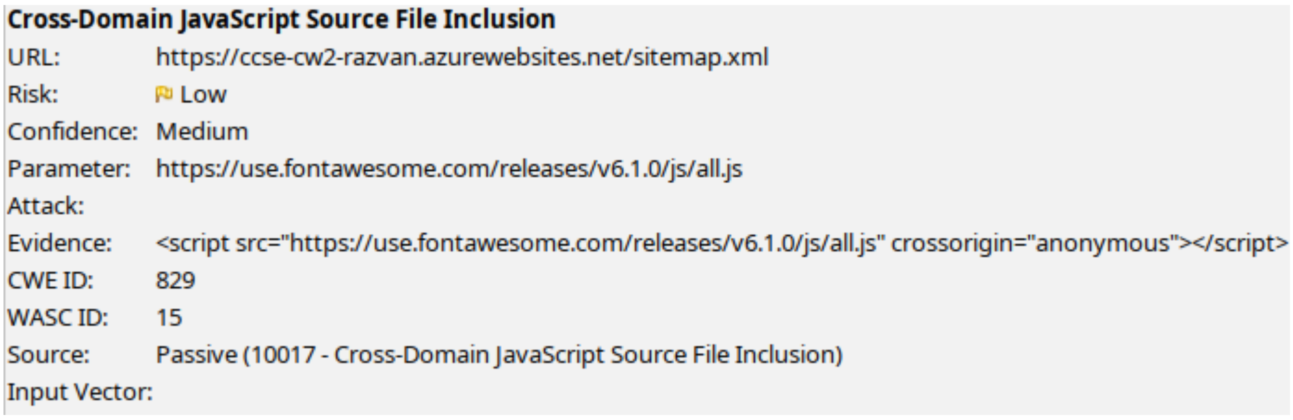
Figure 9: Cookie Without Secure Flag Vulnerability



Cookie without SameSite Attribute

URL:	https://ccse-cw2-razvan.azurewebsites.net/login.php
Risk:	Low
Confidence:	Medium
Parameter:	PHPSESSID
Attack:	
Evidence:	Set-Cookie: PHPSESSID
CWE ID:	1275
WASC ID:	13
Source:	Passive (10054 - Cookie without SameSite Attribute)
Input Vector:	

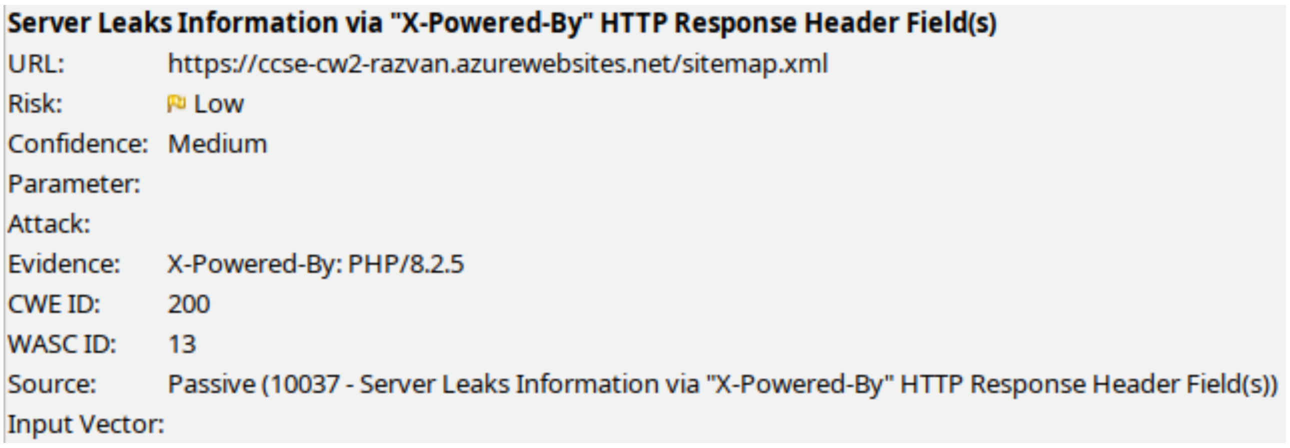
Figure 10: Cookie without SameSite Attribute Vulnerability



Cross-Domain JavaScript Source File Inclusion

URL:	https://ccse-cw2-razvan.azurewebsites.net/sitemap.xml
Risk:	Low
Confidence:	Medium
Parameter:	https://use.fontawesome.com/releases/v6.1.0/js/all.js
Attack:	
Evidence:	<script src="https://use.fontawesome.com/releases/v6.1.0/js/all.js" crossorigin="anonymous"></script>
CWE ID:	829
WASC ID:	15
Source:	Passive (10017 - Cross-Domain JavaScript Source File Inclusion)
Input Vector:	

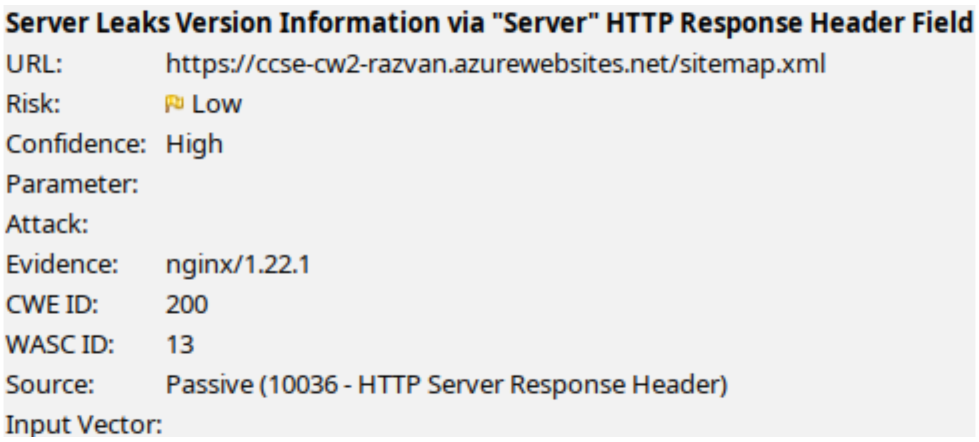
Figure 11: Cross-Domain JavaScript Source File Inclusion Vulnerability



Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)

URL:	https://ccse-cw2-razvan.azurewebsites.net/sitemap.xml
Risk:	Low
Confidence:	Medium
Parameter:	
Attack:	
Evidence:	X-Powered-By: PHP/8.2.5
CWE ID:	200
WASC ID:	13
Source:	Passive (10037 - Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s))
Input Vector:	

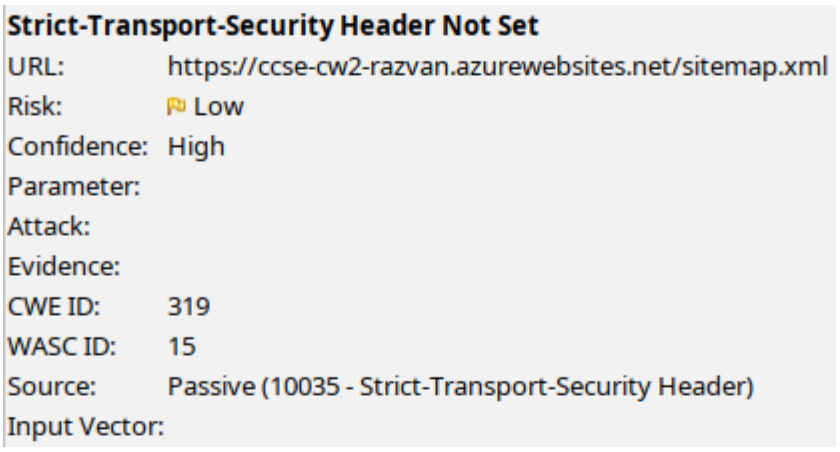
Figure 12: Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s) Vulnerability



Server Leaks Version Information via "Server" HTTP Response Header Field

URL:	https://ccse-cw2-razvan.azurewebsites.net/sitemap.xml
Risk:	Low
Confidence:	High
Parameter:	
Attack:	
Evidence:	nginx/1.22.1
CWE ID:	200
WASC ID:	13
Source:	Passive (10036 - HTTP Server Response Header)
Input Vector:	

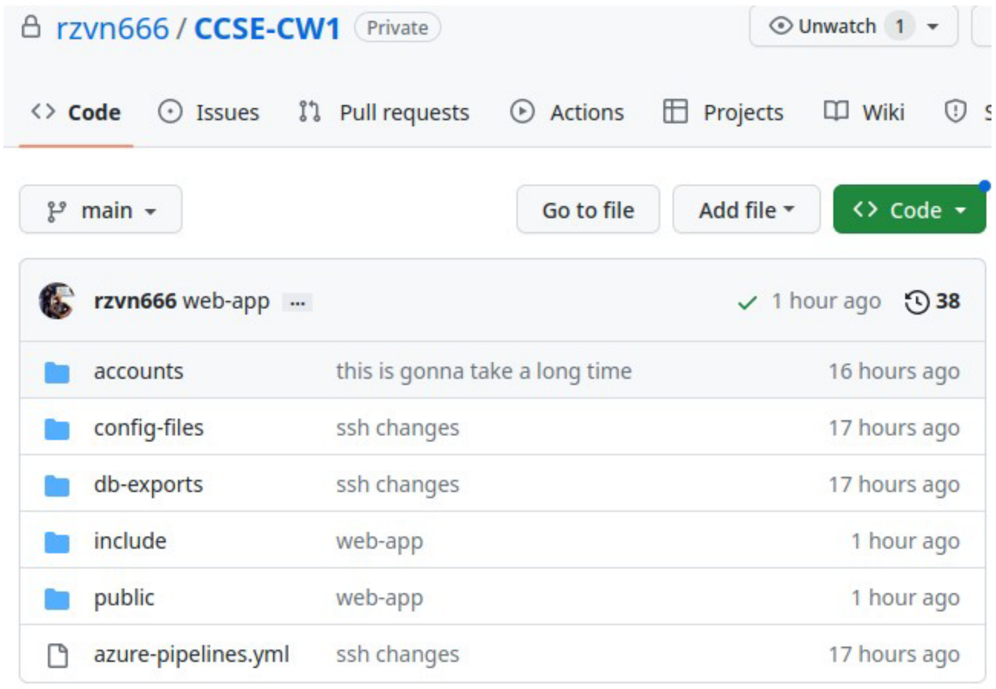
Figure 13: Server Leaks Version Information via "Server" HTTP Response Header Field Vulnerability



Strict-Transport-Security Header Not Set

URL:	https://ccse-cw2-razvan.azurewebsites.net/sitemap.xml
Risk:	Low
Confidence:	High
Parameter:	
Attack:	
Evidence:	
CWE ID:	319
WASC ID:	15
Source:	Passive (10035 - Strict-Transport-Security Header)
Input Vector:	

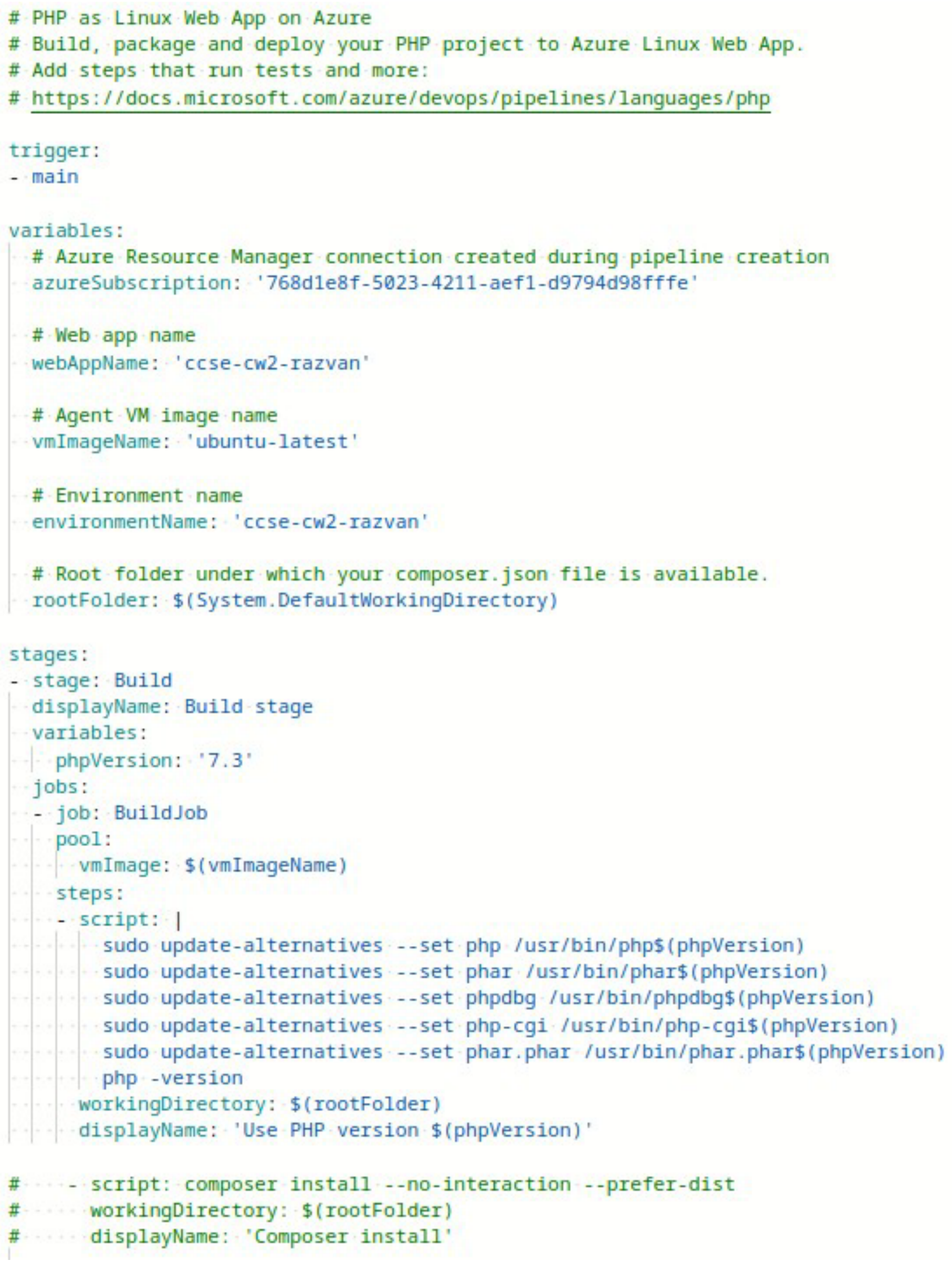
Figure 14: Strict-Transport-Security Header Not Set Vulnerability



rzvn666 web-app ... 1 hour ago 38

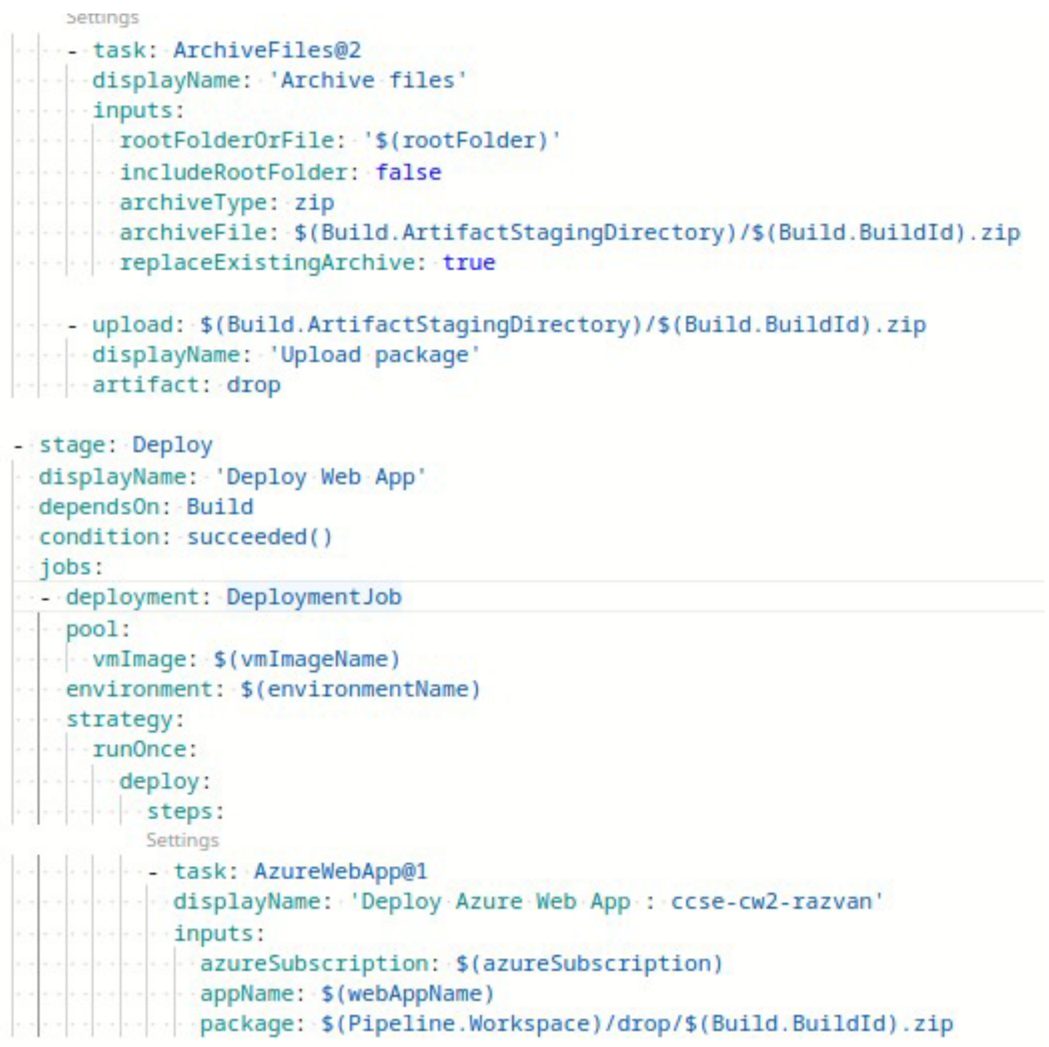
accounts	this is gonna take a long time	16 hours ago
config-files	ssh changes	17 hours ago
db-exports	ssh changes	17 hours ago
include	web-app	1 hour ago
public	web-app	1 hour ago
azure-pipelines.yml	ssh changes	17 hours ago

Figure 15: GitHub Repository



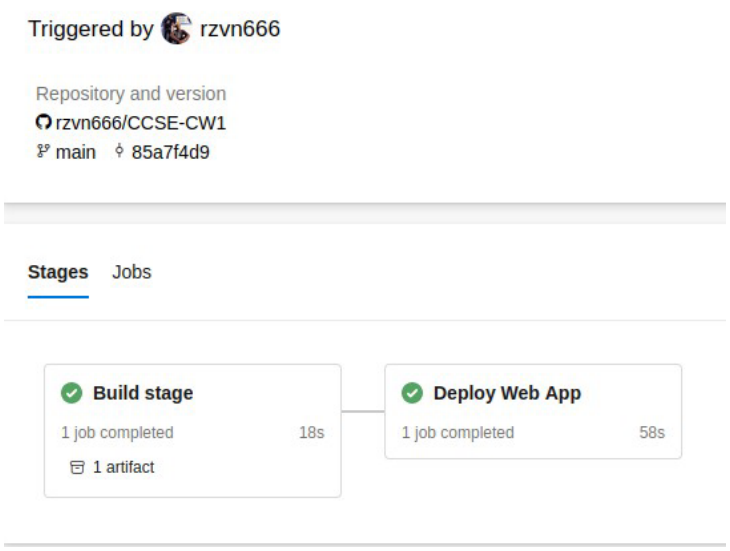
```
# PHP as Linux Web App on Azure
# Build, package and deploy your PHP project to Azure Linux Web App.
# Add steps that run tests and more:
# https://docs.microsoft.com/azure/devops/pipelines/languages/php

trigger:
- main

variables:
  # Azure Resource Manager connection created during pipeline creation
  azureSubscription: '768d1e8f-5023-4211-aef1-d9794d98fffe'

  # Web app name
  webAppName: 'ccse-cw2-razvan'

  # Agent VM image name
  vmImageName: 'ubuntu-latest'

  # Environment name
  environmentName: 'ccse-cw2-razvan'

  # Root folder under which your composer.json file is available.
  rootFolder: $(System.DefaultWorkingDirectory)

stages:
- stage: Build
  displayName: Build stage
  variables:
    phpVersion: '7.3'
  jobs:
  - job: BuildJob
    pool:
      vmImage: $(vmImageName)
    steps:
    - script: |
        sudo update-alternatives --set php /usr/bin/php$(phpVersion)
        sudo update-alternatives --set phar /usr/bin/phar$(phpVersion)
        sudo update-alternatives --set phpdbg /usr/bin/phpdbg$(phpVersion)
        sudo update-alternatives --set php-cgi /usr/bin/php-cgi$(phpVersion)
        sudo update-alternatives --set phar.phar /usr/bin/phar.phar$(phpVersion)
        php -version
      workingDirectory: $(rootFolder)
      displayName: 'Use PHP version $(phpVersion)'

#    - script: composer install --no-interaction --prefer-dist
#      workingDirectory: $(rootFolder)
#      displayName: 'Composer install'
```

Figure 16: Azure DevOps Setup (Part 1)



```
    Settings
    - task: ArchiveFiles@2
      displayName: 'Archive files'
      inputs:
        rootFolderOrFile: '$(rootFolder)'
        includeRootFolder: false
        archiveType: zip
        archiveFile: $(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip
        replaceExistingArchive: true

    - upload: $(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip
      displayName: 'Upload package'
      artifact: drop

- stage: Deploy
  displayName: 'Deploy Web App'
  dependsOn: Build
  condition: succeeded()
  jobs:
  - deployment: DeploymentJob
    pool:
      vmImage: $(vmImageName)
    environment: $(environmentName)
    strategy:
      runOnce:
        deploy:
          steps:
          Settings
          - task: AzureWebApp@1
            displayName: 'Deploy Azure Web App : ccse-cw2-razvan'
            inputs:
              azureSubscription: $(azureSubscription)
              appName: $(webAppName)
              package: $(Pipeline.Workspace)/drop/$(Build.BuildId).zip
```

Figure 17: Azure DevOps Setup (Part 2)



Triggered by rzvn666

Repository and version
rzvn666/CCSE-CW1
main 85a7f4d9

Stages Jobs

Build stage
1 job completed 18s
1 artifact

Deploy Web App
1 job completed 58s

Figure 18: Latest CI/CD pipeline run

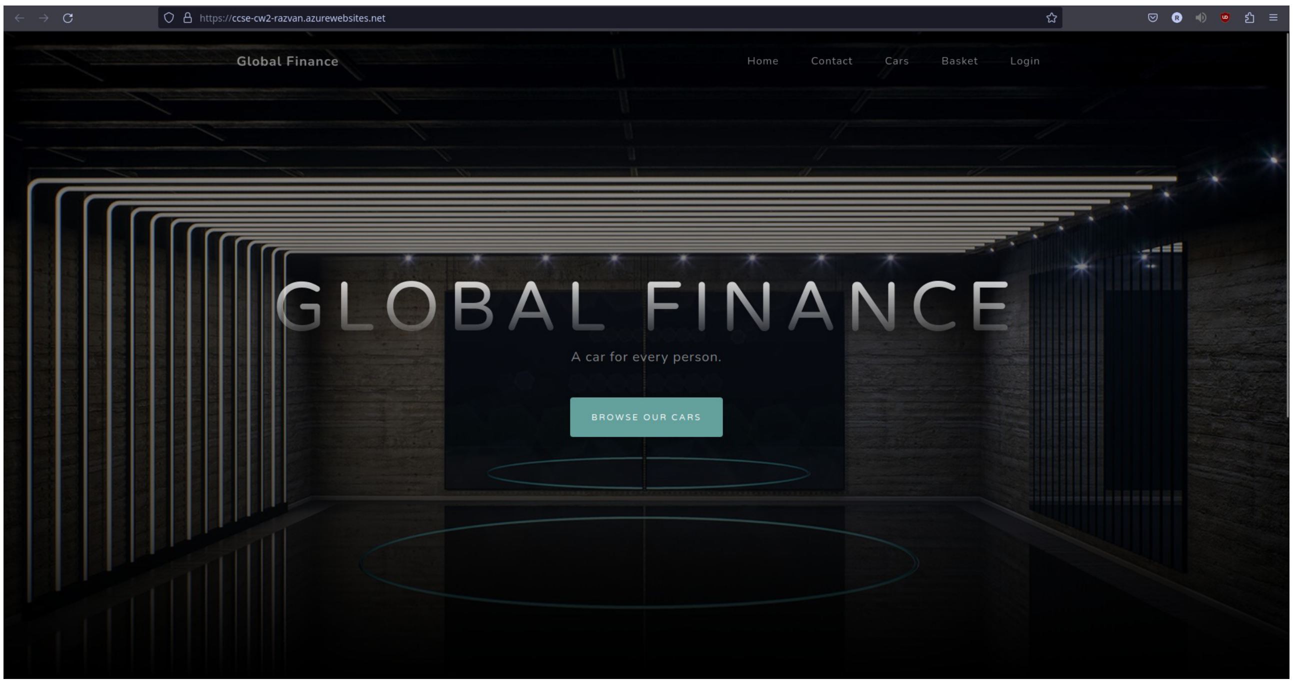


Figure 19: Web App on Azure Public Domain