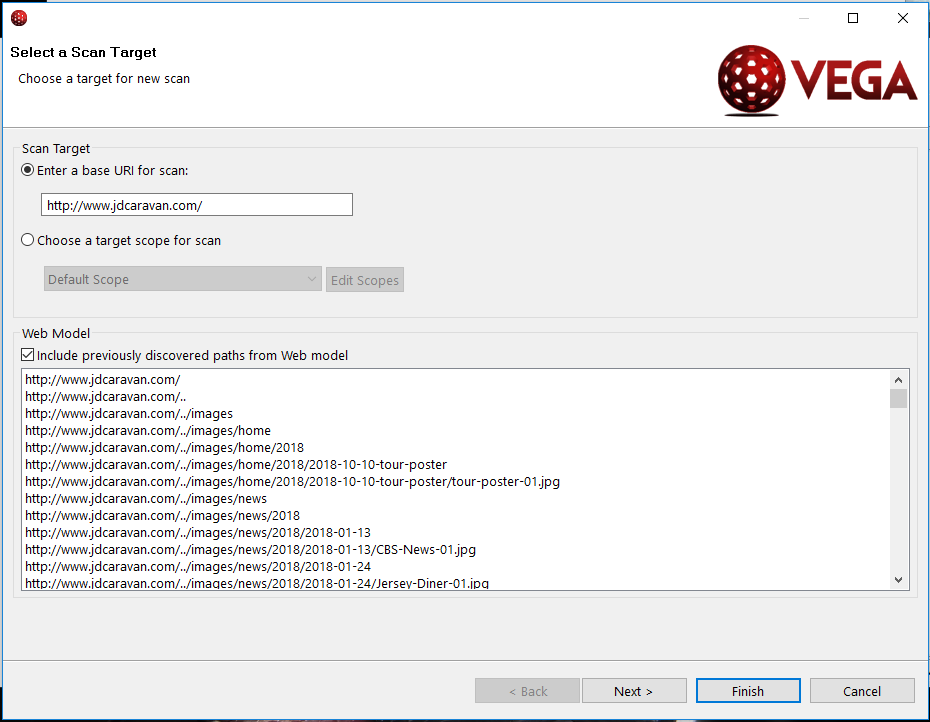
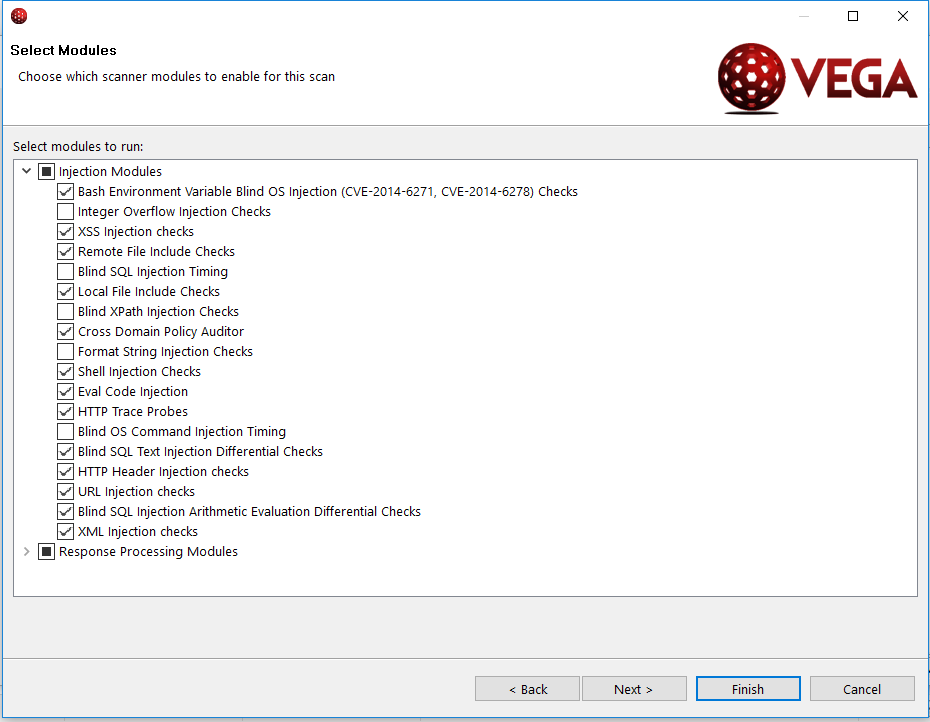
**USER MANUAL**

**1.**



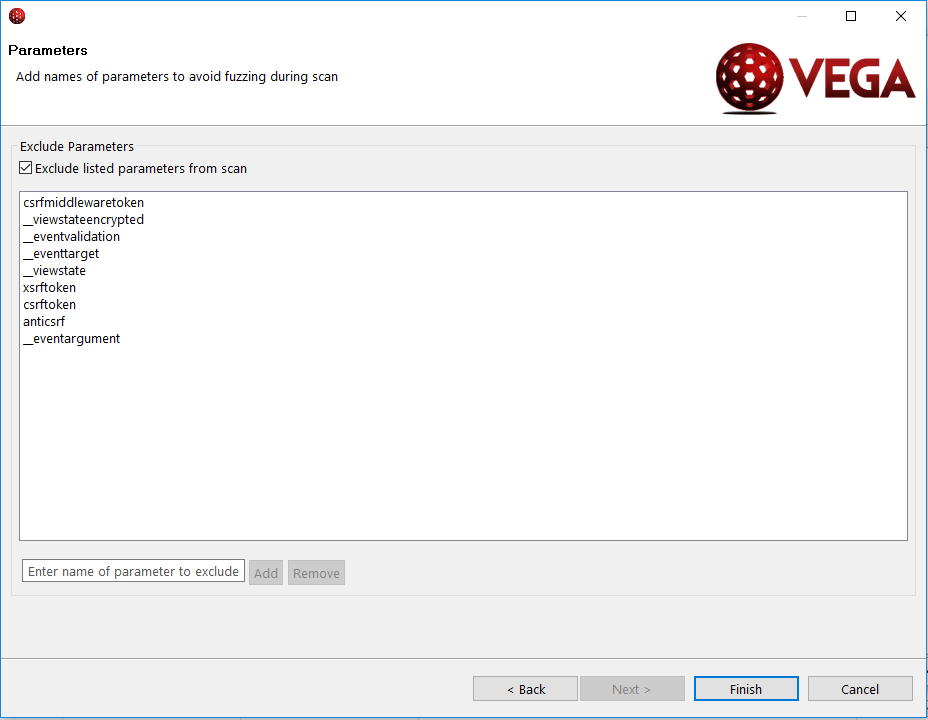
Firstly you start your vega vulnerability and to start a scan, click the new scan icon at the top left corner. Pick your targeted website , copy its URI and paste it in the vega scanner for scanning. Press next to next settings.

**2.**



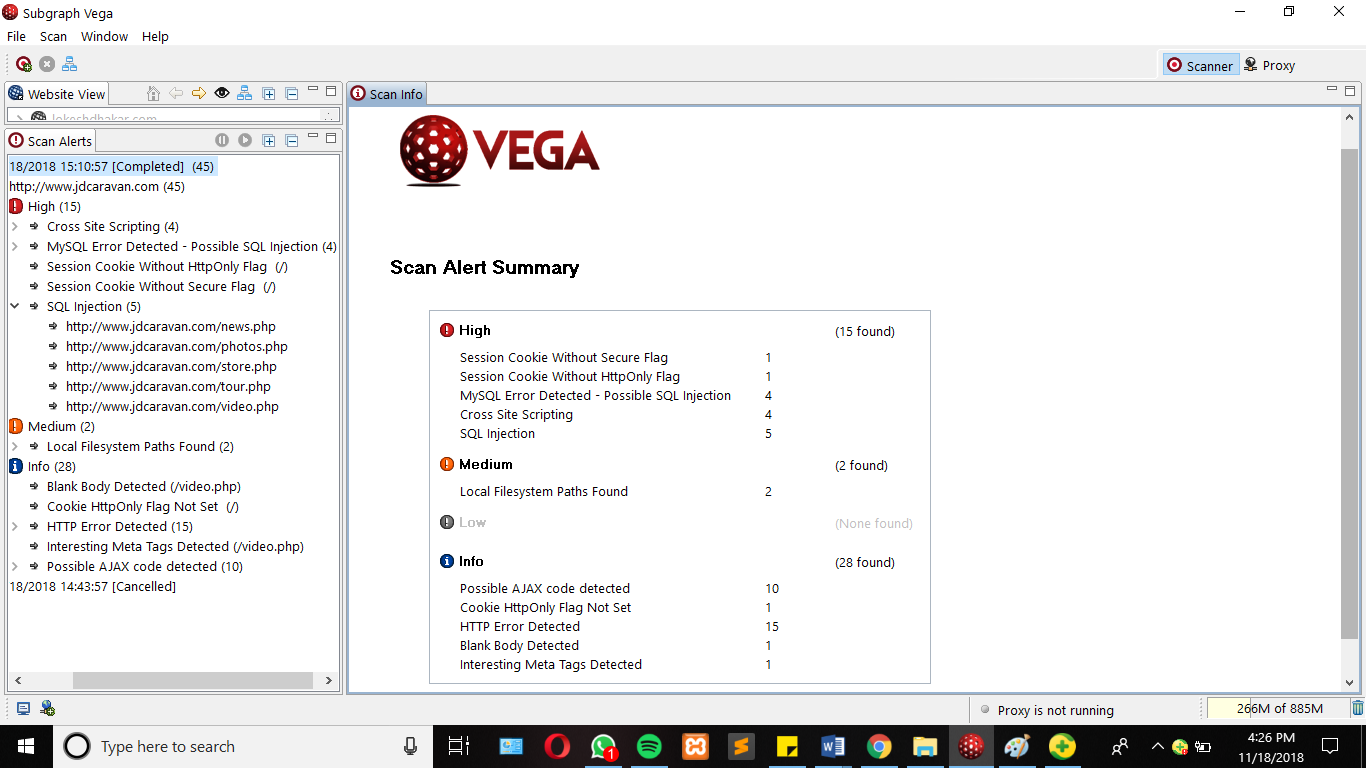
Moving to modules, now you can select which vulnerability to be exploit it the targeted website. Now we select the most common and popular injections to be exploit. If the ticked injections successfully scan it will display in the scan alert.

**3.**



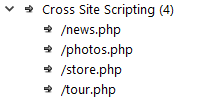
Click finish to start scanning.

**4.**



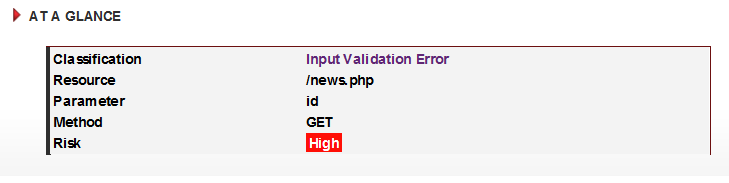
Based on the result there are 15 high and 2 medium vulnerabilities.

**1.CROSS SITE SCRIPTING**



The results shows that the website is vulnerable to cross site scripting in 4 php files.

**1.1 SUMMARY OF THE XSS**



A summary info of XSS on the website. In this example we took the news.php page. The risk is high which means this is a very dangerous vulnerability that can badly destruct the website by an attacker. Method using is GET to request data from the resource and parameter id for the variable

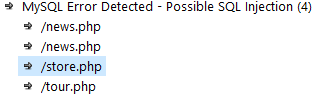
**1.2 IMPACT THAT CAN EFFECT**

* XSS is generally a threat to web applications which have authenticated users or are otherwise security sensitive
* Malicious code may be able to manipulate the content of the site, changing its appearance and/or function for another user.

**1.3 WAYS TO COUNTER THE XSS**

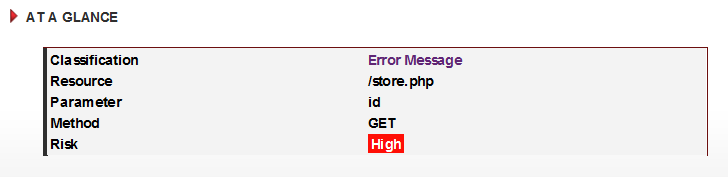
* The developer must identify how the untrustworthy data is being output to the client without adequate filtering.
* There are various language/platform specific techniques for filtering untrustworthy data.
* General rules for preventing XSS can be found in the recommended OWASP XSS Prevention Cheat Sheet (see references).

**2.My SQL ERROR DETECTED – POSSIBLE SQL INJECTION**



The results shows that the website is vulnerable to cross site scripting in 4 php files.

**2.1SUMMARY OF MYSQL ERROR**



For the example we pick store.php to be shown for the summary. The risk is high which means this is a very dangerous vulnerability that can badly destruct the website by an attacker. Method using is GET to request data from the resource and parameter id for the variable. SQL error detected can indicate a possible SQL injection vulnerability. These vulnerabilities are present when externally-supplied input is used to construct a SQL query.

**2.2 IMPACT THAT CAN EFFECT**

* This may indicate an SQL injection vulnerability, though this is not confirmed
* If this is due to an SQL injection condition, exploitation of SQL injection vulnerabilities can also allow for attacks against the logic of the application.
* Attackers may be able to obtain unauthorized access to the server hosting the database.

**2.3 WAYS TO COUNTER THE MYSQL ERROR**

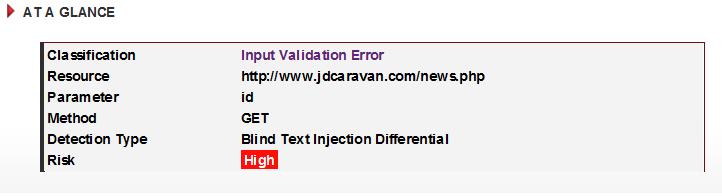
* The developer should review the request and response against the code to manually verify whether or not a vulnerability is present.
* The best defense against SQL injection vulnerabilities is to use parameterized statements
* Sanitizing input can prevent these vulnerabilities. Variables of string types should be filtered for escape characters, and numeric types should be checked to ensure that they are valid.
* Use of stored procedures can simplify complex queries and allow for tighter access control settings.
* Configuring database access controls can limit the impact of exploited vulnerabilities. This is a mitigating strategy that can be employed in environments where the code is not modifiable.

**3.SQL INJECTION**



The results shows that the website is vulnerable to cross site scripting in 5 php files

**3.1 SUMMARY OF SQL INJECTION**



For the example we pick <http://www.jdcaravan.com/news.php>to be shown for the summary. The risk is high which means this is a very dangerous vulnerability that can badly destruct the website by an attacker. Method using is GET to request data from the resource and parameter id for the variable. .These vulnerabilities are present when externally-supplied input is used to construct a SQL query. If precautions are not taken, the externally-supplied input (usually a GET or POST parameter) can modify the query string such that it performs unintented actions

**3.2 IMPACT THAT CAN EFFECT**

* These vulnerabilities can be exploited by remote attackers to gain unauthorized read or write access to the underlying database
* Exploitation of SQL injection vulnerabilities can also allow for attacks against the logic of the application
* Attackers may be able to obtain unauthorized access to the server hosting the database.

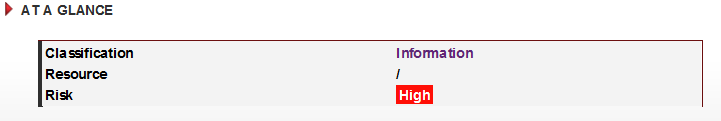
**3.3 WAYS TO COUNTER SQL INJECTION**

* The developer should review the request and response against the code to manually verify whether or not a vulnerability is present
* The best defense against SQL injection vulnerabilities is to use parameterized statements
* Sanitizing input can prevent these vulnerabilities. Variables of string types should be filtered for escape characters, and numeric types should be checked to ensure that they are valid
* Use of stored procedures can simplify complex queries and allow for tighter access control settings
* Configuring database access controls can limit the impact of exploited vulnerabilities. This is a mitigating strategy that can be employed in environments where the code is not modifiable

**4.SESSION COOKIE WITHOUT HTTP ONLY FLAG**



**4.1 SUMMARY OF SESSION COOKIE**



When this flag is not present, it is possible to access the cookie via client-side script code. The HttpOnly flag is a security measure that can help mitigate the risk of cross-site scripting attacks that target session cookies of the victim. If the HttpOnly flag is set and the browser supports this feature, attacker-supplied script code will not be able to access the cookie

**4.2 IMPACT OF EFFECT**

* Information leakage

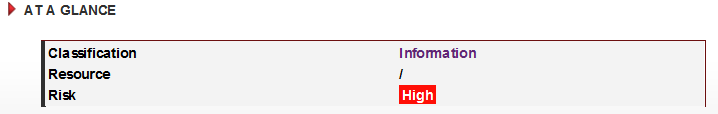
**4.3 WAYS TO PREVENT SESSION COOKIE WITHOUT HTTP ONLY FLAG**

* When creating the cookie in the code, set the HttpOnly flag to true.

**5.SESSION COOKIE WITHOUT SECURE FLAG**



**5.1 SUMMARY OF SESSION COOKIE WITHOUT SECURE FLAG**



This is a high risk vulnerability because attacker can execute ARP poisoning and become man in the middle to eavesdrop any data in the traffic

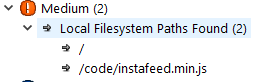
**5.2 IMPACT OF EFFECT**

* Cookies can be exposed to network eavesdroppers
* Session cookies are authentication credentials; attackers who obtain them can get unauthorized access to affected web applications.

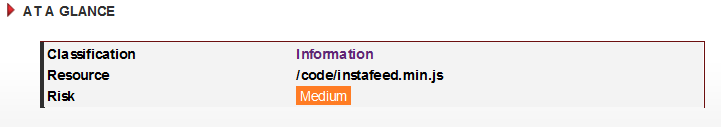
**5.2 WAYS TO COUNTER SESSION COOKIE WITHOUT SECURE FLAG**

* When creating the cookie in the code, set the secure flag to true

**6. LOCAL FILESYSTEM PATH FOUND**



**6.1 SUMMARY OF FILESYSTEM PATH FOUND**



This is categorized in medium risk, this information is sensitive, as it may reveal things about the server environment to an attacker. Knowing filesystem layout can increase the chances of success for blind attacks. Full system paths are very often found in error output.

**6.2 IMPACT THAT CAN EFFECT**

* Disclosure of these paths reveals information about the filesystem layout
* This information can be sensitive, its disclosure can increase the chances of success for other attacks

**6.3 WAYS TO COUNTER FILESYSTEM NOT FOUND**

* Both the system administrators and developers should be made aware, as the problem may be due to an application error or server misconfiguration
* Error output containing sensitive information such as absolute system paths should not be sent to remote clients on production servers
* This output should be sent to another output stream, such as an error log.