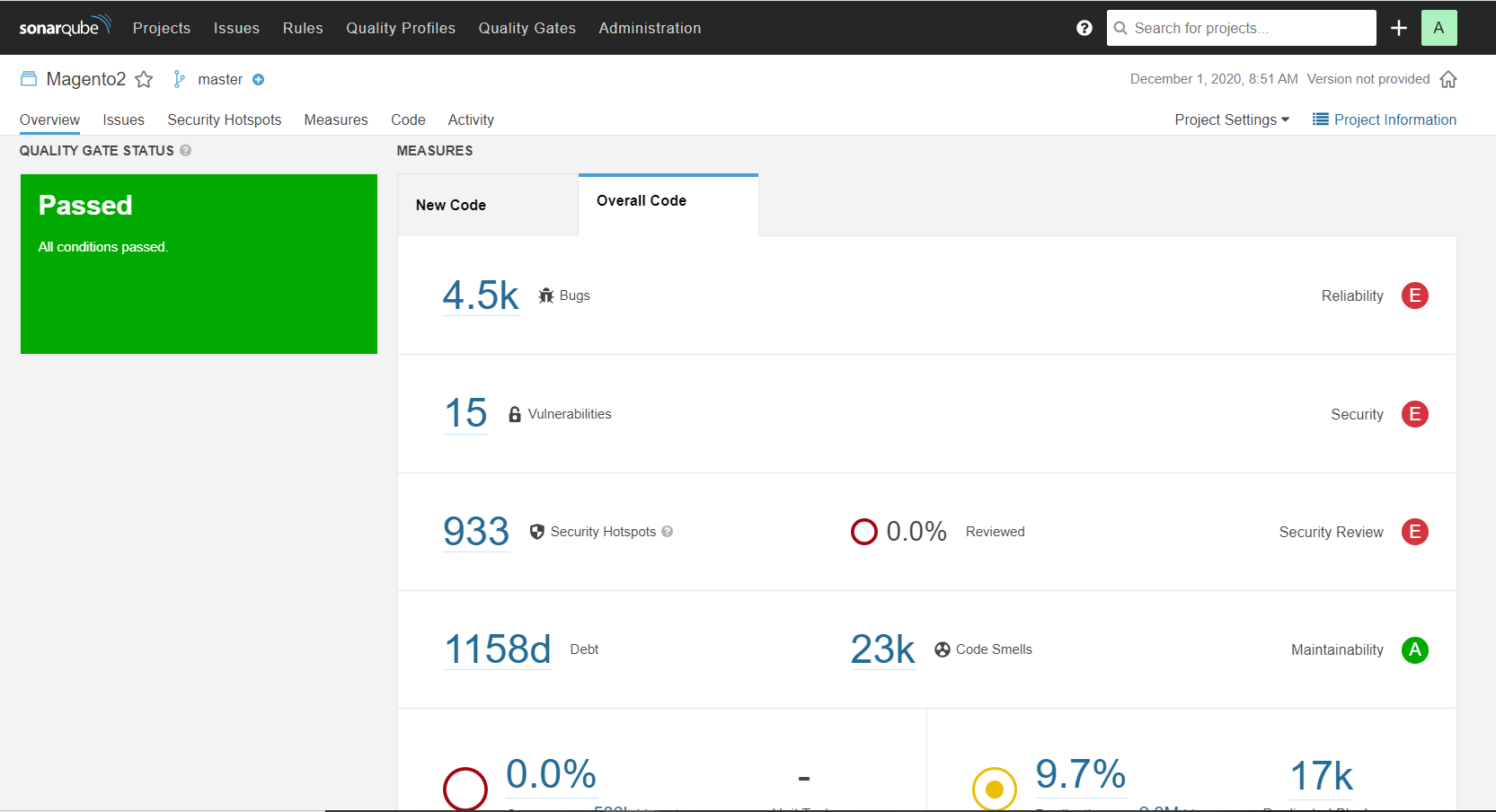
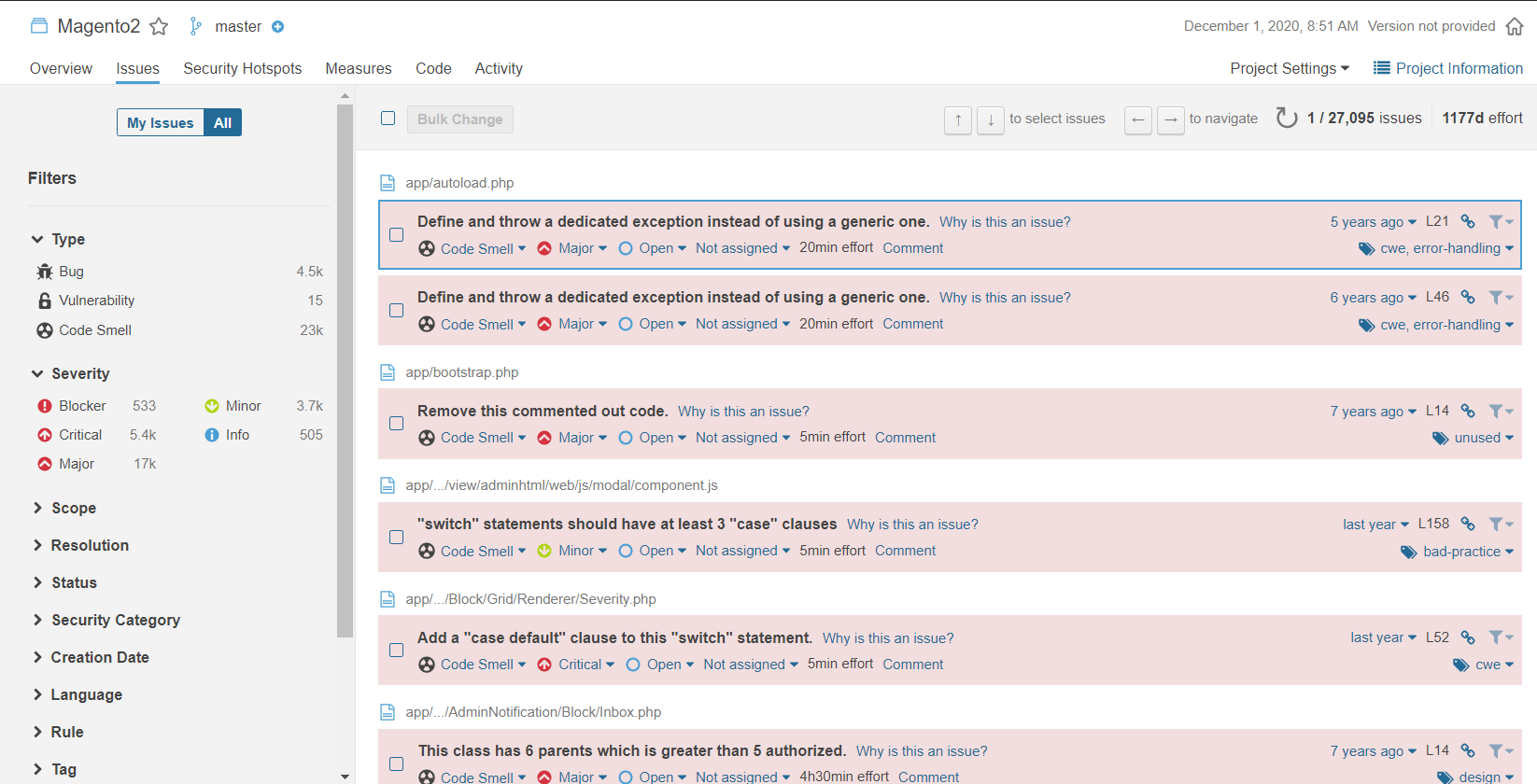
**Automated Code analysis:**

**SonarQube**

Using SonarQube we started Magento code analysis at first. The tool uses a sonar scanner to scan for code issues. The tool was able to find 4.5K bugs and 933 security issues.





It has notified along with the severity of issues such as blocker, critical and major issues. The breakdown of security hotspot issues includes the following:

**High:**

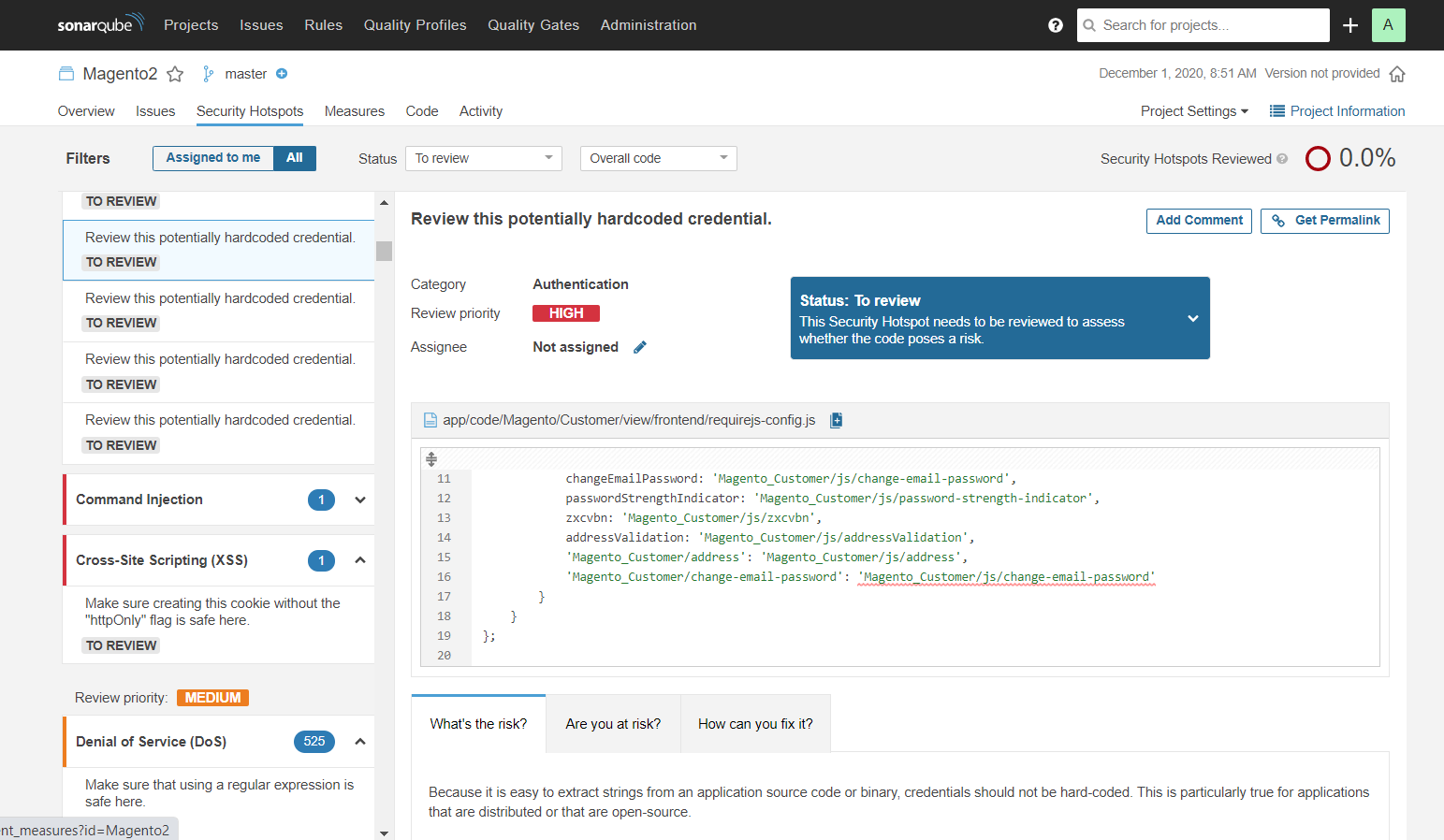
* Authentication = 50
* Command Injection = 1
* Cross-Site Scripting (XSS) = 1043

**Medium:**

* Denial of Service (DoS) = 525
* Code Injection = 31
* Weak Cryptography = 234

**Low:**

* Insecure Configuration = 28
* Log Injection = 6



We first began to analyze these security hotspot issues and documented based on category.

**Issue Analysis:**

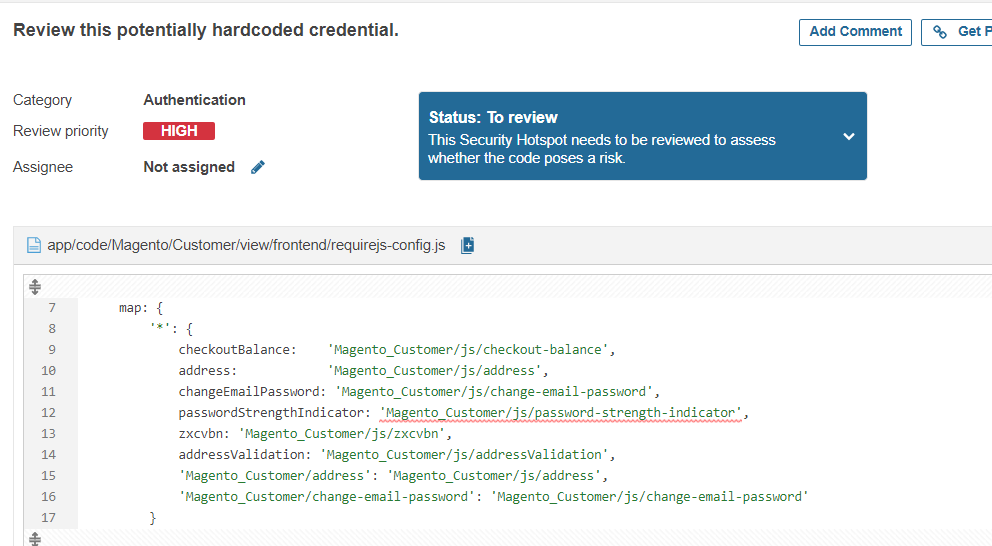
**Authentication:**

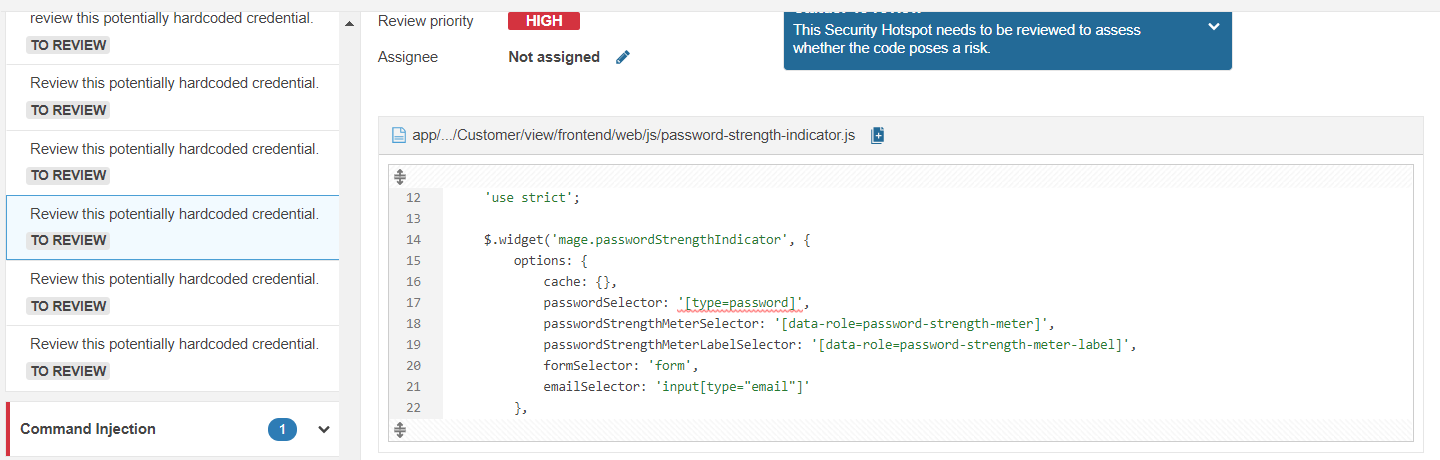
Multiple issues were reported for password authentication, most of the issues look similar, for example, variable name used for password with which it’s easy to identify which is related to password. Tool suggests that credentials should not be hard-coded because it’s easily accessible from application source code or binary. We can find similar suggestions as weaknesses in [CWE-798](http://cwe.mitre.org/data/definitions/798) Use of Hard-coded Credentials and [CWE-259](http://cwe.mitre.org/data/definitions/259) Use of Hard-coded Password. We can find related attack patterns in [CAPEC-191](http://capec.mitre.org/data/definitions/191.html) and [CAPEC-70](http://capec.mitre.org/data/definitions/70.html).

This type of issues has caused the following vulnerabilities:

* [CVE-2019-13466](http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-13466)
* [CVE-2018-15389](http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-15389)

Tool suggests storing the credentials outside of the code in a configuration file, database, or secret management service.





Tool also suggests for sensitive code examples:

$password = "65DBGgwe4uazdWQA"; // Sensitive

$httpUrl = "https://example.domain?user=user&password=65DBGgwe4uazdWQA" // Sensitive

$sshUrl = "ssh://user:65DBGgwe4uazdWQA@example.domain" // Sensitive

We can also check for suggestions from the tool on how to remediate this type of sensitive code.

$user = getUser();

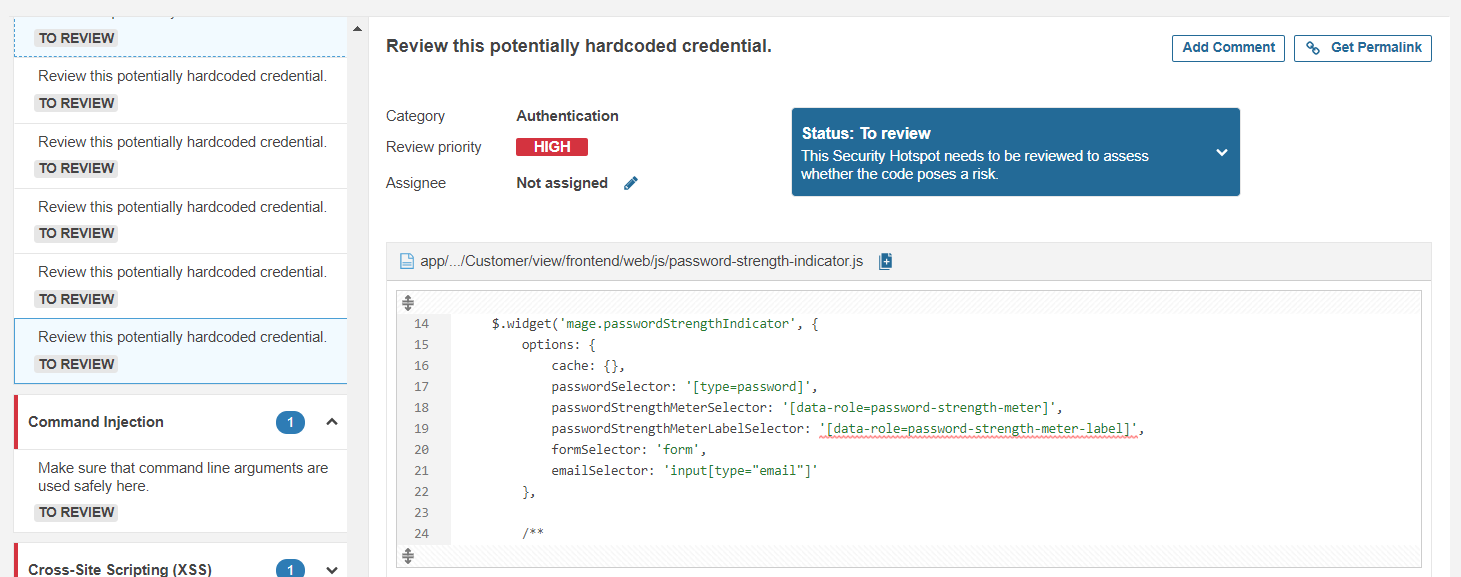
$password = getPassword(); // Compliant

$httpUrl = "https://example.domain?user=$user&password=$password" // Compliant

$sshUrl = "ssh://$user:$password@example.domain" // Compliant

**Command Injection :**

Tool reported some of the issues that could lead to Command Injection as noted below. The suggestions were similar to the Authentication issue. Here also we could see the variable name used for password with which it’s easy to identify that it’s related to password. Tool suggests that credentials should not be hard-coded because it’s easily accessible from application source code or binary. We can find similar suggestions as weaknesses in [CWE-798](http://cwe.mitre.org/data/definitions/798) Use of Hard-coded Credentials and [CWE-259](http://cwe.mitre.org/data/definitions/259) Use of Hard-coded Password.

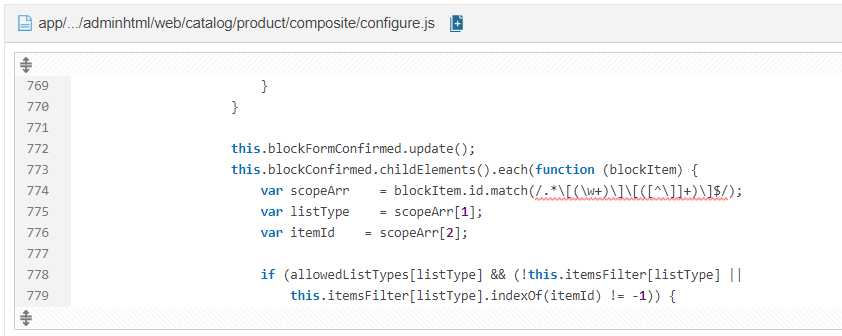


**Denial of service :**

Tool suggests that using regular expressions is security-sensitive. Evaluating regular expressions against input strings is potentially an extremely CPU-intensive task. Specially crafted regular expressions such as (a+)+s will take several seconds to evaluate the input string aaaaaaaaaabs. The problem is that with every additional character added to the input, the time required to evaluate the regex doubles. Evaluating such regular expressions opens the door to [Regular expression Denial of Service (ReDoS)](https://www.owasp.org/index.php/Regular_expression_Denial_of_Service_-_ReDoS) attacks. In the context of a web application, attackers can force the webserver to spend all of its resources evaluating regular expressions thereby making the service inaccessible to genuine users. We can find this type of weakness reported in [CWE-624](https://www.owasp.org/index.php/Regular_expression_Denial_of_Service_-_ReDoS) - Executable Regular Expression Error. However, the equivalent regular expression, a+s (without grouping) is efficiently evaluated in milliseconds and scales linearly with the input size.

This type of issues has caused the following vulnerabilities:

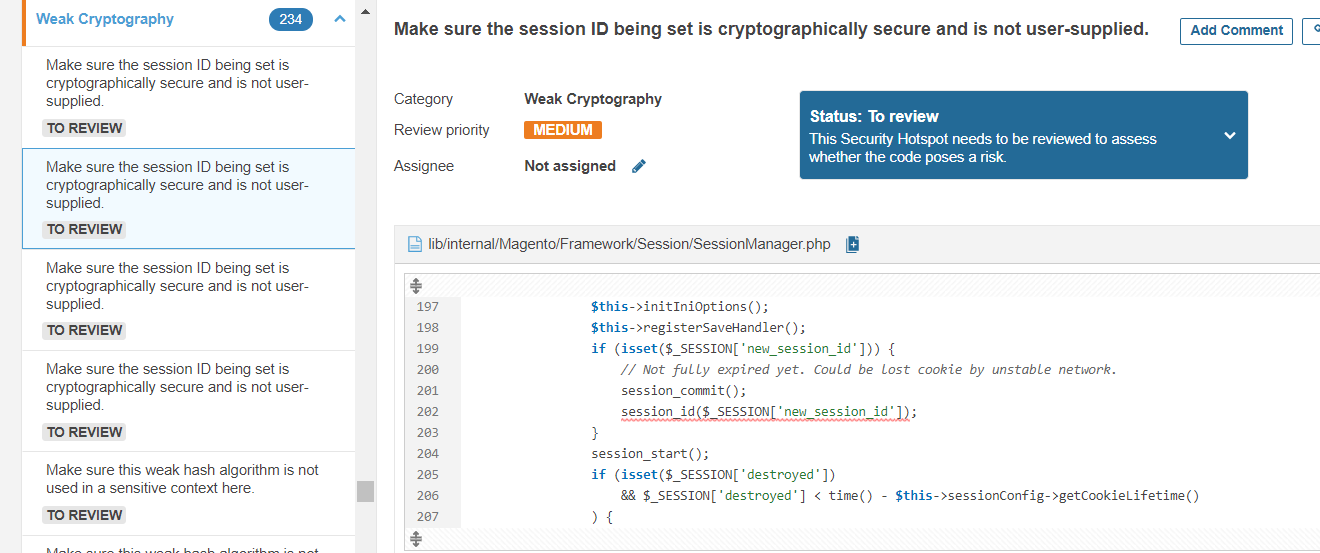
* [CVE-2017-16021](http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-16021)
* [CVE-2018-13863](http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-13863)





**Weak Cryptography :**

The tool suggests that an attacker may be able to hijack another user's session If a session ID can be guessed (not generated with a secure pseudo-random generator, or with insufficient length). If a session id is not unique, set from a user-controlled input, or length is too short then there is a risk and can yield to weakness [CWE-330](https://cwe.mitre.org/data/definitions/330.html) - Use of Insufficiently Random Values.



We could see some of the vulnerabilities reported in <<https://www.cvedetails.com/vendor/15393/Magento.html>>