Using Analyzers to Resolve Security Problems in Web Applications

Using SonarQube

By Kian Sahafi, Fall 2023

What Are Analyzers?

- Security analyzers are tools that are specifically designed to analyze and identify potential security vulnerabilities in software and systems. They can be used to scan and identify potential vulnerabilities, such as **SQL injection**, **corss-site scripting** (XSS), **and insecure file uploads**, in web applications, network infrastructure, and other systems.
- Security analyzers can be divided into two main categories:
    1. Static Analyzers
    2. Dynamic Analyzers

Static Analyzers

- Static Analyzers are tools that analyze the source code of a program without running it.
- Static Analyzers an include code scanners, which can identify vulnerabilities in the source cocde of a program, and configuration analyzers, which can identify vulnerabilities in the configuration of the system.

Dynamic Analyzers

- Dynamic Analyzers are tools that analyze the behavior of a program while it is running.
- Dynamic analyzers can include penetration testing tools, which simulte an attack on a system to identify vulnerabilities, and intrusion detection systems, which monitor a system for signs of an attack.

OK, Then what is SonarQube?

- SonarQube is an open-source platform for static code analysis. It allows developers to identify and fix quality and security issues in their code before it's deployed. The platform uses a set of analyzers to check the code for issues such as bugs, vulnerabilities, and code smells. These analyzers can be configured to check for specific issues and can be integrated with a wide range of programming languages, such as Java, c#, JavaScript, Python, and many others.

- SonarQube provides a web-based interface that allows devleoprs to view the results of the code analysis, including detailed information about the issues found and suggested fixed. It also includes features such as reporting and metrics, which allow developers to **track the quality and security** of their code over time. It also provides a way to manage technical debt, by providing an overview of the codebase and the issues found.

Continue…

- SonarQube's platform is build on a modular architecture, which allows developers to **add** new *analyzers*, *rules*, and *plugins* as needed. This flexibility makes it easy to integrate SonarQube into existing development workflows and to customize it to fit specific needs.
- Ok let’s talk about target market:
- SonarQube is widely used by ***developers***, ***IT administrators***, ***security analysts*** and ***data scientists***. It’s a valuable tool for understanding and troubleshooting complex systems, monitoring and improving the performance of applications, and identifying security threats, it also helps to maintain a high-quality code base by providing a way to measure code complexity, maintainability, and test coverage.

Great… but what is it looking for?

Here is What to expect from security-related rules:

- Well, under the hood, SonarQube is based of different representations of the source code and technologies in order to be able to detect any kind of security issue:

- **Security-injection rules**: there is a vulnerability here when the inputs handled by your application are controlled by a user (potentially an attacker) and *not validated* or *sanitized*. When this occurs, the flow from sources (user-controlled inputs) to sinks (sensitive functions) will be presented. To do this, SonarQube uses well-known taint analysis technology on source code which allows, for example, the detection of:

  1. CWE-89[1]: SQL Injection
  2. SWE-79[2]: Cross-site Scripting
  3. CWE-94[3]: Code Injection

1.https://cwe.mitre.org/data/definitions/89.html
2.https://cwe.mitre.org/data/definitions/79.html
3.https://cwe.mitre.org/data/definitions/94.html

Continue…

- **Security-configuration rules**: here there is a security issue because when calling a sensitive function, *the wrong parameter* (for example invalid cryptographic algorithm or TLS version) *has been set or when a check* (for example, a check_permissions() kind of function) *was not done or not in the correct order, this problem is likely to appear often when the program is executed* (no injected/complex attacks are required unlike the previous category):

  1. CWE-1004[1]: Sensitive Cookie Without ‘HttpOnly' Flag
  2. CWE-297[2]: Improper Validation of Certificate with Host Mismatch
  3. CWE-327[3]: Use of a Broken or Risky Cryptographic Algorithm

1.https://cwe.mitre.org/data/definitions/1004.html
2.https://cwe.mitre.org/data/definitions/297.html
3.https://cwe.mitre.org/data/definitions/327.html

Which security-standards are covered?

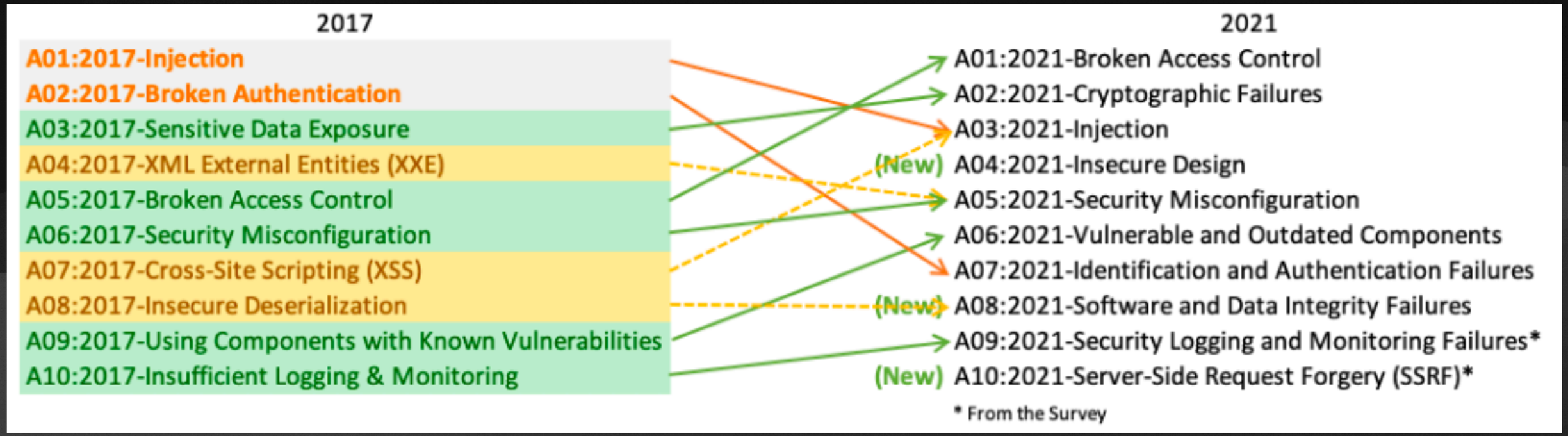
- Their security rules are classified according to well-established security standards such as:
  - CWE
  - OWASP Top 10
  - SANS Top 25 - outdated

CWE

Common Weakness Enumeration
A Community-Developed List of Software & Hardware Weakness Types

Rank	ID	Name	Score	KEV Count (CVEs)	Rank Change vs. 2021
1	CWE-787	Out-of-bounds Write	64.20	62	0
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.97	2	0
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	22.11	7	+3 ▲
4	CWE-20	Improper Input Validation	20.63	20	0
5	CWE-125	Out-of-bounds Read	17.67	1	-2 ▼
6	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	17.53	32	-1 ▼
7	CWE-416	Use After Free	15.50	28	0
8	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.08	19	0
9	CWE-352	Cross-Site Request Forgery (CSRF)	11.53	1	0
10	CWE-434	Unrestricted Upload of File with Dangerous Type	9.56	6	0
11	CWE-476	NULL Pointer Dereference	7.15	0	+4 ▲
12	CWE-502	Deserialization of Untrusted Data	6.68	7	+1 ▲
13	CWE-190	Integer Overflow or Wraparound	6.53	2	-1 ▼
14	CWE-287	Improper Authentication	6.35	4	0
15	CWE-798	Use of Hard-coded Credentials	5.66	0	+1 ▲
16	CWE-862	Missing Authorization	5.53	1	+2 ▲
17	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	5.42	5	+8 ▲
18	CWE-306	Missing Authentication for Critical Function	5.15	6	-7 ▼
19	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	4.85	6	-2 ▼
20	CWE-276	Incorrect Default Permissions	4.84	0	-1 ▼
21	CWE-918	Server-Side Request Forgery (SSRF)	4.27	8	+3 ▲
22	CWE-362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	3.57	6	+11 ▲
23	CWE-400	Uncontrolled Resource Consumption	3.56	2	+4 ▲
24	CWE-611	Improper Restriction of XML External Entity Reference	3.38	0	-1 ▼
25	CWE-94	Improper Control of Generation of Code ('Code Injection')	3.32	4	+3 ▲

OWASP Top 10:


2017
A01:2017-Injection
A02:2017-Broken Authentication
A03:2017-Sensitive Data Exposure
A04:2017-XML External Entities (XXE)
A05:2017-Broken Access Control
A06:2017-Security Misconfiguration
A07:2017-Cross-Site Scripting (XSS)
A08:2017-Insecure Deserialization
A09:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging & Monitoring
2021
A01:2021-Broken Access Control
A02:2021-Cryptographic Failures
A03:2021-Injection
(New) A04:2021-Insecure Design
A05:2021-Security Misconfiguration
A06:2021-Vulnerable and Outdated Components
A07:2021-Identification and Authentication Failures
(New) A08:2021-Software and Data Integrity Failures
A09:2021-Security Logging and Monitoring Failures*
(New) A10:2021-Server-Side Request Forgery (SSRF)*
* From the Survey

And finally, SANS Top 25:

Rank	ID	Name
1	CWE-787	Out-of-bounds Write
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
4	CWE-20	Improper Input Validation
5	CWE-125	Out-of-bounds Read
6	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
7	CWE-416	Use After Free
8	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
9	CWE-352	Cross-Site Request Forgery (CSRF)
10	CWE-434	Unrestricted Upload of File with Dangerous Type
11	CWE-476	NULL Pointer Dereference
12	CWE-502	Deserialization of Untrusted Data
13	CWE-190	Integer Overflow or Wraparound
14	CWE-287	Improper Authentication
15	CWE-798	Use of Hard-coded Credentials
16	CWE-862	Missing Authorization
17	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')
18	CWE-306	Missing Authentication for Critical Function
19	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer
20	CWE-276	Incorrect Default Permissions
21	CWE-918	Server-Side Request Forgery (SSRF)
22	CWE-362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')
23	CWE-400	Uncontrolled Resource Consumption
24	CWE-611	Improper Restriction of XML External Entity Reference
25	CWE-94	Improper Control of Generation of Code ('Code Injection')

Now what problem are they trying to fix?

The likes of SonarQube are designed to solve a number of problems in the software development process, particularly in regards to code quality and security. These problems include:

1. Poor Code Quality: Analyzers help to identify and address issues with code quality, such as bugs, code smells, and poor maintainability. This can improve the *reliability*, *performance*, and *readability of the code*.

2. Incomplete Testing: Analyzers can help to identify areas of code that are *not covered by test*, making it easier to write tests that fully exercise the code.

3. Security Vulnerabilities: Analyzers can help to *identify and remediate security vulnerabilities in code*, such as SQL injection, cross-site scripting(XSS), and insecure file uploads.

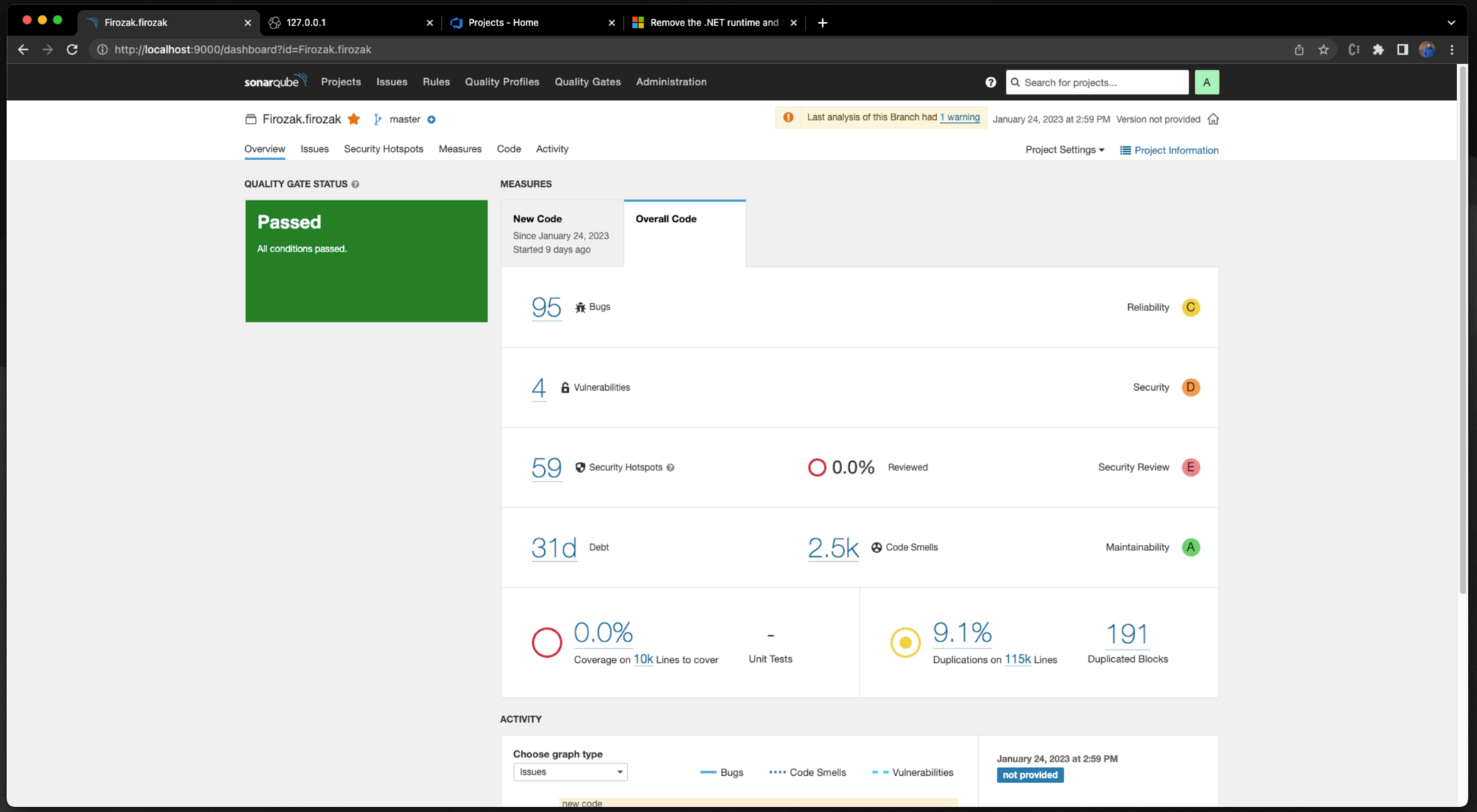
4. Technical Debt: Analyzers provide a way to *track and manage technical debt*, which is the cost of maintaining a codebase over time.

5. Compliance: Analyzers can help organizations to meet regulatory requirements and comply with industry standards.(like OWASM, PCI-DSS, and HIPAA)

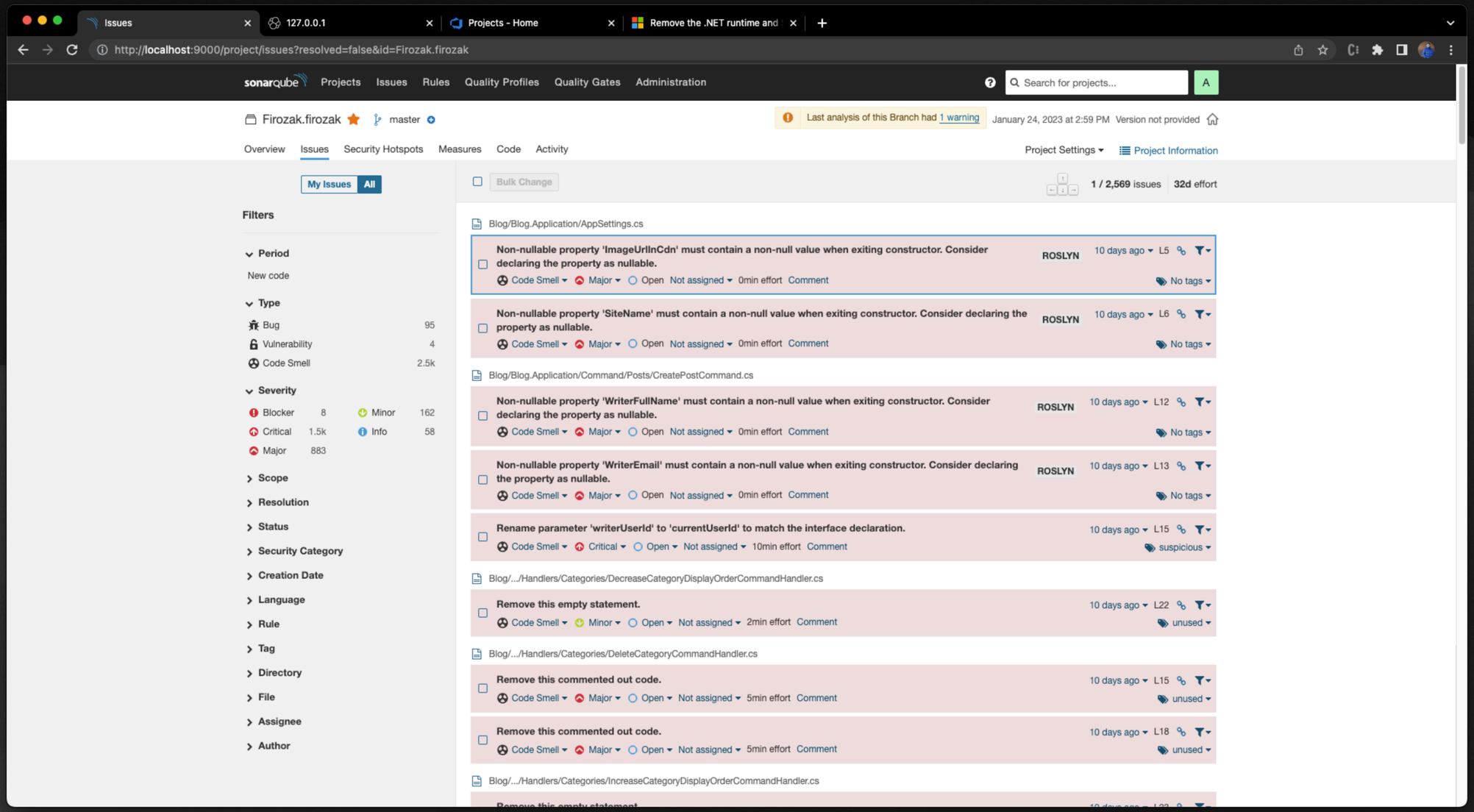
6. Improved Collaboration: Analyzers provide a centralized platform for *code analysis* and *reporting*, making it easier for developers, IT administrators, security analysts, and other stakeholder to collaborate and work together.

Now, What did WE do?

Well the First Report Was Not Great!



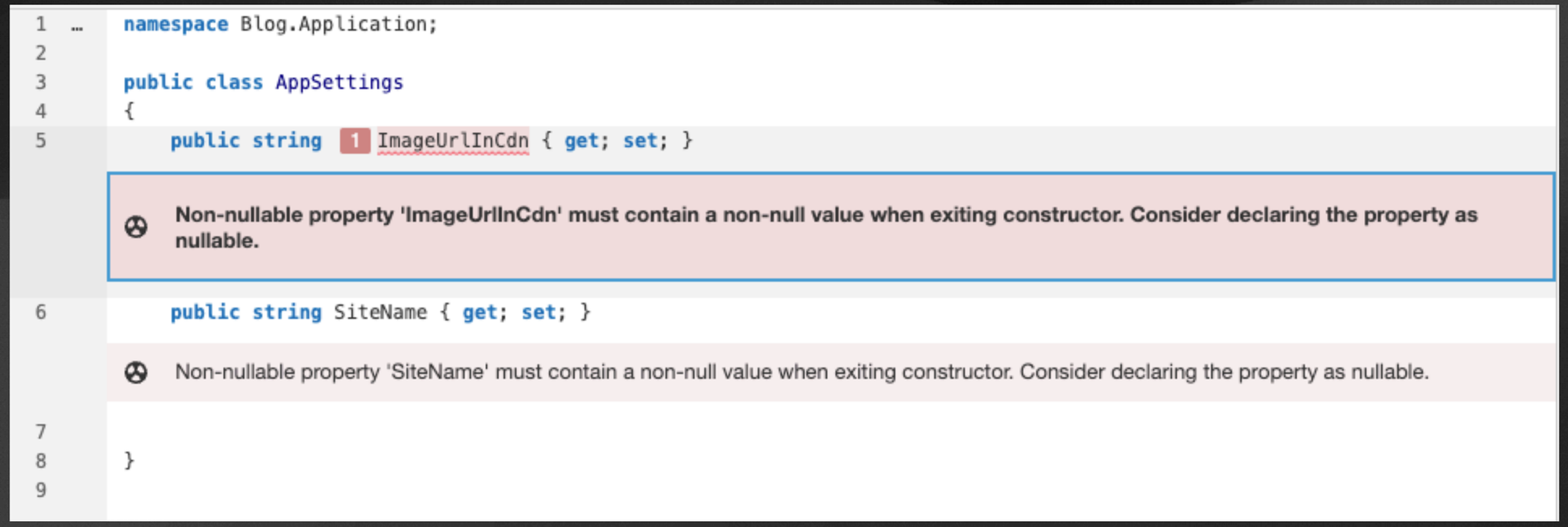
And the bugs Issues Were Like this:



Categorizing the issues:

- I don’t want to bore you with every kind of our problems but the problems were mainly about:

  1. Non-nullable properties in classes without constructors.



2. not matching with the Interface declarations:

```
{
    public Guid CurrentUserId { get; private set; }
    public string WriterFullName { get; private set; }
```

Non-nullable property 'WriterFullName' must contain a non-null value when exiting constructor. Consider declaring the property as nullable.

```
    public Email WriterEmail { get; private set; }
```

Non-nullable property 'WriterEmail' must contain a non-null value when exiting constructor. Consider declaring the property as nullable.

```

    public void SetUser(Guid writerUserId)
```

Rename parameter 'writerUserId' to 'currentUserId' to match the interface declaration.

```
        => CurrentUserId = (writerUserId == Guid.Empty) ?
            throw new DomainException("شناسه مورد نظر صحیح نیست") :
            writerUserId;
    public void SetWriterFullNameANdEmail(string writerFullName, Email email)
    {
        WriterFullName = (string.IsNullOrWhiteSpace(writerFullName)) ?
            throw new DomainException("نام نویسنده مشخص نیست") :
            writerFullName;
        WriterEmail = email;
```

```
partial class Point
{
  partial void MoveVertically(int z);
}

partial class Point
{
  int x = 0;
  int y = 0;
  int z = 0;

  partial void MoveVertically(int y)  // Noncompliant
  {
    this.y = y;
  }
}
```

```
interface IFoo
{
  void Bar(int i);
}

class Foo : IFoo
{
  void Bar(int z) // Noncompliant, parameter name should be i
  {
  }
}
```

3. And finally commented out codes:

```
{
    private readonly IPostCategoryRepository _postCategoryRepository;
    //private readonly IAccountManager _accountManager;
```

Remove this commented out code.

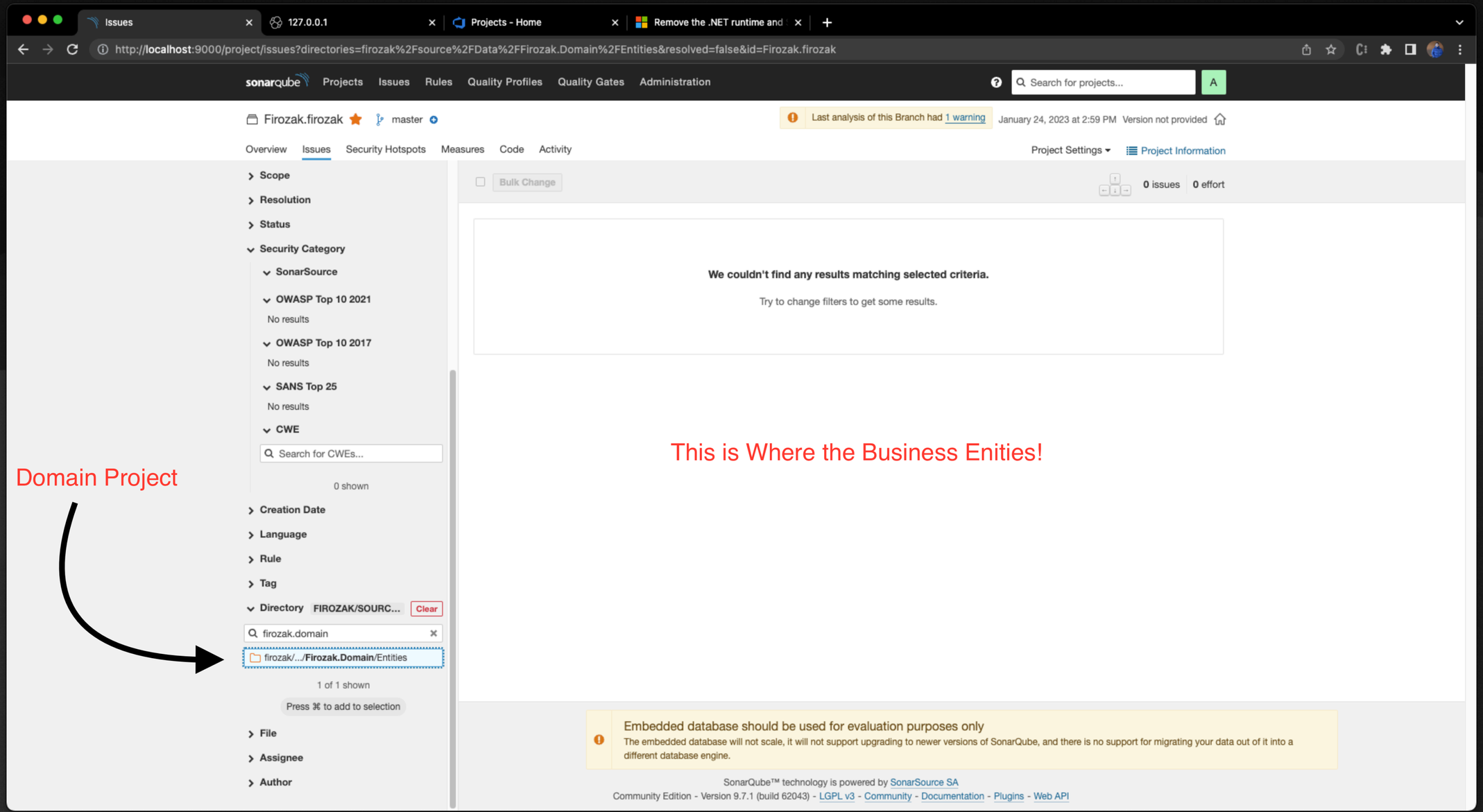
```
    public DeleteCategoryCommandHandler(ILogger<DeleteCategoryCommandHandler> logger, IPostCategoryRepository
postCategoryRepository)
        => _postCategoryRepository = postCategoryRepository ??
                                     throw new ArgumentNullException(nameof(postCategoryRepository));
//_accountManager = accountManager ?? throw new ArgumentNullException(nameof(accountManager));
```

Remove this commented out code.

Programmers should not comment out code as it bloats programs and reduces readability.

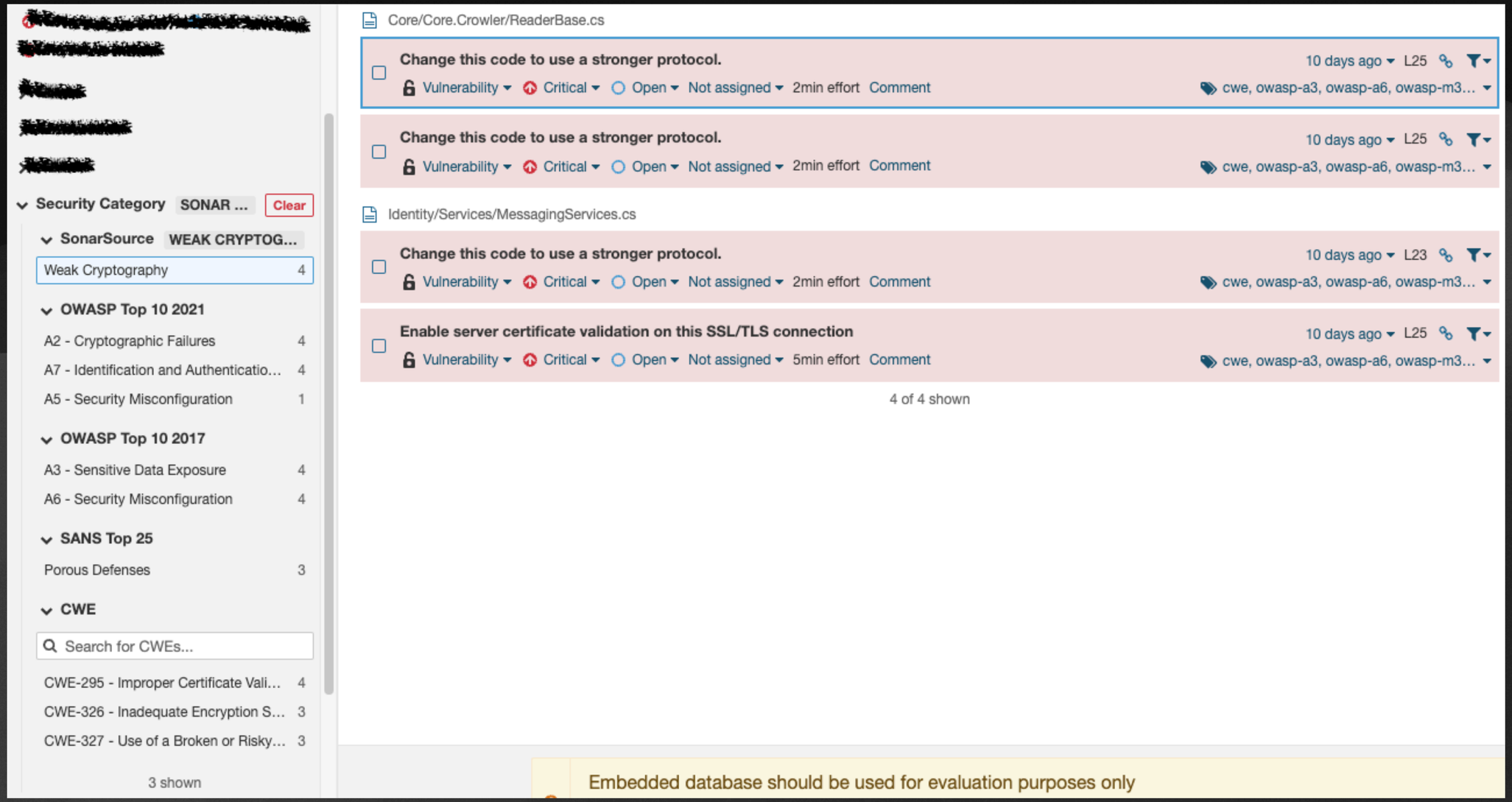
Unused code should be deleted and can be retrieved from source control history if required.

The most important part of our project was the Domain Project which an issue here meant that either we did not understand the business very well or there was an issue with the implementation of the entities in which both can lead to some unhandled errors.
But as you can see, there was not any issues in that Project:

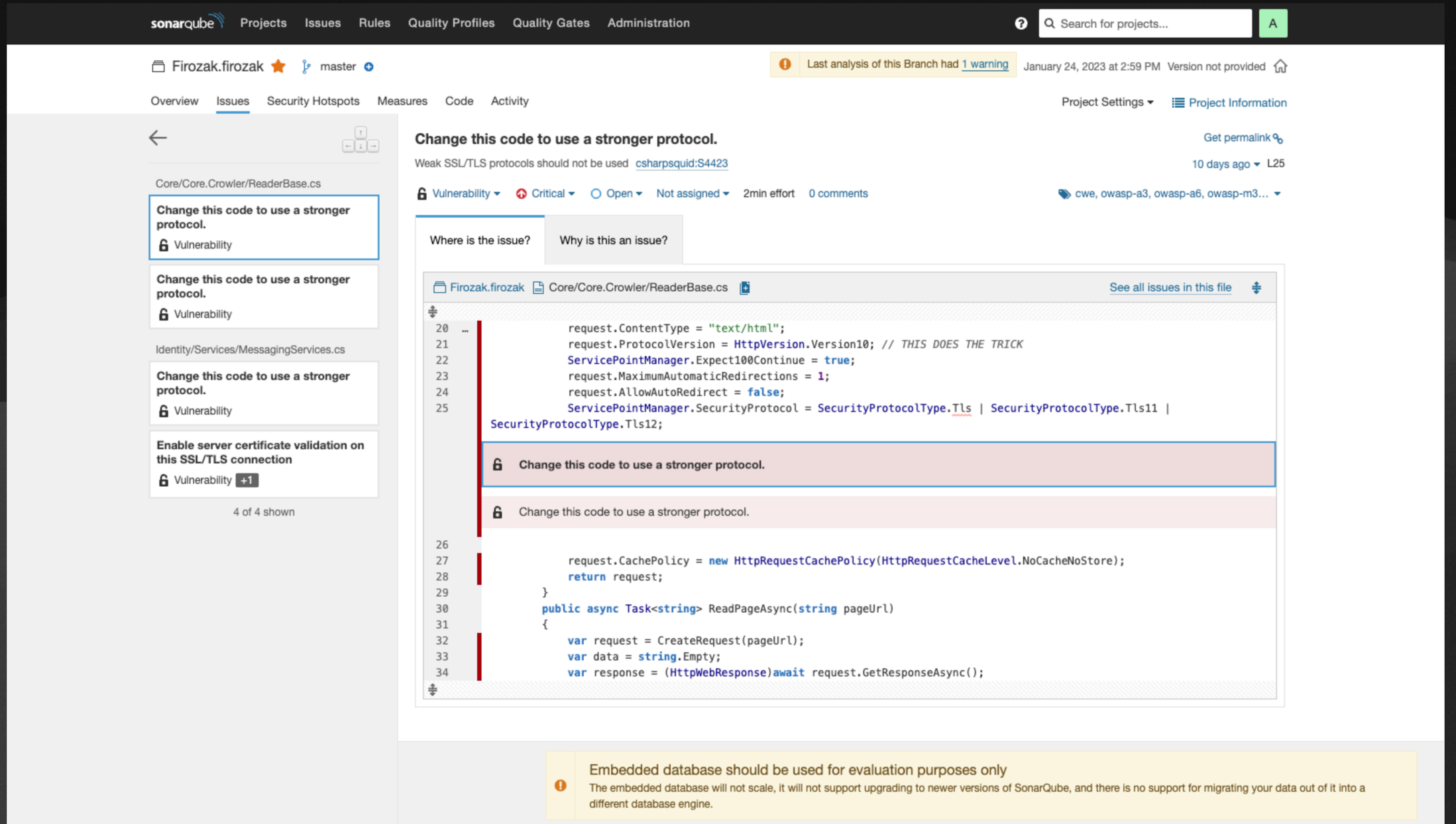


Ok, What About the Three standards that we saw in the previous pages?

- CWE
- OWASP Top 10
- SANS Top 25 - outdated

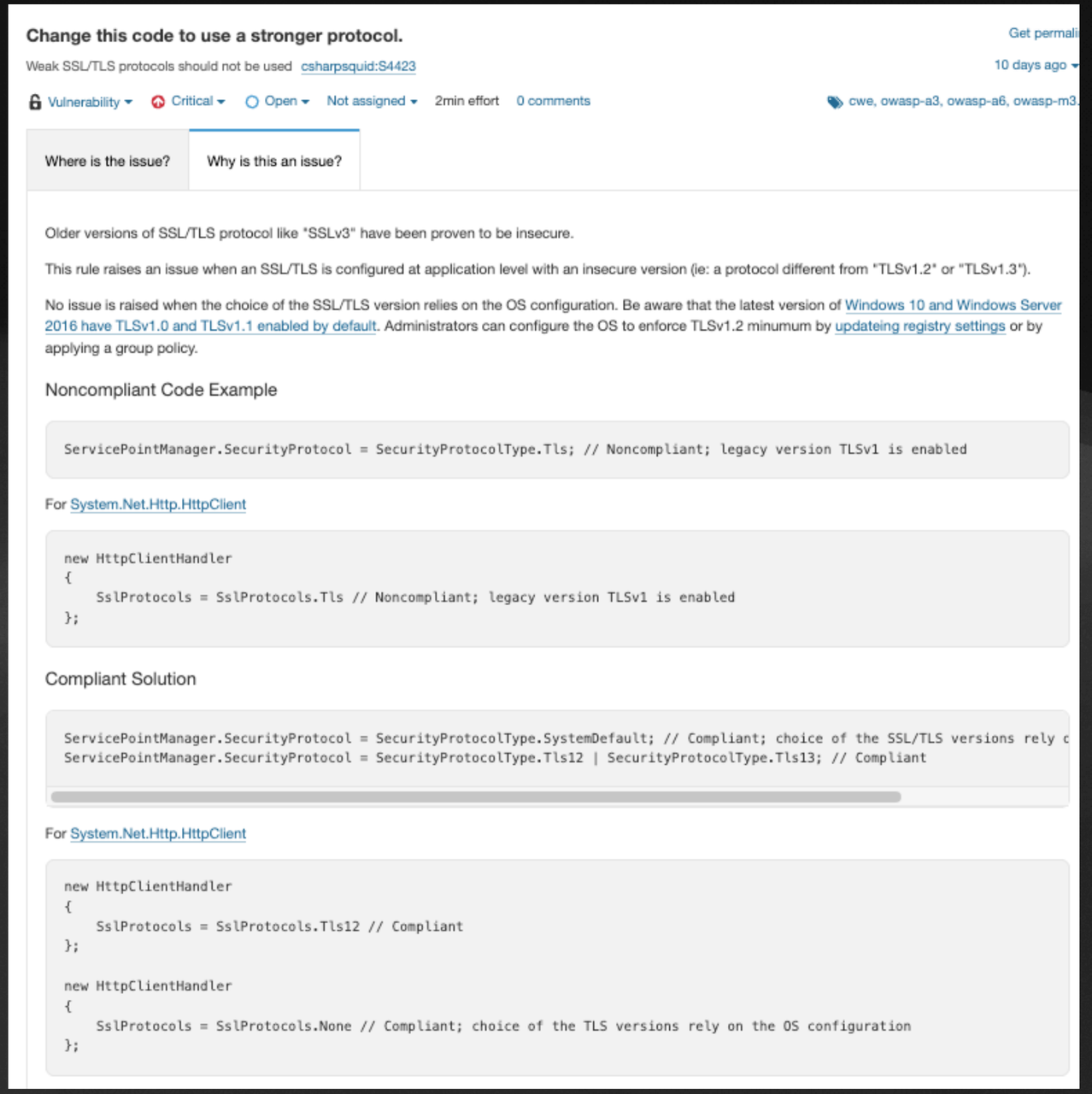


The Issue:


sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Administration
Search for projects...
A
Firozak.firozak
master
Last analysis of this Branch had 1 warning
January 24, 2023 at 2:59 PM Version not provided
Overview
Issues
Security Hotspots
Measures
Code
Activity
Project Settings
Project Information
Core/Core.Crowler/ReaderBase.cs
Change this code to use a stronger protocol.
Vulnerability
Change this code to use a stronger protocol.
Vulnerability
Identity/Services/MessagingServices.cs
Change this code to use a stronger protocol.
Vulnerability
Enable server certificate validation on this SSL/TLS connection
Vulnerability +1
4 of 4 shown
Change this code to use a stronger protocol.
Weak SSL/TLS protocols should not be used csharpsquid:S4423
Get permalink
10 days ago L25
Vulnerability
Critical
Open
Not assigned
2min effort
0 comments
cwe, owasp-a3, owasp-a6, owasp-m3...
Where is the issue?
Why is this an issue?
Firozak.firozak
Core/Core.Crowler/ReaderBase.cs
See all issues in this file
20 …
request.ContentType = "text/html";
21
request.ProtocolVersion = HttpVersion.Version10; // THIS DOES THE TRICK
22
ServicePointManager.Expect100Continue = true;
23
request.MaximumAutomaticRedirections = 1;
24
request.AllowAutoRedirect = false;
25
ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls | SecurityProtocolType.Tls11 | SecurityProtocolType.Tls12;
Change this code to use a stronger protocol.
Change this code to use a stronger protocol.
26
27
request.CachePolicy = new HttpRequestCachePolicy(HttpRequestCacheLevel.NoCacheNoStore);
28
return request;
29
}
30
public async Task<string> ReadPageAsync(string pageUrl)
31
{
32
var request = CreateRequest(pageUrl);
33
var data = string.Empty;
34
var response = (HttpWebResponse)await request.GetResponseAsync();
Embedded database should be used for evaluation purposes only
The embedded database will not scale, it will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data out of it into a different database engine.

"Why is this an issue" Part:

The interesting is that there is a full explanation area that you can read for why is this an issue and find out how to fix the problem.



Change this code to use a stronger protocol.

Get permali

Weak SSL/TLS protocols should not be used csharpsquid:S4423

10 days ago

Vulnerability Critical Open Not assigned 2min effort 0 comments

cwe, owasp-a3, owasp-a6, owasp-m3

Where is the issue? | Why is this an issue?

Older versions of SSL/TLS protocol like "SSLv3" have been proven to be insecure.

This rule raises an issue when an SSL/TLS is configured at application level with an insecure version (ie: a protocol different from "TLSv1.2" or "TLSv1.3").

No issue is raised when the choice of the SSL/TLS version relies on the OS configuration. Be aware that the latest version of Windows 10 and Windows Server 2016 have TLSv1.0 and TLSv1.1 enabled by default. Administrators can configure the OS to enforce TLSv1.2 minumum by updateing registry settings or by applying a group policy.

Noncompliant Code Example

```
ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls; // Noncompliant; legacy version TLSv1 is enabled
```

For System.Net.Http.HttpClient

```
new HttpClientHandler
{
    SslProtocols = SslProtocols.Tls // Noncompliant; legacy version TLSv1 is enabled
};
```

Compliant Solution

```
ServicePointManager.SecurityProtocol = SecurityProtocolType.SystemDefault; // Compliant; choice of the SSL/TLS versions rely c
ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12 | SecurityProtocolType.Tls13; // Compliant
```

For System.Net.Http.HttpClient

```
new HttpClientHandler
{
    SslProtocols = SslProtocols.Tls12 // Compliant
};

new HttpClientHandler
{
    SslProtocols = SslProtocols.None // Compliant; choice of the TLS versions rely on the OS configuration
};
```

Security Hotspots

First a small explanation about what are security hotspots:

- A security hotspot highlights a *security-sensitive piece of code that the developer needs to review*. Upon review, you'll either find that there is no threat or you need to apply a fix to secure the code.
- Another way of looking at hotspots can be the concept of *Defense in depth*(Computing), in which several redundant protection layers are placed in an application so that it becomes more resilient in the event of an attack.

Vulnerability or hotspot?

The mian difference between a hotspot and a vulnerability is *the need for review before deciding whether to apply a fix*:

- With a hotspot, a *security-sensitive piece of code is highlighted*, but the overall application security may not be impacted. it's up to the developer to review the code to determine whether or not a fix is needed to secure the code.
- With a vulnerability, *a problem that impacts the application's security has been discovered and needs to be fixed immediately*.

An example of a hotspot is the RSPEC-2092 where the use of a cookie secure flag is recommended to prevent cookies from being sent over non-HTTPS connections.

A review is needed in this example because:

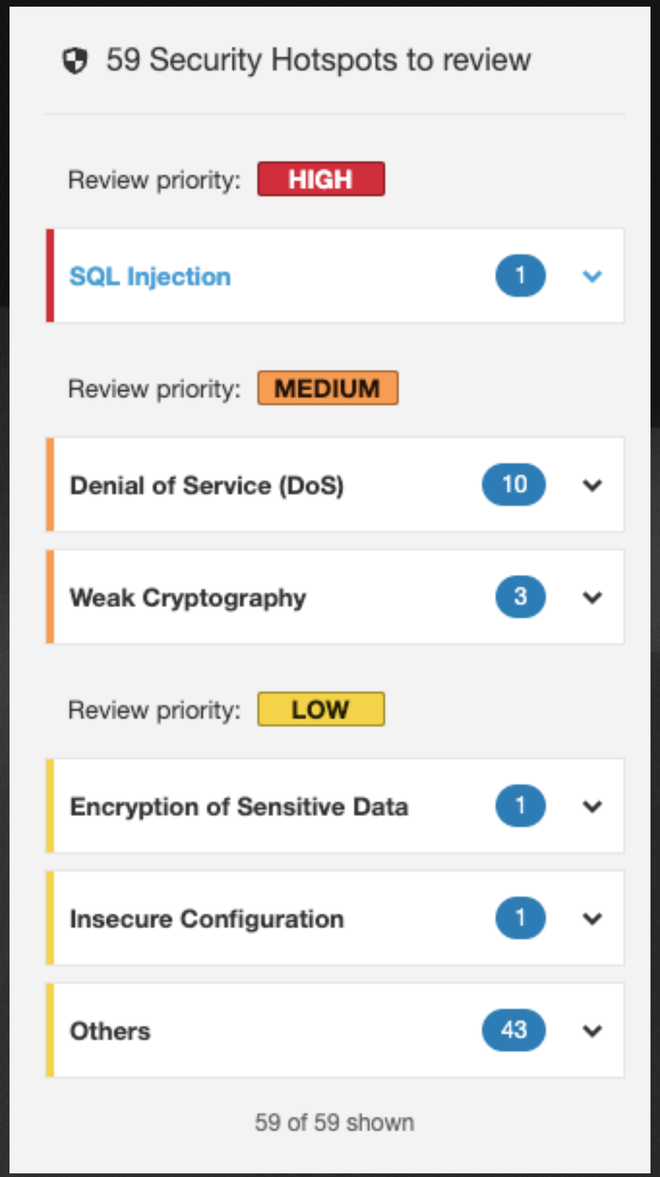
- *HTTPS is the main protection against MITM attacks and so the secure flag is only additional protection in case of some failures of network security.*
- *The cookie may be designed to be sent everywhere (non-HTTPS websites included) because it's a tracking cookie or similar.*

With hotspots, we try to give some freedom to users and educate them on how to choose the most relevant/appropriate protections depending on the context (for example, budgets and threats).

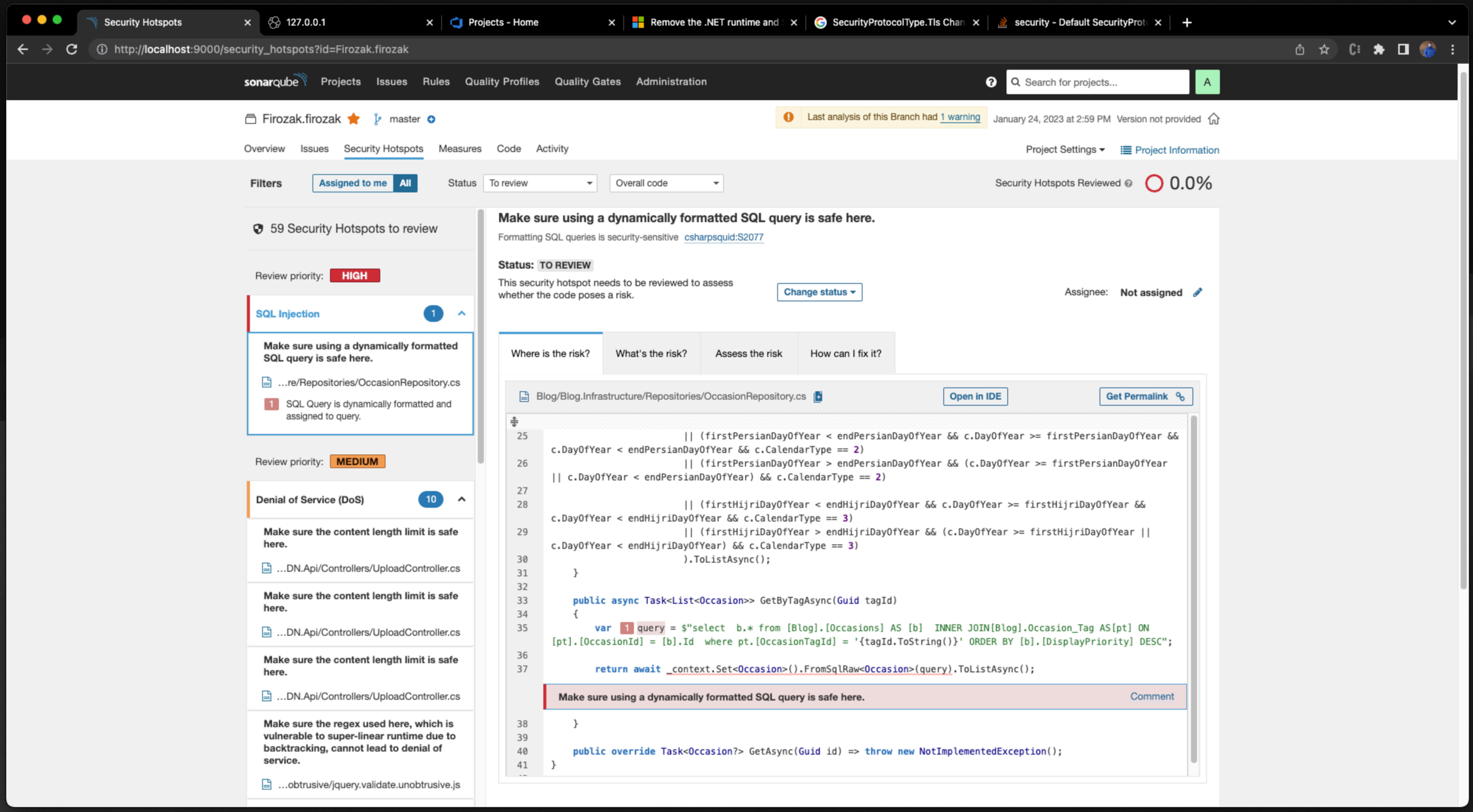
And the most important reason for why they are important are:

Understand the risk, *Identify Protections*, *Identify impacts*.

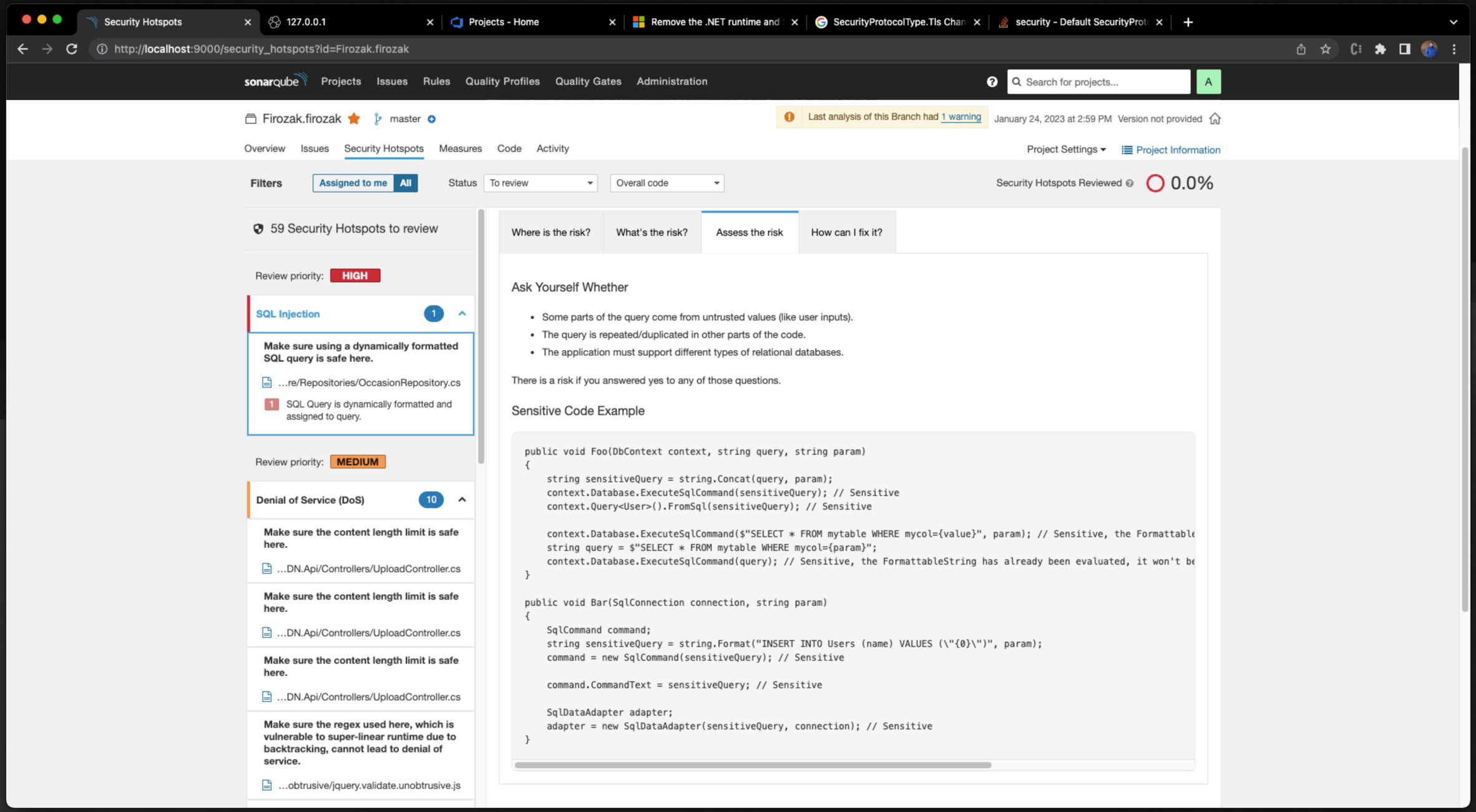
let’s see how many security hotspots we had on our project:



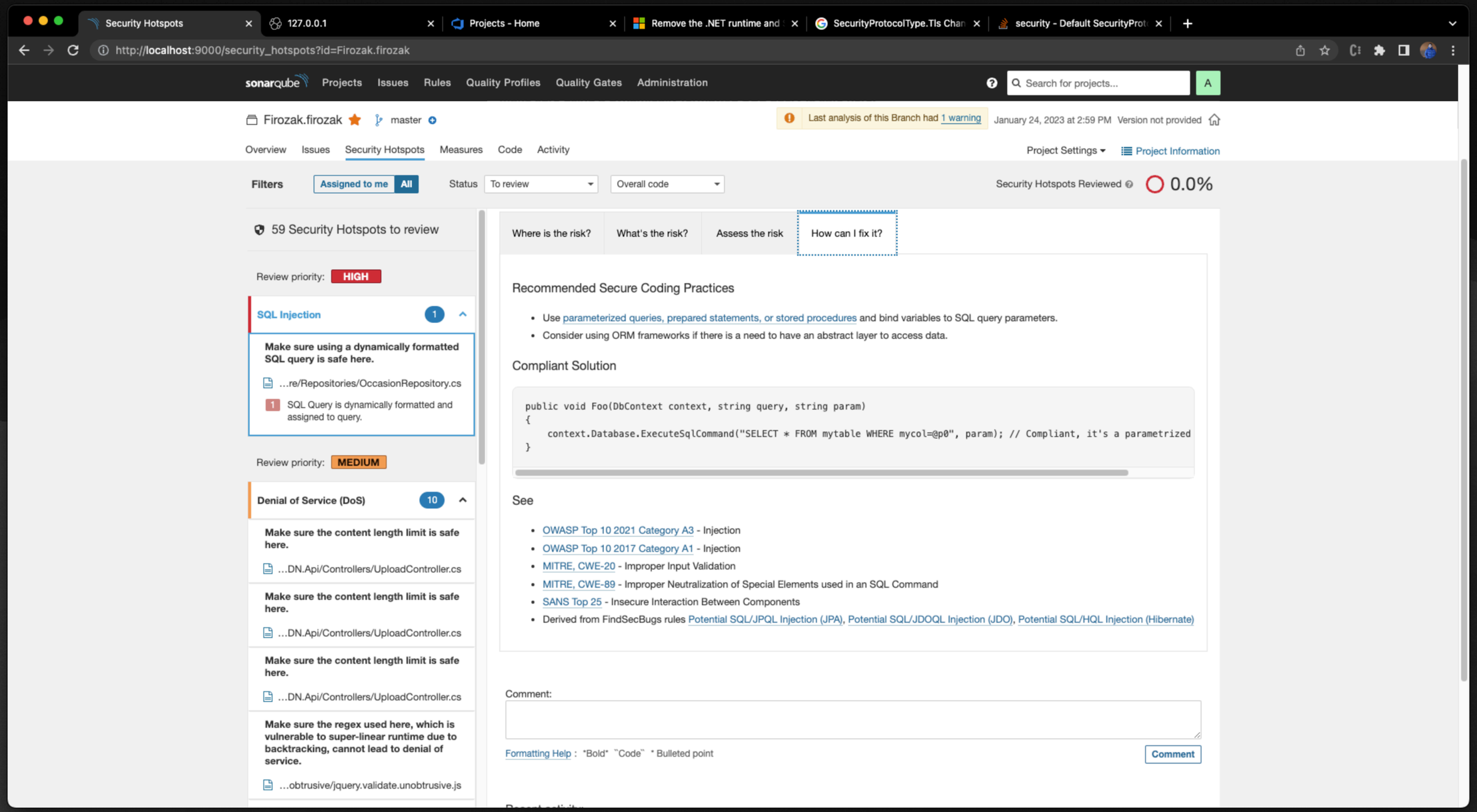
let’s see one of them:



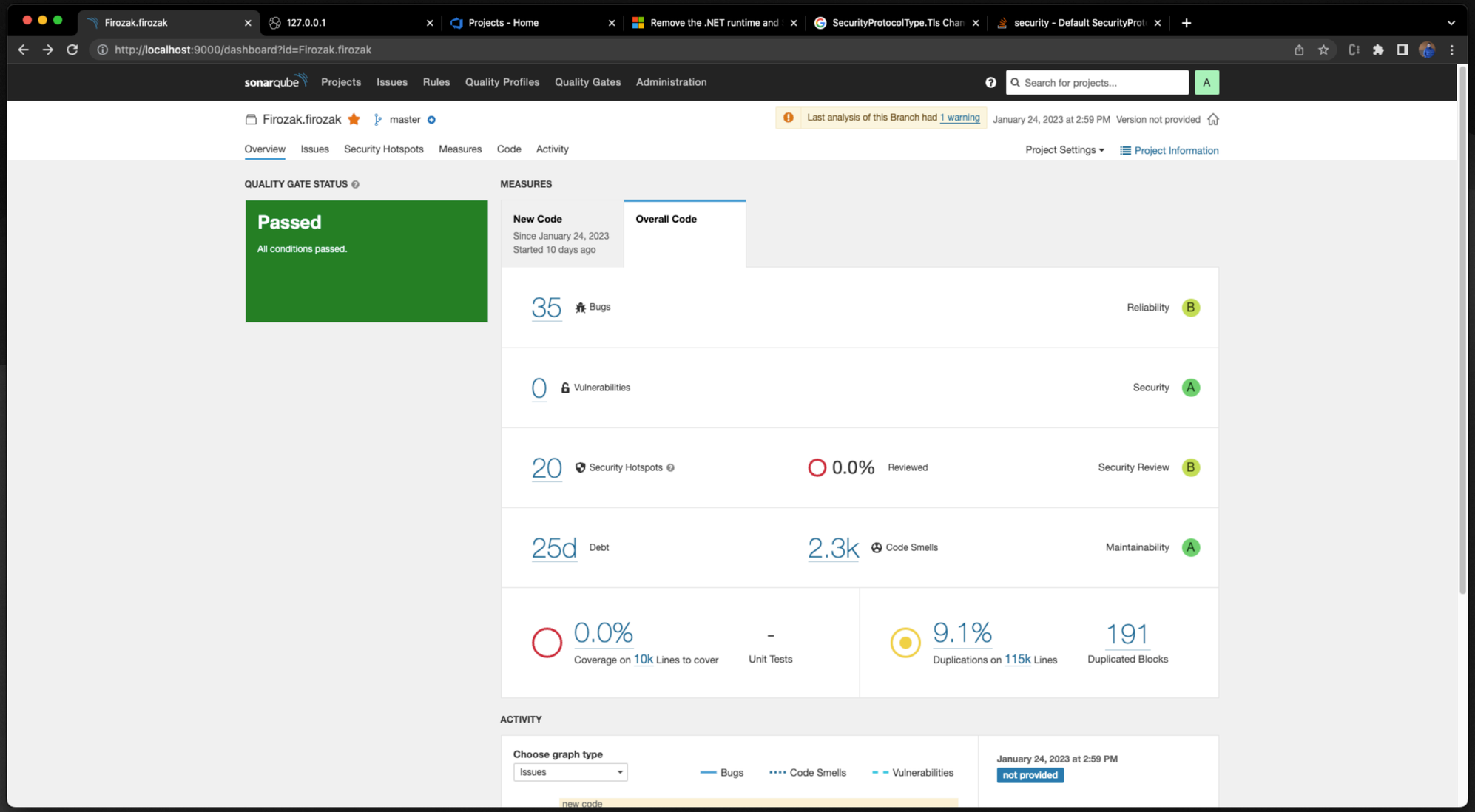
Assess the risk:



How can I fix it?



And finally the results after fixing the Issues:



What we did:

- We reduced Bugs From 95 to 35. (And the other 35 we could not resolve because they raised some other issues in the application.)
- We fixed all 4 Vulnerabilities.
- We fixed 39 Security Hotspots. (from 59 to 20)
- And reduced the Code Smells From 2.5k to 2.3k.

Thank you for listening…

If you have any questions now is the time.😉

Presented By Kian Sahafi

kiansahafi@gmail.com