

# 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, cross-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 can include code scanners, which can identify vulnerabilities in the source code 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 simulate 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](#)<sup>1</sup>: SQL Injection
  2. [SWE-79](#)<sup>2</sup>: Cross-site Scripting
  3. [CWE-94](#)<sup>3</sup>: 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](https://cwe.mitre.org/data/definitions/1004.html)<sup>1</sup>: Sensitive Cookie Without ‘HttpOnly’ Flag
  2. [CWE-297](https://cwe.mitre.org/data/definitions/297.html)<sup>2</sup>: Improper Validation of Certificate with Host Mismatch
  3. [CWE-327](https://cwe.mitre.org/data/definitions/327.html)<sup>3</sup>: 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>

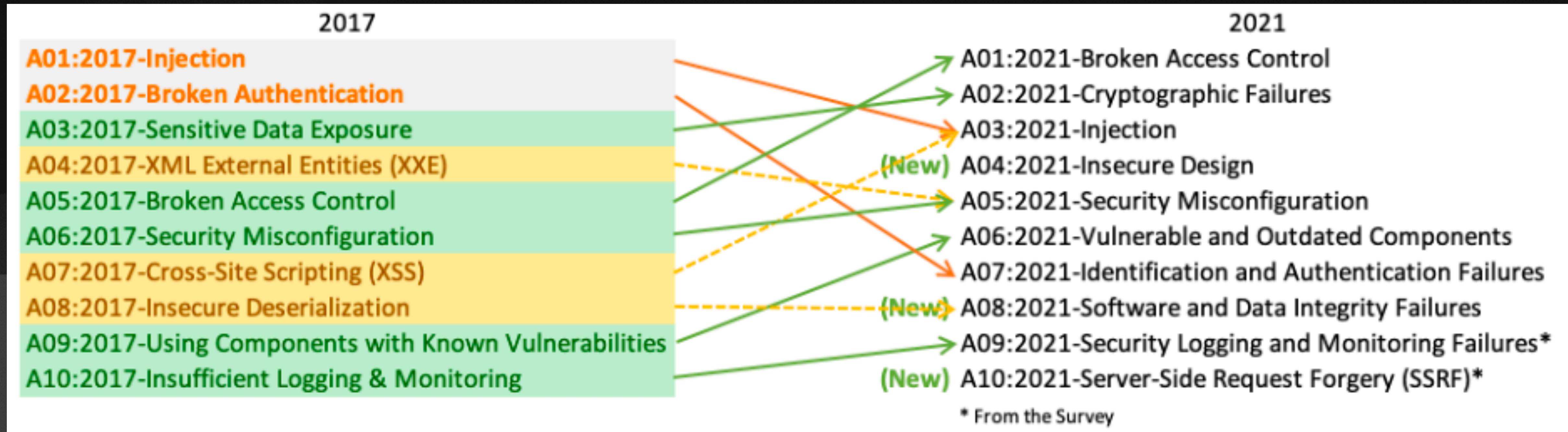
3.<https://cwe.mitre.org/data/definitions/327.html>

# 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

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

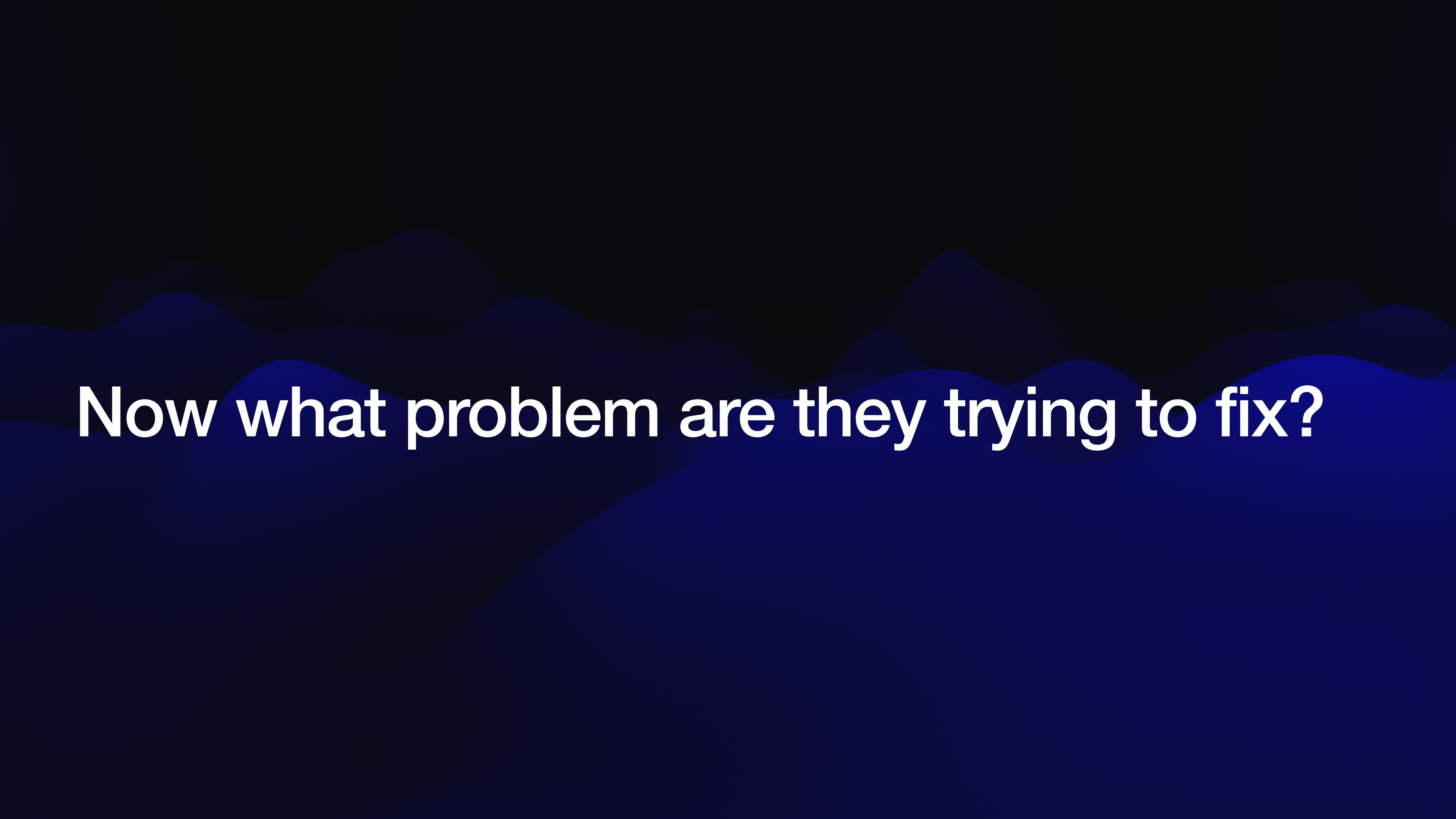
# OWASP Top 10:



# And finally, SANS Top 25:



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

The background features a dark blue gradient with three prominent, wavy horizontal bands. The top band is a lighter shade of blue, while the middle and bottom bands are darker. These waves create a sense of depth and motion.

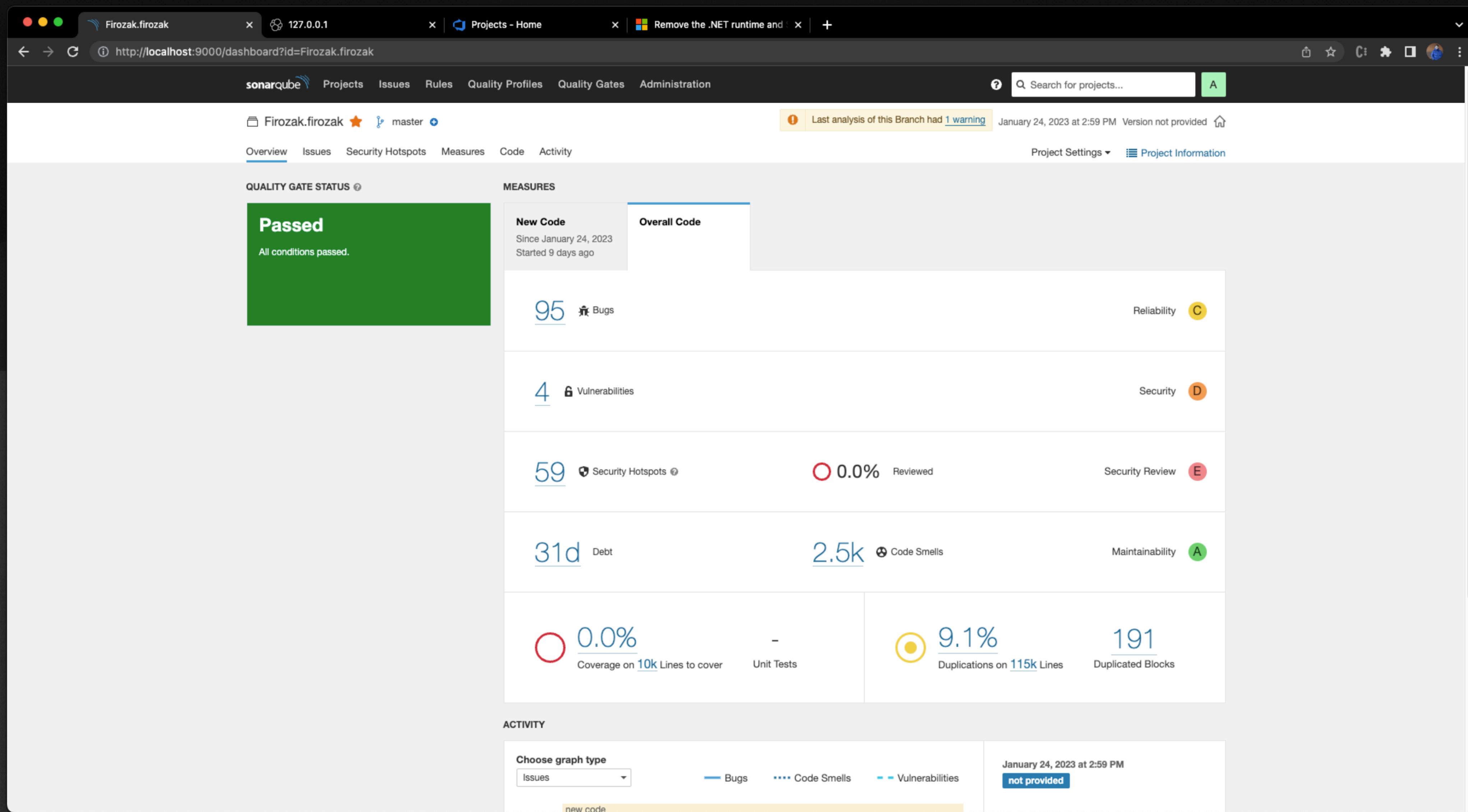
Now what problem are they trying to fix?

**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 OWASD, 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 stakeholders to collaborate and work together.

Now, What did WE do?

# Well the First Report Was Not Great!



# And the bugs Issues Were Like this:

The screenshot shows the SonarQube Issues page for the project "Firozak.firozak" on the master branch. The page displays a list of 2,569 issues across 32d effort. The left sidebar contains filters for Period, Type (Bug, Vulnerability, Code Smell), Severity (Blocker, Critical, Major, Minor, Info), Scope, Resolution, Status, Security Category, Creation Date, Language, Rule, Tag, Directory, File, Assignee, and Author. The main content area shows a grid of issues with columns for file path, message, severity, source, timestamp, and effort. Many issues are categorized as Code Smell, Major, and Open.

File	Message	Severity	Source	Timestamp	Effort
Blog/Blog.Application/AppSettings.cs	Non-nullable property 'ImageUrlInCdn' must contain a non-null value when exiting constructor. Consider declaring the property as nullable.	Code Smell	ROSLYN	10 days ago	L5
Blog/Blog.Application/Command/Posts/CreatePostCommand.cs	Non-nullable property 'WriterFullName' must contain a non-null value when exiting constructor. Consider declaring the property as nullable.	Code Smell	ROSLYN	10 days ago	L6
Blog/.../Handlers/Categories/DecreaseCategoryDisplayOrderCommandHandler.cs	Rename parameter 'writerUserId' to 'currentUserId' to match the interface declaration.	Code Smell	ROSLYN	10 days ago	L15
Blog/.../Handlers/Categories/DeleteCategoryCommandHandler.cs	Remove this empty statement.	Code Smell	ROSLYN	10 days ago	L22
Blog/.../Handlers/Categories/IncreaseCategoryDisplayOrderCommandHandler.cs	Remove this commented out code.	Code Smell	ROSLYN	10 days ago	L15
	Remove this empty statement.	Code Smell	ROSLYN	10 days ago	L18

# 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.

```
1 ... namespace Blog.Application;
2
3 public class AppSettings
4 {
5     public string 1 ImageUrlInCdn { get; set; }

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

6     public string SiteName { get; set; }

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

7
8 }
9
```

## 2. not matching with the Interface declarations:

```
10 ... {
11     public Guid CurrentUserId { get; private set; }
12     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.

13     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.

14
15     public void SetUser(Guid writerUserId)

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

16     => CurrentUserId = (writerUserId == Guid.Empty) ?
17         throw new DomainException("شناسه مورد نظر صحیح نیست");
18         writerUserId;
19     public void SetWriterFullNameAndEmail(string writerFullName, Email email)
20     {
21         WriterFullName = (string.IsNullOrWhiteSpace(writerFullName)) ?
22             throw new DomainException("نام نویسنده مشخص نیست");
23             writerFullName;
24             WriterEmail = email;

    ⓘ 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.
```

```
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:

```
13 ...
14     {
15         private readonly IPostCategoryRepository _postCategoryRepository;
16         //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:

sonarqube Issues 127.0.0.1 Projects - Home Remove the .NET runtime and + http://localhost:9000/project/issues?directories=firozak%2Fsource%2FData%2FFirozak.Domain%2FEntities&resolved=false&id=Firozak.firozak

Firozak.firozak master Overview Issues Security Hotspots Measures Code Activity Bulk Change Project Settings Project Information 0 issues 0 effort

We couldn't find any results matching selected criteria. Try to change filters to get some results.

Domain Project

This is Where the Business Entities!

Creation Date Language Rule Tag Directory FIROZAK/SOURC... Clear firozak.domain firozak.../Firozak.Domain/Entities 1 of 1 shown Press ⌘ to add to selection File Assignee Author

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.

SonarQube™ technology is powered by SonarSource SA  
Community Edition - Version 9.7.1 (build 62043) - [LGPL v3](#) - [Community](#) - [Documentation](#) - [Plugins](#) - [Web API](#)

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

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

The screenshot shows the SonarQube interface for a project. On the left, there's a sidebar with a navigation tree under 'Security Category':

- SonarSource / WEAK CRYPTOGRAPHY
  - Weak Cryptography: 4
- OWASP Top 10 2021
  - A2 - Cryptographic Failures: 4
  - A7 - Identification and Authentication: 4
  - A5 - Security Misconfiguration: 1
- OWASP Top 10 2017
  - A3 - Sensitive Data Exposure: 4
  - A6 - Security Misconfiguration: 4
- SANS Top 25
  - Porous Defenses: 3
- CWE
  - Search for CWEs... (with a search icon)
  - CWE-295 - Improper Certificate Vali...: 4
  - CWE-326 - Inadequate Encryption S...: 3
  - CWE-327 - Use of a Broken or Risky...: 3

On the right, the main panel displays a list of vulnerabilities found in the codebase:

- Core/Core.Crowler/ReaderBase.cs
  - Change this code to use a stronger protocol. (10 days ago, L25 priority, Critical severity, Open status, Not assigned assignee, 2min effort, Comment)
  - Change this code to use a stronger protocol. (10 days ago, L25 priority, Critical severity, Open status, Not assigned assignee, 2min effort, Comment)
- Identity/Services/MessagingServices.cs
  - Change this code to use a stronger protocol. (10 days ago, L23 priority, Critical severity, Open status, Not assigned assignee, 2min effort, Comment)
  - Enable server certificate validation on this SSL/TLS connection (10 days ago, L25 priority, Critical severity, Open status, Not assigned assignee, 5min effort, Comment)

At the bottom, a note says: "Embedded database should be used for evaluation purposes only".

# The Issue:

The screenshot shows the SonarQube interface for a project named "Firozak.firozak". The top navigation bar includes links for Projects, Issues, Rules, Quality Profiles, Quality Gates, and Administration. A search bar and a user profile icon are also present.

The main content area displays the "Issues" tab for the "master" branch. A banner at the top right indicates "Last analysis of this Branch had 1 warning" on January 24, 2023, at 2:59 PM, with a note about "Version not provided".

The left sidebar lists several files with their respective issue counts:

- Core/Core.Crowler/ReaderBase.cs: Change this code to use a stronger protocol. (6 Vulnerability)
- Identity/Services/MessagingServices.cs: Change this code to use a stronger protocol. (6 Vulnerability)
- Enable server certificate validation on this SSL/TLS connection (6 Vulnerability +1)

Below the sidebar, it says "4 of 4 shown".

The central panel shows a specific issue for "Core/Core.Crowler/ReaderBase.cs". The title is "Change this code to use a stronger protocol." and the description states: "Weak SSL/TLS protocols should not be used" (csharp:squid:S4423). The issue is categorized as a Vulnerability, Critical, Open, Not assigned, with 2min effort and 0 comments. It was reported 10 days ago by L25, with tags including cwe, owasp-a3, owasp-a6, owasp-m3... .

The code snippet highlighted in the issue details is:

```
request.ContentType = "text/html";
request.ProtocolVersion = HttpVersion.Version10; // THIS DOES THE TRICK
ServicePointManager.Expect100Continue = true;
request.MaximumAutomaticRedirects = 1;
request.AllowAutoRedirect = false;
ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls | SecurityProtocolType.Tls11 |
SecurityProtocolType.Tls12;
```

Two red boxes highlight sections of the code with the message "Change this code to use a stronger protocol.".

The bottom of the page features a note: "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.  
Weak SSL/TLS protocols should not be used [csharpsquid:S4423](#)  
10 days ago ▾  
Get permalink  
Vulnerability Critical Open Not assigned 2min effort 0 comments  
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 minimum by [updating 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 on the OS configuration
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
};
```

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

# 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 main 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:

🛡️ 59 Security Hotspots to review

Review priority: **HIGH**

SQL Injection 1 ▾

Review priority: **MEDIUM**

Denial of Service (DoS) 10 ▾

Weak Cryptography 3 ▾

Review priority: **LOW**

Encryption of Sensitive Data 1 ▾

Insecure Configuration 1 ▾

Others 43 ▾

59 of 59 shown

Review priority	Hotspot Type	Count
HIGH	SQL Injection	1
MEDIUM	Denial of Service (DoS)	10
MEDIUM	Weak Cryptography	3
LOW	Encryption of Sensitive Data	1
LOW	Insecure Configuration	1
LOW	Others	43

# let's see one of them:

The screenshot shows the SonarQube interface for a project named "Firozak.firozak". The top navigation bar includes tabs for "sonarqube", "Projects", "Issues", "Rules", "Quality Profiles", "Quality Gates", and "Administration". A search bar at the top right contains the placeholder "Search for projects...".

The main content area displays a summary of security hotspots:

- 59 Security Hotspots to review**
- Review priority:** HIGH (1 SQL Injection)
- Review priority:** MEDIUM (10 Denial of Service (DoS))

A specific hotspot is highlighted on the right side:

**Make sure using a dynamically formatted SQL query is safe here.**  
Formatting SQL queries is security-sensitive [csharpsquid:S2077](#)

**Status: TO REVIEW**  
This security hotspot needs to be reviewed to assess whether the code poses a risk. [Change status](#)

**Assignee:** Not assigned [Edit](#)

**Where is the risk?** **What's the risk?** **Assess the risk** **How can I fix it?**

[Blog/Blog.Infrastructure/Repositories/OccasionRepository.cs](#) [Open in IDE](#) [Get Permalink](#)

```
25     || (firstPersianDayOfYear < endPersianDayOfYear && c.DayOfYear >= firstPersianDayOfYear &&
26         c.DayOfYear < endPersianDayOfYear && c.CalendarType == 2)
27         || (firstPersianDayOfYear > endPersianDayOfYear && (c.DayOfYear >= firstPersianDayOfYear ||
28             c.DayOfYear < endPersianDayOfYear) && c.CalendarType == 2)
29
30             || (firstHijriDayOfYear < endHijriDayOfYear && c.DayOfYear >= firstHijriDayOfYear &&
31                 c.DayOfYear < endHijriDayOfYear && c.CalendarType == 3)
32                 || (firstHijriDayOfYear > endHijriDayOfYear && (c.DayOfYear >= firstHijriDayOfYear ||
33                     c.DayOfYear < endHijriDayOfYear) && c.CalendarType == 3)
34                     ).ToListAsync();
35
36             public async Task<List<Occasion>> GetByTagAsync(Guid tagId)
37             {
38                 var 1 query = $"select b.* from [Blog].[Occurrences] AS [b] INNER JOIN[Blog].Occasion_Tag AS[pt] ON
39 [pt].[OccasionId] = [b].Id where pt.[OccasionTagId] = '{tagId.ToString()}' ORDER BY [b].[DisplayPriority] DESC";
40
41                     return await _context.Set<Occasion>().FromSqlRaw<Occasion>(query).ToListAsync();
42             }
43
44             public override Task<Occasion?> GetAsync(Guid id) => throw new NotImplementedException();
45 }
```

**Comment**

**Make sure using a dynamically formatted SQL query is safe here.**

# Assess the risk:

The screenshot shows the SonarQube interface for the project "Firozak.firozak". The top navigation bar includes tabs for Security Hotspots, 127.0.0.1, Projects - Home, Remove the .NET runtime and, SecurityProtocolType.Tls Chan, security - Default SecurityProt, and a plus sign. The main header has links for sonarqube, Projects, Issues, Rules, Quality Profiles, Quality Gates, Administration, and a search bar. A banner at the top right indicates the last analysis had 1 warning on January 24, 2023, at 2:59 PM.

The left sidebar shows two sections of security hotspots:

- SQL Injection** (1):
  - Make sure using a dynamically formatted SQL query is safe here.
  - ...re/Repositories/OccasionRepository.cs
  - 1 SQL Query is dynamically formatted and assigned to query.
- Review priority: HIGH**
- Denial of Service (DoS)** (10):
  - Make sure the content length limit is safe here.
  - ...DN.Api/Controllers/UploadController.cs
  - Make sure the content length limit is safe here.
  - ...DN.Api/Controllers/UploadController.cs
  - Make sure the content length limit is safe here.
  - ...DN.Api/Controllers/UploadController.cs
  - Make sure the regex used here, which is vulnerable to super-linear runtime due to backtracking, cannot lead to denial of service.
  - ...obtrusive/jquery.validate.unobtrusive.js
- Review priority: MEDIUM**

The right panel displays a card for assessing risks, titled "Ask Yourself Whether". It lists three questions:

- Some parts of the query come from untrusted values (like user inputs).
- The query is repeated/duplicated in other parts of the code.
- The application must support different types of relational databases.

Below this, it says "There is a risk if you answered yes to any of those questions." and provides a "Sensitive Code Example" block containing C# code:

```
public void Foo(DbContext context, string query, string param)
{
    string sensitiveQuery = string.Concat(query, param);
    context.Database.ExecuteSqlCommand(sensitiveQuery); // Sensitive
    context.Query<User>().FromSql(sensitiveQuery); // Sensitive

    context.Database.ExecuteSqlCommand($"SELECT * FROM mytable WHERE mycol={value}", param); // Sensitive, the FormattableString is evaluated
    string query = $"SELECT * FROM mytable WHERE mycol={param}";
    context.Database.ExecuteSqlCommand(query); // Sensitive, the FormattableString has already been evaluated, it won't be
}

public void Bar(SqlConnection connection, string param)
{
    SqlCommand command;
    string sensitiveQuery = string.Format("INSERT INTO Users (name) VALUES ('{0}')", param);
    command = new SqlCommand(sensitiveQuery); // Sensitive

    command.CommandText = sensitiveQuery; // Sensitive

    SqlDataAdapter adapter;
    adapter = new SqlDataAdapter(sensitiveQuery, connection); // Sensitive
}
```

# How can I fix it?

The screenshot shows the SonarQube interface for the project **Firozak.firozak**. The main navigation bar includes links for **Projects**, **Issues**, **Rules**, **Quality Profiles**, **Quality Gates**, and **Administration**. A search bar at the top right allows searching for projects. The current view is under the **Security Hotspots** tab.

A message at the top right indicates that the last analysis had **1 warning** on January 24, 2023, at 2:59 PM, with the version not provided.

The main content area displays **59 Security Hotspots to review**, filtered by **Assigned to me** and **All**. The status is set to **To review** and the scope is **Overall code**. The overall completion of reviewing security hotspots is **0.0%**.

On the left, two sections are visible:

- SQL Injection** (1 issue):
  - Review priority: **HIGH**
  - Description: Make sure using a dynamically formatted SQL query is safe here.
  - Code snippet: ...re/Repositories/OccasionRepository.cs
  - Comment: 1 SQL Query is dynamically formatted and assigned to query.
- Denial of Service (DoS)** (10 issues):
  - Review priority: **MEDIUM**
  - Description: Make sure the content length limit is safe here.
  - Code snippet: ...DN.Api/Controllers/UploadController.cs
  - Description: Make sure the content length limit is safe here.
  - Code snippet: ...DN.Api/Controllers/UploadController.cs
  - Description: Make sure the content length limit is safe here.
  - Code snippet: ...DN.Api/Controllers/UploadController.cs
  - Description: Make sure the regex used here, which is vulnerable to super-linear runtime due to backtracking, cannot lead to denial of service.
  - Code snippet: ...obtrusive/jquery.validate.unobtrusive.js

The right side of the interface provides a detailed view of a specific security hotspot:

- Where is the risk?**
- What's the risk?**
- Assess the risk**
- How can I fix it?** (This step is highlighted with a dashed blue border.)

**Recommended Secure Coding Practices**:

- Use [parameterized queries, prepared statements, or stored procedures](#) and bind variables to SQL query parameters.
- Consider using ORM frameworks if there is a need to have an abstract layer to access data.

**Compliant Solution**:

```
public void Foo(DbContext context, string query, string param)
{
    context.Database.ExecuteSqlCommand("SELECT * FROM mytable WHERE mycol=@p0", param); // Compliant, it's a parameterized
}
```

**See**:

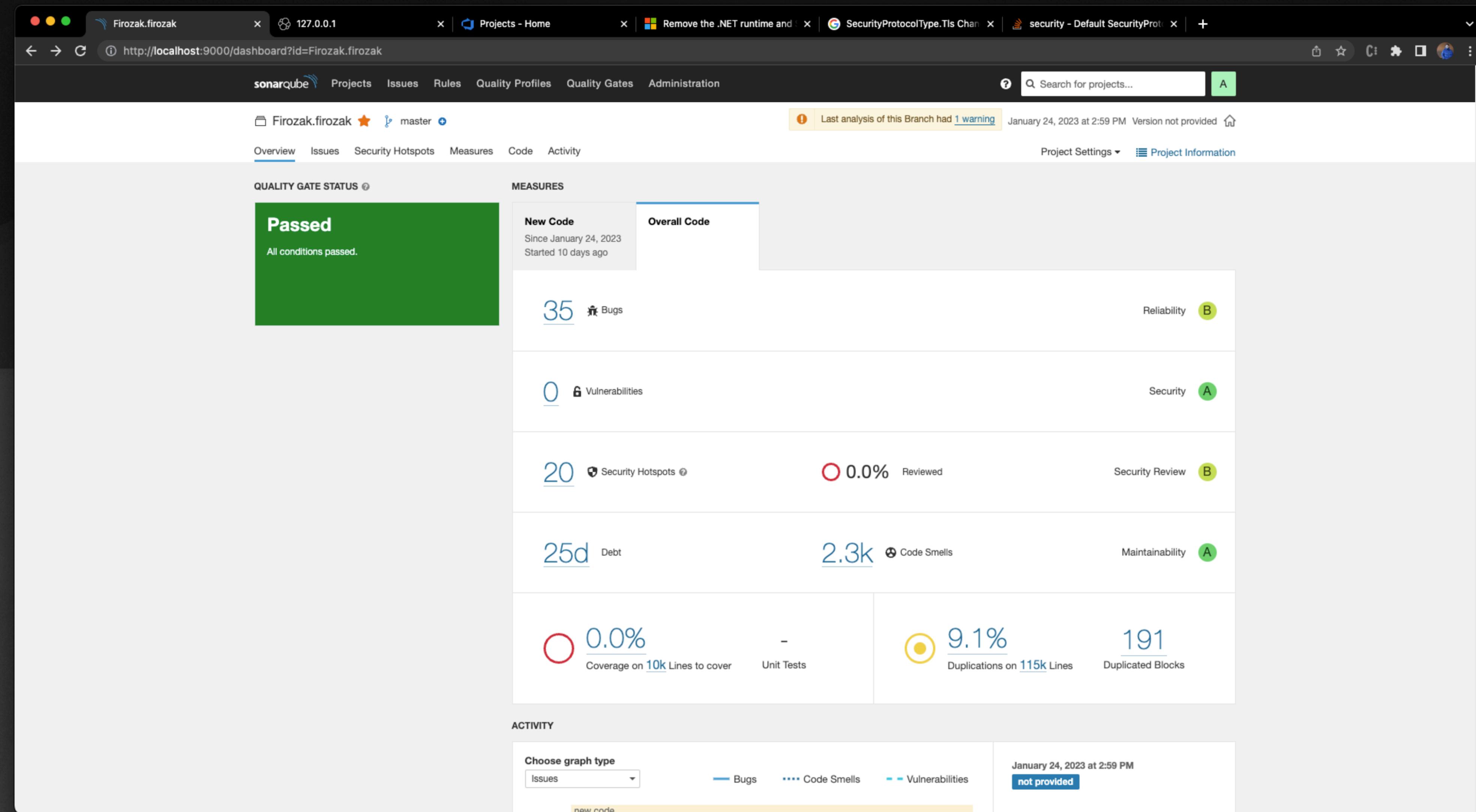
- [OWASP Top 10 2021 Category A3 - Injection](#)
- [OWASP Top 10 2017 Category A1 - Injection](#)
- [MITRE, CWE-20 - Improper Input Validation](#)
- [MITRE, CWE-89 - Improper Neutralization of Special Elements used in an SQL Command](#)
- [SANS Top 25 - Insecure Interaction Between Components](#)
- Derived from FindSecBugs rules [Potential SQL/JPQL Injection \(JPA\)](#), [Potential SQL/JDOQL Injection \(JDO\)](#), [Potential SQL/HQL Injection \(Hibernate\)](#)

**Comment:**

Formatting Help : [Bold](#) [Code](#) [Bulleted point](#)

**Comment**

# 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](mailto:kiansahafi@gmail.com)

