sonar

Find, Fix and Grow with Education from Sonar

What is Education from Sonar?

Software development is complex and involves a range of activities, tools, languages, and frameworks that require different types of support and education. Plus, the evolving nature of technology demands that developers learn new coding practices at such a rapid pace that it's impossible to keep up.

That's why, when a developer encounters an issue, it's common to move out of the coding workflow to investigate why the issue occurred and how to correct it before returning to the task at hand. Oftentimes the investigation can lead to lost time spent reviewing many sources with many opinions and little way to tell which is the best remediation path or why it should be chosen.

When arriving at "the why" behind an issue, developers need relevant education based on the specific context of their work and the ability to address the issue with speed and precision. Beyond effective remediation, developers can go a step further by gaining a deep understanding of the issue, adding it to their knowledge as they grow their expertise. But when developers leave their workflow to investigate, they sacrifice precious time and efficiency in the face of constant deadlines.

For developers to work most effectively, timely, contextualized expert education that's embedded in the coding workflow is essential. With relevant education presented within the development environment that's based on the context of the code being written, developers can quickly understand what the issue is, why it's occurring, and how to fix it. With education from Sonar, or Learn as You Code, developers can use contextualized education at their fingertips to avoid context switching and refine their coding expertise with a deeper understanding of current and related issues, all while delivering timely, high-quality code.

Why is Learn as You Code Important?

Sonar equips developers with the right tools and best practices to write Clean Code. When code is clean, developers spend less time fixing bugs and more time meeting delivery and business goals. Clean Code ensures you get the most value out of your code, and consequently, your software.

When writing Clean Code, it's not enough to find and fix issues. Developers should have the opportunity to find, understand, and fix issues to optimally write Clean Code. To achieve this, we provide the right educational information at the right time inside the workflow. This helps developers learn from their mistakes and get smarter, fix today's issues faster, and internalize best practices to avoid reproducing the same issues in the future.

When developers can confidently understand and fix issues, they grow as individual contributors and team members and, as a result, increase their delivery. As delivery improves, your software and your organization feel the impact. Sonar enables developers to leverage Learn as You Code within their workflow so that they can see the best results possible.

What you achieve with Learn as You Code

Faster issue remediation and delivery

When developers gain an understanding of why an issue occurs without leaving the development workflow they code with greater skill and avoid future issues. With education that stays current, development teams can continuously improve and evolve to support demands.

Reinforced Clean Code best practices

Learn as You Code helps ensure that code stays more succinct and maintainable by surfacing the best path to issue resolution and enabling developers to act quickly. When developers have easy access to why an issue occurs it underpins the knowledge they gain and the Clean Code practices they should consistently apply.

Professional growth and retention

Developers can dedicate less time to figuring out the root cause of an issue when the answer is in front of them which leaves more time to focus on honing their coding skills. When the opportunity for professional growth is part of the daily work, it fosters satisfaction and retention within development teams.

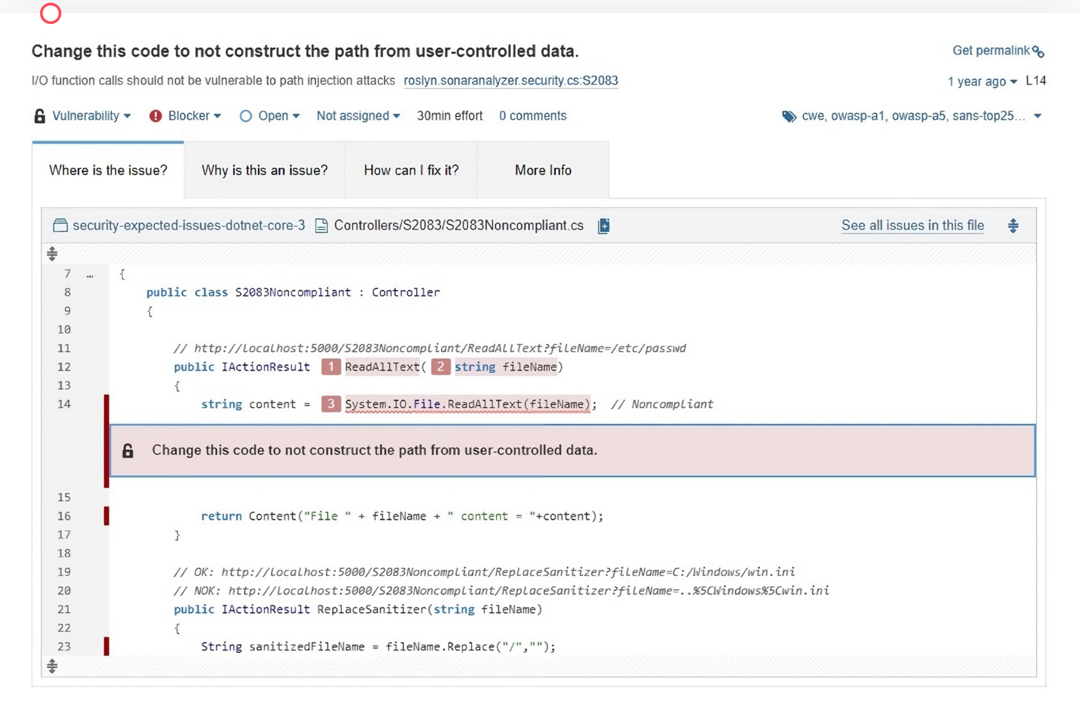
What makes Learn as You Code different?

Sonar helps developers gain access to immediate and contextualized feedback based on years of language analyzer experience. From SonarLint, Sonar's free IDE extension that automatically boosts coding efficiency with its quick fixes feature, to SonarQube and SonarCloud in your CI/CD, education is built in to the entire workflow.

Learn as You Code from Sonar:

- Appears at the right place and the right time with well-structured rule descriptions
- Integrates into the development workflow
- Gives you code samples that suit your coding framework
- Provides specific and contextualized guidance based on the issue
- Educational in nature and helps developers grow their expertise

Using SonarQube and SonarCloud, automated code reviews seamlessly integrate into the development workflow to help developers quickly gain an understanding of what the issue is



Change this code to not construct the path from user-controlled data.

I/O function calls should not be vulnerable to path injection attacks roslyn.sonaranalyzer.security.cs:S2083

```
{
    public class S2083Noncompliant : Controller
    {

        // http://localhost:5000/S2083Noncompliant/ReadAllText?fileName=/etc/passwd
        public IActionResult ReadAllText( string fileName)
        {
            string content = System.IO.File.ReadAllText(fileName);  // Noncompliant
```

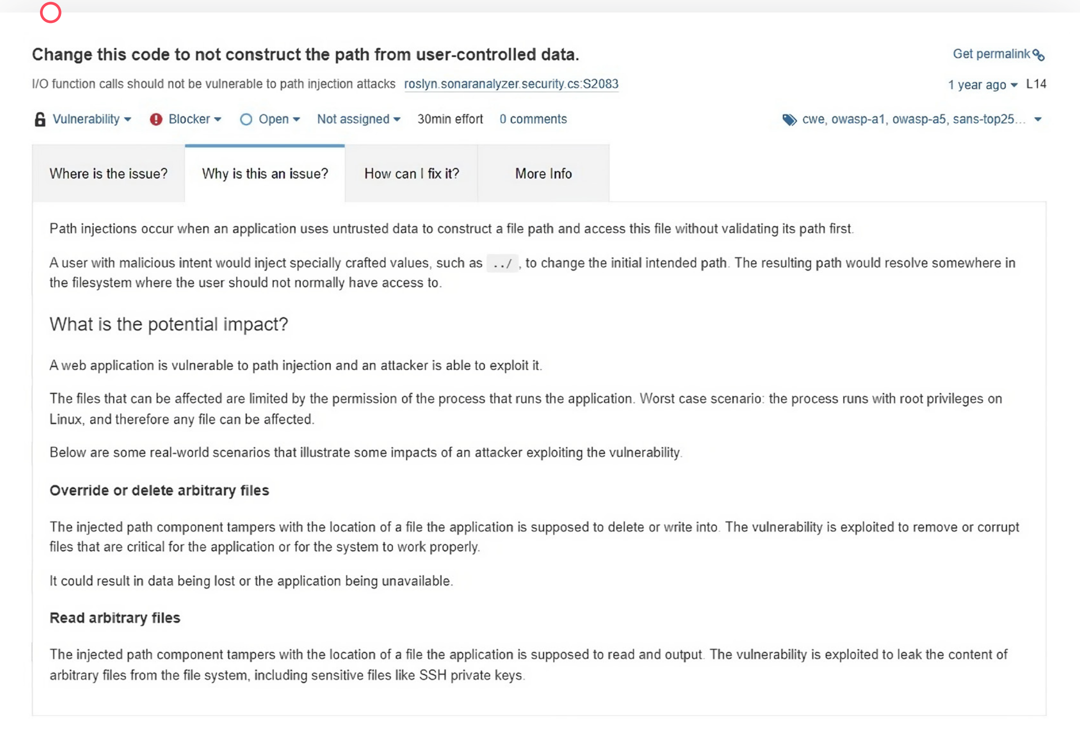
Change this code to not construct the path from user-controlled data.

```

            return Content("File " + fileName + " content = "+content);
        }

        // OK: http://localhost:5000/S2083Noncompliant/ReplaceSanitizer?fileName=C:/Windows/win.ini
        // NOK: http://localhost:5000/S2083Noncompliant/ReplaceSanitizer?fileName=..%5CWindows%5Cwin.ini
        public IActionResult ReplaceSanitizer(string fileName)
        {
            String sanitizedFileName = fileName.Replace("/","");
```

Why it's an issue



Change this code to not construct the path from user-controlled data.

I/O function calls should not be vulnerable to path injection attacks roslyn.sonaranalyzer.security.cs:S2083

Path injections occur when an application uses untrusted data to construct a file path and access this file without validating its path first.

A user with malicious intent would inject specially crafted values, such as `../`, to change the initial intended path. The resulting path would resolve somewhere in the filesystem where the user should not normally have access to.

What is the potential impact?

A web application is vulnerable to path injection and an attacker is able to exploit it.

The files that can be affected are limited by the permission of the process that runs the application. Worst case scenario: the process runs with root privileges on Linux, and therefore any file can be affected.

Below are some real-world scenarios that illustrate some impacts of an attacker exploiting the vulnerability.

Override or delete arbitrary files

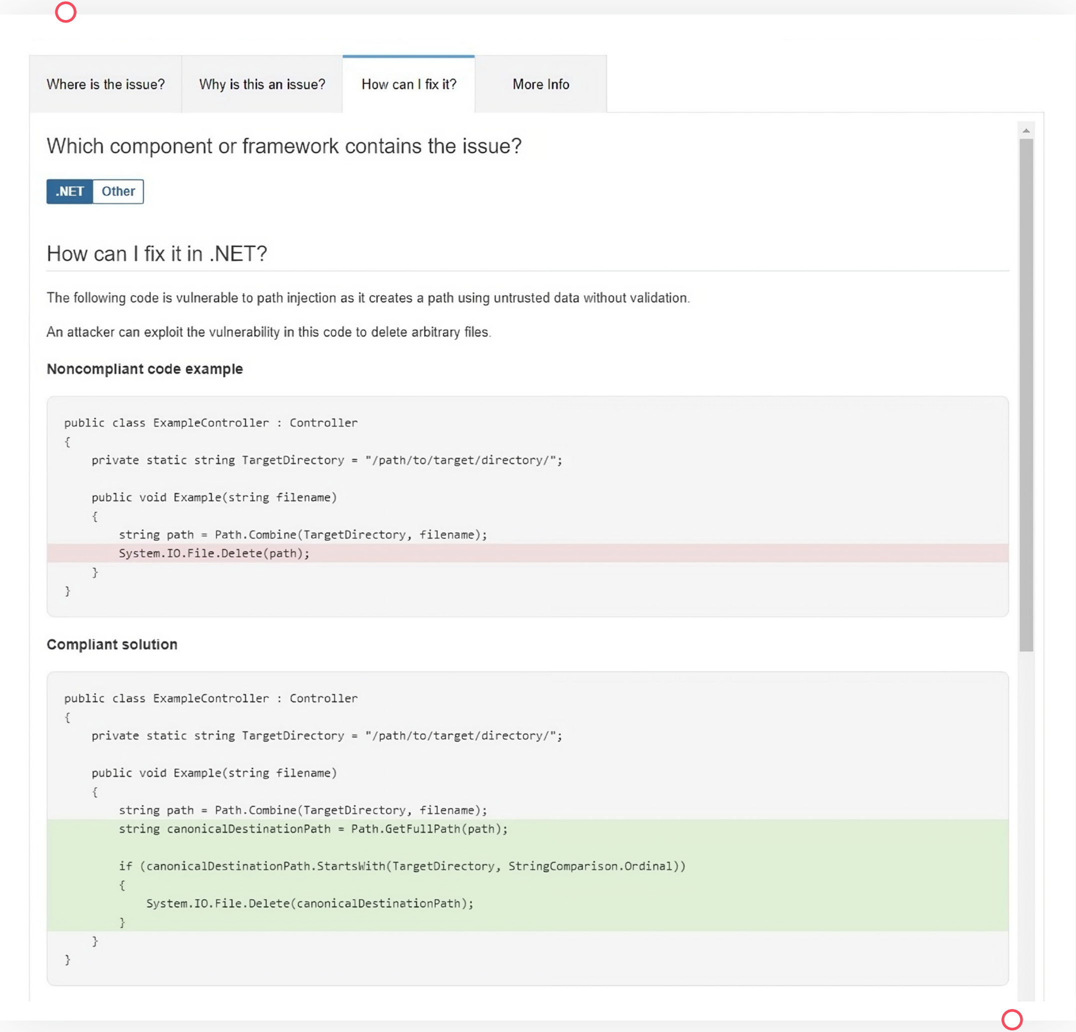
The injected path component tampers with the location of a file the application is supposed to delete or write into. The vulnerability is exploited to remove or corrupt files that are critical for the application or for the system to work properly.

It could result in data being lost or the application being unavailable.

Read arbitrary files

The injected path component tampers with the location of a file the application is supposed to read and output. The vulnerability is exploited to leak the content of arbitrary files from the file system, including sensitive files like SSH private keys.

And how they can fix it



Which component or framework contains the issue?

How can I fix it in .NET?

The following code is vulnerable to path injection as it creates a path using untrusted data without validation.

An attacker can exploit the vulnerability in this code to delete arbitrary files.

Noncompliant code example

```
public class ExampleController : Controller
{
    private static string TargetDirectory = "/path/to/target/directory/";

    public void Example(string filename)
    {
        string path = Path.Combine(TargetDirectory, filename);
        System.IO.File.Delete(path);
    }
}
```

Compliant solution

```
public class ExampleController : Controller
{
    private static string TargetDirectory = "/path/to/target/directory/";

    public void Example(string filename)
    {
        string path = Path.Combine(TargetDirectory, filename);
        string canonicalDestinationPath = Path.GetFullPath(path);

        if (canonicalDestinationPath.StartsWith(TargetDirectory, StringComparison.Ordinal))
        {
            System.IO.File.Delete(canonicalDestinationPath);
        }
    }
}
```

All without having to look outside of the workflow for answers.

It's not enough to simply find and fix issues when your goal is to grow and learn. Learn as You Code from Sonar makes it easier than ever to help developers pursue their passion and grow their coding skills with intuitive, embedded issue-specific education. With the tools and resources from Sonar, writing Clean Code has never been more accessible.

Sonar is the home of Clean Code, trusted by more than 7 million developers and more than 400 thousand organizations worldwide.

VISIT SONARSOURCE.COM --->