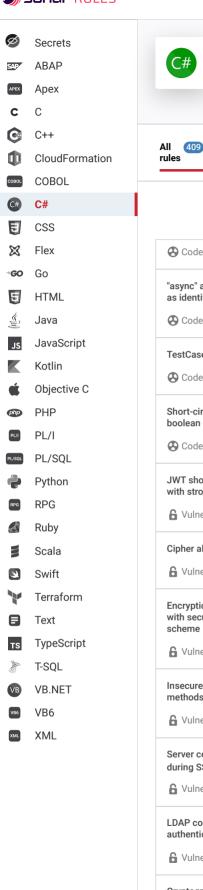
Quick 52 Fix

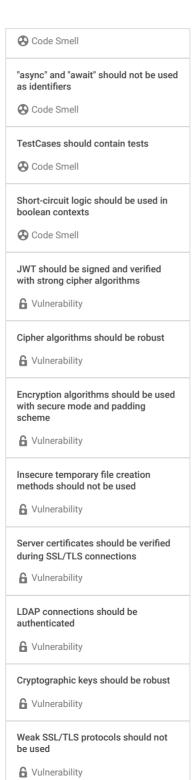




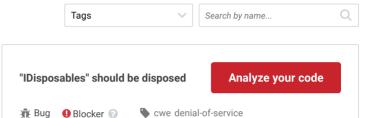


**#** Bug (76)

6 Vulnerability (34)



Cipher Block Chaining IVs should be



⊗ Code Smell

Security

Hotspot

(28)

When writing managed code, you don't need to worry about allocating or freeing memory: The garbage collector takes care of it. For efficiency reasons, some objects such as Bitmap use unmanaged memory, enabling for example the use of pointer arithmetic. Such objects have potentially huge unmanaged memory footprints, but will have tiny managed ones. Unfortunately, the garbage collector only sees the tiny managed footprint, and fails to reclaim the unmanaged memory (by calling Bitmap's finalizer method) in a timely fashion.

Moreover, memory is not the only system resource which needs to be managed in a timely fashion: The operating system can only handle having so many file descriptors (e.g. FileStream) or sockets (e.g. WebClient) open at any given time. Therefore, it is important to Dispose of them as soon as they are no longer needed, rather than relying on the garbage collector to call these objects' finalizers at some nondeterministic point in the future.

This rule tracks private fields and local variables of the following IDisposable types, which are never disposed, closed, aliased, returned, or passed to other methods.

- System.IO namespace
  - o System.IO.FileStream
  - System.IO.StreamReader
  - System.IO.StreamWriter
- System.Net namespace
  - o System.Net.WebClient
- $\bullet \quad {\tt System.Net.Sockets} \ {\tt namespace}$ 
  - System.Net.Sockets.Socket
  - System.Net.Sockets.TcpClient
  - System.Net.Sockets.UdpClient
- System.Drawing namespace
  - o System.Drawing.Image
  - o System.Drawing.Bitmap

which are either instantiated directly using the new operator, or using one of the following factory methods:

- System.IO.File.Create()
- System.IO.File.Open()
- System.Drawing.Image.FromFile()
- System.Drawing.Image.FromStream()

on both private fields and local variables.

# Noncompliant Code Example

```
public class ResourceHolder
{
   private FileStream fs; // Noncompliant; Dispose or Close a
   public void OpenResource(string path)
   {
      this.fs = new FileStream(path, FileMode.Open);
   }
```

# unpredictable

■ Vulnerability

Regular expressions should not be vulnerable to Denial of Service attacks

❸ Vulnerability

Hashes should include an unpredictable salt

Vulnerability

Non-async "Task/Task<T>" methods should not return null

R Bug

Calls to delegate's method

```
public void WriteToFile(string path, string text)
   var fs = new FileStream(path, FileMode.Open); // Noncomp
   var bytes = Encoding.UTF8.GetBytes(text);
   fs.Write(bytes, 0, bytes.Length);
}
```

### **Compliant Solution**

```
public class ResourceHolder : IDisposable
 private FileStream fs;
 public void OpenResource(string path)
   this.fs = new FileStream(path, FileMode.Open);
 public void Dispose()
   this.fs.Dispose();
 public void WriteToFile(string path, string text)
   using (var fs = new FileStream(path, FileMode.Open))
     var bytes = Encoding.UTF8.GetBytes(text):
     fs.Write(bytes, 0, bytes.Length);
}
```

### Exceptions

IDisposable variables returned from a method or passed to other methods are ignored, as are local IDisposables that are initialized with other IDisposables.

```
public Stream WriteToFile(string path, string text)
 var fs = new FileStream(path, FileMode.Open); // Compliant
 var bytes = Encoding.UTF8.GetBytes(text);
 fs.Write(bytes, 0, bytes.Length);
 return fs;
public void ReadFromStream(Stream s)
 var sr = new StreamReader(s); // Compliant as it would clo
  // ...
```

• MITRE, CWE-459 - Incomplete Cleanup

Available In:

sonarlint ⊕ | sonarcloud ♦ | sonarqube