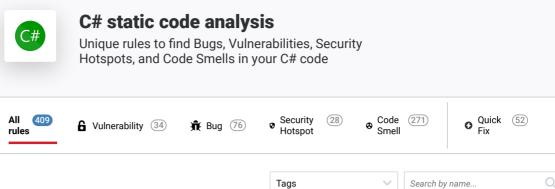
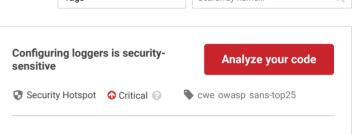


₩ Bug







Configuring loggers is security-sensitive. It has led in the past to the following vulnerabilities:

- CVE-2018-0285
- CVE-2000-1127
- CVE-2017-15113
- CVE-2015-5742

Logs are useful before, during and after a security incident.

- Attackers will most of the time start their nefarious work by probing the system for vulnerabilities. Monitoring this activity and stopping it is the first step to prevent an attack from ever happening
- In case of a successful attack, logs should contain enough information to understand what damage an attacker may have inflicted.

Logs are also a target for attackers because they might contain sensitive information. Configuring loggers has an impact on the type of information logged and how they are logged.

This rule flags for review code that initiates loggers configuration. The goal is to guide security code reviews

# Ask Yourself Whether

- unauthorized users might have access to the logs, either because they are stored in an insecure location or because the application gives access to them.
- the logs contain sensitive information on a production server. This can happen when the logger is in debug mode.
- the log can grow without limit. This can happen when additional information is written into logs every time a user performs an action and the user can perform the action as many times as he/she wants.
- the logs do not contain enough information to understand the damage an attacker might have inflicted. The loggers mode (info, warn, error) might filter out important information. They might not print contextual information like the precise time of events or the server hostname
- the logs are only stored locally instead of being backuped or replicated.

There is a risk if you answered yes to any of those questions.

### **Recommended Secure Coding Practices**

- Check that your production deployment doesn't have its loggers in "debug" mode as it might write sensitive information in logs.
- Production logs should be stored in a secure location which is only accessible to system administrators.
- Configure the loggers to display all warnings, info and error messages. Write relevant information such as the precise time of events and the hostname.
- Choose log format which is easy to parse and process automatically. It is important to process logs rapidly in case of an attack so that the impact is known and limited.
- Check that the permissions of the log files are correct. If you index the logs in some other service, make sure that the transfer and the service are secure too.

Caller information parameters should come at the end of the parameter list



Static fields should appear in the order they must be initialized



Classes directly extending "object" should not call "base" in "GetHashCode" or "Equals"



Anonymous delegates should not be used to unsubscribe from Events



Add limits to the size of the logs and make sure that no user can fill the disk
with logs. This can happen even when the user does not control the logged
information. An attacker could just repeat a logged action many times.

Remember that configuring loggers properly doesn't make them bullet-proof. Here is a list of recommendations explaining on how to use your logs:

- Don't log any sensitive information. This obviously includes passwords and credit card numbers but also any personal information such as user names, locations, etc... Usually any information which is protected by law is good candidate for removal.
- Sanitize all user inputs before writing them in the logs. This includes checking
  its size, content, encoding, syntax, etc... As for any user input, validate using
  whitelists whenever possible. Enabling users to write what they want in your
  logs can have many impacts. It could for example use all your storage space or
  compromise your log indexing service.
- Log enough information to monitor suspicious activities and evaluate the impact an attacker might have on your systems. Register events such as failed logins, successful logins, server side input validation failures, access denials and any important transaction.
- Monitor the logs for any suspicious activity.

#### Sensitive Code Example

.Net Core: configure programmatically

```
using System;
using System.Collections;
using System.Collections.Generic;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;
using Microsoft.AspNetCore;
namespace MvcApp
    public class ProgramLogging
        public static IWebHostBuilder CreateWebHostBuilder(s
            WebHost.CreateDefaultBuilder(args)
                .ConfigureLogging((hostingContext, logging)
                })
                .UseStartup<StartupLogging>();
    public class StartupLogging
        public void ConfigureServices(IServiceCollection ser
            services.AddLogging(logging => // Sensitive
                // ...
            });
        public void Configure(IApplicationBuilder app, IHost
        {
            IConfiguration config = null;
            LogLevel level = LogLevel.Critical;
            Boolean includeScopes = false;
            Func<string, Microsoft. Extensions. Logging. LogLeve
            Microsoft.Extensions.Logging.Console.IConsoleLog
            Microsoft.Extensions.Logging.AzureAppServices.Az
            Microsoft.Extensions.Logging.EventLog.EventLogSe
            // An issue will be raised for each call to an I
            loggerFactory.AddAzureWebAppDiagnostics(); // Se
            loggerFactory.AddAzureWebAppDiagnostics(azureSet
            loggerFactory.AddConsole(); // Sensitive
            loggerFactory.AddConsole(level); // Sensitive
            loggerFactory.AddConsole(level, includeScopes);
            loggerFactory.AddConsole(filter); // Sensitive
            loggerFactory.AddConsole(filter, includeScopes);
            loggerFactory.AddConsole(config): // Sensitive
            loggerFactory.AddConsole(consoleSettings); // Se
            loggerFactory.AddDebug(); // Sensitive
```

```
loggerFactory.AddDebug(level); // Sensitive
loggerFactory.AddDebug(filter); // Sensitive
loggerFactory.AddEventLog(); // Sensitive
loggerFactory.AddEventLog(eventLogSettings); //
loggerFactory.AddEventLog(level); // Sensitive
loggerFactory.AddEventSourceLogger(); // Sensitive
loggerFactory.AddEventSourceLogger(); // Sensiti

IEnumerable<ILoggerProvider> providers = null;
LoggerFilterOptions filterOptions1 = null;
IOptionsMonitor<LoggerFilterOptions> filterOptio

LoggerFactory factory = new LoggerFactory(); //
new LoggerFactory(providers); // Sensitive
new LoggerFactory(providers, filterOptions1); //
new LoggerFactory(providers, filterOptions2); //
}
}
}
```

### Log4Net

```
using System;
using System.IO;
using System.Xml;
using log4net.Appender;
using log4net.Config;
using log4net.Repository;
namespace Logging
    class Log4netLogging
        void Foo(ILoggerRepository repository, XmlElement el
        IAppender appender, params IAppender[] appenders) {
            log4net.Config.XmlConfigurator.Configure(reposit
            log4net.Config.XmlConfigurator.Configure(reposit
            log4net.Config.XmlConfigurator.Configure(reposit
            log4net.Config.XmlConfigurator.Configure(reposit
            log4net.Config.XmlConfigurator.Configure(reposit
            log4net.Config.XmlConfigurator.ConfigureAndWatch
            log4net.Config.DOMConfigurator.Configure(); // S
            log4net.Config.DOMConfigurator.Configure(reposit
            log4net.Config.DOMConfigurator.Configure(element
            log4net.Config.DOMConfigurator.Configure(reposit
            log4net.Config.DOMConfigurator.Configure(configF
            log4net.Config.DOMConfigurator.Configure(reposit
            log4net.Config.DOMConfigurator.Configure(configS
            log4net.Config.DOMConfigurator.Configure(reposit
            log4net.Config.DOMConfigurator.ConfigureAndWatch
            log4net.Config.DOMConfigurator.ConfigureAndWatch
            log4net.Config.BasicConfigurator.Configure(); //
            log4net.Config.BasicConfigurator.Configure(appen
            log4net.Config.BasicConfigurator.Configure(appen
            log4net.Config.BasicConfigurator.Configure(repos
            log4net.Config.BasicConfigurator.Configure(repos
            log4net.Config.BasicConfigurator.Configure(repos
    }
}
```

# NLog: configure programmatically

```
namespace Logging
{
    class NLogLogging
    {
       void Foo(NLog.Config.LoggingConfiguration config) {
            NLog.LogManager.Configuration = config; // Sensi
       }
    }
}
```

## Serilog

```
namespace Logging {
```

```
class SerilogLogging
{
     void Foo() {
         new Serilog.LoggerConfiguration(); // Sensitive
     }
}
```

### See

- OWASP Top 10 2021 Category A9 Security Logging and Monitoring Failures
- OWASP Top 10 2017 Category A3 Sensitive Data Exposure
- OWASP Top 10 2017 Category A10 Insufficient Logging & Monitoring
- MITRE, CWE-117 Improper Output Neutralization for Logs
- MITRE, CWE-532 Information Exposure Through Log Files
- SANS Top 25 Porous Defenses

Available In:

sonarcloud 🔂 | sonarqube

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.

Privacy Policy