

# **ENPM 809W**

# **Introduction to Secure Software Engineering**

**Gananand Kini**

**Lecture 12**

**Error Handling and Logging related security  
bugs - Defenses**



# Outline



- **Error handling defensively**
- **Logging defensively**
- **Ensure debug code is removed**

# Error Handling Defensively

# Error Handling defensively

- **Do catch errors and handle such cases.**
- **Do *not* reveal sensitive information from the error to the user.**
- **Error Checking  $\neq$  Error Handling**
  - Error Checking – program proceeds with normal flow and then explicitly checks for errors in the state of the program and returns values indicating an error.
  - Input validation is similar to error checking (after having received the input you are checking its validity)
  - Error Handling – errors or exceptions are produced by the language or framework that need to be handled immediately. For example, memory allocation where memory might run out (since it can be a finite resource.)
- **Clean up resources being held by the component of the software system where error is being produced.**
  - If error is being thrown and bubbled up, still clean local resources then bubble up the error if possible.

# CVE-2008-4302 Exception Handling



```
1  boolean DoStuff ()
2  {
3      try
4      {
5          while (condition == true)
6          {
7              ThreadLock(TRUE);
8              // do some stuff
9              // an exception may be thrown
10             ThreadLock(FALSE);
11         }
12     }
13     catch (Exception e)
14     {
15         System.err.println("Something bad happened!");
16         return (FAILURE);
17     }
18     return (SUCCESS);
19 }
```

If an exception is thrown while the thread is locked, then the function will return without unlocking the thread.

# CVE-2008-4302 Potential Fix...



```
1  boolean DoStuff ()
2  {
3      try
4      {
5          while (condition == true)
6          {
7              ThreadLock(TRUE);
8              // do some stuff
9              // an exception may be thrown
10             ThreadLock(FALSE);
11         }
12     }
13     catch (Exception e)
14     {
15
16         if (isThreadLocked == TRUE) ThreadLock(FALSE);
17
18         System.err.println("Something bad happened!");
19         return (FAILURE);
20     }
21     return (SUCCESS);
22 }
```

# Error reports



- **Limit error information sent back to user**
  - Information may help attacker
  - *Do* log problems, in ways not available to potential adversaries
- **E.G., login failure**
  - Just tell them “authorization failed” – not “no such user” or “password incorrect” or (worse) “need longer password”

# Error Handling: Calling out to logging/debugging systems



- **Centralize all logging/debugging, use consistently**
  - Simplifies analysis (all data in one place)
  - Eases change/reconfiguration
- **Log *instead* of revealing problem details to users**
  - Ok to say there's a problem, but don't say too much
  - Attackers love it when you give them detailed data!
  - Record important successes & failures
- **Try to reuse existing log systems**
  - Less code, easier to integrate, etc.
  - Existing ones: log4j, java.util.logging, syslog, ...
  - Deployments typically want to centralize logs so they can easily combine data from multiple sources, change how & how much to log, where it's stored, send to separate protected system, etc.



# OLD ASP .Net Error Handling Options

## ■ At the Web.config level:

```
<configuration>
  <system.web>
    <customErrors mode="On" defaultRedirect="ErrorPage.aspx?handler=customErrors%20section%20-%20Web.config">
      <error statusCode="404" redirect="ErrorPage.aspx?msg=404&handler=customErrors%20section%20-%20Web.config"/>
    </customErrors>
  </system.web>
</configuration>
```

**NOT GRANULAR ENOUGH. SORT OF A CATCH ALL.  
CANNOT GET SPECIFICS ON WHAT ERROR  
OCCURRED.**

## ■ At the Application level (Global.asax):

```
void Application_Error(object sender, EventArgs e)
{
    Exception exc = Server.GetLastError();
    if (exc is HttpUnhandledException)
    {
        // Pass the error on to the error page.
        Server.Transfer("ErrorPage.aspx?handler=Application_Error%20-%20Global.asax", true);
    }
}
```

Source: ASP.NET Error Handling. Microsoft Docs. <https://docs.microsoft.com/en-us/aspnet/web-forms/overview/getting-started/getting-started-with-aspnet-45-web-forms/aspnet-error-handling>.

# OLD ASP .Net Error Handling Options



- **At the Page level (which returns user to the page where the error occurred):**
- **At the module/code level (using try ... catch ... finally)**

```
private void Page_Error(object sender, EventArgs e)
{
    Exception exc = Server.GetLastError();

    // Handle specific exception.
    if (exc is HttpUnhandledException)
    {
        ErrorMessageTextBox.Text = "An error occurred on
this page. Please verify your " +
        "information to resolve the issue."
    }
    // Clear the error from the server.
    Server.ClearError();
}
```

```
try
{
    file.ReadBlock(buffer, index, buffer.Length);
}
catch (FileNotFoundException e)
{
    Server.Transfer("NoFileErrorPage.aspx", true);
}
catch (System.IO.IOException e)
{
    Server.Transfer("IOErrorPage.aspx", true);
}

finally
{
    if (file != null)
    {
        file.Close();
    }
}
```

Source: ASP.NET Error Handling. Microsoft Docs. <https://docs.microsoft.com/en-us/aspnet/web-forms/overview/getting-started/getting-started-with-aspnet-45-web-forms/aspnet-error-handling>.

# ASP .NET Core Error Handling



- Provides Exception Handling middleware (remember middleware with dependency injection from HSTS and CORS configuration?)
- Error Handling Docs: <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/error-handling?view=aspnetcore-3.1>

# ASP .NET Core Error Handling Example



```
if (env.IsDevelopment()){
    app.UseDeveloperExceptionPage();
} else {
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

namespace MvcCoreDemo.Controllers
{
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;

        public HomeController(ILogger<HomeController> logger)
        {
            _logger = logger;
        }

        public IActionResult Index()
        {
            throw new Exception("Error occurred");
        }

        public IActionResult Privacy()
        {
            return View();
        }

        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
        public IActionResult Error()
        {
            var exceptionHandlerPathFeature = HttpContext.Features.Get<IExceptionHandlerPathFeature>();
            return View(
                new ErrorViewModel
                {
                    RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier,
                    ErrorMessage = exceptionHandlerPathFeature.Error.Message,
                    Source = exceptionHandlerPathFeature.Error.Source,
                    ErrorPath = exceptionHandlerPathFeature.Path,
                    StackTrace = exceptionHandlerPathFeature.Error.StackTrace,
                    InnerException = Convert.ToString(exceptionHandlerPathFeature.Error.InnerException)
                }
            );
        }
    }
}
```

```
using System;
namespace MvcCoreDemo.Models {
    public class ErrorViewModel {
        public string RequestId { get; set; }
        public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);
        public string ErrorMessage { get; set; }
        public string Source { get; set; }
        public string StackTrace { get; set; }
        public string ErrorPath { get; set; }
        public string InnerException { get; set; }
    }
}

@model ErrorViewModel
@{
    ViewData["Title"] = "Error";
}
<h1 class="text-danger">Error.</h1>
<h2 class="text-danger">An error occurred while processing your request.</h2>
@if (Model.ShowRequestId)
{
    <p>
        <strong>Request ID:</strong> <code>@Model.RequestId</code>
    </p>

    <p>
        <strong>Error Message :</strong> <code>@Model.ErrorMessage</code>
    </p>
    <p>
        <strong>Source :</strong> <code>@Model.Source</code>
    </p>
    <p>
        <strong>ErrorPath :</strong> <code>@Model.ErrorPath</code>
    </p>
    <p>
        <strong>StackTrace :</strong> <code>@Model.StackTrace</code>
    </p>
}
```

Source: <https://www.c-sharpcorner.com/article/asp-net-core-exception-handling/>

# Logging Defensively

# Mature programs log often



- **Logging is pervasive in mature code**
  - “On average, every 30 lines of code contains one line of logging code. Similar density is observed in all the software we studied”
  - They studied widely-used OSS with at least 10 years of development history & large market share (#1 or #2), specifically Apache httpd, OpenSSH, PostgreSQL, and Squid [1].
- **Logging is beneficial for diagnosing production-run failures**
  - “Log messages can speed up the diagnosis time of production-run failures by 2.2 times”

## Sources:

1. D. Yuan, S. Park, and Y. Zhou. Characterizing logging practices in open-source software. In Proceedings of the 34th International Conference on Software Engineering, ICSE'12, pages 102–112, June 2012, [http://opera.ucsd.edu/paper/log\\_icse12.pdf](http://opera.ucsd.edu/paper/log_icse12.pdf).

# Logging and Audit design for Software Systems



- Think about what information is necessary to be logged.
- Debug logs are typically used and implemented by developers of software systems.
- However, there may be requirements to allow the organization using the software system to perform other activities with:
  - Application usage – Who is using the application and at what times
  - Security events – Identity of users that logged in and from which location on the network
- Your information system may also be integrated as part of other systems that exist within an organization.
- See NIST Special Publication 800-92 on Log management:  
<https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-92.pdf>  
(2006).
- Where do these logs go/end up?

# CISA Tool – Logging Made Easy



- Info: <https://www.cisa.gov/resources-tools/services/logging-made-easy>
- GitHub: <https://github.com/cisagov/LME>
- Free and open logging and protective monitoring solution
- Serves as a potential SIEM (Security Information and Event Monitoring) tool



# When to log



- **Logging systems are only useful if the important events are logged**
- **Log all important events, including:**
  - Login, logout, & authorization changes
  - Anything possibly indicating an attack or attempt to work around defenses
- **Categorize messages so operators can configure what gets logged in production. For example:**
  - [Main Application] Started to process the grade files.
  - [File Handler] Accessed and Opened the grade\_file\_1.csv on November 10, 2021 at 10 PM.
  - [File Handler] Finished processing and Deleted the grade files on November 20, 2021 at 10:30 PM.
  - [Main Application] Finished processing all grade files.
  - Here you can only filter on File Handler messages to understand what is happening to just the files.

# If you must roll your own logging/debugging system



- **Record date/time & source**
  - Source = machine & application
  - Sub-second accuracy very helpful
- **Log(category, message)**
- **Allow configuration of:**
  - What to actually record (which categories)
  - Where to send it (file, remote system, etc.)
  - What to do on “log full” (Throw away old? New? Stop running?)
- **Escape and Encode messages**

... but try to reuse a good one instead  
(consider this list a checklist)

# Protect logs

- **Prevent read or write log access by untrusted users**
  - Logs usually sent to *separate* system in operation
- **Logs give away a lot, including:**
  - What you're looking at.. and what you aren't
  - May include sensitive data
- **Logs useful for:**
  - Debugging problems
  - Evidence of attack

# Do *not* include passwords & other sensitive data in logs



- **Logs should normally be private, but:**
  - Sometimes logs will be revealed to others
  - Recipient or recipient's later use may be unauthorized
- **Thus, don't include passwords & very sensitive data in logs**
- **Beware of including data if might include passwords**
  - Ensure URLs don't include passwords!
- **If must include, log encrypted data (or use salted hash)**
- **Example: IEEE log data breach**
  - 99,979 usernames + plaintext (!) passwords
  - Publicly available on their FTP server for at least one month prior to discovery 2012-09-18
  - More info: <https://www.infosecurity-magazine.com/news/ieee-data-breach-offers-up-100k-member-logins/> (26 SEP 2012)

# Improper neutralization of CRLF in Logs: Potential Fix



```
1  string streetAddress = request.getParameter("streetAddress");  
2  
3  if (streetAddress.length() > 150) error();  
4  streetAddress = RemoveCarriageReturns(streetAddress);  
5  
6  logger.info("User's street address: " + streetAddress);
```

**Appropriately filter or quote CRLF sequences in user-controlled input.**

# ASP .NET Logging libraries for use in applications



- Apache Log4Net (<https://logging.apache.org/log4net/>)
  - NLog (<https://nlog-project.org/>)
  - SeriLog (<https://serilog.net/>)
- 
- Logs are often rotated (close and open a new log file when a limit is reached), archived (stored in a compact format somewhere) and compressed.

# Log4J/Log4Shell Explained by Koushik Kothagal



Source: Youtube. JavaBrains Channel. <https://www.youtube.com/watch?v=uyq8yxWO1Is>. Accessed November 14, 2022.

# Log4J/Log4Shell



- Is the vulnerability related to input handling/logging/authentication/?
- What is/are the weakness(es) here?
- Is there not one, but a chain or composite set of weaknesses here?
- How would you fix it?



# CWE Blog



- Did you know that CWE\_CAPEC has a blog?
- Article one Log4Shell: [https://medium.com/@CWE\\_CAPEC/neutralizing-your-inputs-a-log4shell-weakness-story-89954c8b25c9](https://medium.com/@CWE_CAPEC/neutralizing-your-inputs-a-log4shell-weakness-story-89954c8b25c9)

# Next time ...

- Code Analysis

