# ENPM 809W
# Introduction to Secure Software Engineering

Gananand Kini

## Lecture 12

## Error Handling and Logging related security bugs - Defenses

MITRE | SOLVING PROBLEMS FOR A SAFER WORLD™

# 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 ≠ 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

```
boolean DoStuff ()
{
   try
   {
      while (condition == true)
      {
         ThreadLock(TRUE);
         // do some stuff
         // an exception may be thrown
         ThreadLock(FALSE);
      }
   }
   catch (Exception e)
   {
      System.err.println("Something bad happened!");
      return (FAILURE);
   }
   return (SUCCESS);
}
```

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

# CVE-2008-4302 Potential Fix…

```
boolean DoStuff ()
{
   try
   {
      while (condition == true)
      {
         ThreadLock(TRUE);
         // do some stuff
         // an exception may be thrown
         ThreadLock(FALSE);
      }
   }
   catch (Exception e)
   {

      if (isThreadLocked == TRUE) ThreadLock(FALSE);

      System.err.println("Something bad happened!");
      return (FAILURE);
   }
   return (SUCCESS);
}
```

# 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.

# ASP.Net Error Handling Options

- **At the Web.config level:**

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

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

- **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.

# ASP.Net Error Handling Options

- **At the Page level (which returns user to the page where the error occurred):**

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

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

- **At the module/code level (using try … catch … finally)**

```
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.

# 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.

# 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).**

# 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 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: http://ieeelog.com/

# Improper neutralization of CRLF in Logs: Potential Fix

```
string streetAddress = request.getParameter("streetAddress");

if (streetAddress.length() > 150) error();
streetAddress = RemoveCarriageReturns(streetAddress);

logger.info("User's street address: " + streetAddress);
```

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

# ASP .NET Logging libraries

- 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.

# Next time …

- **Debug Mode Code**