

PETER'S INPUT SECURITY INSTALLATION GUIDE



Click on any of these topics to jump to them:

- ◆ Adding Peter's Input Security to a Web Application
 - Add Code Into Global.asax
 - Configure for Your Database
 - Configure for Reporting
 - Configure for Logging
- ◆ Securing The Web Application
 - Showing User Friendly Pages Instead of Exceptions
 - Logging Exceptions
 - Establishing Character Encoding to Limit Script Injection
- ◆ Securing Each Page
- ◆ Troubleshooting
- ◆ Table of Contents

Table of Contents

PETERBLUM.COM AND SECURITY	3
What This Software Will and Will Not Do	3
License Information	4
Platform Support	4
TECHNICAL SUPPORT AND OTHER ASSISTANCE	5
Installation and User's Guides	5
PeterBlum.Com MessageBoard	5
Getting Product Updates	5
Technical Support	5
OVERVIEW	6
Terminology	6
ADDING PETER'S INPUT SECURITY TO A WEB APPLICATION	7
Add Code Into Global.asax	8
Configure for Your Database	9
Configure for Reporting	10
Set up the Reporting Folder	11
Restricting Server Resource Usage	13
Configure for Logging	19
Steps to Configure For Logging	20
Defaults for Tracking Attacks	21
Defaults for Tracking Errors and Exceptions	23
Using the Windows Event Log	25
Using Text File Logging	27
Using Email	29
Using Your Own Logging Code	31
Customize Logging and Response By Attack Type	34
Securing The Web Application	36
Showing User Friendly Pages Instead of Exceptions	37
Logging Exceptions	38
Establishing Character Encoding to Limit Script Injection	40
Securing Each Page	41
TROUBLESHOOTING	42

PeterBlum.com and Security

PeterBlum.com is not a computer security company and does not claim expertise in aspects of security outside of what **Peter's Input Security** addresses. Peter Blum is a programmer who faces the same challenges as other ASP.NET programmers. As Microsoft has promoted the issues behind SQL and script injection attacks, they placed much of the burden on the programmers to solve "input validation". Peter has been able to focus on the problem for an extended period of time to come up with a viable commercial solution for ASP.NET websites. Peter's background includes over 20 years of commercial desktop application development, with extensive OOP, ASP.NET, regex, and database skills. This has been combined with the published knowledge of others in books and articles, many of which you see referenced in this User's Guide, to design Peter's Input Security.

You should expect PeterBlum.com to provide technical support on specific product features and usage.

You should **not** expect PeterBlum.com to provide technical support on general input security issues. PeterBlum.com recommends you seek qualified expertise to answer your general input security issues.

What This Software Will and Will Not Do

Peter's Input Security is designed to greatly reduce the ability for a hacker to attack your website through SQL injection, script injection (aka "Cross Site Scripting") and tampering with inputs from the following sources: HTML form data entry fields, query string parameters and cookies. Even at its highest settings, **it cannot guarantee 100% protection**. Hackers continue to find new ways to attack websites.

This software can only provide its best protection when you carefully follow the directions supplied.

- You are responsible for neutralizing data that was not blocked by this software. The software provides some tools based on a common knowledge of how to defend against hackers. The User's Guide provides additional suggestions.
- Peter's Input Security's validators only look at the data from HTML form data entry fields, hidden fields, query string parameters and cookies. Any other form of input is your responsibility to identify and protect. See the section "Securing a Web Service And Other Inputs" in the User's Guide.
- Peter's Input Security can detect and block most SQL and script injection attacks. However, the settings are highly configurable, allowing you to turn off some or all of the protection. You must write code to neutralize the attacks that get past Peter's Input Security's detection code.
- Peter's Input Security can detect and block some forms of input tampering. You are responsible for detecting, blocking and neutralizing any other case.
- There are many more ways hackers can attack your website. Consult an expert to learn more and determine the best course of action.

License Information

This document includes information for the **Peter's Input Security** module in **Peter's Data Entry Suite**. If you licensed the complete Suite or the "Peter's Input Security" module, you have all features found in this User's Guide, unless otherwise noted.

Platform Support

This product was written for Microsoft ASP.NET. It supports all versions from 1.0 up. It includes assemblies specific to ASP.NET 1.x and ASP.NET 2. It is compatible with all browsers, scaling down automatically when the browser has a limitation. In some cases, that means the control turns off its client-side functionality or turns itself off entirely.

This product is designed to scale properly even when the Page's **ClientTarget** property causes the `HttpBrowserCapabilities (Request.Browser)` to falsely state the browser. In other words, you can't fool these controls with an upLevel clientTarget. This is absolutely necessary because feeding the wrong browser will generate incorrect client side scripts giving the user's scripting errors. It was also considered a requirement to hide features that didn't work on the browser to give the user the best interface. For more, see "Browser Support" in the **General Features Guide**.

Technical Support and Other Assistance

PeterBlum.com offers free technical support. This is just one of the ways to solve problems. This section provides all of your options and explains how technical support is set up.

Installation and User's Guides

These guides are large because they are loaded with content. In many cases, the answers are in them. Both guides include Troubleshooting sections. This information will often save you time.

PeterBlum.Com MessageBoard

Use the message board at <http://groups.yahoo.com/groups/peterblum> to discuss issues and ideas with other users.

Getting Product Updates

As minor versions are released (4.0.1 to 4.0.2 is a minor version release), you can get them for free. Go to <http://www.peterblum.com/DES/Home.aspx>. It will identify the current version at the top of the page. You can read about all changes in the release by clicking "Release History". Click "Get This Update" to get the update. You will need the serial number and email address used to register for the license.

As upgrades are offered (v4.0 to v4.1), PeterBlum.com will determine if there is an upgrade fee at the time. You will be notified of upgrades and how to retrieve them through email.

PeterBlum.com often adds new functionality into minor version releases.

Technical Support

You can contact Technical Support at this email address: Support@PeterBlum.com. I (Peter Blum) make every effort to respond quickly with useful information and in a pleasant manner. As the only person at PeterBlum.com, it is easy to imagine that customer support questions will take up all of my time and prevent me from delivering to you updates and cool new features. As a result, I request the following of you:

- Please review the User's or Installation Guide, including their Troubleshooting sections, first.
- Please try to include as much information about your web form or the problem as possible. I need to fully understand what you are seeing and how you have set things up.
- If you have written code that interacts with my controls or classes, please be sure you have run it through a debugger to determine that it is working in your code or the exact point of failure and error it reports.
- **I am not a security expert.** Tech support cannot provide support on general security issues. See "PeterBlum.com and Security".
- I cannot offer general ASP.NET mentoring. If your problem is due to your lack of knowledge in ASP.NET, I will give you some initial help and then ask you to find assistance from the many tools available to the .Net community. They include:
 - Books
 - <http://www.GotDotNet.com> - for training and many samples on using ASP.NET
 - www.asp.net forums and tutorials
 - Microsoft's usenet newsgroups such as microsoft.public.dotnet.framework.aspnet. See <http://groups.google.com/groups?hl=en&lr=&ie=UTF-8&group=microsoft.public.dotnet>
 - Google searches. (I virtually live in Google as I try to figure things out with ASP.NET.) <http://www.Google.com>. Don't forget to search the "Groups" section of Google!
 - <http://aspnet.4guysfromrolla.com/>, <http://www.dotnetjunkies.com>, <http://www.aspalliance.com/>

As customers identify issues and shortcomings with the software and its documentation, I will consider updating these areas.

Overview

This guide provides detailed step-by-step instructions to setting up **Peter's Input Security**. There are three phases to installation:

1. Adding Peter's Input Security to a Web Application
2. Securing The Web Application
3. Securing Each Page

Use the links to jump around this document. Adobe Reader offers a Previous View command to return to the link. Look for this in the Adobe Reader (shown v6.0):



Terminology

In this documentation, the term **[ProductFolder]** refers to the folder where you installed **Peter's Data Entry Suite**. For example, C:\Program Files\Peters Data Entry Suite v4.0.0.

The term **[webapplicationroot]** refers to the folder that contains the web application on your server. For example, the domain <http://localhost> is usually in C:\inetpub\wwwroot. Web applications are usually in a subfolder. For example, the web app "MyWebApp" is in C:\inetpub\wwwroot\MyWebApp.

Adding Peter's Input Security to a Web Application

Security is a big task. Unfortunately, there is no software that can instantly block what you don't want while permitting what you do. (That would be the "Holy Grail" of input security.) **Peter's Input Security** has been designed to give you the security that works correctly for you. But it comes with some effort. Yet, this effort is *far* smaller than if you had to do it yourself. With **Peter's Input Security**, you have step-by-step guidance to implement security and tools to get many of these tasks done quickly.

There are several major themes to input security:

- Blocking exceptions and diagnostic errors from getting in the hands of hackers which they would exploit
- Logging those exceptions for your own use
- Detecting and blocking hacking attempts on every input of every page in your web application
- Logging the hacking attempts to keep you informed

To accomplish this, **Peter's Input Security** gives you several new tools, all documented in the **Input Security User's Guide**:

- **Security Analysis Report** for determining how secure each page is
- **PageSecurityValidator** for detecting and blocking all forms of attacks on a page
- **FieldSecurityValidator** for detecting and blocking SQL injection and script injection attacks on an individual field
- **LogAndRespond Engine** for logging attacks, exceptions, and errors. It also can redirect to another page or throw an exception after logging.
- **SQL Injection Detection Engine** for detecting SQL injection. It is used by the two validators. You should customize it to understand your database.
- **Script Injection Detection Engine** for detecting script injection. You may customize its filters as you explore the software.

This section will get **Peter's Input Security** set up within a web application. Here are the tasks ahead:

Click on any of these topics to jump to them:

- ◆ Add Code Into Global.aspx
- ◆ Configure for Your Database
- ◆ Configure for Reporting
- ◆ Configure for Logging
- ◆ Securing The Web Application
- ◆ Securing Each Page

Apply each of these in the above order. Do this on each web application.

Add Code Into Global.aspx

Peter's Input Security uses the **Global.aspx** file to set up numerous global settings. It includes the **For Global class.cs** or **For Global class.vb** files with code to copy and paste into your **Global.aspx** file.

1. Open the **Global.aspx** file in an editor. If you use code behind files, open its code behind file.
2. Locate or create the `Application_BeginRequest()` method.

[C#]

```
protected void Application_BeginRequest(Object sender, EventArgs e)
{
}
```

[VB]

```
Sub Application_BeginRequest(ByVal sender As Object, ByVal e As EventArgs)
End Sub
```

Note: Users of Professional Validation And More previously called the `SetupVisualInputSecurity()` from within `Application_Start`. That code can remain until you migrate to IIS 7, which requires the above technique.

3. Open the **For Global class.cs** or **For Global class.vb** file from **[ProductFolder]\Input Security** in an editor.
 - If you use C#, use **For Global class.cs**
 - If you use Visual Basic.Net, use **For Global class.vb**
 - If you use another language, you should be able to adapt the code from one of these files.
4. Follow the directions within those files to update your Global class within the **Global.aspx** file.
 - Add code starting at the `SetupInputSecurity()` method through the end of the file.
 - Add a call to `SetupInputSecurityFromPage()` in `Application_BeginRequest()`.

[C#]

```
protected void Application_BeginRequest(Object sender, EventArgs e)
{
    SetupInputSecurityFromPage();
}
```

[VB]

```
Sub Application_BeginRequest(ByVal sender As Object, ByVal e As EventArgs)
    SetupInputSecurityFromPage()
End Sub
```

WARNING: Do not call the method `SetupInputSecurity()` from `BeginRequest`. `SetupInputSecurity()` loads all configuration settings of Peter's Input Security. That should never happen on every page request!

Configure for Your Database

Peter's Input Security needs to know about your database in order to detect SQL injection attacks against it. You can skip this section if you are not using a database.

In this section, you will identify:

- The type of database that you are using. This will allow Peter's Input Security to detect some of the table and stored procedure names that are pre-installed with your database and often attacked by hackers.
- A list of your own table names and some of the field names likely to be used by hackers. This is used to identify SQL statements, especially those that may be attacks.
- The name of your database. Hackers often discover this name early in their attacks and use it to "drill down" into your database schema.

1. Open the **Global.asax** file in an editor. If you use code behind files, open its code behind file.

2. Within the `SetupInputSecurity()` method, locate this line. (It's near the top):

```
PeterBlum.DES.Security.Globals.UseConfigFiles(  
    PeterBlum.DES.Security.DatabaseTypes.None);
```

3. If you are using a database, change the parameter of `UseConfigFiles()` to identify the database. The enumerated type `PeterBlum.DES.Security.DatabaseTypes` has these values:

- `MSSql` – Microsoft SQL Server
- `MSAccess` – Microsoft Access
- `Oracle`
- `MySql`
- `None` – Use this when no other item applies.

4. Open the **custom.config** file in an editor. The file is in the `[webapplicationfolder]\DES\Security Config Files` folder of your web application.

5. Locate the `<databaseelementnames>` section.

6. Add the name of your database.

```
<item action="add">database name</item>
```

This example uses the Northwind database supplied with MS SQL Server.

```
<item action="add">Northwind</item>
```

7. Add the names of tables from your database. You can choose only those with data that must be protected or all of them.

```
<item action="add">name</item>
```

8. Add the names of the more important fields in your database. Primary key fields and fields used in JOINS are good choices, especially if they are likely to be used by hackers.

Note: Larger lists of items in the `<databaseelementnames>` section will use more CPU time. You must determine the right balance between a complete list and fast processing.

Configure for Reporting

The Security Analysis Report is an important tool within **Peter's Input Security**. It identifies any security risks on the inputs of each page. You will use it to design security on each page. (It is used during development, not in production.)

This report outputs HTML files into a folder that you specify here. Additionally, you can set rules that prevent it from running, such as on a production server. The settings are made within the <appSettings> section of the **web.config** file.

Note: By removing the path to the folder, you turn off reports completely. When you deploy to servers that should not run reports, be sure to remove the path to the folder in the web.config file or programmatically. You can also specify a list of servers that permit reports. See "Restrict by Server Name".

Set up the Reporting Folder

Reports can be written to any folder on the web server, so long as the folder has rights for the NT account used by ASP.NET.

The report files contain sensitive information. Here are some things to consider:

- They should never appear on a production server. You don't want to give hackers a chance to read them and learn the security design of your web pages. When deploying to a production web server, do not set up that server with a Reports folder or configure your **web.config** file to point to that file.
 - On development and testing servers, consider who in your organization should create and see them. Peter's Input Security lets you impose restrictions by location and who should create them.
1. Determine the location of a folder to contain the reporting files. You can put it in any of these places:
 - Within your web application
 - In a shared folder
 - In a private folder
 2. Create a folder there. The recommended name is "**Security Reports**".
 3. Establish the appropriate rights on the folder.
 - If this is within the web application, use IIS to restrict to Read rights only.
 - Folder permissions should allow the ASP.NET user account to read, write, create and delete files.
 - If the folder is not in your web application, you may want to establish it as a shared folder using NT security on the folder.
 4. Establish one of these keys in the <appSettings> section of **web.config**, identifying the path to the folder.

```
<add key="DES_Security_ReportVirtualPath" value="[virtual path]" />
```

```
<add key="DES_Security_ReportFilePath" value="[actual path]" />
```

If you do not have an <appSettings> section in your file, click [here](#) for details.

- Use `DES_Security_ReportVirtualPath` when the folder is in the web application. A virtual path is a URL from the root of the domain folder to the report folder. You can use the tilde (~) as the first character to indicate the web application.

Note: Never supply a domain name or any URL that refers to a different server.

For example, if your domain is `http://localhost`, your web application is `http://localhost/mywebapp` and your report folder is `http://localhost/mywebapp/security reports`, use this:

```
<add key="DES_Security_ReportVirtualPath" value="~/Security Reports" />
```

You have the option of setting this programmatically during application startup. Assign your path to this static/shared property: **PeterBlum.DES.Security.SecurityAnalysisReport.ReportVirtualPath**. It accepts a string.

```
PeterBlum.DES.Security.SecurityAnalysisReport.ReportVirtualPath =  
    "~/Security Reports"
```

- Use `DES_Security_ReportFilePath` when the folder is outside of the web application. If you have set up a shared folder, use a UNC name. Otherwise use a full file path.

Here are examples of each:

```
<add key="DES_Security_ReportFilePath"  
    value="\\mycomputername\security reports" />
```

```
<add key="DES_Security_ReportFilePath"  
    value="C:\documents and settings\all users\security reports" />
```

You have the option of setting this programmatically during application startup. Assign your path to this static/shared property: **PeterBlum.DES.Security.SecurityAnalysisReport.ReportFilePath**. It accepts a string.

```
PeterBlum.DES.Security.SecurityAnalysisReport.ReportFilePath =  
    "\\mycomputername\security reports"
```

C# Users: Remember that the \ character is a special symbol. Either add two of them for each slash or put a @ character in front of the string like this: @"the string".

Restricting Server Resource Usage

When enabled, a page will run a report each time the page is generated. The report takes some time and creates a unique file. This places a higher than usual demand on server resources. Peter's Input Security can be configured to restrict resource usage by establishing these rules to allow a report to run:

- Server name
- IP Address of the user
- The page or folder requested
- Expiration date

Note: Each of these is set in the web.config file. ASP.NET automatically restarts the application after web.config is saved.

Click on any of these topics to jump to them:

- ◆ [Restrict by Server Name](#)
- ◆ [Restrict Users by IP Address](#)
- ◆ [Restrict to specific pages or folders](#)
- ◆ [Letting individual users control pages that output reports](#)
- ◆ [Stop reporting after a certain date](#)

Restrict by Server Name

When you deploy your web application to another server, you must determine if reports should still be enabled. If you do not want them to run, remove the `DES_Security_ReportFilePath` or `DES_Security_ReportVirtualPath` keys from your **web.config** file. What happens if you forget?

Use the `DES_Security_ServersAllowed` key in the `<appSettings>` section of **web.config** to identify a list of server names that allow reports to run. When `ServersAllowed` is not supplied, there are no server name restrictions.

Define server names in a semicolon delimited string like this:

```
<add key="DES_Security_ServersAllowed" value="computername1;computername2" />
```

The software matches these strings to the **System.Environment.MachineName** property. It uses a case insensitive match.

You can include a partial computer name. For example, if you have servers named "Server001", "Server002" and "Server003", you can specify "Server".

You have the option of setting this programmatically during application startup. Assign your path to this static/shared property: **PeterBlum.DES.Security.SecurityAnalysisReport.ServersAllowed**. It accepts a string in the same format as described above.

```
PeterBlum.DES.Security.SecurityAnalysisReport.ServersAllowed =  
    "computername1;computername2"
```

Restrict Users by IP Address

In an environment when several users are accessing the same web application on a server, only a few may be interested in the report feature. Use the `DES_Security_AllowedIPs` key in the `<appSettings>` section of **web.config** to identify a list of IP Addresses that can generate a report. When `DES_Security_AllowedIPs` is not supplied, there are no IP Address restrictions.

Example with one IP Address:

```
<add key="DES_Security_AllowedIPs" value="127.0.0.10" />
```

Multiple IP Addresses can be specified in two ways:

- Semicolon delimited list. For example:

```
<add key="DES_Security_AllowedIPs" value="127.0.0.10;127.0.0.13" />
```

- Partial IP address. Only provide the first few segments of the IP address. All IPs matching those segments are used. Always provide a trailing period. For example:

```
<add key="DES_Security_AllowedIPs" value="127.0.0." />
```

You have the option of setting this programmatically during application startup. Assign your path to this static/shared property: **PeterBlum.DES.Security.SecurityAnalysisReport.AllowedIPs**. It accepts a string in the same format as described above.

```
PeterBlum.DES.Security.SecurityAnalysisReport.AllowedIPs =  
"127.0.0.10;127.0.0.13"
```

Restrict to specific pages or folders

Use the `DES_Security_PagePaths` key in the `<appSettings>` section of **web.config** to identify a list of virtual paths that will generate reports. When used, any page that does not match to this key will not generate a report.

This property looks at the URL from each request and compares it to the entries you make here. You should omit the domain part of the URL. For example, to allow only `http://www.mydomain.com/myfolder/myfile.aspx`, enter `/myfolder/myfile.aspx`.

The value works like there is a wildcard at the end of your text. Suppose you enter `/myfolder` here. This will match to these URLs: `/myfolder/myfile.aspx` and `/myfolder2/myfile.aspx`. If you want to match to all in a specific folder, include the terminating `/` like this: `/myfolder/`.

You can use the tilde (`~`) as the first character to indicate the web application folder. For example, `~/myfolder/`.

You can supply a list of valid page paths, by using semicolon-delimited list. For example, `~/myfolder/;~/myfolder2/;~/myfolder3/myfile1.aspx`.

This example allows all files in the Accounting folder, all files that begin with Save in the Analysis folder, and a single specified page in the Prep folder.

```
<add key="DES_Security_PagePaths"
      value="~/Accounting/;~/Analysis/Save;~/Prep/Collection.aspx" />
```

You have the option of setting this programmatically during application startup. Assign your path to this static/shared property: **PeterBlum.DES.Security.SecurityAnalysisReport.PagePaths**. It accepts a string in the same format as described above.

```
PeterBlum.DES.Security.SecurityAnalysisReport.PagePaths =
    "~/Accounting/;~/Analysis/Save;~/Prep/Collection.aspx"
```

Letting individual users control pages that output reports

You can control which pages output reports using either a query string parameter or the Session collection. When these settings are used, they override the `DES_Security_PagePaths` key (see above).

To allow a page to report, include the query string parameter “DES_EnableReport=1”. To prevent a report, include the query string parameter “DES_EnableReport=0”.

To enable all pages that you use programmatically, assign the “DES_EnableReport” key to the **Session** collection with a value of `true`. To disable all pages that you use programmatically, assign the “DES_EnableReport” key to the Session collection with a value of `false`. You may want to do this by creating a web page designed to set the **Session** collection when it is loaded.

The query string parameter takes precedence over the Session value.

Stop reporting after a certain date

Use the `DES_Security_Expires` key in the `<appSettings>` section of **web.config** to a date after which reports will not run. This makes it easy for the user to run some tests without having to remember to shut off the reporting system later. When `DES_Security_Expires` is not supplied, there is no expiration date.

The value must be in the format `yyyy-MM-dd`. This example shows the expiration date on October 4, 2004.

```
<add key="DES_Security_Expires" value="2004-10-04" />
```

You have the option of setting this programmatically during application startup. Assign your path to this static/shared property: **PeterBlum.DES.Security.SecurityAnalysisReport.Expires**. It accepts a [DateTime](#) structure.

```
PeterBlum.DES.Security.SecurityAnalysisReport.Expires =  
    new DateTime(2004, 10, 2)
```

Configure for Logging

The Log And Respond Engine logs attacks, errors and exceptions from your application and optionally responds by redirecting to another page or throwing an exception. Aside from giving you some very desirable information, it helps hide the real details of an error from a hacker so that they have a harder time creating their attack.

You set up properties on the `PeterBlum.DES.Security.LogAndRespond` class in the `SetupInputSecurity()` method of your **Global.asax** file.

There are four ways you can record an attack, error or exception: the Windows Event Log, a Text File, through Email, and using your own logging code. You can use all four if you like. Here are some guidelines:

- Windows Event Log – When you are on a hosted server, you probably do not have access to the Windows Event Log.
- Text file – Creates a new text file for each date that an attack or error is logged. The same text file will capture all attacks, errors, and exceptions. Each will be time stamped with a header indicating the type of entry. You can set up the Windows Event Log to be a backup to text files in case there are errors writing to text files.
- Email – You must have an SMTP server set up and accessible to the [System.Web.Mail.SmtpMail](#) object (ASP.NET 1.x) or [System.Net.Mail.SmtpClient](#) object (ASP.NET 2.0) . You can define rules to limit the number of emails about an attack for a particular IP Address so that you don't get an email with each attack attempt. However, you will get emails for every error or exception logged. You can set up the Windows Event Log to be a backup to emails in case the `SmtpMail` or `SmtpClient` object throws an exception indicating the server is down.
- Your own code – You can hook up an event handler to direct the data for an attack, error or exception to your own code. Perhaps you want to log into your own database, output to an XML text file, or use an email system that does not use the classes supplied with the .net framework.

Attacks have separate processing rules from errors and exceptions. For example, you may prefer to be emailed about attacks but only log errors. During this setup process, you will define those rules.

Note: The User's Guide section "Using The LogAndRespond Engine" covers the class and its supporting properties and methods in greater detail.

Click on any of these topics to jump to them:

- ◆ Steps to Configure For Logging
- ◆ Defaults for Tracking Attacks
- ◆ Defaults for Tracking Errors and Exceptions
- ◆ Using the Windows Event Log
- ◆ Using Text File Logging
- ◆ Using Email
- ◆ Using Your Own Logging Code
- ◆ Customize Logging and Response By Attack Type

Steps to Configure For Logging

You have already set up **Global.asax** with the `SetupInputSecurity()` method. It contains numerous settings for `LogAndRespond`. They are commented out. The remaining steps will help you decide which to uncomment and modify.

1. Set up properties for logging and responding to attacks on the **LogAndRespond.DefaultTrackAttackArgs** property. See “Defaults for Tracking Attacks”.
2. Set up properties for logging and responding to errors and exceptions on the **LogAndRespond.DefaultTrackErrorArgs** property. See “Defaults for Tracking Errors and Exceptions”.
3. If you are logging to the Windows Event Log, set up the event log. See “Using the Windows Event Log”.
4. If you are logging to a text file, set up the text file folder. See “Using Text File Logging”.
5. If you are sending emails, set up the `System.Web.Mail.SmtpMail` object (ASP.NET 1.x) or `System.Net.Mail.SmtpClient` object (ASP.NET 2.0 and higher). See “Using Email”.
6. If you are using a your own logging code, set up the **LogAndRespond.OnLogAttack** and **OnLogError** properties.
7. The `LogAndRespond` object offers several other properties to customize how attacks are handled:

Note: These properties only apply when `LogAndRespond` is called. The Slow Down Manager monitors attacks but only uses `LogAndRespond` if you set it up to log an attack with the `logfirstattack` attribute on the `SlowDownRules`. Each time it does that, it is counted as one attack by the `LogAndRespond` Engine.

- **AttackTimeout** (integer) – Number of minutes after an attack is recorded before another attack from the same IP Address is considered a new attack. Attacks are tracked by IP address. Each IP Address will record a series of attacks and reset after a delay of this timeout. It defaults to 30 minutes.
- **CountBeforeResponse** (integer) – Number of attacks from a particular IP Address before either the response actions occur. Response actions are redirecting to a URL and throwing an exception. It defaults to 1.

Set it above 1 to give the hacker a false sense that they are not being monitored or blocked after they test your site. You can track their actions and eventually use the responses to forcefully attempt to stop them.

The IP Address count is the total since the web application started up.

- **CustomizeTrackAttackArgs** (event) – Use this event handler to customize the behavior of logging and response for each type of attack. See “Customize Logging and Response By Attack Type”.

Defaults for Tracking Attacks

When tracking attacks, the `LogAndRespond` class follows rules that you set up in its `DefaultTrackAttackArgs` property. By default, all logging and response features are disabled. You set them up in the `SetupInputSecurity()` method.

The `DefaultTrackAttackArgs` property is a `PeterBlum.DES.Security.TrackAttackArgs` class with the following properties:

- **EnableLogging** (boolean) – When `true`, logging is used. You still must set up other properties to determine how attacks will be logged. There are two types of logging available: the Windows Event Log and text file. It defaults to `false`.
- **LoggingText** (string) – The text to write to either the Windows Event Log or text file. It supports tokens that are replaced by actual information about the attack. The tokens are as follows:
 - `{IP}` – IP address from `Request.ServerVariables["REMOTE_ADDR"]`. If `ServerVariables` identifies a proxy server through the `HTTP_VIA` and `HTTP_X_FORWARDED_FOR` variables, they are also embedded into this information using the format:


```
REMOTE_ADDR HTTP_VIA=HTTP_VIA HTTP_X_FORWARDED_FOR=HTTP_X_FORWARDED_FOR
210.123.45.1 HTTP_VIA=210.123.45.1 HTTP_X_FORWARDED_FOR=210.123.45.6
```

 For more information on proxy servers and ways hackers can hide behind them, see http://www.stayinvisible.com/index.pl/anonymity_of_proxy.
 - `{IPTOTAL}` - Total attacks recorded for this IP Address since the web app was started.

Note: IP Addresses may reflect a number of users hidden behind a proxy server.
 - `{USERAGENT}` – The User Agent describing the browser from `Request.ServerVariables["HTTP_USER_AGENT"]`.
 - `{USER}` – Logged in user from `Context.User.Identity.Name`
 - `{URL}` – Complete URL from `Request.Url`
 - `{ATTACKTYPE}` – The type of attack detected: SQL Injection, Script Injection, or `IllegalValue`.
 - `{FIELD}` – ID to the field, cookie or querystring parameter that caused this error.
 - `{INPUTTYPE}` – The type of input attacked: Field, Hidden Field, Cookie, or `QueryString`.
 - `{DETAILS}` – A specific description of what was considered an attack.
 - `{ERRORCODE}` – An error code number associated with the description.
 - `{DATA}` - The text that the user (hacker) entered.

Use the text “\n” to indicate a newline character. *While this is C# syntax, VB.net users should use this syntax too.*

This property defaults to: “`{ATTACKTYPE}\nIP Address: {IP} Total attacks from this address since app started: {IPTOTAL}\nUser Agent: {USERAGENT}\nSite User: {USER}\nURL: {URL}\n{INPUTTYPE}: {FIELD}\nError Code: {ERRORCODE}\nError Details: {DETAILS}\nOffending Text:\n{DATA}`”

- **EnableEmail** (boolean) – When `true`, emailing notices of attacks is enabled. You must have an SMTP Server set up and accessible to the [System.Web.Mail.SmtpMail](#) object (ASP.NET 1.x) or [System.Net.Mail.SmtpClient](#) object (ASP.NET 2 and higher). It defaults to `false`.
- **EmailFrom** (string) – The email address to appear in the **From:** line of an email. Only one is permitted and it must be a valid format.
- **EmailTo** (string) – The email addresses to appear in the **To:** line of an email. Use a semicolon-delimited list for multiple addresses. For example: “`Jon@mydomain.com;Laura@mydomain.com`”
- **EmailSubject** (string) – The email subject line. It defaults to: “`Input validation detected a possible attack`”

- **EmailBody** (string) – The body of the email. It supports the same tokens as shown in the **LoggingText** property.

Use the text “\n” to indicate a newline character. *While this is C# syntax, VB.net users should use this syntax too.*

This property defaults to: “{ATTACKTYPE}\nIP Address: {IP} Total attacks from this address since app started: {IPTOTAL}\nUser Agent: {USERAGENT}\nSite User: {USER}\nURL: {URL}\n{INPUTTYPE}: {FIELD}\nError Code: {ERRORCODE}\nError Details: {DETAILS}\nOffending Text:\n{DATA}”

- **RedirectURL** (string) – The URL to a page that should appear when an attack is detected. When "", redirection is disabled. It defaults to "".
- **ExceptionText** (string) – When assigned and **RedirectURL** is not assigned, throw a `PeterBlum.DES.Security.DESResponseException` with this property’s text as the message.

Example

[C#]

```
public void SetupInputSecurity()
{
    [Code for your License Key here]
    PeterBlum.DES.Security.LogAndRespond vLNR =
        PeterBlum.DES.Security.LogAndRespond.Current;
    vLNR.DefaultTrackAttackArgs.EnableLogging = true;
    vLNR.DefaultTrackAttackArgs.EnableEmail = true;
    vLNR.DefaultTrackAttackArgs.EmailFrom = "You@YourServer.com";
    vLNR.DefaultTrackAttackArgs.EmailTo =
        "You@YourServer.com;Boss@YourServer.com";
    vLNR.DefaultTrackAttackArgs.RedirectURL = "/GoToJail.aspx";

    // in this example, LoggingText, EmailSubject, EmailBody and
    // Exception Text remain at their defaults
}
```

[VB]

```
Public Sub SetupInputSecurity()
    [Code for your License Key here]
    Dim vLNR As PeterBlum.DES.Security.LogAndRespond = _
        PeterBlum.DES.Security.LogAndRespond.Current
    vLNR.DefaultTrackAttackArgs.EnableLogging = True
    vLNR.DefaultTrackAttackArgs.EnableEmail = True
    vLNR.DefaultTrackAttackArgs.EmailFrom = "You@YourServer.com"
    vLNR.DefaultTrackAttackArgs.EmailTo = _
        "You@YourServer.com;Boss@YourServer.com"
    vLNR.DefaultTrackAttackArgs.RedirectURL = "/GoToJail.aspx"
    ' in this example, LoggingText, EmailSubject, EmailBody, and
    ' Exception Text remain at their defaults
End Sub
```

Defaults for Tracking Errors and Exceptions

When tracking errors and exceptions, the `LogAndRespond` class follows rules that you set up in its

DefaultTrackErrorArgs property. By default, all logging and response features are disabled. You set them up in the `SetupInputSecurity()` method.

The **DefaultTrackErrorArgs** property is a `PeterBlum.DES.Security.TrackErrorArgs` class with the following properties:

- **EnableLogging** (boolean) – When `true`, logging is used. You still must set up other properties to determine how errors will be logged. There are two types of logging available: the Windows Event Log and text file. It defaults to `false`.
- **LoggingText** (string) – The text to write to either the Windows Event Log or text file. It supports tokens that are replaced by actual information about the error. The tokens are as follows:
 - {IP} – IP address from `Request.ServerVariables["REMOTE_ADDR"]`. If `ServerVariables` identifies a proxy server through the `HTTP_VIA` and `HTTP_X_FORWARDED_FOR` variables, they are also embedded into this information using the format:


```
REMOTE_ADDR HTTP_VIA=HTTP_VIA HTTP_X_FORWARDED_FOR=HTTP_X_FORWARDED_FOR
210.123.45.1 HTTP_VIA=210.123.45.1 HTTP_X_FORWARDED_FOR=210.123.45.6
```

 For more information on proxy servers and ways hackers can hide behind them, see http://www.stayinvisible.com/index.pl/anonymity_of_proxy.
 - {USERAGENT} – The User Agent describing the browser from `Request.ServerVariables["HTTP_USER_AGENT"]`.
 - {USER} – Logged in user from `Context.User.Identity.Name`
 - {URL} – Complete URL from `Request.Url`
 - {DETAILS} – The text of the error message. When you record an error, you supply this text. When there is an exception, `LogAndRespond.TrackException()` builds this text from the Exception object.
 - {ERRORCODE} – An error code number that you can supply. You define your own error codes. When the error code is 0, this appears as “n/a”.

Use the text “\n” to indicate a newline character. *While this is C# syntax, VB.net users should use this syntax too.*

This property defaults to: “IP Address: {IP}\nUser Agent: {USERAGENT}\nSite User: {USER}\nURL: {URL}\nError Code: {ERRORCODE}\n{DETAILS}”

- **EnableEmail** (boolean) – When `true`, emailing notices of errors is enabled. You must have an SMTP Server set up and accessible to the [System.Web.Mail.SmtpMail](#) object (ASP.NET 1.x) or [System.Net.Mail.SmtpClient](#) object (ASP.NET 2 and higher). It defaults to `false`.
- **EmailFrom** (string) – The email address to appear in the **From:** line of an email. Only one is permitted and it must be a valid format.
- **EmailTo** (string) – The email addresses to appear in the **To:** line of an email. Use a semicolon-delimited list for multiple addresses. For example: “Jon@mydomain.com;Laura@mydomain.com”
- **EmailSubject** (string) – The email subject line. It defaults to: “An error has been recorded in your web application”
- **EmailBody** (string) – The body of the email. It supports the same tokens as shown in the **LoggingText** property.

Use the text “\n” to indicate a newline character. *While this is C# syntax, VB.net users should use this syntax too.*

This property defaults to: “IP Address: {IP}\nUser Agent: {USERAGENT}\nSite User: {USER}\nURL: {URL}\nError Code: {ERRORCODE}\n{DETAILS}”

- **RedirectURL** (string) – The URL to a page that should appear when an error is detected. When “”, redirection is disabled. It defaults to “”.

- **ExceptionText** (string) – When assigned and **RedirectURL** is not assigned, throw a `PeterBlum.DES.Security.DESResponseException` with this property's text as the message.

Note: This property is not used when handling an exception through `LogAndRespond.TrackException()`.

Example

[C#]

```
public void SetupInputSecurity()
{
    [Code for your License Key here]
    PeterBlum.DES.Security.LogAndRespond vLNR =
        PeterBlum.DES.Security.LogAndRespond.Current;
    [Code for DefaultTrackAttackArgs is here]
    vLNR.DefaultTrackErrorArgs.EnableLogging = true;
    vLNR.DefaultTrackErrorArgs.EnableEmail = true;
    vLNR.DefaultTrackErrorArgs.EmailFrom = "You@YourServer.com";
    vLNR.DefaultTrackErrorArgs.EmailTo =
        "You@YourServer.com;Boss@YourServer.com";
    // in this example, LoggingText, EmailSubject, EmailBody,
    // RedirectURL, and ExceptionText remain at their defaults
}
```

[VB]

```
Public Sub SetupInputSecurity()
    [Code for your License Key here]
    Dim vLNR As PeterBlum.DES.Security.LogAndRespond = _
        PeterBlum.DES.Security.LogAndRespond.Current
    [Code for DefaultTrackAttackArgs is here]
    vLNR.DefaultTrackErrorArgs.EnableLogging = True
    vLNR.DefaultTrackErrorArgs.EnableEmail = True
    vLNR.DefaultTrackErrorArgs.EmailFrom = "You@YourServer.com"
    vLNR.DefaultTrackErrorArgs.EmailTo = _
        "You@YourServer.com;Boss@YourServer.com"
    ' in this example, LoggingText, EmailSubject, EmailBody,
    ' RedirectURL, and ExceptionText remain at their defaults
End Sub
```

Using the Windows Event Log

Note: You must decide if Event logging is right for you. You can skip this section if it is not. It is recommended to set it up as a backup to other logging systems. If you are using a hosted site's computer, the event log should be considered off limits. So use file logging instead.

Note: You can come back to this section before deploying to production.

ALERT: If your server is set up as a Partial Trust Environment, this feature requires the `EventLogPermission` for your app. See the section "Installing Into a Partial Trust Environment" in the main **Installation Guide**.

The Windows Event Log provides an effective tool for collecting information on attacks, errors and exceptions. If your site is on a hosted server, the Windows Event Log may not be accessible to you. Otherwise, it is recommended that you use it as a backup to other logging methods in case they fail. It may also be used as a primary logging method.

A Windows Event Log requires that you define an "Event Log Source". This is a name associated with the application that is recording into the log. You can put this name into any of the three standard log groups: Application, System, or Security.

1. Select a name for your event log source. A suggested name is "Peter's Input Security".
2. Create an event log source.

If your web application has rights to write to the registry (which is not common on production servers), you can use the `System.Diagnostics.EventLog.CreateEventSource()` method as shown in the msdn help topic [here](#). Add the code into your `SetupInputSecurity()` method.

If your web application does not have rights to write to the registry, you must edit the Windows registry yourself. Here are the steps:

- a. The event log must already be defined in the registry under:
`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\EventLog`
- b. Use one of these nodes: "Application", "System", or "Security".
- c. Under the selected node, create a node with the name of the Source, such as "Peter's Input Security".
- d. Create a String-type key whose name is "EventMessageFile" and value is [Windows folder]\Microsoft.NET\Framework\[.net version]\EventLogMessages.dll.

For example:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\EventLog\Application\
Peter's Input Security\
```

```
Key name = EventMessageFile
```

```
Value = C:\Windows\Microsoft.NET\Framework\v1.1.4322\EventLogMessages.dll
```

3. Create the `DES_Security_EventLogSource` key with a value of the Event Log Source in the `<appSettings>` section of the **web.config** file. For example:

```
<add key="DES_Security_EventLogSource" value="Peters Input Security" />
```

You have the option of setting this programmatically during application startup. Assign your path to this static/shared property: **PeterBlum.DES.Security.LogAndRespond.EventLogSource**. It accepts a string.

```
PeterBlum.DES.Security.LogAndRespond.EventLogSource =
    "Peters Input Security"
```

4. Within `SetupInputSecurity()`, call the `LogAndRespond.UseEventLog()` method with no parameters. It returns `false` if `DES_Security_EventLogSource` was not found in the **web.config** file. It will throw an exception if it could not create an entry in the event log. By throwing an exception as the application starts up, you will immediately know about a setup problem.
5. By default, the Windows Event Log will be used for all attacks, exceptions, and errors. If you want it to be a backup log when the text file log or email fail, set the **LogAndRespond.EventLogIsBackup** property to `true`.

Example

[C#]

```
public void SetupInputSecurity()
{
    [Code for your License Key here]
    PeterBlum.DES.Security.LogAndRespond vLNR =
        PeterBlum.DES.Security.LogAndRespond.Current;
    [Code for DefaultTrackAttackArgs is here]
    vLNR.DefaultTrackAttackArgs.EnableLogging = true;
    [Code for DefaultTrackErrorArgs is here]
    vLNR.DefaultTrackErrorArgs.EnableLogging = true;
    if (!vLNR.UseEventLog())
        throw new Exception("Need to setup the DES_Security_EventLogSource in web.config.");
    vLNR.EventLogIsBackup = true; // optional
}
```

[VB]

```
Public Sub SetupInputSecurity()
    [Code for your License Key here]
    Dim vLNR As PeterBlum.DES.Security.LogAndRespond = _
        PeterBlum.DES.Security.LogAndRespond.Current
    [Code for DefaultTrackAttackArgs is here]
    vLNR.DefaultTrackAttackArgs.EnableLogging = True
    [Code for DefaultTrackErrorArgs is here]
    vLNR.DefaultTrackErrorArgs.EnableLogging = True
    If Not vLNR.UseEventLog() Then
        Throw New Exception("Need to setup the DES_Security_EventLogSource in web.config.")
    End If
    vLNR.EventLogIsBackup = True ' optional
End Sub
```

Using Text File Logging

There are several benefits to logging to a text file as compared to the Windows Event Log:

- Works on a hosted server environment
- Grouping by date as Peter's Input Security creates a new file for each date
- Ability to archive files and delete individual dates

Note: You must decide if Text File logging is right for you. You can skip this section if it is not.

Note: You can come back to this section before deploying to production.

Peter's Input Security needs a folder where it will write the log files. These steps will show you how to set up the folder.

1. Select a location on the server where the files will be written.
2. Create a folder for logging in that location.
3. If the folder is accessible through IIS, use IIS to remove any rights to access logs through the web. Specifically, there should be no permissions to Read or Write.
4. The folder must have NT permissions on the NT account used by ASP.NET for: create files, delete files, read, and write.
5. Establish one of these keys in the <appSettings> section of **web.config**, identifying the path to the folder.

```
<add key="DES_Security_LogVirtualPath" value="[virtual path]" />
<add key="DES_Security_LogFilePath" value="[actual path]" />
```

- Use `DES_Security_LogVirtualPath` when the folder is in the web application. A virtual path is a URL from the root of the domain folder to the logging folder. You can use the tilde (~) as the first character to indicate the web application.

For example, if your domain is `http://localhost`, your web application is `http://localhost/mywebapp` and your logging folder is `http://localhost/mywebapp/errorlogs`, use this:

```
<add key="DES_Security_LogVirtualPath" value="~/ErrorLogs" />
```

You have the option of setting this programmatically during application startup. Assign your path to this static/shared property: **PeterBlum.DES.Security.LogAndRespond.LogVirtualPath**. It accepts a string.

```
PeterBlum.DES.Security.LogAndRespond.LogVirtualPath = "~/ErrorLogs"
```

- Use `DES_Security_LogFilePath` when the folder is outside of the web application. If you have set up a shared folder, use a UNC name. Otherwise use a full file path.

Here are examples of each:

```
<add key="DES_Security_LogFilePath" value="\\mycomputername\ErrorLogs" />
<add key="DES_Security_LogFilePath"
  value="C:\documents and settings\all users\ErrorLogs" />
```

You have the option of setting this programmatically during application startup. Assign your path to this static/shared property: **PeterBlum.DES.Security.LogAndRespond.LogFilePath**. It accepts a string.

```
PeterBlum.DES.Security.LogAndRespond.LogFilePath =
  "\\mycomputername\ErrorLogs"
```

6. Within `SetupInputSecurity()`, call the `LogAndRespond.UseFileLog()` method with no parameters. It returns `false` if `DES_Security_LogVirtualPath` and `DES_Security_LogFilePath` did not supply any text. It will throw an exception if it could not create a file in the logging folder. By throwing an exception as the application starts up, you will immediately know about a setup problem.
7. Within the `SetupInputSecurity()` method, confirm that these properties on [LogAndRespond.DefaultTrackAttackArgs](#) and [LogAndRespond.DefaultTrackErrorArgs](#) have been set.
 - **EnableLogging** – Set to `true`.

- **LoggingText** – This has a default that you can change.

Example

[C#]

```
public void SetupInputSecurity()
{
    [Code for your License Key here]
    PeterBlum.DES.Security.LogAndRespond vLNR =
        PeterBlum.DES.Security.LogAndRespond.Current;
    [Code for DefaultTrackAttackArgs is here]
    vLNR.DefaultTrackAttackArgs.EnableLogging = true;
    [Code for DefaultTrackErrorArgs is here]
    vLNR.DefaultTrackErrorArgs.EnableLogging = true;
    vLNR.UseEventLog();
    if (!vLNR.UseFileLog())
        throw new Exception("Need to setup the DES_Security_LogVirtualPath
            or DES_Security_LogFilePath key in web.config.");
}
```

[VB]

```
Public Sub SetupInputSecurity()
    [Code for your License Key here]
    Dim vLNR As PeterBlum.DES.Security.LogAndRespond = _
        PeterBlum.DES.Security.LogAndRespond.Current
    [Code for DefaultTrackAttackArgs is here]
    vLNR.DefaultTrackAttackArgs.EnableLogging = True
    [Code for DefaultTrackErrorArgs is here]
    vLNR.DefaultTrackErrorArgs.EnableLogging = True
    vLNR.UseEventLog()
    If Not vLNR.UseFileLog() Then
        Throw New Exception("Need to setup the DES_Security_LogVirtualPath
            or DES_Security_LogFilePath key in web.config.");
    End If
End Sub
```

Using Email

Note: You must decide if Emailing is right for you. You can skip this section if it is not.

Note: You can come back to this section before deploying to production.

Email requires a correctly setup SMTP server to which the [System.Net.Mail.SmtpClient](#) object (ASP.NET 2 and higher) or [System.Web.Mail.SmtpMail](#) object (ASP.NET 1.x) properly connects. When tracking attacks, you have options to avoid sending emails on every attack.

1. Confirm that you have a working SMTP server that is accessible from the web server. The web server computer should be configured with the correct account information.

Note: PeterBlum.com cannot provide technical support on getting ASP.NET to communicate with the SMTP Server. Peter's Input Security uses the System.Web.Mail.SmtpMail or System.Net.Mail.SmtpClient object, just like you would if you wanted to send email from your code.

2. Make sure the Smtp class is configured with the server name and any other attributes.

ASP.NET 2 and higher Users

The [System.Net.Mail.SmtpClient](#) class is configured either in **machine.config** or **web.config** using this basic syntax:

```
<configuration>
  <system.net>
    <mailSettings>
      <smtp>
        <network host="email server" />
      </smtp>
    </mailSettings>
  </system.net>
</configuration>
```

Set the host attribute to the name of the server, such as "mail.myserver.com".

There are other attributes available to the <network> tag, such as username, password, and port. Please refer to the .net documentation if you need to use any of them.

ASP.NET 1.x Users

Within the `SetupInputSecurity()` method, set the [SmtpMail.SmtpServer](#) property to the desired server if needed.

Note: If you are using SmtpMail for other things, you probably have set up SmtpMail.SmtpServer in Application_Start(). You do not need to relocate or modify it.

3. Within the `SetupInputSecurity()` method, confirm that these properties on [LogAndRespond.DefaultTrackAttackArgs](#) and [LogAndRespond.DefaultTrackErrorArgs](#) have been set.
 - **EnableEmail** – Set to `true`.
 - **EmailFrom** and **EmailTo** – Set to the appropriate email addresses.
 - **EmailSubject** and **EmailBody** – These have defaults that you can change.
4. If you want to avoid emails on every attack from a specific IP Address, set these properties within the `SetupInputSecurity()` method.
 - **FirstEmailAfterThisManyAttacks** (integer) – Determines how many attacks within a period before sending the first email. Minimum value of 1. It defaults to 1.

The "attack period" is a time limit between two separate attacks from the same IP Address and is set with the **LogAndRespond.AttackTimeOut** property, described elsewhere.
 - **MoreEmailsAfterThisManyAttacks** (integer) – After the first email has been sent, you can have emails sent on later attacks. When 0, no further emails are sent. When 1, emails are sent with every attack. When higher than 1, it

waits for this many attacks before emailing again. For example, if this is assigned to 3, it will email you on the 3rd, 6th, 9th, etc attack following the first attack. When the attack period is exceeded, it resets counting for emails and will use **FirstEmailAfterThisManyAttacks** on the next attack.

Example

[C#]

```
public void SetupInputSecurity()
{
    [Code for your License Key here]
    PeterBlum.DES.Security.LogAndRespond vLNR =
        PeterBlum.DES.Security.LogAndRespond.Current;
    [Code for DefaultTrackAttackArgs is here]
    vLNR.DefaultTrackAttackArgs.EnableEmail = true;
    vLNR.DefaultTrackAttackArgs.EmailFrom = "me@mydomain.com";
    vLNR.DefaultTrackAttackArgs.EnableEmail = "me@mydomain.com;boss@mydomain.com";
    [Code for DefaultTrackErrorArgs is here]
    vLNR.DefaultTrackErrorArgs.EnableEmail = true;
    vLNR.DefaultTrackErrorArgs.EmailFrom = "me@mydomain.com";
    vLNR.DefaultTrackErrorArgs.EnableEmail = "me@mydomain.com;boss@mydomain.com";
    vLNR.UseEventLog();
    vLNR.UseFileLog();
    System.Web.Mail.SmtpMail.SmtpServer = "mail.mydomain.com"; //ASP.NET 1.x ONLY!
    vLNR.FirstEmailAfterThisManyAttacks = 3;
    vLNR.MoreEmailsAfterThisManyAttacks = 10;
}
```

[VB]

```
Public Sub SetupInputSecurity()
    [Code for your License Key here]
    Dim vLNR As PeterBlum.DES.Security.LogAndRespond = _
        PeterBlum.DES.Security.LogAndRespond.Current
    [Code for DefaultTrackAttackArgs is here]
    vLNR.DefaultTrackAttackArgs.EnableEmail = True
    vLNR.DefaultTrackAttackArgs.EmailFrom = "me@mydomain.com"
    vLNR.DefaultTrackAttackArgs.EnableEmail = "me@mydomain.com;boss@mydomain.com"
    [Code for DefaultTrackErrorArgs is here]
    vLNR.DefaultTrackErrorArgs.EnableEmail = True
    vLNR.DefaultTrackErrorArgs.EmailFrom = "me@mydomain.com"
    vLNR.DefaultTrackErrorArgs.EnableEmail = "me@mydomain.com;boss@mydomain.com"
    vLNR.UseEventLog()
    vLNR.UseFileLog()
    System.Web.Mail.SmtpMail.SmtpServer = "mail.mydomain.com" 'ASP.NET 1.x ONLY!
    vLNR.FirstEmailAfterThisManyAttacks = 3
    vLNR.MoreEmailsAfterThisManyAttacks = 10
End Sub
```

Using Your Own Logging Code

Peter's Input Security can pass all of the details about an attack, error, or exception to your own logging code through event handler methods. To log attacks, attach an event handler method to **LogAndRespond.OnLogAttack**. To log errors and exceptions, attach an event handler method to **LogAndRespond.OnLogError**.

Use these features when you have another solution for logging, such as a database. You can even use it to substitute for the emailing or text file systems supplied by the Log And Respond Engine, simply by defining your own code and disabling the feature in [LogAndRespond.DefaultTrackAttackArgs](#) and [LogAndRespond.DefaultTrackErrorArgs](#).

Note: You can come back to this section before deploying to production.

You have already established two event handler methods in **Global.asax**. They are `LogAttacks()` and `LogErrors()`. Here are the steps to set these methods up.

1. Within the `SetupInputSecurity()` method, locate the commented-out code that sets **OnLogAttack** and **OnLogErrors** to the `LogAttacks()` and `LogErrors()` methods.
2. Remove the comments on those lines.
3. Locate the `LogAttacks()` method below the `SetupInputSecurity()` method.
4. Add your own code to `LogAttacks()`. The parameters are:
 - `pPage` – The Page object that is associated with this error.
 - `pAttackDetails` – Describes the attack in detail. See the topic “AttackDetails Class” in the User’s Guide.
 - `pIPInfo` – The `IPAddressInfo` class describes the IP Address that requested the page along with the total attacks associated with this IP Address. Its properties are:
 - **IPAddress** (string) – The IP address from `Request.ServerVariables["REMOTE_ADDR"]`
 - **TotalAttacks** (integer) – The total attacks recorded since the web application started up
 - **AttacksInPeriod** (integer) – The total attacks for the period defined in **LogAndRespond.AttackTimeOut**.
 - `pArgs` – Describes the `TrackAttackArgs` in use. They have the same properties as shown in [LogAndRespond.DefaultTrackAttackArgs](#).
5. Locate the `LogErrors()` method below the `SetupInputSecurity()` method.
6. Add your own code to `LogErrors()`. The parameters are:
 - `pPage` – The Page object that is associated with this error.
 - `pErrorDetails` – A description of the error.
 - `pErrorCode` – An error code for the error. If 0, no error code was defined.
 - `pArgs` – Describes the `TrackErrorArgs` in use. They have the same properties as shown in [LogAndRespond.DefaultTrackErrorArgs](#).

Example

[C#]

```
public void SetupInputSecurity()
{
    [Code for your License Key here]
    PeterBlum.DES.Security.LogAndRespond vLNR =
        PeterBlum.DES.Security.LogAndRespond.Current;
    [Code for DefaultTrackAttackArgs is here]
    [Code for DefaultTrackErrorArgs is here]
    vLNR.UseEventLog();
    vLNR.UseFileLog();
    [Code for email]
    vLNR.OnLogAttack = new PeterBlum.DES.Security.LogAttackEventHandler(LogAttacks);
    vLNR.OnLogError = new PeterBlum.DES.Security.LogErrorEventHandler(LogErrors);
}

public void LogAttacks(System.Web.UI.Page pPage,
    PeterBlum.DES.Security.AttackDetails pAttackDetails,
    PeterBlum.DES.Security.IPAddressInfo pIPInfo,
    PeterBlum.DES.Security.TrackAttackArgs pArgs)
{
    // Add your code here
} // LogAttacks

public void LogErrors(System.Web.UI.Page pPage, string pErrorDetails,
    int pErrorCode, PeterBlum.DES.Security.TrackErrorArgs pArgs)
{
    // Add your code here
} // LogErrors
```

[VB]

```
Public Sub SetupInputSecurity()
    [Code for your License Key here]
    Dim vLNR As PeterBlum.DES.Security.LogAndRespond = _
        PeterBlum.DES.Security.LogAndRespond.Current
    [Code for DefaultTrackAttackArgs is here]
    [Code for DefaultTrackErrorArgs is here]
    vLNR.UseEventLog()
    vLNR.UseFileLog()
    [Code for email]
    vLNR.OnLogAttack = New PeterBlum.DES.Security.LogAttackEventHandler( _
        AddressOf LogAttacks)
    vLNR.OnLogError = New PeterBlum.DES.Security.LogErrorEventHandler( _
        AddressOf LogErrors)
End Sub

Public Sub LogAttacks(ByVal pPage As System.Web.UI.Page, _
    ByVal pAttackDetails As PeterBlum.DES.Security.AttackDetails, _
    ByVal pIPInfo As PeterBlum.DES.Security.IPAddressInfo, _
    ByVal pArgs As PeterBlum.DES.Security.TrackAttackArgs)
    ' Add your code here
End Sub ' LogAttacks
```

```
Public Sub LogErrors(ByVal pPage As System.Web.UI.Page, _
    ByVal pErrorDetails As String, _
    ByVal pErrorCode As Integer, _
    ByVal pArgs As PeterBlum.DES.Security.TrackErrorArgs)
    ' Add your code here
End Sub ' LogErrors
```

Customize Logging and Response By Attack Type

When `LogAndRespond` records an attack, it determines the logging and response actions from the [LogAndRespond.DefaultTrackAttackArgs](#) object. Suppose you want to change these actions based on the type of attack. Here are some examples:

- Send emails to different recipients. For a SQL Injection attack, notify the Database Administrator. For others, notify the Web Master.
- Respond differently. Send SQL Injection attacks to one page; Script Injections to another.

Use the `LogAndRespond.CustomizeTrackAttackArgs` event handler to preprocess the `TrackAttackArgs` object. It gives you all of the information about the attack and allows you to modify the `TrackAttackArgs` as needed.

The text you added to `Global.asax` already defines a method, `CustomizeArgs()`, and attaches it to the `LogAndRespond.CustomizeTrackAttackArgs` event.

What follows is the definition of the method and an example of how to modify it.

Note: You will probably leave it at its default for now and return here to customize it as needed.

Definition: `PeterBlum.DES.Security.CustomTrackAttackArgsHandler Delegate`

[C#]

```
delegate public void CustomizeTrackAttackArgsHandler(
    ref PeterBlum.DES.Security.AttackType pAttackType,
    ref string pAttackTypeDescription,
    ref PeterBlum.DES.Security.AttackInputType pAttackInputType,
    ref string pAttackInputTypeDescription,
    ref string pErrorDetails,
    PeterBlum.DES.Security.TrackAttackArgs pArgs)
```

[VB]

```
Delegate Public Sub CustomizeTrackAttackArgsHandler( _
    ByRef pAttackType As PeterBlum.DES.Security.AttackType, _
    ByRef pAttackTypeDescription As String, _
    ByRef pAttackInputType As PeterBlum.DES.Security.AttackInputType, _
    ByRef pAttackInputTypeDescription As String, _
    ByRef pErrorDetails As String, _
    ByVal pArgs As PeterBlum.DES.Security.TrackAttackArgs)
```

Parameters

pAttackType

The type of attack. It will be one of these values from the enumerated type `PeterBlum.DES.Security.AttackType`:

- Unknown
- SQLInjection
- ScriptInjection
- IllegalValue

pAttackTypeDescription

A description of the attack type. This text will appear in the `{ATTACKTYPE}` token of `pArgs.LoggingText` and `pArgs.EmailBody`.

pAttackInputType

The type of input that was attacked. It will be one of these values from the enumerated type `PeterBlum.DES.Security.AttackInputType`:

- Unknown
- Field
- HiddenField
- QueryString
- Cookie

pAttackInputTypeDescription

A description of the input type. This text will appear in the {INPUTTYPE} token of **pArgs.LoggingText** and **pArgs.EmailBody**.

pErrorDetails

Textual description of what was detected to indicate an attack. This text will appear in the {DETAILS} token of **pArgs.LoggingText** and **pArgs.EmailBody**.

pArgs

The `PeterBlum.DES.Security.TrackAttackArgs` object that will determine the logging and response actions that follow. You will usually modify the properties of this object. The defaults come from **LogAndRespond.DefaultTrackAttackArgs**. See “Defaults for Tracking Attacks”.

Example

This example will change the email address for SQLInjection attacks and the RedirectURL for an attack on visible fields.

[C#]

```
public void CustomizeArgs(
    ref PeterBlum.DES.Security.AttackType pAttackType,
    ref string pAttackTypeDescription,
    ref PeterBlum.DES.Security.AttackInputType pAttackInputType,
    ref string pAttackInputTypeDescription,
    ref string pErrorDetails,
    PeterBlum.DES.Security.TrackAttackArgs pArgs)
{
    if (pAttackType == PeterBlum.DES.Security.AttackType.SQLInjection)
        pArgs.EmailTo = "dave_DBA@mydomain.com";
    if (pAttackInputType == PeterBlum.DES.Security.AttackInputType.Field)
        pArgs.RedirectURL = "/MyErrors/VisibleFieldResponse.aspx";
}
```

[VB]

```
Public Sub CustomizeArgs( _
    ByRef pAttackType As PeterBlum.DES.Security.AttackType, _
    ByRef pAttackTypeDescription As String, _
    ByRef pAttackInputType As PeterBlum.DES.Security.AttackInputType, _
    ByRef pAttackInputTypeDescription As String, _
    ByRef pErrorDetails As String, _
    ByVal pArgs As PeterBlum.DES.Security.TrackAttackArgs)

    If pAttackType = PeterBlum.DES.Security.AttackType.SQLInjection Then
        pArgs.EmailTo = "dave_DBA@mydomain.com"
    End If
    If pAttackInputType = PeterBlum.DES.Security.AttackInputType.Field Then
        pArgs.RedirectURL = "/MyErrors/VisibleFieldResponse.aspx"
    End If
End Sub
```

Securing The Web Application

This section installs several features that experts recommend for a secure site.

Click on any of these topics to jump to them:

- ◆ Showing User Friendly Pages Instead of Exceptions
- ◆ Logging Exceptions
- ◆ Establishing Character Encoding to Limit Script Injection

Showing User Friendly Pages Instead of Exceptions

Error messages from exceptions are one of the most useful tools for hackers to make progress. For example, when they create a SQL Injection attack on a field, if their text manages to get to the database, most of the time the database will find their text is invalid and throw an exception that details the exact problem. If that exception is provided to the user, the hacker will know more about how to correct their SQL to get what they want.

As a rule, you should log exceptions for your own use and give the user friendly but nonspecific errors.

This section helps you set up a user friendly page and hook it into ASP.NET's own mechanism for showing pages instead of errors.

The next section helps you use Peter's Input Security's LogAndRespond class to log exceptions and select alternative pages for different errors.

ASP.NET has already built a mechanism to block showing errors. These steps will make sure it is set up:

1. Open the **web.config** file and locate the section `<customErrors>` within `<system.web>`. Click [here](#) for details on this section of the **web.config** file.
2. If it is set up, it will have the attribute `Mode` set to "On" or "RemoteOnly". It will have the `defaultRedirect` attribute set to a URL.
3. If the `Mode` and `defaultRedirect` attributes are set up correctly, confirm that the URL in `defaultRedirect` points to an existing web page. If it does, you can skip to step 6.
4. Create a web form (aspx) file to use as the default error page. The information on this page should never tell you the exact error encountered. A friendly message like this will do "We are sorry. You have encountered an error on this site. Use the Back button on your browser to return to the previous page."

Note: Getting the page right may take some time. You can build a basic page now and return later to enhance it.

Suggestion: You may create several web forms for the different types of errors. Consider making a folder in your web application for all of these web forms.

5. Set up the `<customErrors>` section to use your new web form. Click [here](#) for details.

This is an example of the `<customErrors>` section:

```
<configuration>
  <system.web>
    <customErrors defaultRedirect="/ErrorPages/Default.aspx"
                 mode="RemoteOnly">
    </customErrors>
  </system.web>
</configuration>
```

6. Create any additional error pages as you think of them. They will be used in the `Application_Error()` method as described in the following section. Some examples:
 - Email server is down. Please try again shortly.
 - File cannot be downloaded due to an issue on the server. Please contact the web master. (Use this when downloading a file and you have a file system error that prevents saving it. Hide the details.)
 - Site is temporarily down for maintenance. Please try again shortly. (Use this kind of message to handle errors when the database server is shut down. It hides the real reason: the user doesn't need to know.)

Logging Exceptions

As a rule, you should log exceptions for your own use and give the user friendly but nonspecific errors. The `LogAndRespond` class helps you do this. Use its `TrackException()` method when you catch an exception. It will take logging and response actions based on the [LogAndRespond.DefaultTrackErrorArgs](#) object.

There are three places you can use `TrackException()`.

- Within code that throws exceptions. You will write **try...catch** statements and use `TrackException()` within the **catch** statement. This usage is covered in the User's Guide.
 - On an individual page or user control. You will use the [Error](#) event of the Page class. This usage is covered in the "Track exception on the page in the Page.Error event" section of the User's Guide.
 - For all unhandled exceptions throughout your web application. This usage is covered here.
1. Within the **Global.asax** file, create or modify the `Application_Error()` method to call `TrackException()`.

Use the `pRedirectURL` parameter of `TrackException()` to select a page with a user friendly error message. By default, use the error page you developed for the `<customErrors>` section of the **web.config** file in the previous section.

An example follows the definition of `TrackException()`.

Note: `Application_Error()` will be called even if there is an exception within `Application_Start()`. `SetupInputSecurity()` has been set up to throw exceptions that report a bad configuration. The `TrackException()` method will not log exceptions until `SetupInputSecurity()` has successfully run. Instead, the exceptions will appear on the browser.

Definition: PeterBlum.DES.Security.LogAndRespond.TrackException

[C#]

```
public void TrackException(  
    string pRedirectURL,  
    bool pStopException)
```

[VB]

```
Public Sub TrackException( _  
    ByVal pRedirectURL As String, _  
    ByVal pStopException As Boolean)
```

Parameters

pRedirectURL

Determines the URL to redirect to another page. Often you will use different URLs for different types of exceptions. When "", the value from `LogAndRespond.DefaultTrackErrorArgs.RedirectURL` is used.

WARNING: *It is important not to give the hacker any idea that there was a database exception. If you capture any database exceptions, avoid naming the page file in a way that gives the hacker any insight. Even if the page name differs from others, the hacker can tell that there was an error and that is enough to identify a hole in your security.*

pStopException

Determines if the exception is stopped or passed back to show to the user. Set it to `true` to stop exceptions. This is strongly recommended because the goal is to prevent showing the error. It calls `HttpContext.Current.Server.ClearError()` to stop the exception.

This method does not get passed the Exception object. That's because `Application_Error()` does not supply one. Instead, it internally gets the current exception through [HttpContext.Current.Server.GetLastError\(\)](#). You can also use that method to help you customize the `pRedirectURL` parameter.

Example

When a `System.IO.FileNotFoundException` is intercepted, redirect to `"/MyErrors/FileIOError.aspx"`. This example assumes that the default error page URL has been specified in `LogAndRespond.DefaultTrackErrorArgs.RedirectURL`. In your case, if it has not, assign it in the first line.

[C#]

```
protected void Application_Error(Object sender, EventArgs e)
{
    string vRedirectURL = ""; // default
    Exception vException = HttpContext.Current.Server.GetLastError();
    if (vException is System.IO.FileNotFoundException)
        vRedirectURL = "/MyErrors/FileIOError.aspx";
    PeterBlum.DES.Security.LogAndRespond.Current.TrackException(vRedirectURL, true);
}
```

[VB]

```
Protected Sub Application_Error(ByVal sender As Object, ByVal e As EventArgs)
    Dim vRedirectURL As String = "" ' default
    Dim vException As Exception = HttpContext.Current.Server.GetLastError()
    If TypeOf vException Is System.IO.FileNotFoundException Then
        vRedirectURL = "/MyErrors/FileIOError.aspx"
    End If
    PeterBlum.DES.Security.LogAndRespond.Current.TrackException(vRedirectURL, True)
End Sub
```

Establishing Character Encoding to Limit Script Injection

One common technique to limit the types of script injection attacks is to define a character encoding for the web form. There are two parts to this process: settings in the **web.config** file and settings on each web form. This section will cover the **web.config** file. The User's Guide will describe how to set it on each web form in the section "Securing a Page".

1. Determine the character encoding for your page or site.

In Western cultures, the ISO-8859-1 is a [recommended choice](#). It imposes a more restrictive character set than some others, like UTF-8.

2. Assign the encoding to the <globalization> section of the **web.config** file. See this topic from Microsoft: [Selecting an Encoding for Web Forms Globalization](#).

Example of encoding in **web.config**:

```
<configuration>
  <system.web>
    <globalization
      requestEncoding="ISO-8859-1"
      responseEncoding="ISO-8859-1"
    />
  </system.web>
</configuration>
```

Securing Each Page

It's time to move into the **Input Security User's Guide**. See the section "Securing A Page". If you haven't already read the "Overview", "SQL Injection Primer", "Script Injection Primer" and "Input Tampering Primer" sections of the User's Guide, you may want to do this before attempting to design security into your pages.

Remember: To beat your opponent, you must think like your opponent!

Troubleshooting

Here are some issues that you may run into. Remember that technical support is available from support@PeterBlum.com. We encourage you to use this knowledge base first.

This guide contains problems specific to the **Peter's Input Security** module. Please see the "Troubleshooting" section of the **General Features Guide** for an extensive list of other topics including "Handling JavaScript Errors" and "Common Error Messages".

Application_Start() does not appear to be running

The `Application_Start()` method of the Global class of **Global.aspx** is very important to any web application. It is designed to initialize global values of a web application. Yet PeterBlum.com has experienced several customers who cannot get it to run.

Peter's Input Security has been coded with a backup system that detects `SetupInputSecurity()` has not run as pages use its features. If it can, it calls `SetupInputSecurity()` and continues with normal activity. If it cannot find the `SetupInputSecurity()` method, it will throw an exception.

First confirm that the method is present, correctly spelled and using the exact text case. If it is not, fix it and try again. Otherwise, try the following.

Confirm the problem

1. Confirm that the `Application_Start()` method is correctly defined.

[C#]

```
protected void Application_Start(Object sender, EventArgs e)
```

[VB]

```
Protected Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
```

2. Confirm that it is not running. Simply throw an exception from within `Application_Start()`. Compile and run with this code in `Application_Start()`. See if you get an exception page or not.

[C#]

```
throw new Exception("Application_Start running");
```

[VB]

```
Throw New Exception("Application_Start running")
```

Implement a Solution (Workaround)

If you can determine how to fix the problem, do so as `Application_Start()` greatly benefits you. However, Peter's Input Security provides a workaround designed for its own needs. You have already copied text into your **Global.aspx** file. It includes the static/shared method `SetupInputSecurityFromPage()`, which is used in this situation.

Assuming that you do not get the exception, take these steps:

1. Does **Global.aspx** have an `Application_OnStart()` method? Does it work? If so, move the call to `SetupInputSecurity()` into `Application_OnStart()`. If this test works, you are done.
2. Within each web form, add this line to the beginning of the `Page_Load()` method:

ASP.NET 1.1 and higher users

```
ASP.global_asax.SetupInputSecurityFromPage()
```

ASP.NET 1.0 CodeBehind users

```
Global.SetupInputSecurityFromPage()
```

3. Recompile any code behind files and test.

Peter's Input Security reports a file rights error

Peter's Input Security creates files in two places: a reports folder for the Security Analysis Report and a log folder for logging to file. If you get an exception indicating a file rights problem, change the folder's sharing rights.

- With NT, 2000, allow the ASP.NET User Account to create, delete, read and write.
- With XP, set up Sharing with "Allow network users to change my files"

PeterBlum.com Technical Support cannot additional provide user education on setting up sharing rights. If you think that you have the rights correct, create a web form that creates a file in the same folder to test. Here is some code to make it easy. This code belongs in the Page_Load() method. This web form needs a reference to System.IO. Supply the path in vTempFile.

[C#]

```
string vTempFile = "FILE PATH HERE\Temp.txt";
try
{
    if (File.Exists(vTempFile))
        File.Delete(vTempFile);
    StreamWriter vTest = File.CreateText(vTempFile);
    try
    {
        vTest.Write("test");
    }
    finally
    {
        vTest.Close();
    }
    File.Delete(vTempFile);
}
catch (Exception)
{
}
```

[VB]

```
Dim vTempFile As String = "FILE PATH HERE\Temp.txt"
Try
    If File.Exists(vTempFile) Then
        File.Delete(vTempFile)
    End If
    Dim vTest As StreamWriter = File.CreateText(vTempFile)
    Try
        vTest.Write("test")
    Finally
        vTest.Close()
    End Try
    File.Delete(vTempFile)
Catch e As Exception
End Try
```