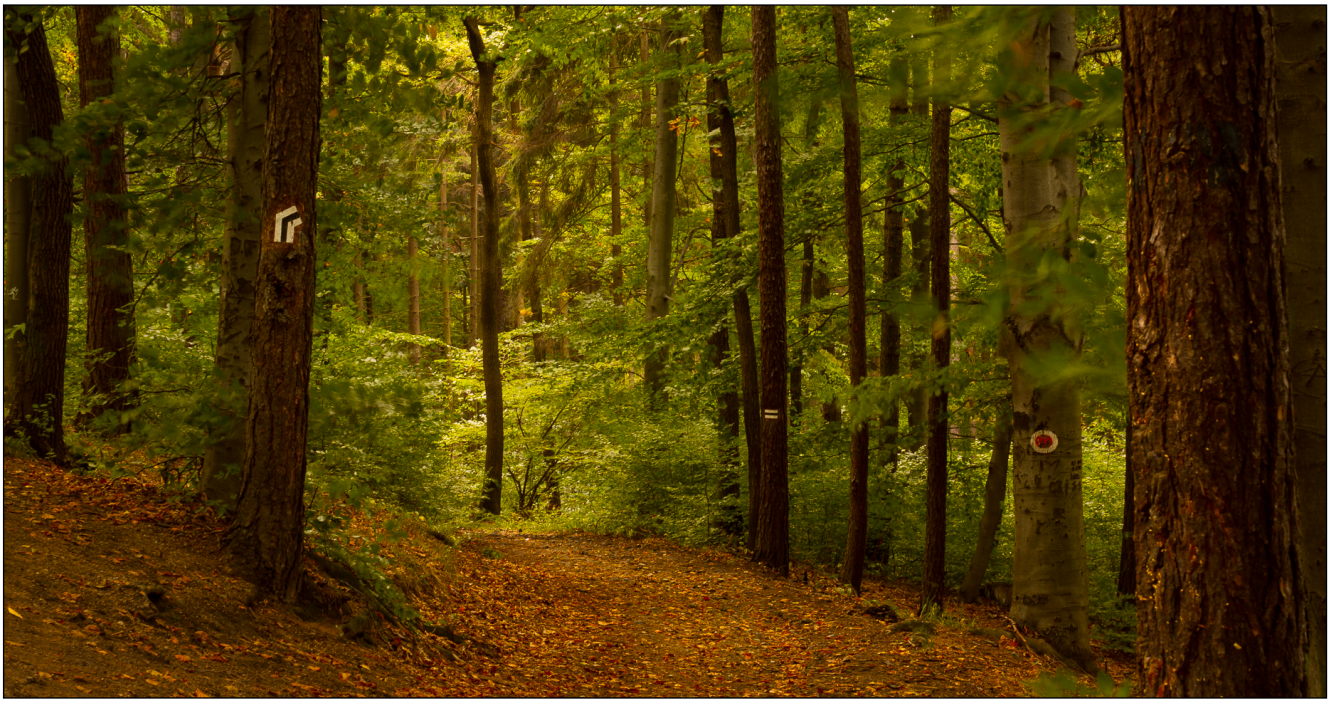
OWASP
Open Web Application
Security Project

Code Review Guide

Version 2.0 Pre-Alpha



The code review guide is currently at release version 1.1 and the second best selling OWASP book in 2008. Many positive comments have been feedback regarding this initial version and believe it's a key enabler for the OWASP fight against software insecurity. It has even inspired individuals to build tools based on its information. The combination of a book on secure code review and tools to support such an activity is very powerful as it gives the developer community a place to start regarding secure application development.

Prefix

This document is a pre Alpha release to demonstrate where we are to date in relation to the OWASP Code Review Guide. OWASP volunteers develop the Code Review Guide, people like you. The aim of the guide is to help developers and code reviewers alike navigate a source code review and pinpoint areas of weakness from a security standpoint.

If you would like to contribute please feel free to contact the team, we are not hard to find on the interwar. If you have feedback, suggestions, or would like to send OWASP lots of donations to assist in developing great documents, please also get in touch….

Thanks Eoin, Larry & Sam

1.1 Forward

The OWASP Code Review guide is the result of initially contributing and leading the Testing Guide. Initially, it was thought to place Code review and testing into the same guide; it seemed like a good idea at the time. But the topic called security code review got too big and evolved into its own stand-alone guide.

Eoin Keary started the Code Review guide in 2006. This current version was started in April 2013 via the OWASP Project Reboot initiative. The OWASP Code Review team consists of a small, but talented, group of volunteers who should really get out more often.

It is common knowledge that more secure software can be produced and developed in a more cost effective way when bugs are detected early on in the systems development lifecycle. Organizations with a proper code review functions integrated into the software development lifecycle (SDLC) produced remarkably better code from a security standpoint. Simply put "We can't hack ourselves secure". Attackers have more time to fine vulnerabilities on a system than the time allocated to a defender. Hacking our way secure amounts to an uneven battlefield; Asymmetric warfare, a losing battle.

By necessity, this guide does not cover all languages; it mainly focuses on .NET and Java, but has a little C/C++ and PHP thrown in also. However, the techniques advocated in the book can be easily adapted to almost any code environment. Fortunately, the security flaws in web applications are remarkably consistent across programming languages.

1.2 Code Review Guide Introduction

Code review is probably the single-most effective technique for identifying security flaws early in the system development lifecycle. When used together with automated tools and manual penetration testing, code review can significantly increase the cost effectiveness of an application security verification effort.

This guide does not prescribe a process for performing a security code review. Rather, this guide focuses on the mechanics of reviewing code for certain vulnerabilities, and provides guidance on how the effort should be structured and executed.

Manual security code review provides insight into the “real risk” associated with insecure code. This is the single most important value from a manual approach. A human reviewer can understand the context of a bug or vulnerability in code. Context requires human understanding of what is being assessed. With appropriate context we can make a serious risk estimate that accounts for both the likelihood of attack and the business impact of a breach. Correct categorization of vulnerabilities helps with priority of remediation and fixing the right things as opposed to wasting time fixing everything.

Why Does Code Have Vulnerabilities?

MITRE has catalogued circa 800 different kinds of software weaknesses in their CWE project. These are all different ways that software developers can make mistakes that lead to insecurity. Every one of these weaknesses is subtle and many are seriously tricky. Software developers are not taught about these weaknesses in school and most do not receive any training on the job about these problems.

These problems have become so important in recent years because we continue to increase connectivity and to add technologies and protocols at a shocking rate. Our ability to invent technology has seriously outstripped our ability to secure it. Many of the technologies in use today simply have not received any security scrutiny.

There are many reasons why businesses are not spending the appropriate amount of time on security. Ultimately, these reasons stem from an underlying problem in the software market. Because software is essentially a black box, it is extremely difficult to tell the difference between good code and insecure code. Without this visibility, buyers won’t pay more for secure code, and vendors would be foolish to spend extra effort to produce secure code.

One goal for this project is to help software buyers gain visibility into the security of software and start to effect change in the software market.

Nevertheless, we still frequently get pushback when we advocate for security code review. Here are some of the (unjustified) excuses that we hear for not putting more effort into security:

'"“We never get hacked (that I know of), we don’t need security”"

'"“We have a firewall that protects our applications”"

'""We trust our employees not to attack our applications" "

Over the last 10 years, the [[team]] involved with the OWASP Code Review Project has performed thousands of application reviews, and found that every single application has had serious vulnerabilities. If you haven’t reviewed your code for security holes, the likelihood that your application has problems is virtually 100%.

Still, there are many organizations that choose not to know about the security of their code. To them, we offer Rumsfeld’s cryptic explanation of what we actually know. If you’re making informed decisions to take risk in your enterprise, we fully support you. However, if you don’t even know what risks you are taking, you are being irresponsible both to your shareholders and your customers.

“*...we know, there are known knowns; there are things we know we know. We also know there are known unknowns; that is to say we know there are some things we do not know. But there are also unknown unknowns -- the ones we don't know we don't know.*”- Donald Rumsfeld

What is Security Code Review?

Security code review is the process of auditing the source code for an application to verify that the proper security and logical controls are present, that they work as intended, and that they have been invoked in all the right places. Code review is a way of helping ensure that the application has been developed so as to be “self-defending” in its given environment.

Security code review is a method helping assure secure application developers are following secure development techniques. A general rule of thumb is that a penetration test should not discover any additional application vulnerabilities relating to the developed code after the application has undergone a proper security code review. - al least very few issues should be discovered.

All security code reviews are a combination of human effort and technology support. At one end of the spectrum is an inexperienced person with a text editor. At the other end of the scale is a security expert with an advanced static analysis (SAST) tool. Unfortunately, it takes a fairly serious level of expertise to use the current application security tools effectively. They also don't understand dynamic data /page flow or business logic. SAST tools are great for coverage and setting a minimum baseline.

Tools can be used to perform this task but they always need human verification. Tools do not understand context, which is the keystone of security code review. Tools are good at assessing large

amounts of code and pointing out possible issues, but a person needs to verify every single result to determine if it is a real issue, if it is actually exploitable, and calculate the risk to the enterprise.

Human reviewers are also necessary to fill in for the significant blind spots where automated tools simply cannot check.

1.2.1 What is source code review and Static Analysis

What is Security Source Code Review?

Source code review is the practice of reviewing developed code for vulnerabilities. There are many ways to review the security of an application and it is recommended to perform more than one method to help ensure more assessment coverage. Penetration testing is great at finding certain bugs such as technical signature or API based issues. Issues related to privacy, information leakage, denial of service is more suited to code review. Source code review is also good practice as you are finding issues early in the SDLC. Locating and fixing issues early in your SDLC makes it cheaper in terms of effort and cost to remediate. It also empowers developers to understand security bugs at the source code level such that they may not repeat the same mistakes.

What is static analysis?

Static Code Analysis is usually performed as part of a Source code review and is carried out at the Implementation phase of SDLC. Static Code Analysis commonly refers to the running of static code analysis tools that attempts to highlight possible vulnerabilities whiting the 'static' (non-running) source code by using techniques such as Taint Analysis, Data Flow Analysis, Control Flow Graph, and Lexical Analysis. When the analysis is performed on a runtime environment, it is referred to as Dynamic Code Analysis. Ideally, such tools would automatically find security flaws with a high degree of confidence that what is found is indeed a flaw. However, this is beyond the state of the art for many types of application security flaws. Thus, such tools frequently serve as aids for an analyst to help them zero in on security relevant portions of code so they can find flaws more efficiently, rather than a tool that simply finds flaws automatically.

1.2.2 What is Code Review (Needs Content)

Lorem Ipsum

1.2.3 Manual Review

Manual review is sited when a risk-based approach to the code review is required. Risk based code review works by.

1. Identification of the trust boundaries in the code. 2. Identification of data paths and storage classes. 3. Identification of authorization components. 4. Identification of authentication components. 5. Review of input validation and encoding methods. 6. Review of logging components.

<more description is required here>

Manual review is good for :

Data leakage detection Resource usage/exhaustion detection Business Logic review* Denial of service Deep Dive review

Not so good for: Business Logic review* Level of coverage

1.2.3.1 Choosing a static analysis tool

Choosing a static analysis tool is a difficult task since there are a lot of choices. The comparison charts below should help you decide which tool is right for you. This list is not exhaustive. The first thing to do is to look to for a tool that supports the programming language of your choice. You also have to decide whether you want a commercial tool or a free one. Usually the commercial tools have more features and are more reliable than the free ones. The major commercial tools are equally effective but their usability might differ. Next, there is the type of analysis you are looking for: Security or Quality, Static or Dynamic analysis. You should also check the compatibility of the tool with your programming environment. This was the easy part to narrow the choice down to a few tools. The next step requires you to do some work since it is quite subjective. The best thing to do is to test a few tools to see if you are satisfied with different aspects such as the user experience, the reporting of vulnerabilities, the level of false positives, the customization, the customer support… The choice should not be based on the number of features, but on the features that you need and how they could be integrated in your SDLC. Also, before choosing the tool, the security expertise of the targeted users should be clearly evaluated in order to choose an appropriate tool.

Free static analysis tools

Name	Type	Programming language									OS
		Obj C	Java	PHP	.Net	Perl	C	C++	Python	Other	
CodePro	Security, Quality		Supported								Linux, Windows, Mac
FindBugs	Security, Quality		Supported								Linux, Windows
FxCop (microsoft)	Security, Quality				Supported						Windows
Flawfinder	Security						Supported	Supported			Linux
Milk	Security		Supported								Linux, Windows
MOPS	Security						Supported				Linux
OWASP Code Crawler	Security		Supported		Supported						Windows
OWASP ORIZON	Security		Supported	Supported			Supported				Linux
OWASP O2 Platform	Security		Supported		Supported					Javascript	Windows
OWASP LAPSE	Security		Supported								Linux, Windows, Mac
PMD	Quality		Supported				Supported	Supported		JavaScript, XML, XSL, + Commercial plugins	Linux, Windows
PREfast (microsoft)	Quality						Supported	Supported			Windows
RIPS-Scanner	Security			Supported							Linux, Windows, Mac
Sonarqube	Quality		Supported	Supported	Supported				Supported	Delphi, Drools, Javascript, XML ...	Linux, Windows, Mac
Splint	Security, Quality						Supported				Linux, Windows
StyleCop	Quality			Supported							Windows
Yasca	Security, Quality		Supported	Supported	Supported	Supported	Supported	Supported		HTML, Javascript, Cobol, ColdFusion, VB	Linux, Windows

Legend

- Security
- Quality
- Supported
- Not suppoted
- Linux
- Windows
- Mac

Commercial static analysis tools

Name	Type	Programming language									OS
		Obj C	Java	PHP	.Net	Perl	C	C++	Python	Other	
dotTest	Security, Quality				Supported						Windows
Jtest	Security, Quality, Dynamic		Supported								Linux, Windows, Mac
C/C++test	Security, Quality, Dynamic						Supported	Supported			Linux, Windows
CodeSonar (GrammaTech)	Security		Supported				Supported	Supported			Linux, Windows, Mac
PVS Studio	Quality						Supported	Supported			Windows
Coverity Security Advisor	Security		Supported				Supported	Supported			Linux, Windows, Mac
Klocwork Insight	Security, Quality				Supported						Linux, Windows, Mac
CxSuite (Checkmarx)	Security, Quality	Supported	Supported	Supported		Supported	Supported	Supported		Javascript, apex, VB6, Ruby	Linux, Windows, Mac
Armorize CodeSecure	Security		Supported	Supported	Supported					VBScript	Linux, Windows
HP Fortify Static code analyser	Security, Quality, Dynamic	Supported	Supported	Supported	Supported		Supported	Supported	Supported	20 supported languages	Linux, Windows, Mac
Seeker (Quotium)	Security		Supported	Supported	Supported						Linux, Windows
Clang	Security, Quality	Supported					Supported	Supported			Mac
Veracode	Security, Quality, Dynamic	Supported	Supported	Supported	Supported		Supported	Supported		ColdFusion, Ruby, VB6, VBS,	Cloud
Appscan (IBM)	Security, Quality, Dynamic	Supported	Supported	Supported	Supported	Supported	Supported	Supported		Javascript, ColdFusion, VB6, PL/SQL, T-SQL, Cobol	Linux, Windows

Legend :

- Security
- Quality
- Dynamic
- Supported
- Not suppoted
- Linux
- Windows
- Mac
- Cloud

1.2.4 Advantages of Code Review to Development Practices

Advantages of Code Review to Development Practices

Integrating code review into your company's development processes can have many benefits that will be explored below. Some of these benefits depend upon the tools you use to perform code reviews, how well that data is backed up, and how well those tools are used. The days of bringing developers into a room and displaying code on a projector, whilst recording the review results on a printed copy and behind us, many tools exist to make code review more efficient and to track the review records/decisions. When the code review process is structured correctly, the act of reviewing code can provide educational, audible and historical benefits to any organization.

The following provides a list of benefits that a code review procedure can add to development team.

Provides an Historical Record

If any developer has joined a company, or moved teams within a company, and had to maintain or enhance a piece of code written years ago, one of the biggest frustrations can be the lack of context the new developer has on the code. Various schools of opinion exist on code documentation, both within the code (comments) and external to the code (design/functional docs, wikis, etc.), opinions ranging from zero-documentation tolerance through to near-NASA level documentation where the size of the documentation far exceeds the size of the code module.

If you think about the discussions that occur during a code review, many of these discussions, if recorded, would provide valuable information (context) to module maintainers and new programmers. From the writer describing the module along with some of their design decisions, to each reviewers comments, stating why they think one SQL query should be restructured, or an algorithm changed, there is a development story unfolding in front of the reviewers eyes which can (and should) be used by future coders on the module, who are probably not involved in the review meetings.

Capturing those review discussions in a review tool automatically, and storing them for future reference, will provide the development organization with a history of the changes on the module which can be queried at a later time by new developers. These discussions can also contain links to any architectural/functional/design/test specifications, bug or enhancement numbers, etc.

Verification Change has been tested

When a developer is about to submit code into the repository, how do you know they have sufficiently tested it? Adding a description of the tests they have run (manually or automated) against the changed code can give reviewers (and management) confidence that the change will work and not cause any regressions. Also by declaring the tests the writer has ran against their change, the author is allowing reviewers to suggest further testing that may have been missed by the author.

In a development scenario where automated unit or component testing exists, the coding guidelines can require the developer include those unit/component tests in the code review. This again allows reviewers within this type of automated environment to ensure the correct unit/component tests are going to be included in the environment, keeping the quality of the continuous integration.

Coding Education for Junior Developers

After you learn the basics of a language and read a few of the best practices book, how can you get good on-the-job skills to learn more? Besides buddy coding (which rarely happens and is never cost effective) and training sessions (brown bag sessions on coding, tech talks, etc.) the design and code decisions discussed during a code review can be a learning experience for junior developers. Many experienced developers may admit to this being a two way street, where new developers can come in with new ideas or tricks that the older developers can learn from. Altogether this cross-pollination of experience and ideas can only be beneficial to a development organization.

Familiarization with Code Base

When a new feature is developed, it is often integrated with the main code base, and here code review can be a conduit for the wider team to learn about the new feature and how it's code will impact the wider product. This helps prevent functional duplication where separate teams end up coding the same small piece of functionality.

This also applies for development environments with silo'ed teams. Here the code review author can reach out to other teams to gain their insight, and allow those other teams to review their code, and everyone then learns a bit more about the company's code.

Pre-warning of Integration Clashes

In a busy code base there will be times (especially on core code) where multiple developers can be writing code affecting the same module. Many people have had the experience of cutting the code and running the tests, only to discover upon submission that some other change has modified the functionality, requiring the author to recode and retest some aspects of their change. Spreading the word on upcoming changes via code reviews gives a greater chance of a developer learning that a change is about to impact their upcoming commit, and development timelines, etc., can be updated accordingly.

Coding Guidelines Touch point

Many development environments have coding guidelines which new code must adhere to. Coding guidelines can take many forms, but it's worth pointing out that security guidelines can be a particularly relevant touch point within a code review as unfortunately, though typically, the security issues are understood by a subset of the development team. Therefore it can be possible to include teams with various technical expertise into the code reviews, i.e. someone from the security team (or that person in the corner who knows all the security stuff) can be invited as a technical subject expert to the review to check the code from their particular angle. This is where the OWASP top 10 could be enforced.

1.2.5 Why Code Review

Security code review aims to identify security flaws in the application related to its features and design along with their exact root causes. With the increasing complexity of the applications and the advent of new technologies the traditional way of testing may fail to detect all the security flaws present in the applications. Thus, there is a need to understand the code of the application, external components, technologies and configurations to be able to uproot all the flaws in different kinds of applications. Such a deep dive into the application code also helps in determining exact mitigation techniques that can be used to avert the security flaws. Let’s look in further detail of the benefits of security code reviews.

Root cause analysis (Source to sink) – There are various reasons why security flaws manifest in the application like lack of validation, parameter mishandling etc. During the assessment, the code is thoroughly studied and such flaws are checked. In the process the exact root cause of the flaws are captured. The complete data flow is traced and source to sink analysis is carried out. Source to sink analysis here means to determine what are possible inputs to the application i.e. source, and how are they being processed by it. Sink refers an insecure code pattern like for instance, a dynamic SQL query. So, its analyzing what is the source (input) and the sink (vulnerable code pattern) for any vulnerability. Consider a scenario where the source is a user input. It flows through the different classes/components of the application and finally falls into a concatenated SQL query i.e. a sink, and there is no proper validation being applied to it in the path. In this case the application will be vulnerable to SQL Injection attack, as identified by the source to sink analysis. Such an analysis exactly helps in understanding, which vulnerable inputs can lead to a possibility of an exploit in the application.

Design Analysis – Design is an important aspect of security code review. The dynamic testing approach does not involve design analysis, which also serves as it one of the limitations. The design

flaws are difficult to uncover and requires knowledge of the entire data flow, input sources, integrations and all the configurations of the application. Thus, the flaws related to mishandling of non-user inputs and external integrations like server-to-server, which are unlikely to get covered in a dynamic testing approach can be easily determined in a security code review. It covers the end-to-end analysis of the application.

All Instances of the Vulnerabilities – Once a flaw is identified the next step of the security code review process is to enumerate its all the possible instances present in the application. Through security code reviews we can uncover insecure patterns present in all the files of the application. Here, all insecure instances includes the list of different the insecure code patterns that lead to a vulnerability and all of their occurrences. For instance, an application can be vulnerable to XSS vulnerability because of use of unvalidated inputs in insecure display methods like scriplets, response.write method etc. at several places.

Uncommon Security Flaws – Security code reviews is very specific to the application being reviewed. It may highlight some flaws that are new or specific to the code implementation of the application like insecure termination of execution flow, synchronization error etc. These flaws can only be uncovered if we understand the application code flow and its logic well. Thus, security code review is not just about scanning the code for set of unknown insecure code patterns. But it also involves understanding the code implementation of the application and enumerating the flaws specific to it.

Limitations of Existing Security Controls – The application being reviewed might have been designed with some security controls in place like centralized blacklist validation etc. or there could be some inbuilt validation/security controls present in the application platform being used. These security controls must be studied carefully to identify if they are foolproof. According to the implementation of the control, the nature of attack or any specific attack vector that can be used to bypass it, must be analyzed. Enumerating the weakness in the existing security control is another important aspect of the security code reviews.

Specific Recommendations – The security code reviews also helps to come up with mitigation techniques that can best suit the application, instead of a generic one.

1.2.5.1 Scope and Objective of secure code review (Needs Content)

Lorem Ipsum

1.2.6 We can't hack ourselves secure

We can't hack ourselves secure**.** Penetration testing is generally a point in time test. As source code changes the value of the findings of a penetration test degrade with time. There are also privacy, compliance and stability and availability concerns, which are generally not covered by penetration testing. Data information leakage in a cloud environment for example may not be discovered via a penetration test.

1.2.7 360 Review: Coupling source code review and Testing / Hybrid Reviews (Needs Content)

Lorem Ipsum

1.2.8 Can static code analyzers do it all?

Secure code review is a process of enumerating the flaws in the application. The flaws may exist in the application due to insecure code, design or configuration. Out of which the flaws that arise due to insecure code can be enumerated to a great extent through automated analysis, as most of them are associated with insecure patterns.

Automated analysis can be carried out through any of the following three options:

Static Code analyzers or scanners

Custom scripts based on some pattern search

Open source tools

Though some scripts and some open source tools are efficient enough in finding insecure code patterns but they often lack the capability of tracing the data flow. The use of static code analyzers or the scanners, which identify the insecure code pattern along with source to the sink analysis, fulfill this limitation.

With this we come to the next big question i.e. Can scanners i.e. static code analyzers do it all?

Static code analyzers or the scanners are the most comprehensive options to automate the process of review.

Some of the advantages of static code scanners are:

1. Reduction in manual efforts – Secure code review can be at times be a tedious process of analyzing thousands of lines of code for a host of vulnerabilities. Moreover as the type of patterns to be scanned almost remains common across application the task also tends to get a bit repetitive. In

such a scenario, scanners play a big role is automating the process of searching the vulnerabilities through big numbers of lines of code.

2. Time efficient – Scanners are must time efficient than manual reviews. In most cases the scanners have proved to be much faster than manual process of reviewing the source code.

3. Finds all the instances of the vulnerabilities - Scanners are very effective in identifying all the instances of a particular vulnerability with their exact location. This is really helpful for larger code base where tracing for flaws in all the files is difficult.

4. Source to sink analysis – Most scanners today trace the code and identify the vulnerabilities through source to sink analysis. They identify the inputs to the application and trace them thoroughly throughout the code till they find them to be associated with any insecure code pattern. Such a source to sink analysis helps the developers in understanding the flaws better as they get a complete root cause analysis of the flaw.

5. Exhaustive coverage of vulnerability patterns – Most of the scanners have well-defined set of test cases covering all the well-known vulnerabilities. Thus through the use of scanners we can ensure that the code gets thoroughly checked for all the pre-defined set of flaws.

6. Elaborate reporting format – Scanners provide a detailed report on the observed vulnerabilities with exact code snippets, risk rating and complete description of the vulnerabilities. This helps the development teams to easily understand the flaws and implement necessary controls.

Limitations of static code analyzers:

1. Business Logic Flaws remain untouched – The flaws that are related to application's logic, transactions, and specific sensitive data remain untouched by the scanners. The security controls that needs to be implemented in the application specific its features and design are often not pointed by the scanners. This stands as a biggest limitation of the static code analyzers.

2. Limited scope – Static code analyzers are often designed for specific frameworks or certain set of vulnerable patterns, they fail to address the issues not covered in their search pattern repository. So the scanners often fail in catching up the vulnerabilities of the new versions of the framework that keeps coming up.

3. Custom validations - Most of the static analyzers tool miss out the custom validations added in the application while identifying the flaws. These could include blacklist or whitelist validation present in the application before the input sources. It could also mean the customization added by the developers to the existing design frameworks and inbuilt framework based API, the scanners that go by pattern based search usually miss out in understanding such intricate details of the code.

4. Design flaws – Design flaws are lessen known issues and static code analyzers often focus more on the code than the design. Mainly if the application design is custom built it becomes challenging for the scanners to trace the code flow.

5. Application specific recommendations – Scanners usually provide a generic solution and do not point out application specific code changes. If the solutions are customized as per the design and the feasibility of the application it will be clearer to the developers and require less code change.

Parameters to be considered for commercial scanners:

1. Source to sink analysis Capabilities
2. Support for frameworks
3. Ability to understand customized code or validations
4. Coverage of Business logic cases specially the ones related to authorization
5. Option to customize the pattern search

2 Methodology (Missing Content)

We need to add content that introducing readers to what the methodology is.

2.1 The code review approach (Needs Content)

Lorem Ipsum

2.1.1 Preparation and context

Laying the Groundwork

In order to effectively review a code baseline, it is critical that the review team understands the business purpose of the application and the most critical business impacts. This will guide them in their search for serious vulnerabilities. The team should also identify the different threat agents, their motivation, and how they could potentially attack the application.

All this information can be assembled into a high-level threat model of the application that represents all of information that is relevant to application security. The goal for the reviewer is to verify that the key risks have been properly addressed by security controls that work properly and are used in all the right places. In some cases the threat model will already be created, in other cases the reviewers might need to draft one up a threat model.

Ideally the reviewer should be involved in the design phase of the application, but this is almost never the case. However regardless of the size of the code change, the engineer initiating the code review should direct reviewers to any relevant architecture or design documents. The easiest way to do this is to include a link to the documents (assuming they're stored in an online document repository) in the initial e-mail, or in the code review tool if that is supported.

Performing code review can feel like an audit, and most developers hate being audited. The way to approach this is to create an atmosphere of collaboration between the reviewer, the development team, the business representatives, and any other vested interests. Portraying the image of an advisor and not a policeman is very important if you wish to get full co-operation from the development team.

Before We Start

The reviewer(s) need to be familiar with:

1. Code: The language(s) used, the features and issues of that language from a security perspective. The issues one needs to look out for and best practices from a security and performance perspective.

2. Context: The working of the application being reviewed. All security is in context of what we are trying to secure. Recommending military standard security mechanisms on an application that vends apples would be over-kill, and out of context. What type of data is being manipulated or processed, and what would the damage to the company be if this data was compromised? Context is the "Holy Grail" of secure code inspection and risk assessment… we'll see more later.

3. Audience: The intended users of the application. Is it externally facing or internal to "trusted" users? Does this application talk to other entities (machines/services)? Do humans use this application?

4. Importance: The size of the consequences of failure. Shall the enterprise be affected in any great way if the application cannot perform its functions as intended?

Discovery: Gathering the Information

The reviewers will need certain information about the application in order to be effective. The information should be assembled into a threat model that can be used to prioritize the review. Frequently, this information can be obtained by studying design documents, business requirements, functional specifications, test results, and the like. However, in most real-world projects, the documentation is significantly out of date and almost never has appropriate security information. If the development organization has procedures and templates for architecture and design documents, the reviewer can suggest updates to ensure security is considered at these phases.

One of the most effective ways to get started is to talk with the developers and the lead architect for the application. This does not have to be a long meeting; it could be a whiteboard session for the development team to share some basic information about the key security considerations and controls. A walkthrough of the actual running application is very helpful to give the reviewers a good idea about how the application is intended to work. Also, a brief overview of the structure of the codebase and any libraries used can help the reviewers get started.

If the information about the application cannot be gained in any other way, then the reviewers will have to spend some time doing reconnaissance and sharing information about how the application appears to work by examining the code.

Context, Context, Context

Security code review is not simply about reviewing code. It's important to remember that the reason that we review code is to ensure that the code adequately protects the information and assets it has been entrusted with, such as money, intellectual property, trade secrets, lives, or data.

The context in which the application is intended to operate is a very important issue in establishing potential risk. If reviewers do not understand the business context, they will not be able to find the most important risks and may focus on issues that are inconsequential to the business.

As preparation for a security code review, a high-level threat model should be prepared which includes the relevant information. This process is described more fully in a later section, but the major areas are listed here:

• Threat Agents

• Attack Surface (including any public and backend interfaces)

• Possible Attacks

• Required Security Controls (both to stop likely attacks and to meet corporate policy)

• Potential Technical Impacts

• Important Business Impacts

Defining context should provide us with the following information:

• Establish the importance of the application to the enterprise.

• Establish the boundaries of the application context.

• Establish the trust relationships between entities.

• Establish potential threats and possible controls.

The reviewers can use simple questions like the following to gather this information from the development team:

- "What type/how sensitive is the data/asset contained in the application?":

This is a keystone to security and assessing possible risk to the application. How desirable is the information? What effect would it have on the enterprise if the information were compromised in any way?

- "Is the application internal or external facing?", "Who uses the application, and are they trusted users?"

This is a bit of a false sense of security, as attacks take place by internal/trusted users more often than is acknowledged. It does give us context that the application should be limited to a finite number of identified users, but it's not a guarantee that these users shall all behave properly. Even if a application is sitting behind a firewall, defense in depth can be considered when the data is sensitive.

- "Where does the application host sit?"

Users should not be allowed past the DMZ into the LAN without being authenticated. Internal users also need to be authenticated. No authentication = no accountability and a weak audit trail.

If there are internal and external users, what are the differences from a security standpoint? How do we identify one from another? How does authorization work?

- "How important is this application to the enterprise?".

Is the application of minor significance or a Tier A / Mission critical application, without which the enterprise would fail? Any good web application development policy would have additional requirements for different applications of differing importance to the enterprise. It would be the analyst's job to ensure the policy was followed from a code perspective also.

A useful approach is to present the developers with a checklist, which asks the relevant questions pertaining to any web application.

The Checklist

Defining a generic checklist that can be filled out by the development team is of high value, if the checklist asks the correct questions in order to give us context. The checklist is a good barometer for the level of security the developers have attempted or thought of. If security code review becomes a common requirement, then this checklist can be incorporated into a development procedure so that the information is always available to code reviewers. The checklist should cover the most critical security controls and vulnerability areas such as:

- Data Validation
- Authentication
- Session Management
- Authorization
- Cryptography
- Error Handling
- Logging
- Security Configuration
- Network Architecture

	Category	Vulnerable Area	Facts to ANALYSE
Design	Code Flow – Division of code based on MVC	Presence of backdoor parameters/functions/files	1. Are there backdoor/unexposed business logic classes?
			2. Are there unused configurations related to business logic?
			3. If request parameters are used to identify business logic methods, is there a proper mapping of user privileges and methods/actions allowed to them?
		Placement of checks	1. Are security checks placed before processing inputs?
		Insecure Data Binding Mechanism	1. Check if unexposed instance variables are present in form objects that get bound to user inputs. If present, check if they have default values.
			2. Check if unexposed instance variables present in form objects that get bound to user inputs. If present, check if they get initialized before form binding.
	Authentication and Access Control Mechanism	Insecure authentication and access control logic	1. Is the placement of authentication and authorization check correct?
			2. Is there execution stopped/terminated after for invalid request? I.e. when authentication/authorization check fails?
			3. Are the checks correct implemented? Is there any backdoor parameter?
			4. Is the check applied on all the required files and folder within web root directory?
		Redundant configuration	1. Is there any default configuration like Access- ALL?
			2. Does the configuration get applied to all files and users?
			3. Incase of container managed authentication - Is the authentication based on web methods only?
			4. Incase of container managed authentication - Does the authentication get applied on all resources?
		Insecure Session management	1. Does the design handle sessions securely?
		Weak Password Handling	1. Is Password Complexity Check enforced on the password?
			2. Is password stored in an encrypted format?
			3. Is password disclosed to user/written to a file/logs/console?
	Data Access Mechanism	Presence of sensitive data in configuration/code files	1. Are database credentials stored in an encrypted format?
		Presence/support for different insecure data sources and their related flaws	1. Does the design support weak datastores like flat files
	Centralized Validation and Interceptors	Weakness in any existing security control	1. Does the centralized validation get applied to all requests and all the inputs?
			2. Does the centralized validation check block all the special characters?
			3. Does are there any special kind of request skipped from validation?
			4. Does the design maintain any exclusion list for parameters or features from being validated?
Architecture	Entry Points	Insecure Data handling and validation	1. Are all the untrusted inputs validated?
	External Integrations	Insecure data transmission	1. Is the data sent on encrypted channel? Does the application use HTTPClient for making external connections?
			2. Does the design involve session sharing between components/modules? Is session validated correctly on both ends?
		Elevated privilege levels	3. Does the design use any elevated OS/system privileges for external connections/commands?
Configuration	External API's used	Known flaws present in 3rd party APIs/functions	1. Is there any known flaw in API's/Technology used? For eg: DWR
	Inbuilt Security Controls	Common Security Controls	1. Does the design framework provide any inbuilt security control? Like <%: %> in ASP.NET MVC.
			2. Are are there any flaw/weakness in the existing inbuilt control?
			3. Are all security setting enabled in the design?

2.1.2 Understanding Code layout/Design/Architecture (Needs Content)

Lorem Ipsum

2.2 SDLC Integration (Needs Content)

Lorem Ipsum

2.2.1 Deployment Models (Needs Content)

Lorem Ipsum

2.2.1.1 Secure deployment configurations (Needs Content)

Lorem Ipsum

2.2.1.2 Metrics and code review (Needs Content)

Lorem Ipsum

2.2.1.3 Source and sink reviews (Needs Content)

Lorem Ipsum

2.2.1.4 Code review coverage (Needs Content)

Lorem Ipsum

2.2.1.5 Design Reviews (Needs Content)

Introduction

This project highlights some of the vital areas of design security. We are all aware of “secure coding” and practice it to great extent while developing applications. But do we give equal attention to – “Secure Design”? Most of us would probably say, NO. Design level flaws are lesser-known concepts

but their presence is a very big risk to the applications. Such flaws are hard to find in static or dynamic application scans and instead requires deep understanding of application architecture and layout to uncover them manually. With increasing business needs the complexities in application design and architecture are also increasing. There is a rise in the use of custom design techniques and diverse technologies in the applications today, which makes the need for design reviews imperative. This project focuses on highlighting some important secure design principles that developers and architects must adapt to build a secure application design. With the help of some design flaws we will see the areas of design that are exposed to security risks and what measures can be taken to avoid them in our design. It also includes mitigation techniques that can be implemented in the applications to prevent them.

Understanding the design

- What is an application design?

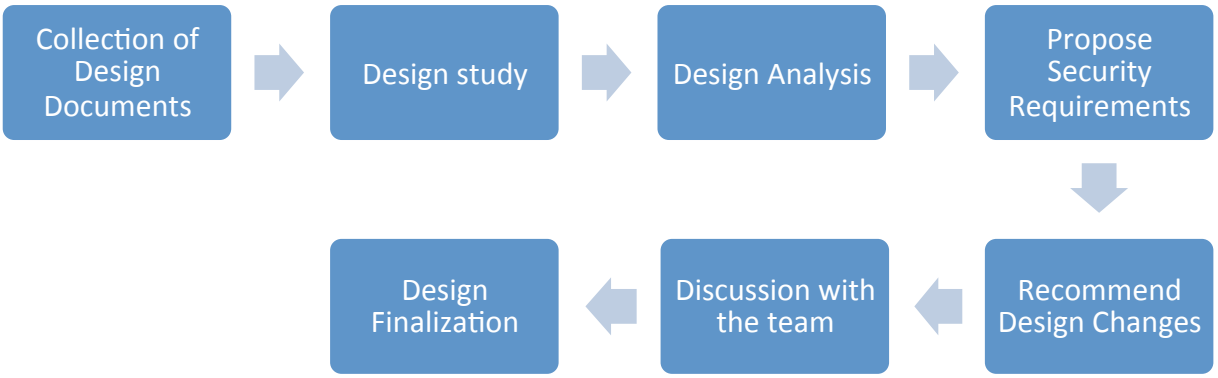
A design is a blueprint of an application; it lays a foundation for its development. It illustrates the layout of the application and identifies different application components needed for it. It is a structure that determines execution flow of the application. Most of the application designs are based on a concept of MVC. In such designs different components interact with each other in an ordered sequence to serve any user request.

- Why should be review the design?

Design review should be an integral part of secure software development process. If the application is reviewed for security at the design level many inherent backdoors can be uncovered. Design reviews also help to implementing the security requirements in a better way.

Methodology

The methodology to be followed for design reviews is explained below:



- Collection of Design Documents:

This phase involves collecting required information of the proposed design. It would involve all kinds of documentation maintained by the development team about the design like flow charts, sequence diagrams, class diagrams etc. Requirements documents are also needed to understand the objective of the proposed design.

- Design Study:

In this phase the design is thoroughly studied mainly with respect to the data flow, different application components and their interactions, data handling etc. This is achieved through manual analysis and discussions with the design or technical architect's team. The design and the architecture of the application must be understood thoroughly to analyse vulnerable areas that can lead to security breaches in the application. The key areas of the design that must be considered during threat analysis are given below.

• Data Flow/Code Layout

• Access control

• Existing/Built-in Security controls

• Entry points of non-user inputs

• Integrations with external services

• Location of configurations file and data sources

• Add-ons and customization present (in case of built-in design framework)

This will help in identifying the trust boundaries for an application and thus aid in taking decisions about the vulnerabilities and their risk levels posed to the application.

- Design Analysis:

After understanding the design the next phase is to analyse the threats for the design. This phase involves “threat modeling” the design.

The threats must be identified for different design areas that were identified in the previous step. It involves observing the design from an attacker’s perspective and uncovering the backdoors and insecure areas present in it. The analysis can be done broadly on the basis of 2 important criteria:

1. Insecure Implementation – This would mean the design has a loophole, which can lead to a security breach in the application for instance, insecure reference to business logic functions.

2. Lack of secure implementation – This would mean the design has not incorporated secure practices. For instance, in connection to external server different security requirements to protect confidentiality and integrity of the data are not present.

Similar instances are listed below to illustrate the points that should be broadly considered while analysing different design areas:

• Data Flow -

a. Are user inputs used to directly reference a business logic class/function

b. Is there a data binding flaw?

c. Does it expose any backdoor parameter to invoke business logic?

d. Is the execution flow of the application correct?

• Authentication and access control -

a. Does the design implement access control for all the files?

b. Does it handle session securely?

c. Is there SSO, does it leave any backdoor?

• Existing/built-in Security Controls -

a. Weakness in any existing security control

b. Is the placement of the security controls correct?

• Architecture –

a. Is there validation for all input sources?

b. Is the connection to external servers secure?

• Configuration/code files and datastores -

a. Are sensitive data present in configuration files?

b. Does it support any insecure data source?

A detailed checklist is available here - Excel Doc was here at the end of this activity we get a list of threats or insecure areas applicable to the design.

- Propose Security Requirements:

After analysing the insecure areas in the design in this step a list of security requirements corresponding to them must be created. Requirements are high level changes or additions to be incorporated in the design, for instance: Validate the inputs fetched from the webservice response before processing them. Any protection that is needed for resolving the vulnerable area identified in the design would go as a security requirement for the design. This phase involves listing all the security requirements for the design along with security risk associated with them. This risk-based approach would help the development teams in prioritizing the security requirements.

- Recommend Design Changes:

In this phase every security requirement must be associated with a security control. A security control best suited for the design is proposed and documented. These security controls are an elaborate view of the security requirements. Here, we would identify exact changes or additions to be incorporated in the design that are needed to meet any requirement or mitigate a threat. The changes or controls recommended for the design should be clear and detailed, as given in the instances below:

a. Elaborate validation strategy with respect to:

b. Identifying right application component like servlet filters, interceptors, validator classes etc.

c. Placement of check c. Validation mechanism

d. Use of 3rd party security API’s or inbuilt design features of the frameworks

e. Encryption techniques

f. Design Patterns And so on depending on the control to be built in the design.

- Discussion with the design team:

The list of security requirements and proposed controls must be then discussed with the development teams. The queries of the teams should be addressed and feasibility of incorporating the controls must be determined. Exceptions, if any must be taken into account and alternate recommendations should be proposed. In this phase a final agreement on the security controls is achieved.

- Design Finalization:

The final design incorporated by the development teams must be reviewed again and finalized for further development process.

Design flaws

This section describes some of the important design flaws that can leave a backdoor in the application to access it without authentication or manipulate its business logic. We will understand such flaws and secure design recommendations in detail.

- Business Logic Decision

During testing it is crucial to identify the key parameters related to business logic and understand how application handles them. This section will focus on insecure business logic decisions that are based on such parameters. Two such cases are listed below, it is important to look for such scenarios in the application while testing.

1. Use of non-editable controls – Applications may use the values of non-editable controls, drop-down menus, hidden fields or query string parameters for business logic processing. If such fields contain values like the type of the user, nature of the request, status of the transaction, etc. the attackers will get a chance to manipulate them and perform unauthorized operations. The application developers must understand that such fields are non-editable only in the context of the proxy tool. The attackers can easily modify their values using a proxy editor tool and try to manipulate business logic.

2. Business logic decision based on presence or absence of certain parameters - This is especially observed in ASP.NET applications where there is provision to make the server side controls hidden/invisible for certain users. However, in most cases it has been observed that if the users add the parameters corresponding to the UI elements that are kept hidden/invisible to them into the request, they are able to change the behavior of the server side logic. Consider a scenario where only

admin user can change password of other users of the system, as a result the field to enter username is only made visible to the admin user. However, if a normal a user tries to add username parameter in the request he/she will be able to trick the server in believing that the request has come from an admin user and try to change password of other users. Thus there exists a hole in such applications where the server side behavior can be influenced with request parameters. Users can perform unauthorized operations in the application by supplying the values for the inputs fields that are hidden from them. Secure Design Recommendation:

• The application must not expose such parameters to the users.

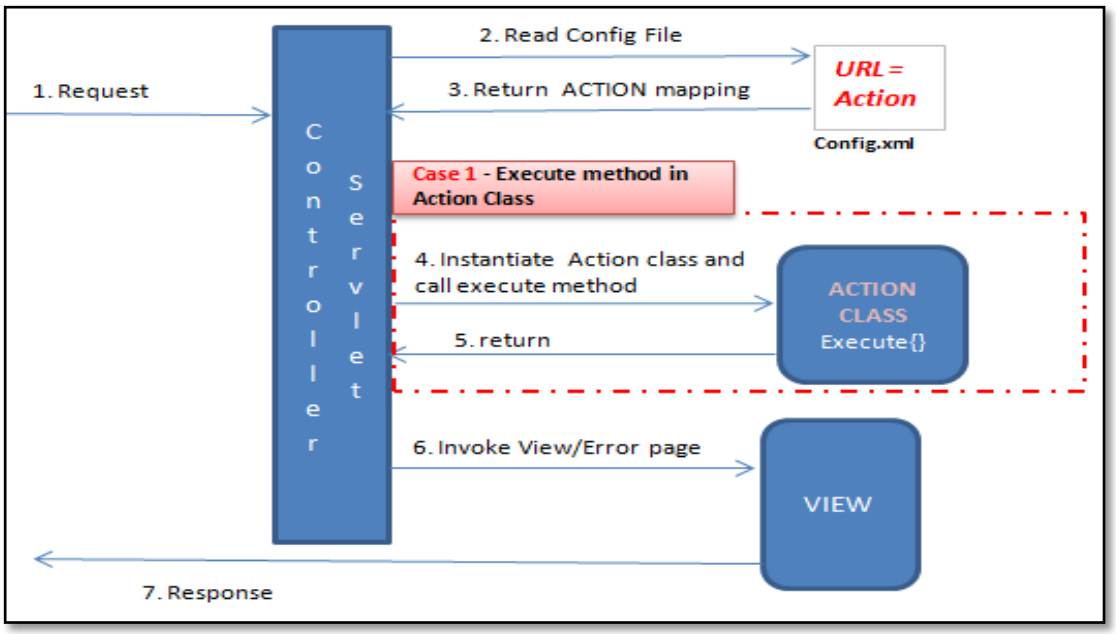
• If they are exposed, the application must not rely on request parameters for logical decisions. It mustmaintain a separate copy of such values at the server side and use the same for business logic processing.

• Apply proper authorization checks on the server side for all transactions, wherever necessary. Do not depend on presence of a user input for such decisions.

- Business Logic Invocation Technique

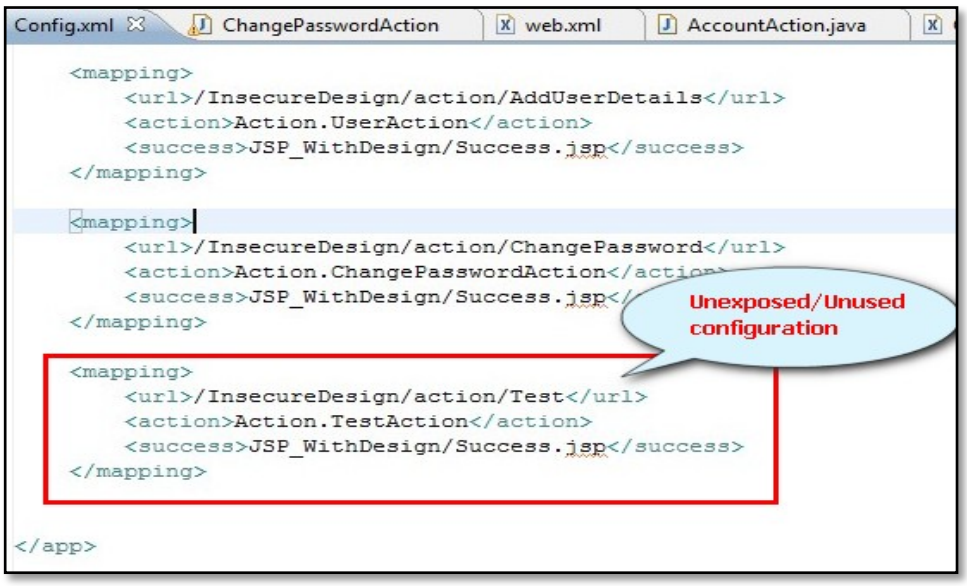
In most of the design techniques the request parameters or the URL's serve as sole factors to determine the processing logic. In such a scenario the elements in the request, which are used for such identifications, may be subject to manipulation attacks to obtain access to restricted resources or pages in the application.

Consider a design below; here the business logic class is identified based on a configuration file that keeps the mapping of the request URL and the business logic class i.e. action class.

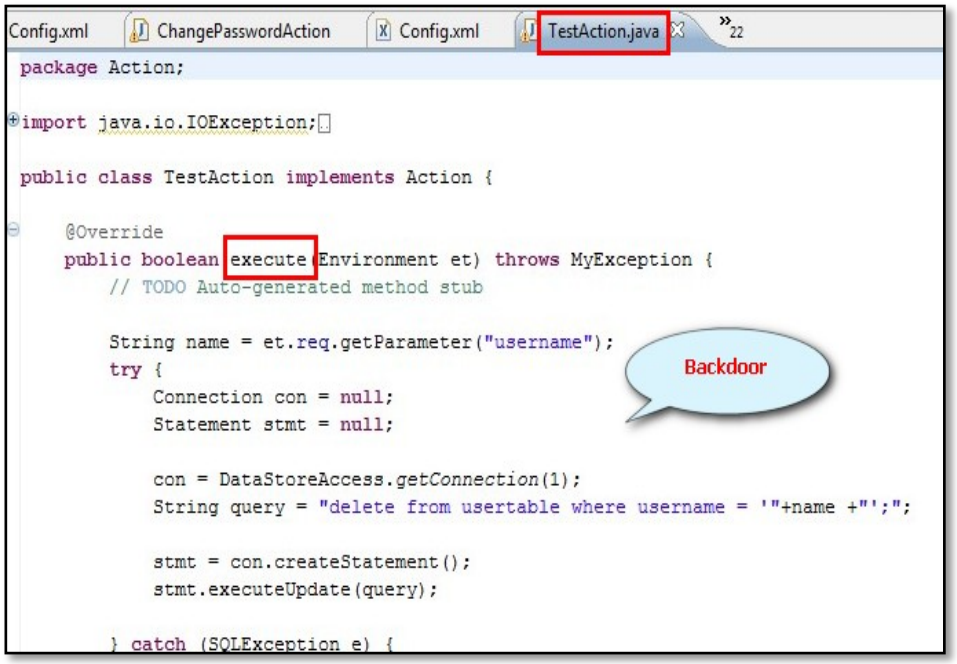


What is the flaw?

A flaw in such a design could be unused configurations present in the configuration file. Such configurations that are not exposed as valid features in the application and could serve as a potential backdoor to it. An unused configuration present in the configuration file of the application is shown below:

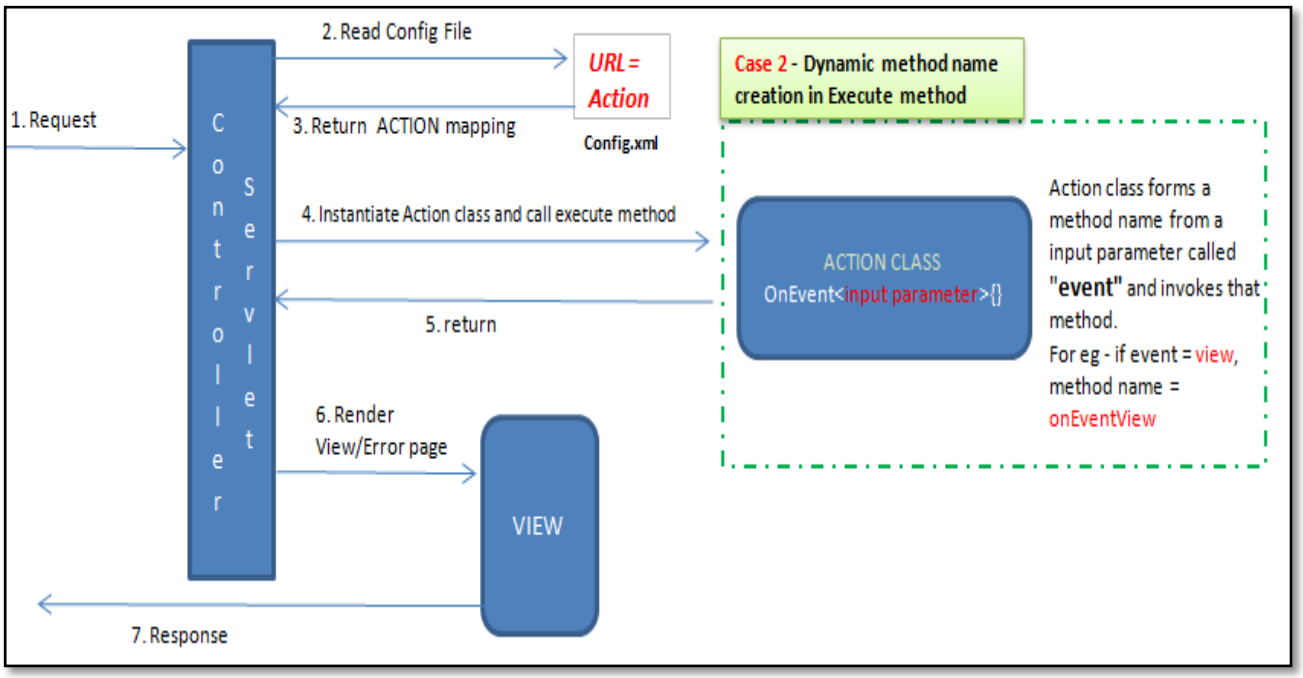


Observe that the “TestAction” has an insecure logic to delete records from the system. This can act as a potential backdoor to the application.



Consider another scenario

In the some designs request parameters are used to identify business logic methods. In the figure shown below a request parameter named “event” is used to identify and invoke the corresponding event handling methods of the business logic/action class.



What is the flaw?

Here, the user can attempt to invoke the methods of the events that are not visible to the user.

Secure Design Recommendation:

The applications must ensure to:

• Remove ALL redundant/test/unexposed business logic configurations from the file

• Apply necessary authorization check before processing business logic method

• Maintain a mapping of method/class/view names with the privilege level of the users, wherever applicable and restrict access of the users to restricted URLs/methods/views.

Review Criteria

Understand the business logic invocation technique used in the design of any application. Check if the user inputs are directly (i.e. without any restriction) used to determine any of the following elements (as applicable):

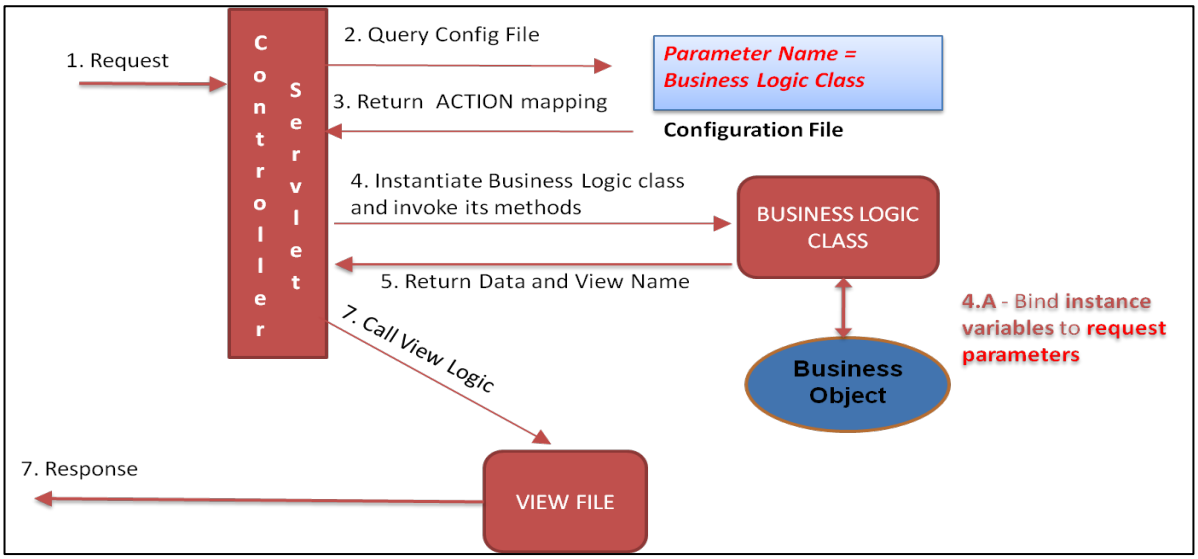
• Business logic class

• Method names

• View component

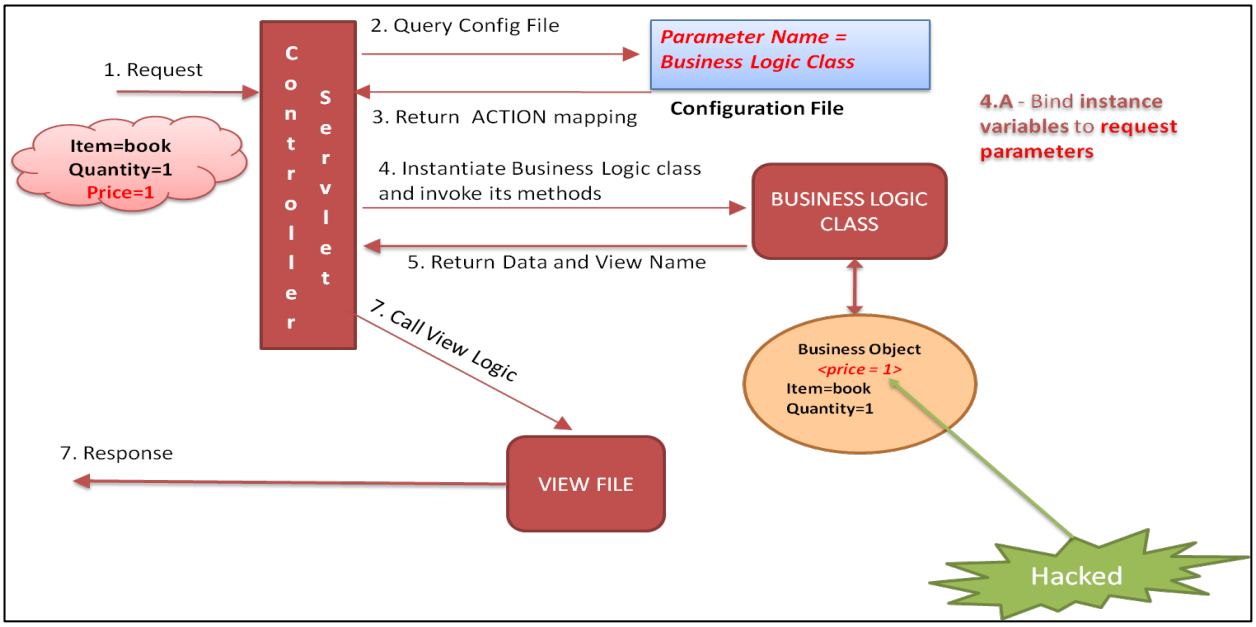
Data Binding Technique

Another popular feature seen in most of the design frameworks today is data binding, where the request parameters get directly bound to the variables of the corresponding business/command object. Binding here means that the instance variables of such classes get automatically initialized with the request parameter values based on their names. Consider a sample design given below; observe that the business logic class binds the business object with the request parameters.



What is the flaw?

The flaw in such design is that the business objects may have variables that are not dependent on the request parameters. Such variables could be key variables like price, max limit, role etc. having static values or dependent on some server side processing logic. A threat in such scenarios is that an attacker may supply additional parameters in request and try to bind values for unexposed variable of business object class. As illustrated in the figure below, the attacker sends an additional “price” parameter in the request and binds with the unexposed variable “price” in business object, thereby manipulating business logic.



Secure Design Recommendation

• An important point to be noted here is that the business/form/command objects must have only those instance variables that are dependent on the user inputs.

• If additional variables are present those must not be vital ones like related to the business rule for the feature.

• In any case the application must accept only desired inputs from the user and the rest must be rejected or left unbound. And initialization of unexposed of variables, if any must take place after the binding logic.

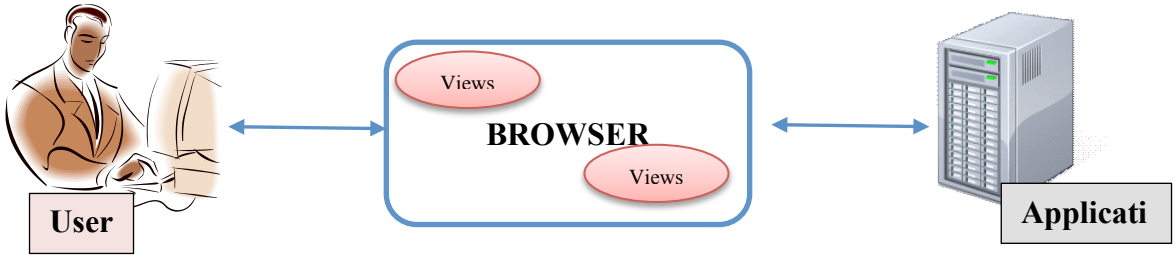
Review Criteria

Review the application design and check it is incorporates a data binding logic. In case it does, check if business objects/beans that get bound to the request parameters have unexposed variables that are meant to have static values. If such variables are initialized before the binding logic this attack will work successfully.

Placement of Security Controls

Placement of security checks is a vital area of review in an application design. Incorrect placements can render the applied security controls null and void. So, it is important to study the application design and spot the correctness of such checks in the overall execution flow of the design. Most of the

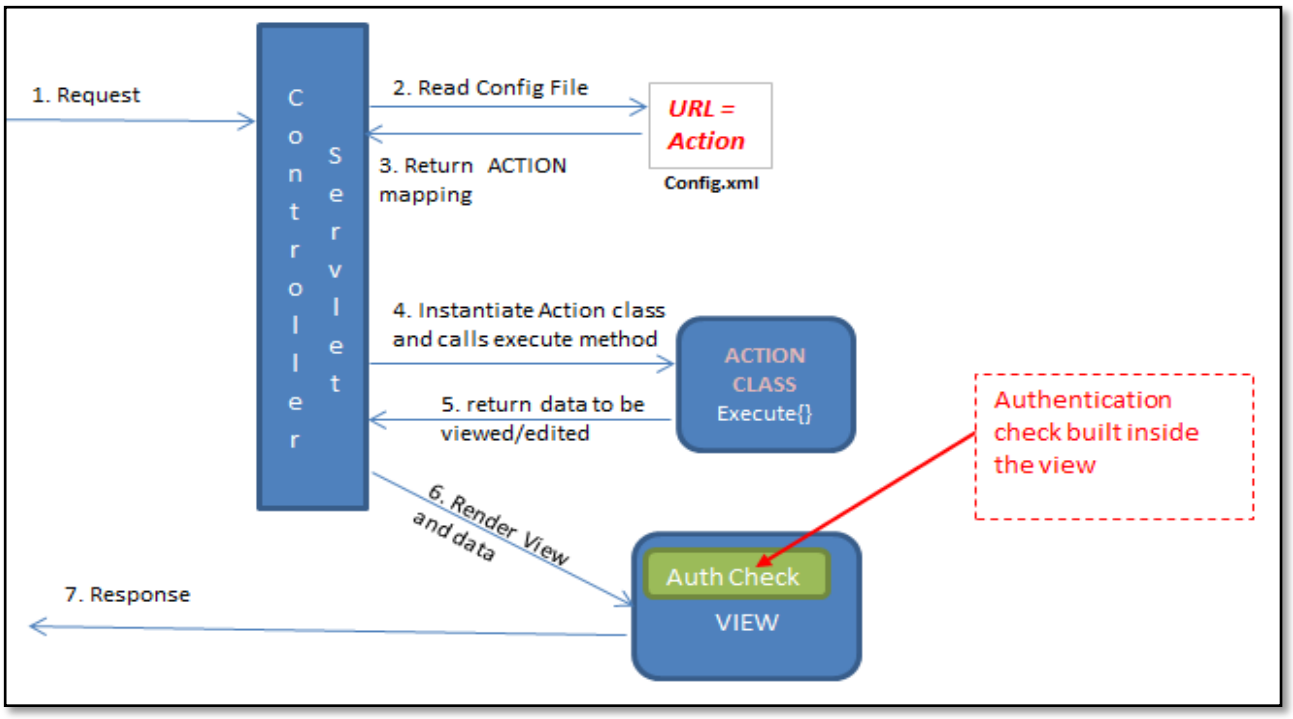
application designs are based on the concept of Model-View-Controller (MVC). They have a central controller, which listens to all incoming request and delegates control to appropriate form/business processing logic. And ultimately the user is rendered with a view. In such a layered design, when there are many entities involved in processing a request, developers often go wrong in placing the security controls at the right place. Most application developers feel “view” is the right place to have the security checks like authentication check etc.



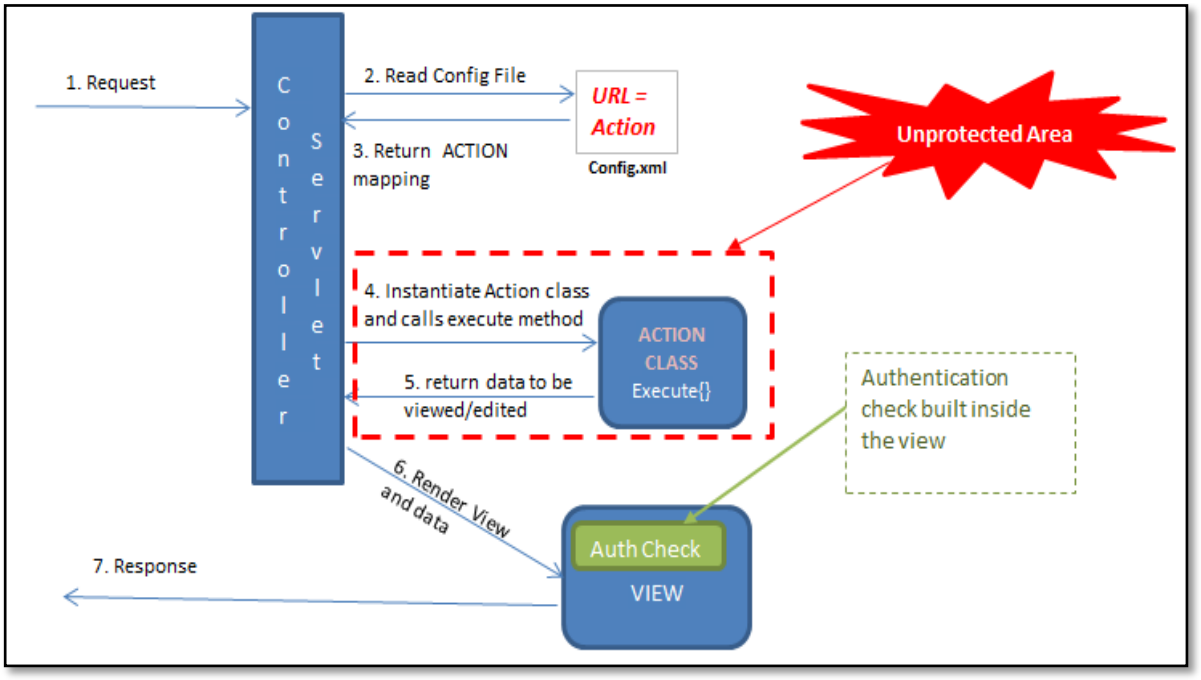
What is the flaw?

It thus seems logical that if you restrict the users at the page/view level they won’t be able to perform any operation in the application. But what if instead of requesting for a page/view an unauthorized user tries to request for an internal action like to action to add/modify any data in the application? It will get processed but the resultant view will be denied to the user; because the flaw lies in just having a view based access control in the applications. I am sure you will agree that a lot of processing for a request is done before the “view” comes into picture in any design. So the request to process any action will get processed successfully without authorization.

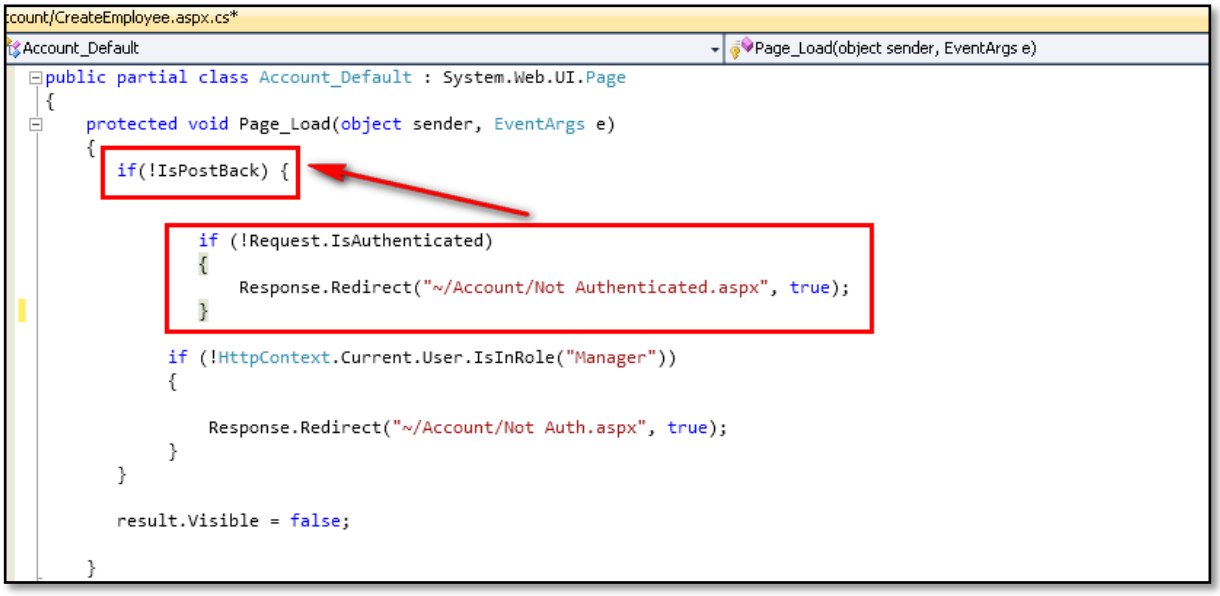
Consider a MVC based given in the figure below. Observe in the figure that the authentication check is present only in the view pages.



Observe that neither the controller servlet (central processing entity) nor the action classes have any access control checks. So here, if the user requests for an internal action like add user details, etc. without authentication it will get processed, but the only difference is that the user will be shown an error page as resultant view will be disallowed to the user.



Insecure POST-BACK's in ASP.NET A similar flaw is predominantly observed in ASP.NET applications where the developers tend to mix the code for handling POSTBACK's and authentication checks. Usually it is observed that the authentication check in the ASP.NET pages are not applied for POSTBACKs, as indicated below. Here, if an attacker tries to access the page without authentication an error page will be rendered. Instead, if the attacker tries to send an internal POSTBACK request directly without authentication it would succeed.



Secure Design Recommendation:

It is imperative to place all validation checks before processing any business logic and in case of ASP.NET applications independent of the POSTBACKs.

Review criteria

Check if the placement of the security checks is correct. The security controls like authentication check must be place before processing any request.

Execution Flow

Execution flow is another important consideration of design. The execution flow must terminate appropriately in case of an error condition. However, due to mishandling of some programming entities there could be a big hole in the application, which would allow unrestricted access to applications. One such flaw is related to – “sendRedirect” method in J2EE applications.

response.sendRedirect(“home.html”);

This method is used to send a redirection response to the user who then gets redirected to the desired web component who’s URL is passesd an argument to the method. One such misconception is that execution flow in the Servlet/JSP page that is redirecting the user stops after a call to this method. Take a look at the code snippet below, it checks for authenticated session using an “if” condition. If the condition fails the response.sendRedirect() is used to redirect the user to an error page.

```
<%@page import="DataAccess.DataStoreAccess"%>
<%@page import="DataAccess.QueryStore"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Add Branch Details</title>
</head>
<body>
<font face="tahoma"> <% String username = (String)session.getAttribute("user");
if(username ==null){
response.sendRedirect("../ErrorPage.jsp?message=User Not Authenticated");
}
    %> <%
    String branchname = request.getParameter("branchname");
    String address = request.getParameter("address");
    String phone = request.getParameter("phone");
    String fax = request.getParameter("fax");
    String city = request.getParameter("city");

    Connection con = null;
    PreparedStatement stmt = null;

    con = DataStoreAccess.getConnection(1);
    String query = QueryStore.addBranch;
    stmt = con.prepareStatement(query);
```

Note that there is code present after the If condition, which continues to fetch request parameters and processes business logic for instance adding a new branch entry of a bank in this case.

What is the flaw?

This flaw manifests as a result of the misconception that the execution flow in the JSP/Servlet page stops after the “sendRedirect” call. However it does not; in this case the execution of the servlet would continue even if an invalid session is detected by the “if” condition and thus the business logic will get processed for unauthenticated requests.

Note: The fact that execution of a servlet or JSP continues even after sendRedirect() method, also applies to Forward method of the RequestDispatcher Class. However, <jsp:forward> tag is an exception, it is observed that the execution flow stops after the use of <jsp:forward> tag.

Secure Design Recommendation:

Since this flaw results from the assumption made by developers that control flow execution terminates after a sendRedirect call, the recommendation would be to terminate the flow using a “return” statement.

Review criteria

Check if there is an appropriate logic to terminate the execution flow is present in case of an error condition. Check for similar instances of insecure security controls built using “sendRedirect” method.

References:

http://artechtalks.blogspot.in/

http://packetstormsecurity.com/files/119129/Insecure-Authentication-Control-In-J2EE.html

2.2.1.6 A Risk based approach to code review (Needs Content)

Lorem Ipsum

2.2.2 Crawling Code

Crawling code is the practice of scanning a code base of the review target in question. It is, in effect, looking for key pointers wherein possible security vulnerability might reside. Certain APIs are related to interfacing to the external world or file IO or user management, which are key areas for an attacker to focus on. In crawling code we look for APIs relating to these areas. We also need to look for business logic areas, which may cause security issues, but generally these are bespoke methods

which have bespoke names and can not be detected directly, even though we may touch on certain methods due to their relationship with a certain key API.

We also need to look for common issues relating to a specific language; issues that may not be *security* related but which may affect the stability/availability of the application in the case of extraordinary circumstances. Other issues when performing a code review are areas such a simple copyright notice in order to protect one's intellectual property. Generally these issues should be part of your companies Coding Guidelines, and should be enforceable during a code review (i.e. a reviewer can fail code review because the code violates something in the Coding Guidelines, regardless of whether or not the code would work in its current state, and regardless on whether the original developer agrees or not).

Crawling code can be done manually or in an automated fashion using automated tools. Crawling code manually is probably not effective, as (as can be seen below) there are plenty of indicators, which can apply to a language. Tools as simple as grep or wingrep can be used. Other tools are available which would search for key words relating to a specific programming language. If you are using a particular review tool, which allows you to specify strings to be highlighted in a review (e.g. Python based review tools using pygments syntax highlighter, or an in-house tool for which you can change the source code) then you could add the relevant string indicators from the lists below and have them highlighted to reviewers automatically.

The following sections shall cover the function of crawling code for Java/J2EE, .NET and Classic ASP. This section is best used in conjunction with the transactional analysis section also detailed in this guide.

- Searching for Key Indicators

The basis of the code review is to locate and analyse areas of code that may have application security implications. Assuming the code reviewer has a thorough understanding of the code, what it is intended to do, and the context in which it is to be used, firstly one needs to sweep the code base for areas of interest.

This can be done by performing a text search on the code base looking for keywords relating to APIs and functions. Below is a guide for .NET framework 1.1 & 2.0

2.2.2.1 Searching for Code in .NET

Firstly one needs to be familiar with the tools one can use in order to perform text searching, following this one needs to know what to look for.

In this section we will assume you have a copy of Visual Studio (VS) .NET at hand. VS has two types of search "Find in Files" and a cmd line tool called Findstr.

To start off, one could scan thorough the code looking for common patterns or keywords such as "User", "Password", "Pswd", "Key", "Http", etc... This can be done using the "Find in Files" tool in VS or using findstring as follows:

findstr /s /m /i /d:c:\projects\codebase\sec "http" *.*

- HTTP Request Strings

Requests from external sources are obviously a key area of a security code review. We need to ensure that all HTTP requests received are data validated for composition, max and min length, and if the data falls with the realms of the parameter white-list. Bottom-line is this is a key area to look at and ensure security is enabled.

request.acceptypes	request.browser	request.files	request.headers	request.httpmethod	request.item
request.querystring	request.form	request.cookies	request.certificate	request.rawurl	request.server variables
request.url	request.urlreferrer	request.useragent	request.userlanguages	request.IsSecureConnection	request.Total Bytes
request.BinaryRead	InputStream	HiddenField.Value	TextBox.Text	recordSet	

- HTML Output

Here we are looking for responses to the client. Responses which go unvalidated or which echo external input without data validation are key areas to examine. Many client side attacks result from poor response validation. XSS relies on this somewhat.

response.write	<% =	HttpUtility	HtmlEncode	UrlEncode

innerText	innerHTML			

- SQL & Database

Locating where a database may be involved in the code is an important aspect of the code review. Looking at the database code will help determine if the application is vulnerable to SQL injection. One aspect of this is to verify that the code uses either SqlParameter, OleDbParameter, or OdbcParameter(System.Data.SqlClient). These are typed and treat parameters as the literal value and not executable code in the database.

<link to SQL injection section>

exec sp_	select from	insert	update	delete from where	delete
execute sp_	exec xp_	exec @	execute @	executestatement	executeSQL
setfilter	executeQuery	GetQuery ResultIn XML	adodb	sqloledb	sql server
driver	Server.CreateObject	.Provider	.Open	ADODB.recordset	New OleDbConnection
ExecuteReader	DataSource	SqlCom mand	Microsoft .Jet	SqlDataReader	ExecuteReader
GetString	SqlDataAdapter	Comman dType	StoredPro cedure	System.Data.sql	

- Cookies

Cookie manipulation can be key to various application security exploits, such as session hijacking/fixation and parameter manipulation. One should examine any code relating to cookie functionality, as this would have a bearing on session security.

System.Net.Cookie	HTTPOnly	document.cookie

- HTML Tags

Many of the HTML tags below can be used for client side attacks such as cross site scripting. It is important to examine the context in which these tags are used and to examine any relevant data validation associated with the display and use of such tags within a web application.

HtmlEncode	URLEncode	<applet>	<frameset>	<embed>	<frame>	<html>
<iframe>	<img>	<style>	<layer>	<ilayer>	<meta>	<object>
<frame security	<iframe security	<body>				

- Input Controls

The input controls below are server classes used to produce and display web application form fields. Looking for such references helps locate entry points into the application.

htmlcontrols.htmlinputhidden	webcontrols.hiddenfield	webcontrols.hyperlink	webcontrols.textbox	webcontrols.label
webcontrols.linkbutton	webcontrols.listbox	webcontrols.checkboxlist	webcontrols.dropdownlist	

- WEB.Config

The .NET Framework relies on .config files to define configuration settings. The .config files are text-based XML files. Many .config files can, and typically do, exist on a single system. Web applications refer to a web.config file located in the application's root directory. For ASP.NET applications, web.config contains information about most aspects of the application's operation.

requestEncoding	responseEncoding	trace	authorization	compilation	CustomErrors
httpCookies	httpHandlers	httpRuntime	sessionState	maxRequestLength	debug
forms protection	appSettings	ConfigurationSettings	appSettings	connectionStrings	authentication mode
allow	deny	credentials	identity impersonate	timeout	remote

- Global.asax

Each application has its own Global.asax if one is required. Global.asax sets the event code and values for an application using scripts. One must ensure that application variables do not contain sensitive information, as they are accessible to the whole application and to all users within it.

Application_OnAuthenticateRequest	Application_OnAuthorizeRequest	Session_OnStart	Session_On End

- Logging

Logging can be a source of information leakage. It is important to examine all calls to the logging subsystem and to determine if any sensitive information is being logged. Common mistakes are logging userID in conjunction with passwords within the authentication functionality or logging database requests which may contains sensitive data.

log4net	Console.WriteLine	System.Diagnostics.Debug	System.Diagnostics.Trace

- Machine.config

Its important that many variables in machine.config can be overridden in the web.config file for a particular application.

validateRequest	enableViewState	enableViewStateMac

- Threads and Concurrency

Locating code that contains multithreaded functions. Concurrency issues can result in race conditions which may result in security vulnerabilities. The Thread keyword is where new threads objects are created. Code that uses static global variables that hold sensitive security information may cause session issues. Code that uses static constructors may also cause issues between threads. Not synchronizing the Dispose method may cause issues if a number of threads call Dispose at the same time, this may cause resource release issues.

Thread	Dispose

- Class Design

Public and Sealed relate to the design at class level. Classes that are not intended to be derived from should be sealed. Make sure all class fields are Public for a reason. Don't expose anything you don't need to.

Public	Sealed

- Reflection, Serialization

Code may be generated dynamically at runtime. Code that is generated dynamically as a function of external input may give rise to issues. If your code contains sensitive data, does it need to be serialized?

Serializable	AllowPartiallyTrustedCallersAttribute	GetObjectData	StrongNameIdentityPermission
StrongNameIdentity	System.Reflection		

- Exceptions & Errors

Ensure that the catch blocks do not leak information to the user in the case of an exception. Ensure when dealing with resources that the finally block is used. Having trace enabled is not great from an information leakage perspective. Ensure customized errors are properly implemented.

catch	finally	trace enabled	customErrors mode

- Crypto

If cryptography is used then is a strong enough cipher used, i.e. AES or 3DES? What size key is used? The larger the better. Where is hashing performed? Are passwords that are being persisted hashed? They should be. How are random numbers generated? Is the PRNG "random enough"?

RNGCryptoServiceProvider	SHA	MD5	base64	xor
DES	RC2	System.Random	Random	System.Security.Cryptography

- Storage

If storing sensitive data in memory, I recommend one uses the following.

SecureString	ProtectedMemory

- Authorization, Assert & Revert

Bypassing the code access security permission? Not a good idea. Also below is a list of potentially dangerous permissions such as calling unmanaged code, outside the CLR.

.RequestMinimum	.RequestOptional	Assert	Debug.Assert
CodeAccessPermission	ReflectionPermission.MemberAccess	SecurityPermission.ControlAppDomain	SecurityPermission.UnmanagedCode
SecurityPermission.SkipVerification	SecurityPermission.ControlEvidence	SecurityPermission.SerializationFormatter	SecurityPermission.ControlPrincipal
SecurityPermission.ControlDomainPolicy	SecurityPermission.ControlPolicy		

- Legacy Methods

Some standard functions that should be checked in any context include the following.

printf	strcpy

2.2.2.2 Searching for Code in Java

- Input and Output Streams

These are used to read data into one's application. They may be potential entry points into an application. The entry points may be from an external source and must be investigated. These may also be used in path traversal attacks or DoS attacks

<are some of these a bit wide ranging? java.io?>

java.io	java.util.zip	java.util.jar	FileInputStream	ObjectInputStream
FilterInputStream	PipedInputStream	SequenceInputStream	StringBufferInputStream	BufferedReader
ByteArrayInputS	CharArrayReader	File	ObjectInputStream	PipedInputStream

tream				
StreamTokenizer	getResourceAsStream	java.io.FileReader	java.io.FileWriter	java.io.RandomAccessFile
java.io.File	java.io.FileOutputStream	mkdir	renameTo	

- Servlets

These API calls may be avenues for parameter, header, URL, and cookie tampering, HTTP Response Splitting and information leakage. They should be examined closely as many of such APIs obtain the parameters directly from HTTP requests.

javax.servlet.*	getParameterNames	getParameterValues	getParameter	getParameterMap
getScheme	getProtocol	getContentType	getServerName	getRemoteAddr
getRemoteHost	getRealPath	getLocalName	getAttribute	getAttributeNames
getLocalAddr	getAuthType	getRemoteUser	getCookies	isSecure
HttpServletRequest	getQueryString	getHeaderNames	getHeaders	getPrincipal
getUserPrincipal	isUserInRole	getInputStream	getOutputStream	getWriter
addCookie	addHeader	setHeader	setAttribute	putValue
javax.servlet.http.Cookie	getName	getPath	getDomain	getComment
getMethod	getPath	getReader	getRealPath	getRequestURI
getRequestURL	getServerName	getValue	getValueNames	getRequestedSessionId

- Cross Site Scripting

These API calls should be checked in code review as they could be a source of Cross Site Scripting vulnerabilities.

javax.servlet.ServletOutputStream.print	javax.servlet.jsp.JspWriter.print	java.io.PrintWriter.print

- Response Splitting

Response splitting allows an attacker to take control of the response body by adding extra CRLFs into headers. In HTTP the headers and bodies are separated by 2 CRLF characters, and thus if an attackers input is used in a response header, and that input contained 2 CRLFs, then anything after the CRLFs would be interpreted as the response body. In code review ensure you are sanitizing any information being put into headers.

javax.servlet.http.HttpServletResponse.sendRedirect	addHeader	setHeader

- Redirection

Any time your application is sending a redirect response, ensure that the logic involved cannot be manipulated by an attackers input. Especially when input is used to determine where the redirect goes to.

sendRedirect	setStatus	addHeader	setHeader

- SQL & Database

Searching for Java Database related code this list should help you pinpoint classes/methods which are involved in the persistence layer of the application being reviewed.

jdbc	executeQuery	select	insert	update
delete	execute	executestatement	createStatem	java.sql.Result

			ent	Set.getString
java.sql.ResultSet.getObject	java.sql.Statement.executeUpdate	java.sql.Statement.executeQuery	java.sql.Statement.execute	java.sql.Statement.addBatch
java.sql.Connection.prepareStatement	java.sql.Connection.prepareCall			

- SSL

Looking for code which utilizes SSL as a medium for point to point encryption. The following fragments should indicate where SSL functionality has been developed.

com.sun.net.ssl	SSLContext	SSLSocketFactory
TrustManagerFactory	HttpsURLConnection	KeyManagerFactory

- Session Management

The following APIs should be checked in code review when they control session management.

getSession	invalidate	getId

- Legacy Interaction

Here we may be vulnerable to command injection attacks or OS injection attacks. Java linking to the native OS can cause serious issues and potentially give rise to total server compromise.

java.lang.Runtime.exec	java.lang.Runtime.getRuntime

- Logging

We may come across some information leakage by examining code below contained in one's application.

java.io.PrintStream.write	log4j	jLo	Lumberjack	MonoLog
qflog	just4log	log4Ant	JDLabAgent	

- Architectural Analysis

If we can identify major architectural components within that application (right away) it can help narrow our search, and we can then look for known vulnerabilities in those components and frameworks:

Ajax

XMLHTTP

Struts

org.apache.struts

Spring

org.springframework

Java Server Faces (JSF)

import javax.faces

Hibernate

import org.hibernate

Castor

org.exolab.castor

JAXB

javax.xml

JMS

JMS

- Ajax and JavaScript

Look for Ajax usage, and possible JavaScript issues:

document.write	eval	document.cookie

window.location	document.URL	window.createRequest

2.2.2.3 Searching for Code in Classic ASP

- Inputs

These API in ASP are commonly used to retrieve the input from the request. Therefore code review should ensure these requests (and dependent logic) cannot be manipulated by an attacker.

Request	Request.QueryString	Request.Form
Request.ServerVariables	Query_String	hidden
include	.inc	

- Output

These APIs are used by ASP to write the response body, that will be sent to the end user. Code review should check these requests are used in a proper manner and no sensitive information can be returned.

Response.Write	Response.BinaryWrite	<%=

- Cookies

Cookies can be a source of information leakage.

.cookies

- Error Handling

<link to error handling section> Ensure errors in your application are handled properly, otherwise an attacker could use error conditions to manipulate you application.

err.	Server.GetLastError	On Error Resume Next
On Error GoTo 0		

- Information in URL

These APIs are used to extract information from the URL object in the request. Code review should check that the information extracted from the URL is sanitized.

location.href	location.replace	method="GET"

- Database

These APIs can be used to interact with a database, which can lead to SQL attacks. Code review can check these API calls use sanitized input.

commandText	select from	update
insert into	delete from where	exec
execute	.execute	.open
ADODB.	commandtype	ICommand
IRowSet		

- Session

These API calls can control session within ASP applications.

session.timeout	session.abandon	session.removeall

- DOS Prevention

The following ASP APIs can help prevent DOS attacks against your application.

server.ScriptTimeout	IsClientConnected

- Logging

Leaking information to a log can be of use to an attacker; hence the following API call can be checked in code review to ensure no sensitive information is being written to logs.

WriteEntry

- Redirection

Do not allow attacker input to control when and where rejection occurs.

Response.AddHeader	Response.AppendHeader	Response.Redirect
Response.Status	Response.StatusCode	Server.Transfer
Server.Execute		

2.2.2.4 Searching for Code in Javascript and AJAX

Ajax and JavaScript have brought functionality back to the client side, which has brought a number of old security issues back to the forefront. The following keywords relate to API calls used to manipulate user state or the control the browser. The event of AJAX and other Web 2.0 paradigms has pushed security concerns back to the client side, but not excluding traditional server side security concerns.

Look for Ajax usage, and possible JavaScript issues:

eval	document.cookie	document.referrer	document.attachEvent
document.body	document.body.innerHtml	document.body.innerText	document.close

document.create	document.execCommand	document.forms[0].action	
document.location	document.open	document.URL	document.URLUnencoded
document.write	document.writeln	location.hash	location.href
location.search	window.alert	window.attachEvent	window.createRequest
window.execScript	window.location	window.open	window.navigate
window.setInterval	window.setTimeout	XMLHTTP	

2.2.2.5 Searching for Code in C++ and Apache

Commonly when a C++ developer is building a web service they will build a CGI program to be invoked by a web server (though this is not efficient) or they will use the Apache httpd framework and write a handler or filter to process HTTP requests/responses. To aid these developers, this section deals with generic C/C++ functions used when processing HTTP input and output, along with some of the common Apache APIs that are used in handlers.

- Legacy C/C++ Methods

For any C/C++ code interacting with web requests, code that handles strings and outputs should be checked to ensure the logic does not have any flaws.

exec	sprintf	snprintf	fprintf
printf	stdio	FILE	strcpy
strncpy	strcat	cout	cin
cerr	system	popen	stringstream

fstringstream	malloc	free	

- Request Processing

When coding within Apache, the following APIs can be used to obtain data from the HTTP request object.

headers_in	ap_read_request	post_read_request

- Response Processing

Depending on the type of response you wish to send to the client, the following Apache APIs can be used.

headers_out	ap_rprintf	ap_send_error_response
ap_send_fd	ap_vprintf	

- Cookie Processing

Cookie can be obtained from the list of request headers, or from specialized Apache functions.

headers_in	headers_out	ap_cookie_write
ap_cookie_write2	ap_cookie_read	ap_cookie_check_string

- Logging

Log messages can be implemented using custom loggers included in your module (e.g. log4cxx, boost::log, etc), by using the Apache provided logging API, or by simply writing to standard out or standard error.

cout	cerr	ap_open_stderr_log

ap_log_error	ap_log_perror	ap_log_rerror
ap_error_log2stderr		

- HTML Encoding

When you have got a handle for the HTML input or output in the C/C++ handler, the following methods can be used to ensure/check HTML encoding.

ap_unescape_all	ap_unescape_url	ap_unescape_url_keep2f
ap_unescape_urlencoded	ap_escape_path_segment	

2.2.3 Code Reviews and Compliance (Needs Content)

Lorem Ipsum

3 Reviewing by Technical Control (Missing Content)

We need to add content here to introduce the section.

3.1 Reviewing code for Authentication controls (Needs Content)

Lorem Ipsum

3.1.1 Forgot Password

Overview

If your web site needs to have user authentication then most likely it will require user name and password to authenticate user accesses. However as computer system have increased in complexity, so has authenticating users has also increased. As a result the code reviewer needs to be aware of the benefits and drawbacks of user authentication referred to as “Direct Authentication” pattern in this section. This section is going to emphasis design patterns for when users forget user id and or password and what the code reviewer needs to consider when reviewing how user id and passwords can be retrieved when forgotten by the user and how to do this in a secure manner.

General considerations

Notified user by (phone sms, email) an email where the user has to click a link in the email that takes them to your site and ask the user to enter a new password.

Ask user to enter login credentials they already have (Facebook, Twitter, Google, Microsoft Live, OpenID etc) to validate user before allowing user to change password.

Send notification to user to confirm register and or forgot password.

Send notifications that account information has been changed for registered email. Set appropriate time out value. I.e. If user does not respond to email within 48 hours then user will be frozen out of system until user re-affirms password change.

- General Considerations

1. The identity and shared secret/password must be transferred using encryption to provide data confidentiality. HTTPS should also be used but in itself should not be the only mechanism used for data confidentiality.

2. A shared secret can never be stored in clear text format, even if only for a short time in a message queue.

3. A shared secret must always be stored in hashed or encrypted format in a database.

4. The organization storing the encrypted shared secret does not need the ability to view or decrypt users passwords. User password must never be sent back to a user.

5. If the client must cache the username and password for presentation for subsequent calls to a Web service then a secure cache mechanism needs to be in place to protect user name and password.

6. When reporting an invalid entry back to a user, the username and or password should no be identified as being invalid. User feed back/error message must consider both user name and password as one item “user credential”. I.e. “The username or password you entered is incorrect.”

- AntiPatterns: Forget password.

1. Validate all fields have been completed correctly
    1. Avoid password fields being wiped out.
    2. Retain user’s email address between log-in and “Forgotten password” page.

2. CAPTCHA should be used as last resort or not all. CAPTCHA can be hacked.

3. OpenID does have security implications (e.i. if a hacker gains access to your OpenID, the hacker now potentially have access to all the sites you use with that OpenID.

4. Make sure security questions don’t ask for information that can easily be found on social sites like Facebook. E.I. “Mother’s maiden name”.

5. Do not mask user input of password. Show character until next character is typed in. Masking every character only provides security if someone is standing directly over you.

6. Do not send a onetime password to allow user to reset his/her password. This password would be stored even if for a short time in clear text and email storage is not the place to store passwords.

7. Do not have an error message saying account does not exists for this email address. This could be used to find out if user has an account for a porn or another site if hacker knows users email address.

8. Important: Do not allow listing of users. The password reset tokens should be uniquely generated, and be cryptographically secure random. Otherwise a listing of users can be generated from forget password feature.

3.1.2 Authentication (Needs Content)

Lorem Ipsum

3.1.3 CAPTCHA

CAPTCHA (an acronym for "Completely Automated Public Turing test to tell Computers and Humans Apart".) is an access control technique.

CAPTCHA is used to prevent automated software from gaining access to webmail services like Gmail, Hotmail and Yahoo to create e-mail spam, automated postings to blogs, forums and wikis for the purpose of promotion (commercial, and or political) or harassment and vandalism and automated account creation.

CAPTCHA’s have proved useful and their use has been upheld in court. Circumventing CAPTCHA has been upheld in US Courts as a violation Digital Millennium Copyright Act anti-circumvention section 1201(a)(3) and European Directive 2001/29/EC.

Code review of CAPTCHA’s the reviewer needs to pay attention to the following rules to make sure the CAPTCHA is built with strong security principals.

1. Do not allow the user to enter multiple guesses after an incorrect attempt.

2. The software designer and code review need to understand the statics of guessing. I.e. One CAPTCHA design shows four (3 cats and 1 boat) pictures, User is requested to pick the picture where it is not in the same category of the other pictures. Automated software will have a success rate of 25% by always picking the first picture. Second depending on the fixed pool of CAPTCHA images over time an attacker can create a database of correct answers then gain 100% access.

3. Consider using a key being passed to the server that uses a HMAC (Hash-based message authentication code) the answer.

Text base CAPTCHA's should adhere to the following security design principals...

1. Randomize the CAPTCHA length: Don't use a fixed length; it gives too much information to the attacker.

2. Randomize the character size: Make sure the attacker can't make educated guesses by using several font sizes / several fonts.

3. Wave the CAPTCHA: Waving the CAPTCHA increases the difficulty for the attacker.

4. Don't use a complex charset: Using a large charset does not improve significantly the CAPTCHA scheme's security and really hurts human accuracy.

5. Use anti-recognition techniques as a means of strengthening CAPTCHA security: Rotation, scaling and rotating some characters and using various font sizes will reduce the recognition efficiency and increase security by making character width less predictable.

6. Keep the line within the CAPTCHAs: Lines must cross only some of the CAPTCHA letters, so that it is impossible to tell whether it is a line or a character segment.

7. Use large lines: Using lines that are not as wide as the character segments gives an attacker a robust discriminator and makes the line anti-segmentation technique vulnerable to many attack techniques.

CAPTCHA does create issues for web sites that must be ADA (Americans with Disabilities Act of 1990) compliant. Code reviewer may need to be aware of web accessibilities and security to review the CAPTCHA implementation where web site is required to be ADA complaint by law.

Examples of a CAPTCHA

References:

1. UNITED STATES of AMERICA vs KENNETH LOWSON, KRISTOFER KIRSCH, LOEL STEVENSON Federal Indictment. February 23, 2010. Retrieved 2012-01-02.

2. http://www.google.com/recaptcha/captcha [[1]]

3. http://www.ada.gov/anprm2010/web%20anprm_2010.htm [[2]]

4. Inaccessibility of CAPTCHA - Alternatives to Visual Turing Tests on the Web http://www.w3.org/TR/turingtest/ [[3]]

- Notes from Renchie Joan...

If we can add a small section like, what are the uses of CAPTCHA when mapped to vulnerability categories. eg,: Since CAPTCHA is basically a challenge-response process, it can be used an effective mechanism against XSRF (CSRF) attack.

Or in preventing Dictionary attacks in a login screen.

Please let me know if you also think these are relevant and fit in the scope of the guide. If yes, then we will discuss and work on it.

3.1.4 Out of Band Considerations (Needs Content)

Lorem Ipsum

3.2 Reviewing code for Authorization weakness

- Authorisation in .NET MVC 4

The usage of filters is recommended when authorization is being implemented in MVC 4 .NET MVC 3 introduced a method in global.asax called RegisterGlobalFilters.The can be used to **DEFAULT DENY** access to URL's in the application.

```
public static void RegisterGlobalFilters(GlobalFilterCollection filters)
{
    filters.Add(new HandleErrorAttribute());
    filters.Add(new System.Web.Mvc.AuthorizeAttribute());
}
```

Is is recommended when reviewing MVC3/4 .NET to take a look at how authorization is being implemented. The line above, **filters.Add(newSystem.Web.Mvc.AuthorizeAttribute());** pretty much default denies access to any request without a valid session. If this is implemented we may need to

provide unauthorized access to certain pages such as a registration page, public welcome page or a login page. How do we do this?

AllowAnonymous is used to provide access to public pages with no valid session required. The code may look like this:

```
[AllowAnonymous]
public ActionResult LogMeIn(string returnUrl)
```

One must be careful that the pages which have AllowAnonymous enabled are actually designed for public consumption.

3.2.1 Checking authorization upon every request

Authorization is as important as authentication. Every functionality as well as every data access should be authorized. For data access authorization, application logic should check if the data belongs to the authenticated user, or if the user should be able to access that data.

Functionality authorization can be achieved through access control lists on small systems (such as an embedded system such as a router), but not via ACLs in an enterprise application. The complexity of an authorization model cannot be implemented by ACLs, and will definitely lead to human-errors that put the integrity of the system at risk.

- Role Based Access Control for Functionality

RBAC means assigning users to roles, and then roles to permissions. This is a more logical modeling of actual system authorization. On top of that, allows administrators to fine-grain and re-check role-permission assignments, and make sure that every role has exactly the permissions it is supposed to have (and nothing more or less). Then assigning users to roles will yield minimal human-error.

There are 4 levels of RBAC standardized by NIST, level 3 and 4 are almost never found. Level 2 introduces role hierarchy on top of level 1 (the simple RBAC), and has a better matching to enterprise model. Extended level 2 introduces hierarchical permissions as well, as one permission per functionality is required in a system, and in big systems, the number of available permissions soon

introduce human-errors. Depending on the size of the application, usage of different levels of RBAC systems is strongly advised.

Unfortunately there are not many fast enough implementations of the RBAC model, since it is very complex within. OWASP RBAC project introduces a very fast NIST Level 2 Extended RBAC implementation.

- Authorization Checklist

1. Every entry point should be authorized. Every functionality that an application performs, is a function, and should be authorized. Authorization should be check for every dynamic (generated) application access.

2. Every function should be authorized. Changing password, logging out, editing a certain record, and etc. are sample functions. Everything should be authorized.

3. Authorization checks should be fast and easy. Requiring multiple lines of complicated code for a single authorization is not recommended.

4. Authorization can be forced, or checked (depending on tolerance of application). For example:

```
if ($RBAC->hasAuthority($CurrentUser,"/users/passwords/change")) ShowChangePasswordLink();
//checked authority for visual manipulation in a view

$RBAC->authorize("/users/passwords/change"); //force authorization on a user management model
ChangePassword(...);
```

In case that a forced authorization fails, a HTTP 403 not authorized page can be shown. If the user is not logged in yet, a login page with not authorized error is more appropriate.

3.2.2 Reducing the attack surface (Needs Content)

Lorem Ipsum

3.2.3 SSL/TLS Implementations

Ensuring SSL with MVC.NET When reviewing MVC .NET is is important to make suer the application transmitts and received over a secure link. It is not recommended to only have the login pages over SSL and then default to clear. We also need to protect our session cookie as it is pretty much as useful as users credentials.

```
public static void RegisterGlobalFilters(GlobalFilterCollection filters)
{
    ......
    ......
    filters.Add(new RequireHttpsAttribute());
}
```

In the global.asax file we can review the RegisterGlobalFilters method. The attribute RequireHttpsAttribute() can be used to make sure the application runs over SSL/TLS It is recommended that this is enabled for SSL/TLS sites.

3.2.4 Reviewing code for session handling

General Considerations

1. If the system is critical, Session IDs should be cryptographically secure (i.e non determinable)

2. In big systems, sessions should not be stored in files (default PHP behavior). They should be stored in memory or in databases, to prevent DOS attacks on new sessions.

3. As soon as a confidential or higher session is formed for a user, they should have all their traffic transmitted through SSL. SessionID is almost as important as passwords.

4. A policy should be defined and forced on an application, to define the number of sessions a user can have. (One, Many, etc.) If this is left vague, it usually leads to security flaws.

5. Sessions require a general timeout, which happens at a certain time after creation (usually a week), and an idle timeout, which happens after a certain time of the session being idle (usually 30 minutes).

6. The idle timeout can be changed depending on the nature of the application (smaller for banking applications, larger for email composing clients)

7. The idle timeout doesn't have to be precise. The application can check for it every 2 minutes, and flush all timed-out idle sessions.

Data Type	Context	Defense
String	HTML Body	HTML Entity Encode
String	HTML Attribute	Minimal Attribute Encoding
String	GET Parameter	URL Encoding
String	Untrusted URL	URL Validation, avoid javascript: URLs, Attribute encoding, safe URL verification
String	CSS	Strict structural validation, CSS Hex encoding, good design
HTML	HTML Body	HTML Validation (JSoup, AntiSamy, HTML Sanitizer)
Any	DOM	DOM XSS Cheat Sheet
Untrusted JavaScript	Any	Sandboxing
JSON	Client Parse Time	JSON.parse() or json2.js

8. Sessions should be rolled when they are elevated. Rolling means that the session-id should be changed, and the session information should be transferred to the new id.

9. Sessions need to be cleared out on logout. It is a good idea to dispose of the session-id on logout as well.

Session Attacks

Generally three sorts of session attacks are possible:

1. Session Hijacking: stealing someone's session-id, and using it to impersonate that user.

2. Session Fixation: setting someone's session-id to a predefined value, and impersonating them using that known value

3. Session Elevation: when the importance of a session is changed, but its ID is not.

- Session Hijacking

1. Mostly done via XSS attacks, mostly can be prevented by HTTP-Only session cookies (unless Javascript code requires access to them).

2. It’s generally a good idea for Javascript not to need access to session cookies, as preventing all flavors of XSS are usually the toughest part of hardening a system.

3. Session-ids should be placed inside cookies, and not in URLs. URL informations are stored in browser's history, and HTTP Referrers, and can be accessed by attackers.

4. Geographical location checking can help detect simple hijacking scenarios. Advanced hijackers use the same IP (or range) of the victim.

5. An active session should be warned when it is accessed from another location.

6. An active users should be warned when s/he has an active session somewhere else (if the policy allows multiple sessions for a single user).

- Session Fixation

1. If the application sees a new session-id that is not present in the pool, it should be rejected and a new session-id should be advertised. This is the sole method to prevent fixation.

2. All the session-ids should be generated by the application, and then stored in a pool to be checked later for. Application is the sole authority for session generation.

- Session Elevation

1. Whenever a session is elevated (login, logout, certain authorization), it should be rolled.

2. Many applications create sessions for visitors as well (and not just authenticated users). They should definitely roll the session on elevation, because the user expects the application to treat them securely after they login.

3. When a down-elevation occurs, the session information regarding the higher level should be flushed.

3.2.5 Reviewing client side code (Needs Content)

Lorem Ipsum

3.2.5.1 Javascript

Three points of validity are required for Javascript codes:

1. Have all the logic server-side, Javascript is only the butler

2. Check for all sorts of XSS DOM Attacks

3. Check for insecure Javascript libraries and update them frequently.

Javascript uses strings to create DOM elements. This can lead to XSS attacks. All input should be sanitized before being converted to DOM objects.

Javascript libraries are not prone to attack. Most of them have flaws in them, recent jQuery flaw (evaluating the document.location.hash, allowing XSS to be embedded after # in location) caused Drupal (which is generally a safe system) to allow admin user creation for attackers!

3.2.5.2 JSON (Needs Content)

Lorem Ipsum

3.2.5.3 Content Security Policy (Needs Content)

Lorem Ipsum

3.2.5.4 “Jacking”/Framing

In order to help prevent clickjacking or UI redress attacks one of the following headers should be in all HTTP response headers.

X-Frame-Options HTTP Response Header

// to prevent all framing of this content response.addHeader("X-FRAME-OPTIONS", "DENY");

// to allow framing of this content only by this site response.addHeader("X-FRAME-OPTIONS", "SAMEORIGIN");

// to allow framing from a specific domain response.addHeader("X-FRAME-OPTIONS", "ALLOW-FROM X");

Older browsers dont usderstand the above headers. In order to help prevent regress attacks we may see the following code on the client side files.

Legacy Browser Clickjacking Defense

```
<style id="antiCJ">body{display:none !important;}</style>
<script type="text/javascript">
if (self === top)
{ var antiClickjack = document.getElementByID("antiCJ");
antiClickjack.parentNode.removeChild(antiClickjack)
}
else { top.location = self.location; }
</script>
```

3.2.5.5 HTML 5? (Needs Content)

Lorem Ipsum

3.2.5.6 Browser Defenses Policy (Needs Content)

Lorem Ipsum

3.2.5.7 Etc… (Needs Content)

Lorem Ipsum

3.2.6 Review code for input validation (Needs Content)

Lorem Ipsum

3.2.6.1 Regex Gotchas (Needs Content)

Lorem Ipsum

3.2.6.2 ESAPI (Needs Content)

Lorem Ipsum

3.2.7 Review code for contextual encoding

When untrusted data is to be rendered to the UI it MUST under both input validation and encoding. Encoding is of significant importance given it can protect the user from client side scripting attacks. XSS (Cross-Site-Scripting) attacks may consist of exploiting or breaching a users system:

-Session Hijacking

–Site Defacement

–Network Scanning

–Undermining CSRF Defenses

–Site Redirection/Phishing

–Load of Remotely Hosted Scripts

–Data Theft

–Keystroke Logging

The following is a suggested encoding matrix. When reviewing code is is important the untrusted data undergoes one of the following encoding schemes depending on the context of where is data sits on the page.

3.2.7.1 HTML Attribute

HTML Attribute Encoding: HTML attributes may contain untrusted data. It is important to determine if any ot the HTML attributes on a given page contains data from outside the trust boundary.

Some HTML attributes are considered safeer than others such as

align, alink, alt, bgcolor, border, cellpadding, cellspacing, class, color, cols, colspan, coords, dir, face, height, hspace, ismap, lang, marginheight, marginwidth, multiple, nohref, noresize, noshade, nowrap, ref, rel, rev, rows, rowspan, scrolling, shape, span, summary, tabindex, title, usemap, valign, value, vlink, vspace, width

when reviewing code for XSS we need to look for HTML attributes such as the folloiwng

```
<input type="text" name="fname" value="UNTRUSTED DATA">
```

Attacks may take the following format:

```
"><script>/* bad stuff */</script>
```

- What is Attribute encoding?

HTML attribute encoding replaces a subset of characters that are important to prevent a string of characters from breaking the attribute of an HTML element.

We replace ", &, and < with ", &, and >.

This is because the nature of attributes, the data they contain, and how they are parsed and interpreted by a browser or HTML parser is different than how an HTML document and its elements are read.

[XSS Prevention CheatSheet]: Except for alphanumeric characters, escape all characters with ASCII values less than 256 with the &#xHH; format (or a named entity if available) to prevent switching out of the attribute. The reason this rule is so broad is that developers frequently leave attributes unquoted. Properly quoted attributes can only be escaped with the corresponding quote. Unquoted attributes can be broken out of with many characters, including [space] % * + , - / ; < = > ^ and l.

Attribute encoding may be performed in a number of ways.

HttpUtility.HtmlAttributeEncode

http://msdn.microsoft.com/en-us/library/wdek0zbf.aspx

OWASP Java Encoder Project

OWASP Java Encoder Project https://www.owasp.org/index.php/OWASP_Java_Encoder_Project

3.2.7.2 HTML Entity

HTML elements which contain user controlled data or data from untrusted sourced should be reviewed for contextual output encoding. In the case of HTML entities we need to help ensure HTML Entity Encoding is perfromed:

- Example HTML Entity containing untrusted data:

HTML Body Context

UNTRUSTED DATA

OR

<body>...UNTRUSTED DATA </body>

OR

UNTRUSTED DATA

- HTML Entity Encoding is required

& --> &

< --> <

> --> >

" --> "

' --> '

It is recommended to review where/if untrusted data is placed within entity objects. searching the source code fro the following encoders may help establish if HTML entity encoding is being done in the application and in a consistent manner.

- OWASP Java Encoder Project

https://www.owasp.org/index.php/OWASP_Java_Encoder_Project

```
<input type="text" name="data" value="<%= Encode.forHtmlAttribute(dataValue) %>" />
```

- OWASP ESAPI

http://code.google.com/p/owasp-esapi-java/source/browse/trunk/src/main/java/org/owasp/esapi/codecs/HTMLEntityCodec.java

```
String safe = ESAPI.encoder().encodeForHTML( request.getParameter( "input" ) );
```

- PHP

http://php.net/manual/en/function.htmlentities.php

- .NET

http://msdn.microsoft.com/en-us/library/w3te6wfz.aspx

3.2.7.3 Javascript Parameters

Untrusted data, if being placed inside a Javascript function/code requires validation. Unvalidated data may break out of the data context and wind up being executed in the code context on a users browser.

- Examples of exploitation points (sinks) which are worth reviewing for:

```
<script>var currentValue='UNTRUSTED DATA';</script>
<script>someFunction('UNTRUSTED DATA');</script>
attack: ');/* BAD STUFF */
```

- Potential solutions:

OWASP HTML sanatiser Project

OWASP JSON Sanitizer Project

ESAPI javascript escaping can be call in this manner:

```
String safe = ESAPI.encoder().encodeForJavaScript( request.getParameter( "input" ) );
```

- Please note there are some JavaScript functions that can never safely use untrusted data as input - EVEN IF JAVASCRIPT ESCAPED!

For example:

```
<script>
window.setInterval('...EVEN IF YOU ESCAPE UNTRUSTED DATA YOU ARE XSSED HERE...');
</script>
```

- eval()

```
var txtField = "A1";
var txtUserInput = "'test@google.ie';alert(1);";
eval(   "document.forms[0]." + txtField + ".value =" + A1);
```

- jquery

```
var txtAlertMsg = "Hello World: ";
var txtUserInput = "test<script>alert(1)<\/script>";
$("#message").html(   txtAlertMsg +"" + txtUserInput + "");
Safe usage (use text, not html)
$("#userInput").text(   "test<script>alert(1)<\/script>");<-- treat user input as text
```

Nested Contexts Best to avoid such nested contexts: an element attribute calling a Javascript function etc These contexts can really mess with your mind.

```
<div onclick="showError('<%=request.getParameter("errorxyz")%>')" >An error occurred ....</div>
```

Here we have a HTML attribute(onClick) and within a nested Javascript function call (showError).

When the browser processes this it will first HTML decode the contents of the onclick attribute. It will pass the results to the JavaScript Interpreter. So we have 2 contextx here...HTML and Javascript (2 browser parsers). We need to apply “layered” encoding in the RIGHT order:

1) JavaScript encode

2) HTML Attribute Encode so it "unwinds" properly and is not vulnerable.

```
<div onclick="showError
  ('<%= Encoder.encodeForHtml(Encoder.encodeForJavaScript( request.getParameter("error")%>')))"
  >An error occurred ....</div>
```

3.2.7.4 JQuery (Needs Content)

Lorem Ipsum

3.2.8 Reviewing file and resource handling code (Needs Content)

Lorem Ipsum

3.2.9 Resource Exhaustion - error handling (Needs Content)

Lorem Ipsum

3.2.9.1 Native Calls (Needs Content)

Lorem Ipsum

3.2.10 Reviewing logging code - Detective Security (Needs Content)

Lorem Ipsum

3.2.11 Reviewing Error handling and Error messages

Error Handling

Proper error handling is important in two ways: <are these the only 2 ways?>

1. It may affect the state of the application

The initial failure to prevent the error may cause the application to traverse into an insecure state. This covers the key premise of failing securely <include reference to antipatterns>, errors induced should not leave the application in an insecure state. Resources should be locked down and released, sessions terminated (if required), and calculations or business logic should be halted (depending on the type of error, of course).

2. It may leak system information to a user

1. An important aspect of secure application development is to prevent information leakage. Error messages give an attacker great insight into the inner workings of an application. Weak error handling also aids the attacker, as the errors returned may assist them in constructing correct attack vectors.

A generic error page for most errors is recommended, this approach makes it more difficult for attackers to identify signatures of potentially successful attacks such as blind SQL injection using booleanization <should include reference> or analysis of response time characteristics.

- Reviewing Error Handling

The purpose of reviewing the Error Handling code is to assure that the application fails safely under all possible error conditions, expected and unexpected. No sensitive information is presented to the user when an error occurs. A company coding guidelines should include sections on Error Handling and how it should be controlled by an application suite, this will allow developers to code against this guidelines as well as review against them.

For example, SQL injection is much tougher to successfully execute without some healthy error messages. It lessens the attack footprint, and an attacker would have to resort to using “blind SQL injection” which is more difficult and time consuming.

A well-planned error/exception handling guideline is important in a company for three reasons:

1. Good error handling does not give an attacker any information which is a means to an end, attacking the application

2. A proper centralized error strategy is easier to maintain and reduces the chance of any uncaught errors “bubbling up” to the front end of an application.

3. Information leakage could lead to social engineering exploits, for example if the hosting companies name is returned, or some employees name can be seen.

Regardless of whether the development language provide checked exceptions or not, reviewers should remember:

1. Not all errors are exceptions

    1. Don’t rely on exception handling to be your only way of handling errors, handle all case statement 'default' sections, ensure all 'if' statements have their 'else' clauses covered, ensure that all exits from a function (e.g. return statements, exceptions, etc) are covered. RAII concepts (e.g. auto pointers and the like) are an advantage here. In languages like Java and C#, remember that errors are different from exceptions (different hierarchy) and should be handled.

2. Catching an exception is not automatically handling it

    1. You’ve caught your exception, so how do you handle it? For many cases this should be obvious enough, based on your business logic, but for some (e.g. out of memory, array index out of bounds, etc) the handling many not be so simple.

3. Don’t catch more that you can handle

    1. Catch all clauses (e.g. 'catch(Exception e)' in Java & C# or 'catch(...) in C++) should be avoided as you will not know what type of exception you are handling, and if you don't know the exception type, how do you accurately handle it? It could be that the downstream server is not responding, or a user may have exceeded their quota, or you may be out of memory, these issues should be handled in different ways and thus should be caught in exception clauses that are specific.

When an exception or error is thrown, we also need to log this occurrence. Sometimes this is due to bad development, but it can be the result of an attack or some other service your application relies on failing. This has to be imagined in the production scenario, if your application handles 'failing securely' by returning an error response to the client, and since we don't want to leak information that error will be generic, we need to have some way of identifying why the failure occurred. If your customer reports 1000's of errors occurred last night, you know that customer is going to want to know why. If you don't have proper logging and traceability coded into your application then you will not be able to establish if those errors were due to some attempted hack, or an error in your business logic when handling a particular type of error.

All code paths that can cause an exception to be thrown should check for success in order for the exception not to be thrown. This could be hard to impossible for a manual code review to cover, especially for large bodies of code. However if there is a debug version of the code, then modules/functions could throw relevant exceptions/errors and an automated tool can ensure the state and error responses from the module is as expected. This then means the code reviewer has the job of ensuring all relevant exceptions/errors are tested in the debug code.

- Centralized Error Handling

When reviewing code it is recommended that you assess the commonality within the application from a error/exception handling perspective. Frameworks have error handling resources which can be exploited to assist in secure programming, and such resources within the framework should be reviewed to assess if the error handling is "wired-up" correctly. A generic error page should be used for all exceptions if possible as this prevents the attacker from identifying internal responses to error states. This also makes it more difficult for automated tools to identify successful attacks.

For JSP structs this could be controlled in the struts-config.xml file, a key file when reviewing the wired-up struts environment:

```
<exception key=”bank.error.nowonga”
        path=”/NoWonga.jsp”
        type=”mybank.account.NoCashException”/>
```

Specification can be done for JSP in web.xml in order to handle unhandled exceptions. When unhandled exceptions occur, but are not caught in code, the user is forwarded to a generic error page:

```
<error-page>
    <exception-type>UnhandledException</exception-type>
    <location>GenericError.jsp</location>
</error-page>
```

Also in the case of HTTP 404 or HTTP 500 errors during the review you may find:

```
<error-page>
 <error-code>500</error-code>
 <location>GenericError.jsp</location>
</error-page>
```

For IIS development the 'Application_Error()' handler will allow the application to catch all uncaught exceptions and handle them in a consistent way. Note this is important or else there is a chance your exception information could be sent back to the client in the response.

For Apache development, returning failures from handlers or modules can prevent an further processing by the Apache engine and result in an error response from the server. Response headers, body, etc can be set by by the handler/module or can be configured using the "ErrorDocument" configuration.

We should use a localized description string in every exception, a friendly error reason such as “System Error – Please try again later”. When the user sees an error message, it will be derived from this description string of the exception that was thrown, and never from the exception class which may contain a stack trace, line number where the error occurred, class name, or method name.

Do not expose sensitive information like exception messages. Information such as paths on the local file system is considered privileged information; any internal system information should be hidden from the user. As mentioned before, an attacker could use this information to gather private user information from the application or components that make up the app.

Don’t put people’s names or any internal contact information in error messages. Don’t put any “human” information, which would lead to a level of familiarity and a social engineering exploit.

- Failing Securely

There can be many different reasons why an application may fail, for example:

• The result of business logic conditions not being met.

• The result of the environment wherein the business logic resides fails.

• The result of upstream or downstream systems upon which the application depends fail.

• Technical hardware / physical failure.

Failures are like the Spanish Inquisition; popularly nobody expected the Spanish Inquisition (see Monty Python) but in real life the Spanish knew when an inquisition was going to occur and were prepared for it, similarly in an application, though you don't expect errors to occur your code should be prepared for them to happen. In the event of a failure, it is important not to leave the "doors" of the application open and the keys to other "rooms" within the application sitting on the table. In the course of a logical workflow, which is designed based upon requirements, errors may occur which can be programmatically handled, such as a connection pool not being available, or a downstream server returning a failure.

Such areas of failure should be examined during the course of the code review. It should be examined if resources should be released such as memory, connection pools, file handles etc.

The review of code should also include pinpointing areas where the user session should be terminated or invalidated. Sometimes errors may occur which do not make any logical sense from a business logic perspective or a technical standpoint, for example a logged in user looking to access an account which is not registered to that user. Such conditions reflect possible malicious activity. Here we should review if the code is in any way defensive and kills the user’s session object and forwards the user to the login page. (Keep in mind that the session object should be examined upon every HTTP request).

How to Locate the Potentially Vulnerable Code

- Java

In Java we have the concept of an error object; the Exception object. This lives in the Java package java.lang and is derived from the Throwable object. Exceptions are thrown when an abnormal occurrence has occurred. Another object derived from Throwable is the Error object, which is thrown when something more serious occurs. The Error object can be caught in a catch clause, but cannot be handled, the best you can do is log some information about the Error and then re-throw it.

Information leakage can occur when developers use some exception methods, which 'bubble' to the user UI due to a poor error handling strategy. The methods are as follows:

printStackTrace()

getStackTrace()

Also important to know is that the output of these methods is printed in System console, the same as System.out.println(e) where there is an Exception. Be sure to not redirect the outputStream to PrintWriter object of JSP, by convention called "out", for example:

printStackTrace(out);

Note it is possible to change where system.err and system.out write to (like modifing fd 1 & 2 in bash or C/C++), using the java.lang.system package:

setErr() for the System.err field and setOut() for the System.out field.

This could be used on a process wide basis to ensure no output gets written to standard error or standard out (which can be reflected back to the client) but instead write to a configured log file.

- .NET

In .NET a System.Exception object exists and has commonly used child objects such as ApplicationException and SystemException are used. It is not recommended that you throw or catch a SystemException this is thrown by runtime.

When an error occurs, either the system or the currently executing application reports it by throwing an exception containing information about the error, similar to Java. Once thrown, an exception is handled by the application or by the default exception handler. This Exception object contains similar methods to the Java implementation such as:

StackTrace

Source

Message

HelpLink

In .NET we need to look at the error handling strategy from the point of view of global error handling and the handling of unexpected errors. This can be done in many ways and this article is not an exhaustive list. Firstly, an Error Event is thrown when an unhandled exception is thrown.

This is part of the TemplateControl class, see http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfSystemWebUITemplateControlClassErrorTopic.asp

Error handling can be done in three ways in .NET, executed in the following order:

• On the aspx or associated codebehind page in the Page_Error.

• In the global.asax file's Application_Error (as mentioned before).

• In the web.config file's customErrors section.

It is recommended to look in these areas to understand the error strategy of the application.

- Classic ASP

Unlike Java and .NET, classic ASP pages do not have structured error handling in try-catch blocks. Instead they have a specific object called "err". This makes error handling in a classic ASP pages hard to do and prone to design errors on error handlers, causing race conditions and information leakage. Also, as ASP uses VBScript (a subtract of Visual Basic), sentences like "On Error GoTo label" are not available.

- C++

In the C++ language, any object or built in type can be thrown. However there is a STL type std::exception which is supposed to be used as the parent of any user defined exception, indeed this type is used in the STL and many libraries as the parent type of all exceptions. Having std::exception encourages developers to create a hierarchy of exceptions which match the exception usage and means all exceptions can be caught by catching a std::exception object (instead of a 'catch (...)' block). Unlike Java, even errors that you can't recover from (e.g. std::bad_alloc which means your out of memory) derive from std::exception, so a 'catch(std::exception& e)' is similar to 'catch (...)' except that it allows you access to the exception so you can know what occurred and possibly print some error information using e.what().

There are many logging libraries for C++, so if your codebase uses a particular logging class look for usages of that logger for anywhere sensitive information can be written to the logs.

Vulnerable Patterns for Error Handling in IIS

- Page_Error

Page_Error is page level handling which is run on the server side in .NET. Below is an example but the error information is a little too informative and hence bad practice.

```
<script language="C#" runat="server">
Sub Page_Error(Source As Object, E As EventArgs)
Dim message As String = Request.Url.ToString()& Server.GetLastError().ToString()
Response.Write(message) // display message
End Sub
</script>
```

The text in the example above has a number of issues:

• Firstly, it displays the HTTP request to the user in the form of Request.Url.ToString(). Assuming there has been no data validation prior to this point, we are vulnerable to cross site scripting attacks!

• Secondly, the error message and stack trace is displayed to the user using Server.GetLastError().ToString() which divulges internal information regarding the application.

After the Page_Error is called, the Application_Error sub is called.

- Global.asax

When an error occurs, the Application_Error function is called. In this method we can log the error and redirect to another page. In fact catching errors in Application_Error instead of Page_Error would be an example of centralizing errors as described earlier,

```
<%@ Import Namespace="System.Diagnostics" %>
<script language="C#" runat="server">
void Application_Error(Object sender, EventArgs e) {
  String Message = "\n\nURL: http://localhost/" + Request.Path
                   + "\n\nMESSAGE:\n " + Server.GetLastError().Message
                   + "\n\nSTACK TRACE:\n" + Server.GetLastError().StackTrace;
  // Insert into Event Log
  EventLog Log = new EventLog();
  Log.Source = LogName;
  Log.WriteEntry(Message, EventLogEntryType.Error);
  Server.Redirect(Error.htm) // this shall also clear the error
}
</script>
```

Above is an example of code in Global.asax and the Application_Error method. The error is logged and then the user is redirected. Non-validated parameters are being logged here in the form of Request.Path. Care must be taken not to log or display non-validated input from any external source. <link to XSS>

- Web.config

Web.config has custom error tags which can be used to handle errors. This is called last and if Page_error or Application_error is called and has functionality, that functionality shall be executed first. If the previous two handling mechanisms do not redirect or clear (Response.Redirect or a Server.ClearError), this will be called and you shall be forwarded to the page defined in web.config in the customErrors section, which is configured as follows:

```
<customErrors mode="<On|Off|RemoteOnly>" defaultRedirect="<default redirect page>">
 <error statusCode="<HTTP status code>" redirect="<specific redirect page for listed status code>"/>
</customErrors>
```

The "mode" attribute value of "On" means that custom errors are enabled whilst the "Off" value means that custom errors are disabled. The "mode" attribute can also be set to "RemoteOnly" which specifies that custom errors are shown only to remote clients and ASP.NET errors are shown to requests coming from the the local host. If the "mode" attribute is not set then it defaults to "RemoteOnly".

When an error occurs, if the status code of the response matches one of the error elements, then the relevant 'redirect' value is returned as the error page. If the status code does not match then the error page from the 'defaultRedirect' attribute will be displayed. If no value is set for 'defaultRedirect' then a generic IIS error page is returned.

An example of the customErrors section completed for an application is as follows:

```
<customErrors mode="On" defaultRedirect="error.html">
    <error statusCode="500" redirect="err500.aspx"/>
    <error statusCode="404" redirect="notHere.aspx"/>
```

```
    <error statusCode="403" redirect="notAuthz.aspx"/>
</customErrors>
```

Vulnerable Patterns for Error Handling in Apache

In Apache you have two choices in how to return error messages to the client:

1. You can write the error status code into the req object and write the response to appear the way you want, then have you handler return 'DONE' (which means the Apache framework will not allow any further handlers/filters to process the request and will send the response to the client.

2. Your handler or filter code can return pre-defined values which will tell the Apache framework the result of your codes processsing (essentially the HTTP status code). You can then configure what error pages should be returned for each error code.

In the interest of centralizing all error code handling, option 2 can make more sense. To return a specific pre-defined value from your handler, refer to the Apache documentation for the list of values to use, and then return from the handler function as shown in the following example:

```
static int my_handler(request_rec *r)
{
    if ( problem_processing() )
    {
      return HTTP_INTERNAL_SERVER_ERROR;
    }

    ... continue processing request ...
}
```

<what happens if an uncaught exception occurs in the filter/handler? Will httpd core like any C++ thread?>

In the httpd.conf file you can then specify which page should be returned for each error code using the 'ErrorDocument' directive. The format of this directive is as follows:

ErrorDocument <3-digit-code> <action>

... where the 3 digit code is the HTTP response code set by the handler, and the action is a local or external URL to be returned, or specific text to display. The following examples are taken from the Apache ErrorDocument documentation (https://httpd.apache.org/docs/2.4/custom-error.html) which contains more information and options on ErrorDocument directives:

ErrorDocument 500 "Sorry, our script crashed. Oh dear"

ErrorDocument 500 /cgi-bin/crash-recover

ErrorDocument 500 http://error.example.com/server_error.html

ErrorDocument 404 /errors/not_found.html

ErrorDocument 401 /subscription/how_to_subscribe.html

Leading Practice for Error Handling

- Try & Catch (Java/.NET/C++)

Code that might throw exceptions should be in a try block and code that handles exceptions in a catch block. The catch block is a series of statements beginning with the keyword catch, followed by an exception type and an action to be taken.

Example:

Java Try-Catch:

```
public class DoStuff {
    public static void Main() {
        try {
            StreamReader sr = File.OpenText("stuff.txt");
            Console.WriteLine("Reading line {0}", sr.ReadLine());
```

```
        }
        catch(MyClassExtendedFromException e) {
            Console.WriteLine("An error occurred. Please leave to room”);
                logerror(“Error: “, e);
        }
    }
}
```

- .NET Try–Catch

```
public void run() {
            while (!stop) {
                try {

                    // Perform work here

                } catch (Throwable t) {
                    // Log the exception and continue
                            WriteToUser(“An Error has occurred, put the kettle on”);
                    logger.log(Level.SEVERE, "Unexception exception", t);
                }
            }
        }
```

- C++ Try–Catch

```
void perform_fn() {
  try {
     // Perform work here
```

```
  } catch ( const MyClassExtendedFromStdException& e) {
     // Log the exception and continue
     WriteToUser(“An Error has occurred, put the kettle on”);
     logger.log(Level.SEVERE, "Unexception exception", e);
  }
}
```

In general, it is best practice to catch a specific type of exception rather than use the basic catch(Exception) or catch(Throwable) statement in the case of Java.

- The Order Of Catching Exceptions

Keep in mind that many languages will attempt to match the thrown exception to the catch clause even if it means matching the thrown exception to a parent class. Also remember that catch clauses are checked in the order they are coded on the page. This could leave you in the situation where a certain type of exception might never be handled correctly, take the following example where 'non_even_argument' is a subclass of 'std::invalid_argument':

```
class non_even_argument : public std::invalid_argument {
public:
 explicit non_even_argument (const string& what_arg);
};

void do_fn()
{
   try
   {
      // Perform work that could throw
   }
   catch ( const std::invalid_argument& e )
```

```
  {
    // Perform generic invalid argument processing and return failure
  }
  catch ( const non_even_argument& e )
  {
    // Perform specific processing to make argument even and continue processing
  }
}
```

The problem with this code is that when a 'non_even_argument is thrown, the catch branch handling 'std::invalid_argument' will always be executed as it is a parent of 'non_even_argument' and thus the runtime system will consider it a match (this could also lead to slicing). Thus you need to be aware of the hierarchy of your exception objects and ensure that you list the catch for the more specific exceptions first in your code.

Finally

If the language in question has a finally method, use it. The finally method is guaranteed to always be called. The finally method can be used to release resources referenced by the method that threw the exception. This is very important. An example would be if a method gained a database connection from a pool of connections, and an exception occurred without finally, the connection object shall not be returned to the pool for some time (until the timeout). This can lead to pool exhaustion. finally() is called even if no exception is thrown.

```
try {
      System.out.println("Entering try statement");
      out = new PrintWriter(new FileWriter("OutFile.txt"));
    //Do Stuff….

  } catch (IOException e) {
      System.err.println("Input exception ");
```

```
} catch (Exception e) {
    System.err.println("Error occurred!”);

} finally {

    if (out != null) {
        out.close(); // RELEASE RESOURCES
    }
}
```

A Java example showing finally() being used to release system resources.

- Classic ASP Error Handling

In classic ASP there are two ways to do error handling, the first is using the err object with a "On Error Resume Next" and "On Error GoTo 0".

```
Public Function IsInteger (ByVal Number)
  Dim Res, tNumber
  Number = Trim(Number)
  tNumber=Number
  On Error Resume Next                    'If an error occurs continue execution
  Number = CInt(Number)                   'if Number is a alphanumeric string a Type Mismatch error will occur
  Res = (err.number = 0)                  'If there are no errors then return true
  On Error GoTo 0                         'If an error occurs stop execution and display error
  re.Pattern = "^[\+\-]? *\d+$"       'only one +/- and digits are allowed
  IsInteger = re.Test(tNumber) And Res
End Function
```

The second is using an error handler on an error page (http://support.microsoft.com/kb/299981).

```
Dim ErrObj
set ErrObj = Server.GetLastError()
'Now use ErrObj as the regular err object
```

- Releasing resources and good housekeeping

RAII

RAII is Resource Acquisition Is Initialization, which is a way of saying that when you first create an instance of a type, it should be fully setup (or as much as possible) so that it's in a good state. Another advantage of RAII is how objects are disposed of, effectively when an object instance is no longer needed then it resources are automatically returned when the object goes out of scope (C++) or when it's 'using' block is finished (C# 'using' directive which calls the Dispose method, or Java 7's try-with-resources feature) <add reference to http://docs.oracle.com/javase/7/docs/technotes/guides/language/try-with-resources.html >.

RAII has the advantage that programmers (and users to libraries) don't need to explicitly delete objects, the objects will be removed themselves, and in the process of removing themselves (destructor or Dispose)

- Classic ASP

For Classic ASP pages it is recommended to enclose all cleaning in a function and call it into an error handling statement after an "On Error Resume Next".

Centralized exception handling (Struts Example)

Building an infrastructure for consistent error reporting proves more difficult than error handling. Struts provides the ActionMessages and ActionErrors classes for maintaining a stack of error messages to be reported, which can be used with JSP tags like <html: error> to display these error messages to the user.

To report a different severity of a message in a different manner (like error, warning, or information) the following tasks are required:

1. Register, instantiate the errors under the appropriate severity

2. Identify these messages and show them in a consistent manner.

Struts ActionErrors class makes error handling quite easy:

```
ActionErrors errors = new ActionErrors()
errors.add("fatal", new ActionError("...."));
errors.add("error", new ActionError("...."));
errors.add("warning", new ActionError("...."));
errors.add("information", new ActionError("...."));
saveErrors(request,errors); // Important to do this
```

Now that we have added the errors, we display them by using tags in the HTML page.

```
<logic:messagePresent property="error">
<html:messages property="error" id="errMsg" >
   <bean:write name="errMsg"/>
</html:messages>
</logic:messagePresent >
```

- Classic ASP

For classic ASP pages you need to do some IIS configuration, follow http://support.microsoft.com/kb/299981 for more information.

3.2.12 Reviewing Security alerts (Needs Content)

Lorem Ipsum

3.2.13 Reviewing for active defense

Active Defense

Attack detection undertaken at the application layer has access to the complete context of an interaction and enhanced information about the user. The application knows what is a high-value issue and what is noise. Input data are already decrypted and canonicalized within the application and therefore application-specific intrusion detection is less susceptible to advanced evasion techniques. This leads to a very low level of attack identification false positives, providing appropriate detection points are selected.

The fundamental requirements are the ability to perform four tasks:

• Detection of a selection of suspicious and malicious events

• Use of this knowledge centrally to identify attacks

• Selection of a predefined response

• Execution of the response.

Applications can undertake a range of responses, which may include changes to a user's account or other changes to the application's defensive posture. It can be difficult to detect active defense in dynamic analysis since the responses may be invisible to the tester. Code review is the best method to determine the existence of this defense.

Other application functionality like authentication failure counts and lockout, or limits on rate of file uploads are localized protection mechanisms. This sort of standalone logic is not active defense equivalents in the context of this review, unless they are rigged together into an application-wide sensory network and centralized analytical engine.

It is not a bolt-on tool or code library, but instead offers insight to an approach for organizations to specify or develop their own implementations – specific to their own business, applications, environments and risk profile – building upon existing standard security controls. However, some developers may have used the following components:

• Category:OWASP Enterprise Security API

• AppSensor demonstration code

Purpose of Code Review

In this case, a code review is being used to detect the presence of a defense, and it is the absence of this that is a weakness. Note that active defense cannot defend an application that has known vulnerabilities, and therefore the other parts of this guide are extremely important. The code reviewer should note the absence of active defense as a **vulnerability**.

The purpose of code review is not necessarily to determine the efficacy of the active defense, but could simply be to determine if such capability exists.

How to Locate the Attack Detection and Response Code

- Detection Points

Detection points can be integrated into presentation, business and data layers of the application. Application-specific intrusion detection does not need to identify all invalid usage, to be able to determine an attack. There is no need for “infinite data” or “big data” and therefore the location of "detection points" may be very sparse within source code.

A useful approach for identifying such code is to find the name of the only guidance in this area by searching for the string:

AppSensor

Additionally search for AppSensor's detection point type identities:

RE1, RE2, ... RE8

AE1, AE2, ... AE12

SE1, SE2, ... SE6

ACE1, ACE2, ... ACE4

IE1, IE2, ... IE6

EE1, EE2

CIE1, CIE2, ... CIE4

FIO1, FIO2

HT1, HT2, HT3

UT1, UT2, ... UT4

STE1, STE2, STE3

RP1, RP2, RP3, RP4

Additionally search for any tagging based on Mitre's Common Attack Pattern Enumeration and Classification (CAPEC) such as strings like:

CAPEC-212, CAPEC-213, etc

The AppSensor detection point type identifiers and CAPEC codes will often have been used in configuration values (e.g. in ESAPI for Java), parameter names and security event classification. Also, examine error logging and security event logging mechanisms as these may be being used to collect data that can then be used for attack detection. Identify the code or services called that perform this logging and examine the event properties recorded/sent. Then identify all places where these are called.

An examination of error handling code relating to input and output validation is very likely to reveal the presence of detection points. For example, in a whitelist type of detection point, additional code may have been added adjacent, or within error handling code flow:

```
if ( var !Match this ) {
    Error handling
    Record event for attack detection
```

```
}
```

In some situations attack detection points are looking for blacklisted input, and the test may not exist otherwise, so brand new code is added:

```
if ( var !Match that ) {
    Record event for attack detection
}
```

Identification of detection points should also have found the locations where events are recorded (the "event store").

If detection points cannot be found, continue to review the code for execution of response, as this may provide insight into the existence of active defense.

- Attack Identification

The event store has to be analysed in real time or very frequently, in order to identify attacks based on predefined criteria. The criteria should be defined in configuration settings (e.g. in configuration files, or read from another source such as a database).

A process will examine the event store to determine if an attack is in progress - typically this will be attempting to identify an authenticated user, but it may also consider a single IP address, range of IP addresses, or groups of users such as one or more roles, users with a particular privilege or even all users.

- Selection of Response

Once an attack has been identified, the response will be selected based on predefined criteria. Again an examination of configuration data should reveal the thresholds related to each detection point, groups of detection points or overall thresholds.

Additionally search for AppSensor's response type identities as they may have been used in configuration settings, parameter names or in logical operations:

ASR-A, ASR-B, ... ASR-N, ASR-P

The most common response actions are user warning messages, log out, account lockout and administrator notification. However, as this approach is connected into the application, the possibilities of response actions are limited only by the coded capabilities of the application.

- Execution of Response

Search code for any global includes that poll attack identification/response identified above. Response actions (agains a user, IP address, group of users, etc) will usually be initiated by the application, but in some cases other applications (e.g. alter a fraud setting) or infrastructure components (e.g. block an IP address range) may also be involved.

Examine configuration files and any external communication the application performs. The following types of responses may have been coded:

• Logging increased

• Administrator notification

• Other notification (e.g. other system)

• Proxy

• User status change

• User notification

• Timing change

• Process terminated (same as traditional defenses)

• Function amended

• Function disabled

• Account log out

• Account lock out

• Application disabled

• Collect data from user.

Other capabilities of the application and related system components can be repurposed or extended, to provide the selected response actions. Therefore review the code associated with any localised security measures such as account lock out.

Leading Practice for Active Defense

The guidance for adding active response to applications given in the OWASP_AppSensor_Project, and in particular the AppSensor Guide v2.

3.2.14 Reviewing Secure Storage (Needs Content)

Lorem Ipsum

3.2.15 Hashing & Salting - When, How, and Where

Introduction

A cryptographic hash algorithm; also called a hash "function" is a computer algorithm designed to provide a random mapping from an arbitrary block of data (string of binary data) and return a fixed-size bit string known as a “message digest” and achieve certain security. Cryptographic hashing functions are used to create digital signatures, message authentication codes (MACs), other forms of authentication and many other security applications in the information infrastructure. They are also used to store user passwords in databases instead of storing the password in clear text and help prevent data leakage in session management for web applications. The actual algorithm used to create a cryptology function varies per implementation (SHA-256, SHA-512, etc.)

The code reviewer needs to be aware of three main things when reviewing code that uses cryptographic hashing functions.

• Legality of the cryptographic hashing functions if the source code is being exported to another country.

• The life cycle of the cryptographic hashing function being used.

• Basic programming of cryptographic hashing functions.

Legal

In the United States in 2000, the department of Commerce Bureau of Export revised encryption export regulations. The results of the new export regulations it that the regulations have been greatly relaxed. However if the code is to be exported outside of the source country current export laws for the export and import counties should be reviewed for compliance.

Case in point is if the entire message is hashed instead of a digital signature of the of message the National Security Agency (NSA) considers this a quasi-encryption and State controls would apply.

It is always a valid choice to seek legal advice within the organization that the code review is being done to ensure legal compliance.

Lifecycle

With security nothing is secure forever. This is especially true with cryptographic hashing functions. Some hashing algorithms such as Windows LanMan hashes are considered completely broken. Others like MD5, while currently considered safe for password hash usage, have known issues like collision attacks (note that collision attacks do not affect password hashes). The code reviewer needs to understand the weaknesses of obsolete hashing functions as well as the current best practices for the choice of cryptographic algorithms.

Programming/Vulnerabilities

The most common programmatic issue with hashing is not using a salt value or if using a salt the salt value is too short and or the same salt value is used in multiple hashes. The purpose of a salt is to make it harder for an attacker to perform pre-computed hashing attack (e.g., using rainbow tables) but other benefits of a salt can include making it difficult for an attacker to perform even password guessing attacks by obsfucating the hashed value.

Salt

One way to generate a secure salt value is using a pseudo-random number generator. Note that a salt value does not need to possess the quality of a cryptographically secure randomness. Best practices is to use a cryptographically function to create the salt, salt value should be created for each hash value, and a minimum value of 128 bits. The bits are not costly so don't save a few bits thinking you gain something back in performance instead use a value of 256-bit salt value. It is highly recommended.

- .Net Salt

```
  private int minSaltSize = 8;
  private int maxSaltSize = 24;
  private int saltSize;

  private byte[] GetSalt(string input) {
        byte[] data;
        byte[] saltBytes;
        RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider();
        saltBytes = new byte[saltSize];
        rng.GetNonZeroBytes(saltBytes);
        data = Encoding.UTF8.GetBytes(input);
        byte[] dataWithSaltBytes =
           new byte[data.Length + saltBytes.Length];
        for (int i = 0; i < data.Length; i++)
           dataWithSaltBytes[i] = data[i];
        for (int i = 0; i < saltBytes.Length; i++)
           dataWithSaltBytes[data.Length + i] = saltBytes[i];
        return dataWithSaltBytes;
     }
```

This method uses an agile approach to calling a hash function. It is explained below.

```
private string computeHashWithSalt(HashAlgorithm myHash, string input) {
        byte[] data;
        data = myHash.ComputeHash(GetSalt(input));
        sb = new StringBuilder();
        for (int i = 0; i < data.Length; i++) {
           sb.Append(data[i].ToString("x2"));
        }
```

```
        return sb.ToString();
    }
```

Microsoft .Net Notes on Hashing

Microsoft does not recommend using MD5 or SHA-1. With .Net 3.5 and above Microsoft supports the Suite B set of cryptographic algorithms published by the National Security Agency (NSA).

Java – java.security.SecureRandom

PHP - ???

Ruby - ???

Perl - ???

C++ none managed code on CLR or none windows ????

Javascript ?????

The salt value does not need to be secret and can be stored along with the hash value. Some may use a combination of account details (username, user full name, ID, creation date, etc.) as the salt for hash to further obsfucate the hash computation: for example salt = (username|lastname|firstname|ID|generated_salt_value).

Best Practices

Industry leading Cryptographer's are advising that MD5 and SHA-1 should not be used for any applications. The United State FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION (FIPS) specifies seven cryptographic hash algorithms — SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, and SHA-512/256 are approved for federal use. The code reviewer should consider this standard because the FIPS is also widely adopted by the information technology industry.

The code reviewer should raise a red flag if MD5 and SHA-1 are used and a risk assessment be done to understand why these functions would be used instead of other better-suited hash functions. FIPS does allow that MD5 can be used only when used as part of an approved key transport scheme (e.g. SSL v3.1) where no security is provided by the algorithm.

FIPS disapproves the following functions DES; MD51; RC4; Blowfish; Diffie-Hellman2; Diffie-Hellman3 (key agreement); EC Diffie-Hellman2 (key agreement); AES4 (non-compliant); Diffie-Hellman5 (key agreement); EC Diffie-Hellman4 (vendor affirmed); RSA4 (key agreement); RSA2 (key wrapping).

.Net Agile Code example for hashing

App Code File: <add key="HashMethod" value="SHA512"/>

C# Code:

```
1:  preferredHash =
HashAlgorithm.Create((string)ConfigurationManager.AppSettings["HashMethod"]);
 2:
 3:  hash = computeHash(preferredHash, testString);
 4:
 5:  private string computeHash(HashAlgorithm myHash, string input) {
 6:       byte[] data;
 7:       data = myHash.ComputeHash(Encoding.UTF8.GetBytes(input));
 8:       sb = new StringBuilder();
 9:       for (int i = 0; i < data.Length; i++) {
10:            sb.Append(data[i].ToString("x2"));
11:       }
12:       return sb.ToString();
13:  }
```

Line 1 let's us get our hashing algorithm we are going to use from the config file. If we use the machine config file our implementation would be server wide instead of application specific. Line 3 allows us to use the config value and set it according as our choice of hashing function. ComputHash can be SHA-256 or SHA-512.

The drawback to this method is key size. I would suggest of giving yourself twice the size of the largest key of hashing algorithm you could possible use to store hash values. This means we need a varchar of 1024 if we are going to store our hash value in the database.

Afterword

Lastly, never accept in a code review an algorithm created by the programmer for hashing or copy a hashing function taken from the Internet. Always use cryptographic functions that are provided by the language framework the code is written in. These functions are well vetted and well tested by experience cryptographers.

References:

http://valerieaurora.org/hash.html (Lifetimes of cryptographic hash functions)

http://docs.oracle.com/javase/6/docs/api/java/security/SecureRandom.html

http://msdn.microsoft.com/en-us/library/system.security.cryptography.rngcryptoserviceprovider.aspx

http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf

Ferguson and Schneier (2003) Practical Cryptography (see Chapter 6; section 6.2 Real Hash Functions)

4 Reviewing by Vulnerability

We need to add content here to introduce the section.

4.1 Review Code for XSS

Where can XSS occur?

- HTML Body Context

UNTRUSTED DATA

===**HTML Attribute Context**===

<input type="text" name="fname" value="UNTRUSTED DATA">

attack: "><script>/* bad stuff */</script>

===**HTTP GET Parameter Context**===

clickme

attack: " onclick="/* bad stuff */"

===**URL Context**===

clickme <iframe src="UNTRUSTED URL" />

attack: javascript:/* BAD STUFF */

===**CSS Value Context**===

Selection

attack: expression(/* BAD STUFF */)

===**JavaScript Variable Context**===

<script>var currentValue='UNTRUSTED DATA';</script>

<script>someFunction('UNTRUSTED DATA');

</script> attack: ');/* BAD STUFF */

===**JSON Parsing Context**===

JSON.parse(UNTRUSTED JSON DATA)

4.2 Persistent - The Anti Pattern (Needs Content)

Lorem Ipsum

4.2.1 .NET

.NET Anti-Pattern: Mishandled Concurrency

The correct concurrency management techniques is absolutely necessary in order to guarantee data integrity. A way to implement proper concurrency consists in creating a concurrency token which will be checked from the moment the entity object in the database was read until the moment when the submission will be executed. Prior to commit the final changes, the application must execute control where the concurrency token will be compared. If the token differs, conclusions can be drawn that another user has changed indeed the data.

The Entity Framework supports optimistic concurrency, unfortunately exceptions derived from errors encountered between the updates is not automatically handled, neither this will protect your data from corrupting.

Fetching Strategy issues

Main purpose of the ORM (Object Relational Mapper) is to create an abstraction, a representation of the database easier to manipulate and access data contained in it than retrieving it using SQL statements. Unfortunately, many developers assume that because the ORM is the middle layer handling the communication and manipulation of records, the data persistence will be handled without issues. The developer should analyze and observe what occurs at the database level once the ORM in question (such as EntityFramework, NHibernate or Ideablade for example) has been implemented. Concurrency issues can arise from incorrect fetching strategy implementations such as for example using 'CachedOnly' or 'DataSourceThenCached'(IdeaBlade). It is highly recommended that the developer revises the proper application depending on the implementation scenario, such as multiple users accessing the same data at the same time in order to avoid data corruption or when data is cached in an action by a certain Entity but the another one is not aware of this change.

Avoid the DTO pattern

Known to many NHibernate developers, the use of DTO pattern seems as something that sounds good but could be highly inconvenient. Main reasons for not implementing are: Contrary to their name, DTO's are no objects but a state which is defined through their behavior. By replicating multiple objects, a developer might end up having to implement multiple changes in the domain model and the DTO classes when for example, introducing a new property in the domain model class.

Avoid Locks

Another anti-pattern approach used by many developers is to lock regions in the database.Most of the time , this is implemented as way to avoid concurrency, however web applications are not properly suited for using locking which will indeed freeze the application. Locking data for the time the request takes place will not solve this problem. Using locks in database are absolutely not recommend since they required careful implementation planning and design.(Freeman, pg 179 ,2011)

Race conditions

If the following is run on more than one thread, it will randomly crash. It is not possible to know deterministically whether the code will throw an ArgumentOutOfRangeException. Sometimes it will, sometimes it won't.(Mclean, 2010)

```
IList<string> list = new List<string>();
list.Add("Hello");
```

```
…
// multi-threaded code
if(list.Count > 0)
{
   list.RemoveAt(0);
}
```

In that case a locking can be used, however using locks as mentioned earlier should be consider as an option if it is absolutely necessary.

Example locking code

```
object lockObj = new object();
IList<string> list = new List<string>();
list.Add("Hello");
…
 // multi-threaded code
 lock(lockObj)
 {
   if(list.Count > 0)
   {
       list.RemoveAt(0);
   }
  }
```

Recommendations

• The best option in this case is to alert the user who initiated the second request that his changes cannot be applied. "This is largely because, by definition, the first request will already have

completed".(Freeman,pg 179,2011)

• A recommended pattern when using the Entity Framework consists in "making a copy of the entity on the client and send back both the original version unmodified and the modified version or to write the client in such a way that it does not modify the concurrency token".(Simmons, 2009)

• Keep in mind that Detached objects (such as in the case of NHibernate or IdeaBlade ORM's)may no longer be guaranteed to be synchronized with database state; they're no longer under the management of NHibernate.However they could still contain persistent data. These ORM's allow you to reuse these instance by associating them with a new Persistence manager.

• Avoid the use DTO pattern by properly handling detached objects

References

Simmons, D. (2009, June). Anti-Patterns To Avoid In N-Tier Applications. MSDN Magazine. Retrieved from http://msdn.microsoft.com/en-us/magazine/dd882522.aspx#id0420025

Freeman, A (2011). Applied ASP .NET 4 in Context. Apress, New York, USA

McLean, G. (2010).Pro WPF and Silverlight MVVM: Effective Application Development with Model-View-ViewModel. Apress, New York, USA

4.2.2 Java

Java Persistence anti-patterns

Spring –Hibernate Anti-patterns

Some of the following anti-patterns are an important concern in the security area of Java applications. A related problem with these anti-patterns is data integrity.

- Lazy loading

This feature reduces the handling of data in an asynchronous way, which avoids unnecessary requests to the database, however it can causes problems with persistence. Errors associated with Lazy loading are:

org.hibernate.StaleObjectStateException: Row was updated or deleted by another transaction (or unsaved-value mapping was incorrect)

- N+1 Select issue

This problem occurs when the collection is returned from the database, containing n+1 separate queries instead of a single join query. This issue is quite challenging to solve because it depends on the specific implementation of the code, therefore look for the following executions:

• Control that mapping configurations are updated for affected domain classes

• Add the @ManyToMany @Fetch(FetchMode.JOIN) as a query strategy to override the Lazy behavior if necessary

• Review Tuning fetching strategies from Hibernate reference (http://docs.jboss.org/hibernate/core/3.3/reference/en/html/performance.html#performance-fetching-custom)

- Sessions per Operation anti-pattern

This anti pattern is caused by the opening and closing of individual sessions for each call executed to the database. In order to avoid this issue, make sure the calls are planned in sequence. Control proper implementation of persistence context. Problem occurs when DAO uses different persistence context for each one, in other words, a different Session or EntityManager.

Example of anti-pattern : Using the following code for each operation

session = sessionFactory.openSession();

session.close()

- Open Conversation anti-pattern

Keeping a database session alive while a user 'a' is editing data meanwhile, a lock is exerted to avoid concurrency issues can cause serious performance and bottlenecks in the application. It is not

advisable to do this in order to keep data integrity. There are other Hibernate features that allows the developer to handle sessions in a much efficient way such as automatic optimistic concurrency

Long Term Persistence Security Issues

Long-term persistence is a model that enables beans to be saved in XML format.(Java Tutorial, 2013). For this purpose, a programmer can use XMLEncoder class to pass through output files for textual representation of Serializable objects.In the example provided in the Java Tutorial, the programmer can invoke and create an instance of javax.swing.JButton such as this

```
<object class="javax.swing.JButton">
  <void method="setText">
    <string>Cancel</string>
      </void>
</object>
```

The vulnerability occurs when instead of passing acquitted XML code, the attacker sends dangerous Payloads. This vulnerability was shown by Dinis Cruz, Alvaro Muñoz and Abraham Kang in DefCon Conference 2013 “Resting on Your Laurels will get you Pwned: Effectively Code Reviewing REST Applications to avoid getting powned”

As explained by Dinis Cruz (2013) in his blog "there are two key scenarios where this ‘feature’ becomes a spectacular vulnerability:

• Server-side backend system that process attacker-controlled XML files using XMLDecoder

• REST APIs that uses XMLDecoder to create strongly type objects from the HTTP Request data

And the 2nd case is exactly what happens with Restlet REST API , which wraps XMLDecode in its org.restlet.representation.ObjectRepresentation<T> feature/class."(Cruz, 2013)

An Example of an attack which can be fond in Github from many examples created by Cruz , creates an item

```
<?xml version="1.0" encoding="UTF-8"?>
<java>
<object class="firstResource.Item">
<string>a aName</string>
<string>a Description</string>
```

</object>

</java>

Recommendations

It's clear that proper understanding of certain features and Java methods is essential to avoid certain vulnerabilities associated with the use of Persistence in frameworks, ORM'S and specific Java classes

References

NHibernate, 2013 "Transactions and concurrency control" available at http://docs.jboss.org/hibernate/orm/4.1/devguide/en-US/html/ch02.html#session-per-operation accessed on 4rd October 2013

Oracle, 2013 "Java Tutorials" available at http://docs.oracle.com/javase/tutorial/javabeans/advanced/longpersistence.html Accessed on 3rd October 2013

Dinis Cruz, 2013 "Using XMLDecoder to execute server-side Java Code on an Restlet application (i.e. Remote Command Execution)" available at

http://blog.diniscruz.com/2013/08/using-xmldecoder-to-execute-server-side.html Accessed on 3rd October 2013

https://github.com/o2platform/DefCon_RESTing/blob/master/Demos/_O2_Scripts/XmlEncoder%20-%20Restlet/exploits/1%20-%20create%20item%20%28Simple%29.xml

4.2.3 PHP

It is pretty easy to remove all persistent XSS attacks from PHP, just remove all instances of output functions (such as echo and print) with their safe counterparts from OWASP PHP Security Core Library, and then whenever you need HTML elements to be outputted, used the appropriate functions or PHP tags. There's a scanner in PHP Security Project that scans for this and can replace it effectively as well.

4.2.4 Ruby (Needs Content)

Lorem Ipsum

4.3 Reflected - The Anti Pattern (Needs Content)

Lorem Ipsum

4.3.1 .NET

The classes in the System.Reflection namespace and with System.Type, enable us to obtain information about loaded assemblies and the types defined within them, such as classes, interfaces, and value types. Reflection can be used to create type instances at run time, and to invoke and access them.

Depending on the .Net framework version, the use of Reflection could represent a major security risk.

The risk with using Reflection is that the original application code could be replaced by malicious code. The use of .NET framework 4 and on offers a higher level of protection since only trusted code can use reflection to access security-critical members. Furthermore, only trusted code can use reflection to access nonpublic members that would not be directly accessible to compiled code. Finally, code that uses reflection to access a safe-critical member must have whatever permissions the safe-critical member demands, just as with compiled code.

For other .NET framework versions beneath version 4, there are important recommendations to follow.

Recommendations

- Avoid writing public APIs that take MethodInfo parameters

Avoid it especially for highly trusted code. Such APIs might be more vulnerable to malicious code. For example, consider a public API in highly trusted code that takes a MethodInfo parameter. Assume that the public API indirectly calls the Invoke method on the supplied parameter. If the public API does not perform the necessary permission checks, the call to the Invoke method will always succeed, because the security system determines that the caller is highly trusted. Even if malicious code does not have the permission to directly invoke the method, it can still do so indirectly by calling the public API.

- Using ReflectionPermission class

To discover information about nonpublic members, callers must have the ReflectionPermission that represents the ability to obtain type information. Without this permission, code cannot use reflection to

obtain information about nonpublic members (even of its own class) through the Get methods on Type, Assembly, and Module.

To use reflection to invoke methods or access fields that are inaccessible according to the common type system accessibility rules, the code must be granted the ReflectionPermission for member access.

- Security policy deny ReflectionPermission to code that originates from the Internet

Because ReflectionPermission can provide access to non-public types and members,it is recommend that you do not grant ReflectionPermission to Internet code, except with the Reflection Permission Flag.RestrictedMemberAccess flag. RestrictedMemberAccess allows access to non-public members, with the restriction that the grant set of the non-public members must be equal to, or a subset of, the grant set of the code that accesses the non-public members.

4.3.2 Java

Reflection Security Issues

Java reflection is a mechanism used by Java programs given them the ability to change the runtime actions of the application running within the Java Virtual Machine (JVM). It makes it easier for developers to write programs because it helps gather information to implement proper analysis by the software itself (Schildt, 2011), however it compromises the systems because malware can easily bypass the security around the JVM.

Two security vulnerabilities found regarding the use of Java Reflection are CVE-2012-4681 and CVE-2012-5076. Both of them are related to Java Applets and another common factor is the use of Java reflection.

What to look in the code

In order to avoid this security issues, make sure that

• Java Runtime Environment (JRE) is higher that Java SE 7 Update 6 version

• Correct implementation of classes such as com.sun.beans.finder.ClassFinder.findClass

• Avoid Private structure using AccessibleObject.setAccessible because it breaks the encapsulation

• Avoid Use of sun.misc.Unsafe because it provides direct access to memory

• Verify correct Implementation of java.lang.reflect.ReflectPermission following best practices as described in Oracle Documents, September 2011

References

Schildt Hebert, 2011 ‘Java: The complete Reference, 8th Edition ‘ McGraw-Hill

Common Vulnerabilities and Exposure, 2012 ‘CVE-2012-4681’, available at (http://cve.mitre.org/cgi-bin/cvename.cgi?name=2012-4681) last viewed on October 3rd, 2013

Oracle Documents, 2011 “Permissions in the Java 2 SDK’ available at http://docs.oracle.com/javase/1.4.2/docs/guide/security/permissions.html#ReflectPermission, last viewed on October 3rd, 2013

4.3.3 PHP

To mitigate reflected XSS attacks fully, a PHP code should never output variables using echo, print and other output generating functions. If the output needs to be complex (for example a HTML list of variables) the HTML part should be outside PHP tags, and the rest should be inside and using safe output functions (available in OWASP PHP Security Project Core Library). For example:

```
 foreach ($list as $item)
  {
  ?>
  <li><?php phpsec\exho($item);?></li>
  <?php
  }
```

Or

```
  foreach ($list as $item)
  {
     phpsec\printf("<li>%s</li>\n",$item);
  }
```

4.3.4 Ruby (Needs Content)

Lorem Ipsum

4.4 Stored - The Anti Pattern (Needs Content)

Lorem Ipsum

4.4.1 .NET (Needs Content)

Lorem Ipsum

4.4.2 Java

Bad Session Stores

As described in the research paper written by V.Benjamin Livshits(2005), Bad session stores occurs when objects stored in attributes of javax.servlet.http.HttpSession are not subclasses of java.io.Serializable.

As further described by Livshits, it causes issues because HttpSessions objects could be written out to disk especially when all objects stored are handled as attributes that must be serialized, if not done properly this will cause exceptions or data corruption.

What to look for in the code

Parameters of HttpSession.set Attribute

Control if javax.servlet.httpSession is a subclass of java.io.Serializable

References

V. Benjamin Livshits, "Findings Security Errors in Java Applications Using Lightweight Static Analysis" 2005 available at (http://research.microsoft.com/en-us/um/people/livshits/papers/pdf/acsac04v.pdf) Last Viewed October 3rd 2013

4.4.3 PHP (Needs Content)

Lorem Ipsum

4.4.4 Ruby (Needs Content)

Lorem Ipsum

4.5 DOM XSS

CRV2 DOMXSS

XSS stands for Cross-site scripting. XSS is a common vulnerability found in web sites. Cross-site scripting accounts for 60 to 80% of all security vulnerabilities. This section od the Code Review Guide is focused DOM-based XSS vulnerabilities instead of the more traditionally cross-site scripting. The difference between the two is that traditional XSS occur in server-side code response for preparing HTML response to be served client-sided and DOM-XSS vulnerabilities occur in the content pessing on the client side typically done with client-side JavaScript.

DOM-XSS is mitigation is hampered by a lack of standardization of browsers and a large attack surface.

Reducing the threat:

The most common way to reduce is with encoding/escaping of string input. The code reviewer talk with development staff to see if any frameworks were used to help eliminated common XSS vulnerabilities.

• OWASP ESAPI (Java)

• ValidateRequest (ASP.NET)

• Anti-XSS library (ASP.NET)

• AntiSamy (Java)

• strip_tags, sanitize (Ruby on Rails)

• Django template escaping (Python Django)

• Coverity Security Library (Java)

• xss validator (Node.js)

• HTML Purifier

• Google Gaja

As with standard XSS the code reviewer should always pay attention to the following bullet points….

• How the programmer is validating input data. Not validating data is the root of all evil.

• Making sure data is escaped when the script writes out the page.

• Code Reviewer should review methods that make it hard to escape data and require an understanding of each browsers JavaScript engine. The code reviewer should red flag code that uses the following. These methods can be used securely but are prone to vulnerabilities. Best practice is not to use them.

• Element's .innerHTML() and .outerHTML() methods.

• Using user data in jQuery's element creation.

• jQuery's append.

• Using user data in a string passed to eval, setTimeout, an object's event handler, or javascript: url targets.

• Using user data in strings that generate CSS.

Microsoft ASPX .Net

• On ASPX .Net pages code review should check to make sure web config file does not turn off page validation. <pages validateRequest="false" />

• .Net framework 4.0 does not allow page validation to be turned off. Hence if the programmer wants to turn of page validation the developer will need to regress back to 2.0 validation mode. <httpRuntime requestValidationMode="2.0" />

• Code reviewer needs to make sure page validation is never turned off on anywhere and if it is understand why and the risks it opens the organization to. <%@ Page Language="C#" ValidationRequest="false"

OWASP Resources:

OWASP DOM BASED XSS [1]

OWASP DOM BASED Cheat Sheet
[www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet]

4.6 JQuery Mistakes (Needs Content)

Lorem Ipsum

4.7 Reviewing code for SQL Injection (Needs Content)

Lorem Ipsum

4.7.1 PHP

Introduction

An SQL injection Attack consists of injecting sql query portions in the back-end database system via the client interface in the web application. The consequence of a successful exploitation of an SQL injection varies from just reading data to modifying data or executing system commands. SQL Injection in PHP remains the number one attack vector, and also the number on reason for DATA COMPROMISES

Data Validation and prepared statements

It is as simple as this the absence of data validation and prepared statements or stored procedures will increase the possibility that your code contain SQL injections. If your application gives the users the possibility to change parameters and those parameters are not verified and inserted in an unprepared statement than your code contain an SQL Injection.

Example 1 :

```
<?php
$pass=$_GET["pass"];
$con = mysql_connect('localhost', 'owasp', 'abc123');
mysql_select_db("owasp_php", $con);
$sql="SELECT card FROM users WHERE password = '".$pass."'";
$result = mysql_query($sql);
?>
```

Suspicious Validation

The most common ways to prevent SQL Injection in PHP are using functions such as addslashes() and mysql_real_escape_string() but those function can always cause SQL Injections in some cases.

addslashes :

addslashes() will only work if the query string is wrapped in quotes.A string such as the following would still be vulnerable to an SQL injection

```
<?php

$id = addslashes( $_GET['id'] );
$query = 'SELECT title FROM books WHERE id = ' . $id;

?>
```

mysql_real_escape_string():

mysql_real_escape_string() is a little bit more powerful than addslashes() as it calls MySQL's library function mysql_real_escape_string, which prepends backslashes to the following characters: \x00, \n, \r, \, ', " and \x1a. As with addslashes(), mysql_real_escape_string() will only work if the query string is wrapped in quotes. A string such as the following would still be vulnerable to an SQL injection:

```
<?php
```

```
$bid = mysql_real_escape_string( $_GET['id'] );
$query = 'SELECT title FROM books WHERE id = ' . $bid;

?>
```

Canonicalization

Canonicalization is the process by which various equivalent forms of a name can be resolved to a single standard name, or the "canonical" name.

The most popular encodings are UTF-8, UTF-16, and so on (which are described in detail in RFC 2279). A single character, such as a period/full-stop (.), may be represented in many different ways: ASCII 2E, Unicode C0 AE, and many others.

With the myriad ways of encoding user input, a web application's filters can be easily circumvented if they're not carefully built.

Bad Example

```
public static void main(String[] args) {
    File x = new File("/cmd/" + args[1]);
    String absPath = x.getAbsolutePath();
}
```

Good Example

```
public static void main(String[] args) throws IOException {
    File x = new File("/cmd/" + args[1]);
    String canonicalPath = x.getCanonicalPath();
}
```

References

See Reviewing code for Data Validation (in this guide) Reviewing code for Data Validation

See the OWASP ESAPI Project

The OWASP ESAPI project provides a reference implementation of a security API which can assist in providing security controls to an application.

4.7.2 Java

Java SQL Injections

SQL injections occur when input to a web application is not controlled or sanitized before executing to the back-end database The attacker tries to exploit this vulnerability by passing SQL commands in her/his input and therefore will create a undesired response from the database such as providing information that bypasses the authorization and authentication programmed in the web application

An example of a vulnerable java code (Livshits and Lam, 2005)

```
HttpServletRequest request = ...;
String userName = request.getParameter("name");
Connection con  = ...
String query    = "SELECT * FROM Users " + " WHERE name = '" + userName + "'";
con.execute(query);
```

The input parameter “name” is passed to the String query without any proper validation or verification. The query ‘SELECT* FROM users where name" is equal to the string ‘username’ can be easily misused to bypass something different that just the ‘name’. For example, the attacker can attempt to pass instead

" OR 1=1.

In this way accessing all user records and not only the one entitled to the specific user

More of SQL Injection Vulnerabilities

See the OWASP article on SQL Injection Vulnerabilities. http://www.owasp.org/index.php/SQL_Injection See the OWASP article on Blind_SQL_Injection Vulnerabilities. http://www.owasp.org/index.php/Blind_SQL_Injection

How to Avoid SQL Injection Vulnerabilities

See the OWASP Development Guide article on how to Avoid SQL Injection Vulnerabilities. http://www.owasp.org/index.php/Guide_to_SQL_Injection

How to Test for SQL Injection Vulnerabilities

See the OWASP Testing Guide article on how to Test for SQL Injection Vulnerabilities. http://www.owasp.org/index.php/Testing_for_SQL_Injection

References

Livshits and Lam, 2005 "Finding Security Vulnerabilities in Java Applications with Static Analysis" available at https://www.usenix.org/legacy/event/sec05/tech/full_papers/livshits/livshits_html/#sec:sqlinjexample Accessed on 3rd October, 2013

4.7.3 .NET (Needs Content)

Lorem Ipsum

4.7.4 HQL (Needs Content)

Lorem Ipsum

4.8 The Anti Pattern

Introduction

In software engineering, a design pattern is a reusable solution to a common occurring problem that can be generalized to be used a numerous contexts of software design.

Anti-patterns are commonly used patterns used in software engineering but are ineffective, counterproductive, and may result in software vulnerabilities.

The Code Review Guide will focus on anti-design patterns that help create in-secure code/applications.

Information Disclosure Through Exception:

Error information for a hacker is like a bone to a dog; “Something good to chew on“. Without controlling what error information is shown to a user the application may release information such as platform target code is running on, database being used, computer language, etc. The significance of each piece of information allows the hacker to quickly narrow what tools he uses and what vulnerabilities he might try to exploit.

- .Net

What is the flaw?

} catch (exception e) {YourLogger.Log(e);}

This exception was not handled. The exception was log but the program is allowed to continue executing. This is contrary to good secure design. When exceptions occur the code needs to handled the exception not just log it. Programs need to fail and fail fast and securely. If the program is not allow to fail because of business rules it needs to know how to handled it's state to be able to recover in a secure manner.

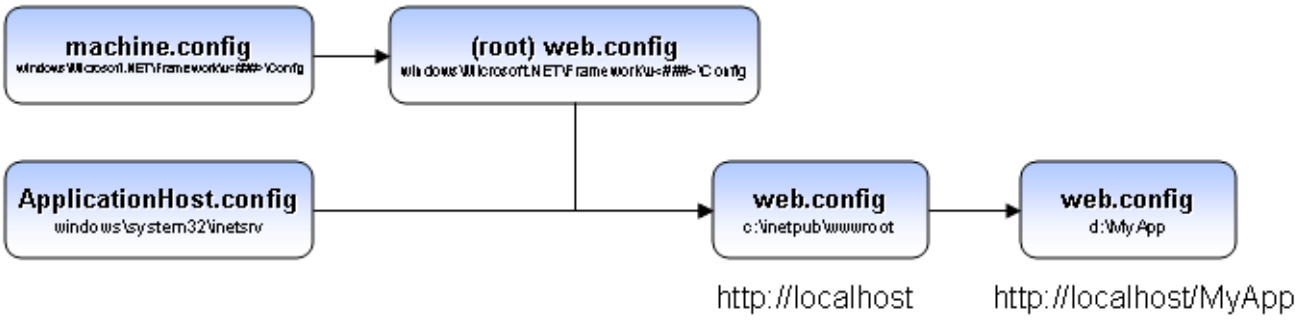
} catch (exception e) { YourLogger.Log(e); throw e; }

is incorrect.

} catch (exception e) { YourLogger.Log(e); throw new Exception("Your exception description");

is incorrect.

} catch (exception e) { YourLogger.Log(e); throw new Exception("Your exception description ",e);



Throw new Exception could be valid if you want to hide the original error, perhaps a security related issue.

Secure Design Recommendation:

At minimum exception handling should have...

} catch (Exception ex) {YourLogger.Log(ex); throw;}

Review Criteria

Static analysis tools like Cat.Net or FxCop are good at finding information leakage from exceptions. Code reviewer needs to understand how exceptions and unhanded exceptions are handled by the program.

Need link here on what information is valid to log

4.8.1 PHP (Needs Content)

Lorem Ipsum

4.8.2 Java (Needs Content)

Lorem Ipsum

4.8.3 .NET (Needs Content)

Lorem Ipsum

4.8.4 Ruby (Needs Content)

Lorem Ipsum

4.8.5 Cold Fusion

Lorem Ipsum

4.9 Reviewing code for CSRF Issues

Having CSRF-proof forms and actions is a complex task, and very prone to human-error. The most effective means of mitigating it is incorporating it into a widget library, for example OWASP PHP Security Widget library, which automatically uses CSRF protection.

CSRF Protection for GET and COOKIE elements is hard and not recommended, therefore all operations that change the state of the application in someway should be implemented using HTTP Post (or other HTTP state changing requests).

Generally, CSRF protection is achieved by generating cryptographically secure, required parameters into HTML forms, and checking them back when they are submitted. If they are submitted and valid, they should get expired.

4.10 Transactional logic / Non idempotent functions / State Changing Functions (Needs Content)

Lorem Ipsum

4.11 Reviewing code for poor logic /Business logic/Complex authorization (Needs Content)

Lorem Ipsum

4.12 Reviewing Secure Communications (Needs Content)

Lorem Ipsum

4.12.1 .NET Config

ASP.NET Configurations

Sensitive data passes across networks. This data might include passwords, credit card numbers, social security numbers, you name it. To protect against disclosure of this information from unwelcome users, it is essential to protect the data in transit from unwanted alteration.

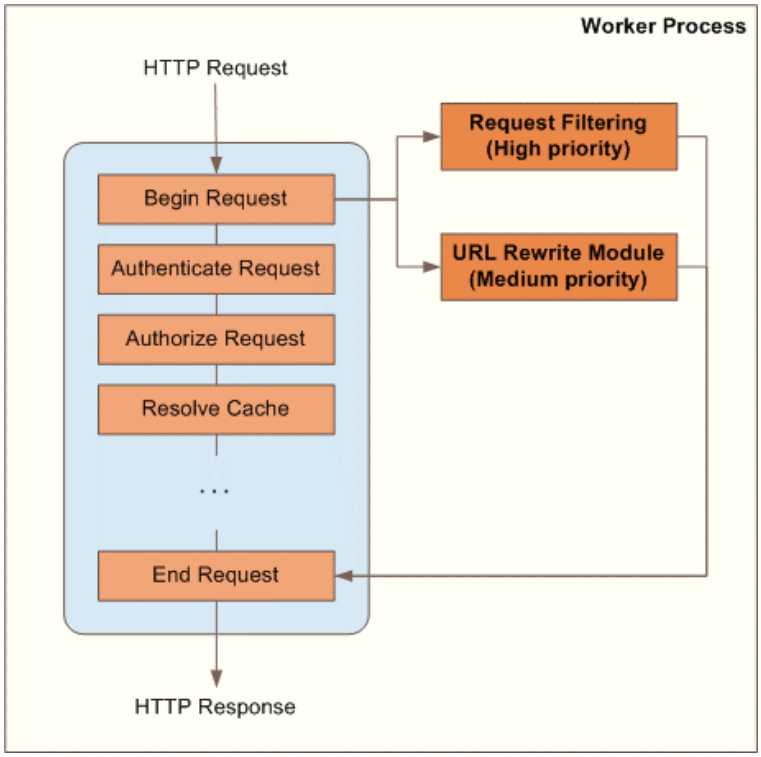
Since web requests through physical tiers of an application crosses different communication channels, it must be considered how the application must be secure for each of these channels. Most of the security configurations in ASP.NET occurs on different files found in the application such as the web.config file or in the IIS server such as the policy files. The following information highlights the most important aspects to secure communications in ASP.NET applications

IIS 7 Configurations

In IIS 7 there is a new configuration system which affects the hierarchy level and how one file can inherit from another. The following figure resumes how this work and the location of each file (Aguilar, 2006)

- Filtering Requests and URL Rewriting

Request Filtering was introduced in IIS7 and it has replaced the functionality UrlScan add-on for IIS 6.0. This built-in security feature allows to filter undesired URL request but it is also possible to configure different kinds of filtering. To begin with, it is important to understand how the IIS pipeline works when a request is done. The following diagram shows the order in these modules



(Yakushev, 2008)

Request filtering can be setup through the IIS interface or on the web.config file. Example:

```
<configuration>

  <system.webServer>
    <security>
      <requestFiltering>
        <denyUrlSequences>
          <add sequence=".." />
          <add sequence=":" />
        </denyUrlSequences>
        <fileExtensions allowUnlisted="false" />
        <requestLimits maxUrl="2048" maxQueryString="1024" />
        <verbs allowUnlisted="false" />
      </requestFiltering>
```

```
    </security>
  </system.webServer>

</configuration>
```

(Yakushev, 2008) This can also be done through the application code , for example:

```
using System;
using System.Text;
using Microsoft.Web.Administration;
internal static class Sample
{
   private static void Main()
   {
      using (ServerManager serverManager = new ServerManager())
      {
         Configuration config = serverManager.GetWebConfiguration("Default Web Site");
         ConfigurationSection requestFilteringSection = config.GetSection("system.webServer/security
         /requestFiltering");
        ConfigurationElementCollection denyUrlSequencesCollection =
        requestFilteringSection.GetCollection("denyUrlSequences");
        ConfigurationElement addElement = denyUrlSequencesCollection.CreateElement("add");
        addElement["sequence"] = @"..";
        denyUrlSequencesCollection.Add(addElement);
        ConfigurationElement addElement1 = denyUrlSequencesCollection.CreateElement("add");
        addElement1["sequence"] = @":";
        denyUrlSequencesCollection.Add(addElement1);
        ConfigurationElement addElement2 = denyUrlSequencesCollection.CreateElement("add");
```

```
        addElement2["sequence"] = @"\";
        denyUrlSequencesCollection.Add(addElement2);
        serverManager.CommitChanges();
      }
    }
}
```

(Yakushev, 2008)

- Filtering Double –Encoded Requests

This attack technique consists of encoding user request parameters twice in hexadecimal format in order to bypass security controls or cause unexpected behavior from the application. It's possible because the webserver accepts and processes client requests in many encoded forms.

By using double encoding it’s possible to bypass security filters that only decode user input once. The second decoding process is executed by the backend platform or modules that properly handle encoded data, but don't have the corresponding security checks in place.

Attackers can inject double encoding in pathnames or query strings to bypass the authentication schema and security filters in use by the web application.

There are some common characters sets that are used in Web applications attacks. For example, Path Traversal attacks use “../” (dot-dot-slash) , while XSS attacks use “<” and “>” characters. These characters give a hexadecimal representation that differs from normal data.

For example, “../” (dot-dot-slash) characters represent %2E%2E%2f in hexadecimal representation. When the % symbol is encoded again, its representation in hexadecimal code is %25. The result from the double encoding process ”../”(dot-dot-slash) would be %252E%252E%252F:

The hexadecimal encoding of “../” represents "%2E%2E%2f"

Then encoding the “%” represents "%25"

Double encoding of “../” represents "%252E%252E%252F"

If you do not want IIS to allow doubled-encoded requests to be served, use the following (IIS Team,2007) :

```
<configuration>
  <system.webServer>
   <security>
     <requestFiltering
      allowDoubleEscaping="false">
     </requestFiltering>
   </security>
  </system.webServer>
</configuration>
```

- Filter High Bit Characters

This allows or rejects all requests to IIS that contain non-ASCII characters . When this occurs error code 404.12. is displayed to the user . The UrlScan(IIS6 add-on) equivalent is AllowHighBitCharacters.

```
<configuration>
 <system.webServer>
  <security>
    <requestFiltering
    allowHighBitCharacters="true">
    </requestFiltering>
  </security>
 </system.webServer>
</configuration>
```

- Filter Based on File Extensions

Using this filter you can allow IIS to a request based on file extensions, the error code logged is 404.7. The AllowExtensions and DenyExtensions options are the UrlScan equivalents.

```
<configuration>
  <system.webServer>
   <security>
    <requestFiltering>
     <fileExtensions allowUnlisted="true" >
     <add fileExtension=".asp" allowed="false"/>
     </fileExtensions>
    </requestFiltering>
   </security>
  </system.webServer>
 </configuration>
```

- Filter Based on Request Limits

When IIS rejects a request based on request limits, the error code logged is: • 404.13 if the content is too long. • 404.14 if the URL is too large. • 404.15 if the query string is too long. This can be used to limit a long query string or too much content sent to an application which you cannot change the source code to fix the issue.

```
<configuration>
   <system.webServer>
     <security>
      <requestFiltering>
       <requestLimits
        maxAllowedContentLength="30000000"
        maxUrl="260"
```

```
        maxQueryString="25"
        />
      </requestFiltering>
     </security>
    </system.webServer>
  </configuration>
```

- Filter by Verbs

When IIS reject a request based on this feature, the error code logged is 404.6. This corresponds to the UseAllowVerbs, AllowVerbs, and DenyVerbs options in UrlScan. In case you want the application to use only certain type of verb, it is necessary to firt set the allowUnlisted to ‘false’ and then set the verbs that you would like to allow(see example)

```
  <configuration>
   <system.webServer>
    <security>
     <requestFiltering>
      <verbs allowUnlisted="false">
        <add verb="GET" allowed="true" />
      </verbs>
     </requestFiltering>
    </security>
   </system.webServer>
</configuration>
```

- Filter Based on URL Sequences

This feature defines a list of sequences that IIS reject when it is part of a request. When IIS reject a request for this feature, the error code logged is 404.5.This corresponds to the DenyUrlSequences feature in UrlScan. This is a very powerful feature. This avoids a given character sequence from ever being attended by IIS:

```
<configuration>
 <system.webServer>
  <security>
   <requestFiltering>
    <denyUrlSequences>
     <add sequence=".."/>
    </denyUrlSequences>
   </requestFiltering>
  </security>
 </system.webServer>
</configuration>
```

- Filter Out Hidden Segments

In case you want IIS to serve content in binary directory but not the bin, you can apply this configuration.

```
<configuration>
 <system.webServer>
  <security>
   <requestFiltering>
    <hiddenSegments>
     <add segment="BIN"/>
    </hiddenSegments>
   </requestFiltering>
  </security>
 </system.webServer>
</configuration>
```

Password protection and sensitive information

The web.config files might include sensitive information in the connection strings such as database passwords, mail server user names among others.

Sections that are required to be encrypted are:

<appSettings>. This section contains custom application settings.

<connectionStrings>. This section contains connection strings.

<identity>. This section can contain impersonation credentials.

<sessionState>. This section contains the connection string for the out-of-process session state provider.

Passwords and user names contained in a <connectionstring> section should be encrypted. ASP.NET allows you to encrypt this information by using the functionality aspnet_regiis. This utility is found in the installed .NET framework under the folder

%windows%\Microsoft.NET\Framework\v2.0.50727

You can specify the section you need to encrypt by using the command:

aspnet_regiis -pef sectiontobeencryoted .

Encrypting sections in Web.Config file

Even though encrypting sections is possible, not all sections can be encrypted, specifically, sections that are read before user code is run. The following sections cannot be encrypted:

*<processModel>

*<runtime>

*<mscorlib>

*<startup>

*<system.runtime.remoting>

*<configProtectedData>

*<satelliteassemblies>

*<cryptographySettings>

*<cryptoNameMapping>

*<cryptoClasses>

Machine-Level RSA key container or User-Level Key Containers

Encrypting a single file has its disadvantages when this file is moved to another servers. In this case, the user of an RSA key container is strongly advice. The RSAProtectedConfigurationProvider supports machine-level and user-level key containers for key storage.

RSA machine key containers are stored in the following folder:

\Documents and Settings\All Users\Application Data\Microsoft\Crypto\RSA\MachineKey

- User Key Container

When the application that needs to be protected is in a shared hosting environment and protection of sensitive data cannot be accessible to other applications, the user key container is strongly recommended. In this case each application should have a separate identity. RSA user-level key containers are stored in the following folder:

\Documents and Settings\{UserName}\Application Data\Microsoft\Crypto\RSA

IIS configurations

Depending on the version of IIS that must be configured, it is important to revise some of its settings, which can comprise security in the server.

- Trust level

The trust level is a set of Code Access Security permissions granted to an application within a hosting environment. These are defined using policy files. Depending on the trust level that must be configured, it is possible to grant FULL, HIGH, MEDIUM, LOW or MINIMAL level. The ASP.NET host does not apply any additional policy to applications that are running at the full-trust level.

Example:

```
<system.web>
  <securityPolicy>
    <trustLevel name="Full" policyFile="internal"/>
  </securityPolicy>
</system.web>
```

- Lock Trust Levels

In the .NET framework web.config file is possible to lock applications from changing their trust level This file is found at:

C:\Windows\Microsoft.NET\Framework\{version}\CONFIG

The following example shows how to lock 2 different application configuration trust levels (MSDN, 2013)

```
<configuration>
  <location path="application1" allowOverride="false">
    <system.web>
      <trust level="High" />
    </system.web>
```

```
  </location>
  <location path="application2" allowOverride="false">
    <system.web>
      <trust level="Medium" />
    </system.web>
  </location>
</configuration>
```

References

Yakushev Ruslan , 2008 "IIS 7.0 Request Filtering and URL Rewriting " available at http://www.iis.net/learn/extensions/url-rewrite-module/iis-request-filtering-and-url-rewriting (Last accessed on 14 July, 2013)

OWASP, 2009 "Double Encoding" available at https://www.owasp.org/index.php/Double_Encoding (Last accessed on 14 July, 2013)

IIS Team, 2007 "Use Request Filtering " available at http://www.iis.net/learn/manage/configuring-security/use-request-filtering (Last accessed on 14 July, 2013)

Aguilar Carlos ,2006 "The new Configuration System in IIS 7" available at http://blogs.msdn.com/b/carlosag/archive/2006/04/25/iis7configurationsystem.aspx (Last accessed on 14 July, 2013)

MSDN, 2013 . How to: Lock ASP.NET Configuration Settings available at http://msdn.microsoft.com/en-us/library/ms178693.aspx (Last accessed on 14 July, 2013)

4.12.2 Spring Config (Needs Content)

Lorem Ipsum

4.12.3 HTTP Headers (Needs Content)

Lorem Ipsum

4.13 Tech-Stack Pitfalls (Needs Content)

Lorem Ipsum

4.14 Framework Specific Issues (Needs Content)

Lorem Ipsum

4.14.1 Spring

Spring Mass assignment

The mass assignment problem relates to the universal web framework pattern of automatic binding request parameters into model objects. See also MVC/.NET section previously <link here>

Model objects are an object-oriented representation of user input. They provide methods to get, set etc associated parameters from user input. The following frameworks provide a mechanism for binding request parameters into request bound objects based on matching request parameter names to object attribute names (based on matching public getter and setter methods).

1. .NET MVC --> Controller method Parameters
2. Struts 1 --> ActionForms
3. Struts 2 --> Action Attributes
4. spring mvc --> Command Object
5. Ruby Rails & Grails --> bound request parameters to objects

- Anti-Pattern

Example Code

```
public class User{ <-- object to place request data
    public long id;
    public String fname;
```

```
        public String lname;
        public boolean isAdmin;
    }

 <form action”/updateUser” method=”post” > <-- intened use
        <input name=”user.fname” />
        <input name=”user.lname” />
    </form>
```

The attacker may post a request such as

```
 <form action”/updateUser” method=”post” >
        <input name=”user.fname” value="Eoin" />
        <input name=”user.lname” value = "OWASP" />
        <input name=”user.isAdmin” = "True" /><-- injection
    </form>
```

This shall update the user Object isAdmin boolean even though the developer did not intend to do so.

What to look for

When reviewing ORM code such as applications using the frameworks above it is important to veify they they are protected against mass binding attacks. It is suggested to look for the following...

1 Rails: In Ruby on Rails, attr_accessible allows you to specify which attributes of a model can be altered via mass-assignment (most notably by update_attributes(attrs) and new(attrs)). Any attribute names you pass as parameters will be alterable via mass-assignment, and all others won’t be.

Check config/application.rb for

config.active_record.whitelist_attributes = true

And also at the top of each model object

attr_accessible :fname, :lname

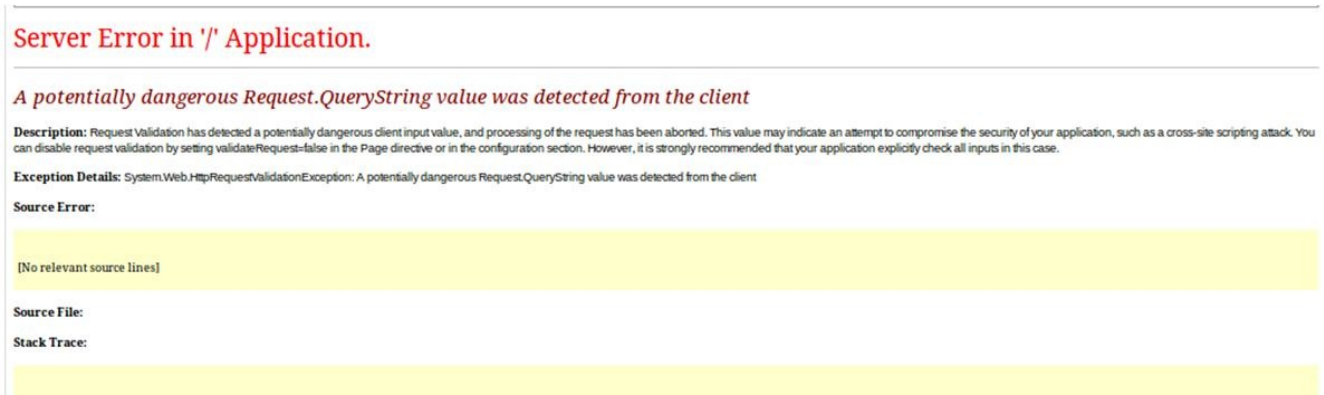
2 .NET MVC

Bind(include = "name") should be used to define which attributes can be updated.

```
[Bind(Include = "fname")]
  public class Enquiry
  {
     public string fname { get; set; }
     public boolean isAdmin{ get; set; }
  }
```

3 Grails: See http://blog.adamcreeger.com/2012/03/grails-rails-github-and-mass-assignment.html

4 Spring MVC: Use DataBinder.setAllowedFields()
http://static.springsource.org/spring/docs/current/javadoc-



api/org/springframework/validation/DataBinder.html

4.14.2 Structs (Needs Content)

Lorem Ipsum

4.14.3 Drupal (Needs Content)

Lorem Ipsum

4.14.4 Ruby on Rails (Needs Content)

Lorem Ipsum

4.14.5 Django (Needs Content)

Lorem Ipsum

4.14.6 .NET Security / MVC

ASP.NET Security in MVC

Binding issues in MVC .NET

- A.K.A Over-Posting A.K.A Mass assignments

In MVC framework, mass assignments are a mechanism that allows us to update our models with data coming in a request in HTTP form fields. As the data that needs to be updated comes in a collection of form fields, a user could send a request and modify other fields in the model that may not be in the form and the developer didn't intend to be updated.

Depending on the models you create, there might be sensitive data that you would not like to be modified. The vulnerability is exploited when a malicious user modifys a model's fields, which are not exposed to the user via the view, and the malicious user to change hidden model values adds additional model parameters.

public class user

```
{
    public int ID { get; set; }  <- exposed via view
    public string Name { get; set; } <- exposed via view
    public bool isAdmin{ get; set; } <-hidden from view
}
```

Corresponding view (HTML)

```
ID: <%= Html.TextBox("ID") %>

Name:  <%= Html.TextBox("Name")  %>

            <-- no isAdmin here!
```

The corresponding HTML for this model contain 2 fields: ID and Name. If an attacker adds the isAdmin parameter to the form and submits they can change the model object above. So a malicious attacker may change isAdmin=true

Recommendations:

-1 Use a model which does not have values the user should not edit.

-2 Use the bind method and whitelist attributes which can be updated.

-3 Use the controller.UpdateModel method to exclude certain attribute updates.

Anti-XSS

Traditional ASP.NET applications do not suffer from XSS attacks, contrary to MVC ASP.NET applications. When MVC web apps are exposed to malicious XSS code, they will not throw an error like the following one:

To avoid this vulnerability, make sure that use use the following code snippet:

<%server.HtmlEncode(stringValue)%>

The HTMLEncode method applies HTML encoding to a specified string. This is useful as a quick method of encoding form data and other client request data before using it in your Web application. Encoding data converts potentially unsafe characters to their HTML-encoded equivalent.(MSDN,2013)

- Razor Syntax MVC3

An option to for XSS protection is the use of Razor syntax for ASP.NET MVC3 applications use the following code for this purpose:

@message

- MVC4 anti XSS feature

in ASP.NET MVC4 It is possible to override the standard HTML encoder by using the XSS encoder For this, download the code, compile it and add the library as a reference to the application.

In the web.config, add the following line in the <system.web> section:

<httpRuntime encoderType=

"Microsoft.Security.Application.AntiXssEncoder, AntiXssLibrary"/>

- Sanitize object before saving to a database

The Anti XSS library contains a Sanitize object that can be called to clean the HTML before is stored in a database in case the web application is using a WYSIWYG editor

example:

```
using Microsoft.Security.Application;
...
...
string wysiwygData = "before <script>alert('bip ')</script> after ";
string cleanData = Sanitizer.GetSafeHtmlFragment(wysiwygData);
```

Protection against SQL injections

The best solution to avoid this OWASP #1 in the top ten list of security vulnerabilities is to use Parameterized queries .Equivalent to this solution, the use of Stored procedures is also a form of parameterized queries, however the way you implement them could still be prone to vulnerabilities

- Parameter collections

Parameter collections such as SqlParameterCollection provide type checking and length validation. If you use a parameters collection, input is treated as a literal value, and SQL Server does not treat it as executable code, and therefore the payload can not be injected. Using a parameters collection lets you enforce type and length checks. Values outside of the range trigger an exception. Make sure you handle the exception correctly. Example of the SqlParameterCollection:

```
using System.Data;
using System.Data.SqlClient;
using (SqlConnection conn = new SqlConnection(connectionString))
{
 DataSet dataObj = new DataSet();
 SqlDataAdapter sqlAdapter = new SqlDataAdapter( "StoredProc", conn);
 sqlAdapter.SelectCommand.CommandType = CommandType.StoredProcedure;
 //specify param type
 sqlAdapter.SelectCommand.Parameters.Add("@usrId", SqlDbType.VarChar, 15);
 sqlAdapter.SelectCommand.Parameters["@usrId "].Value = UID.Text; // Add data from user
 sqlAdapter.Fill(dataObj); // populate and execute proc
```

}

- Stored procedures don’t always protect against SQL injection

Even though Stored procedures are a form of Parametarized queries, they do not offer total protectioon agains SQL injections. The following code for example demonstrates this issue:

```
CREATE PROCEDURE dbo.RunAnyQuery
@parameter NVARCHAR(50)
AS
EXEC sp_executesql @parameter
GO
```

The above procedure shall execute any SQL you pass to it because is using unfiltered content from the end user. The directive sp_executesql is a system stored procedure in Microsoft® SQL Server™

Lets pass it.

```
DROP TABLE ORDERS;
```

Guess what happens? So we must be careful of not falling into the “We’re secure, we are using stored procedures” trap!

- Use an ORM(Object Relational Mapper)

ORM’s are a real blessing regarding protection against SQL injection. By default, the use of ORM will automatically send all SQL request as parameterized queries, however, it’s important to keep in mind that this form of security can be easily bypassed if the developer uses unparameterized HQL or Entity SQL queries dynamically with string concatenations

Request Validation feature against XSS attacks

The ASP .NET framework contains a validator framework, which has made input validation easier and less error prone than in the past. This feature was added in the ASP.NET version 1.1, in addition this feature is enabled by default. Once a malformed request containing any HTML tags in send, ASP.NET will simply display an error as shown in the following figure

The validation solution for .NET also has client and server side functionality akin to Struts (J2EE). What is a validator? According to the Microsoft (MSDN) definition it is as follows:

"A validator is a control that checks one input control for a specific type of error condition and displays a description of that problem."

The main point to take out of this from a code review perspective is that one validator does one type of function. If we need to do a number of different checks on our input we need to use more than one validator. The .NET solution contains a number of controls out of the box:

• RequiredFieldValidator – Makes the associated input control a required field.

• CompareValidator – Compares the value entered by the user into an input control with the value entered into another input control or a constant value.

• RangeValidator – Checks if the value of an input control is within a defined range of values.

• RegularExpressionValidator – Checks user input against a regular expression.

- Disadvantages

Unfortunately, this feature can also create issues when legitimate requests are sent by users who need to submit data containing certain kind of characters such as brackets.

Another disadvantage is that this does not avoid any attacks originated from other application or if stored in the database, neither will offer any protection when input is injected in HTML attributes.

- Use Microsft's Anti-XSS library

Unfortunately, HtmlEncode or validation feature is not enough to deal with XSS, especially if the user input needs to be added to JavaScript code, tag attributes, XML or URL. In this case a good option is the Anti-XSS library

MVC's CSFR anti-forgery system

This is one handy feature found in .NET which contra rest the #8 owasp top 10 security issue.

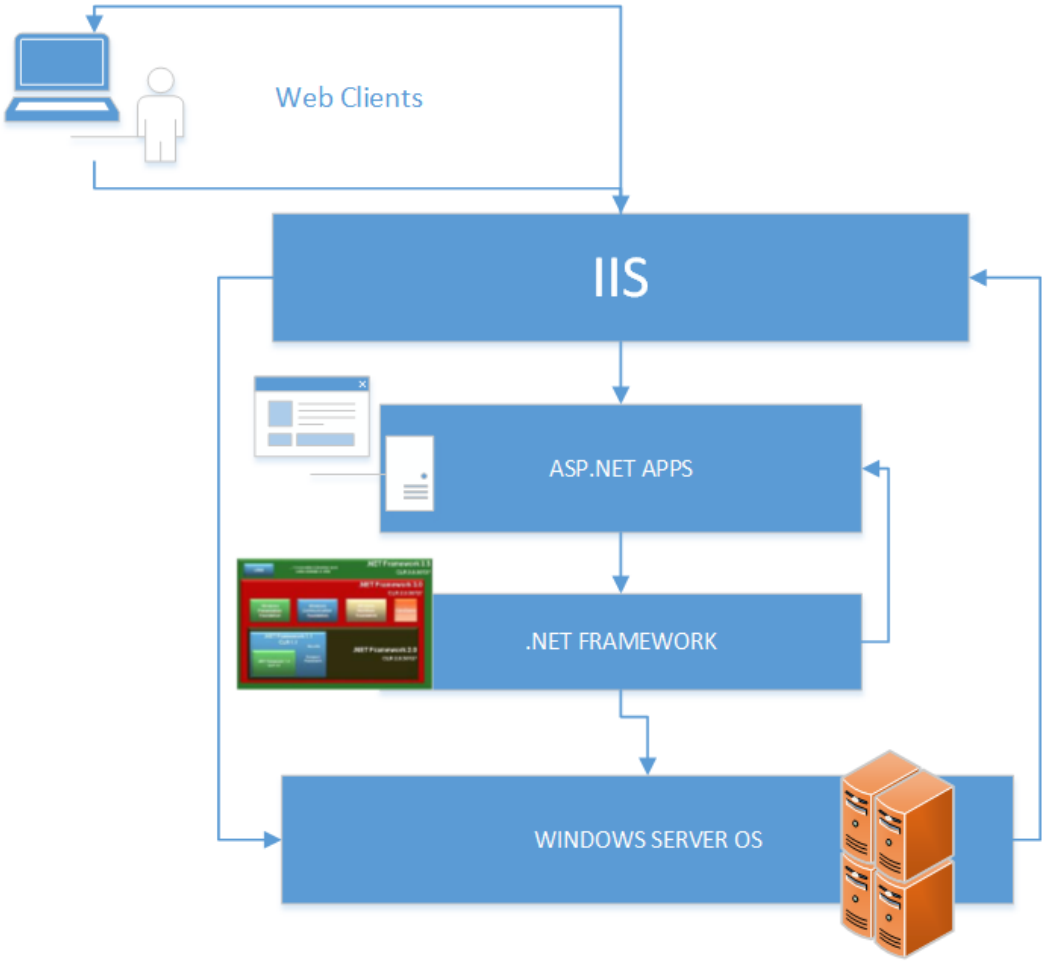
- Use Anti-forgery Helpers

There are 2 methods which a developer can use to avoid CSFR attacks, these are Html.AntiForgeryToken() and the filter [ValidateAntiForgeryToken]. To use these features, call the AntiForgeryToken method from within your form, and add the ValidateAntiForgeryTokenAttribute to the action method you want to protect. A combination between the Html.AntiForgeryToken() and

Ajax.ActionLink is a recommended way to go in order to make sure that no attacker can send a false deletion request

```
$.ajaxPrefilter(
    function (options, localOptions, jqXHR) {
        if (options.type !== "GET") {
            var token = GetAntiForgeryToken();
            if (token !== null) {
                if (options.data.indexOf("X-Requested-With") === -1) {
                    options.data = "X-Requested-With=XMLHttpRequest" + (options.data === "") ? "" : "&" +
options.data;
                }
```

```
        options.data = options.data + "&" + token.name + '=' + token.value;
    }
```




```
                }
            }
        );
```

- Limitations

• Users must accept cookies otherwise the

[ValidateAntiForgeryToken] will deny their form's posts

• Works only with POST request

• Can be bypassed if the application has XSS vulnerabilities since it will be possible to read _RequestVerificationToken value

References

MSDN, 2013 - Server.HTMLEncode Method available at http://msdn.microsoft.com/en-us/library/ms525347%28v=vs.90%29.aspx (last viewed: 2nd July, 2013)

MSDN, 2013 HtmlHelper.AntiForgeryToken Method available at http://msdn.microsoft.com/en-us/library/dd470175%28v=vs.108%29.aspx (last viewed: 2nd July, 2013)

MSDN, 2013 Anti-Cross Site Scripting Library available at http://msdn.microsoft.com/en-us/security/aa973814.aspx (last viewed: 2nd July, 2013)

4.14.7 Security in ASP .NET applications

ASP.NET Security

Securing web applications in ASP.NET requires an integration of configurations between the .NET framework and IIS. ASP.NET applications contains a web.config file where you can define many access and privileges for example, but they alone are not sufficient to protect the resources of your application. IIS plays a major role in protecting the website's assets contained in it too. It is important to understand the interaction between these components in order to implement proper securi

Integrating Authentication with IIS

Enable and configure the necessary type of authentication based on the security level required by your application. ASP.NET membership and ASP.NET login controls implicitly work with forms authentication.

The authentication methods used in IIS 7 are the following:

• Anonymous

• ASP.NET impersonation

• Basic

• Client certificate mapping,

• Digest

• Forms

• Windows Integrated Security (NTLM or Kerberos)

ASP.NET configuration works only for its resources. Keep in mind that if you need to configure access to resources of files contained in your application such as .txt, .gif, .jpg, these are done through the IIS permissions. For example, even though the ASP.NET resources in a directory might be forbidden by a Web.config file, users can still seethe files located in that directory if directory browsing is turned on and no other restrictions are in place.

Reference

http://msdn.microsoft.com/en-us/library/bwd43d0x%28v=vs.85%29.aspx

4.14.7.1 Strongly Named Assemblies

Overview Strongly Named assemblies

During the build process either QA or Developers are going to publish the code into executable formats. Usually this consists of an exe or and one or several DLL's. During the build/publish process a decision needs to be made to sign or not sign the code. Signing your code is called creating "strong names" by Microsoft. If you create a project using Visual Studio and use Microsofts "Run code analysis" most likely your will encounter a Microsoft design error if the code is not strong named; "Warning 1 CA2210 : Microsoft.Design : Sign 'xxx.exe' with a strong name key."

Code review needs to be aware if strong naming is being used, benefits and what threat vectors strong naming helps prevent or understand the reasons for not using strong naming.

A strong name is a method to sign an assembly's identity using its text name, version number, culture information, a public key and a digital signature.(Solis, 2012)

• Strong naming guarantees a unique name for that assembly.

• Strong names protect the version lineage of an assembly. A strong name can ensure that no one can produce a subsequent version of your assembly. Users can be sure that a version of the assembly they are loading comes from the same publisher that created the version the application was built with.

The above two point are very important if you are going to use Global Assembly Cache (GAC).

• Strong names provide a strong integrity check and prevent spoofing. Passing the .NET Framework security checks guarantees that the contents of the assembly have not been changed since it was built. Note, however, that strong names in and of themselves do not imply a level of trust like that provided, for example, by a digital signature and supporting certificate. If you use the GAC assemblies remember the assemblies are not verified each time they load since the GAC by design is a locked-down, admin-only store.

What strong names can't prevent is a malicious user from stripping the strong name signature entirely, modifying the assembly, or re-signing it with the malicious user's key.

The code reviewer needs to understand how the strong name private key will be kept secure and managed. This is crucible if you decide strong name signatures are a good fit for your organization.

If principal of least privilege is used so code is not or less susceptible to be access by the hacker and the GAC is not being used strong names provides less benefits or no benefits at all.

How to use Strong Naming

- Signing tools

In order to create a Strongly name assembly there is a set of tools and steps that you need to follow

- Using Visual Studio

In order to use Visual Studio to create a Strongly Named Assembly, it is necessary to have a copy of the public/private key pair file. Its is also possible to create this pair key in Visual Studio

In Visual Studio 2005, the C#, Visual Basic, and Visual J# integrated development environments (IDEs) allow you to generate key pairs and sign assemblies without the need to create a key pair using Sn.exe(Strong Name Tool). These IDEs have a Signing tab in the Project Designer. . The use of the AssemblyKeyFileAttribute to identify key file pairs has been made obsolete in Visual Studio 2005.

The following figure illustrates the process done by the compiler:

Using Strong Name tool

The Sign Tool is a command-line tool that digitally signs files, verifies signatures in files, or time stamps files. The Sign Tool is not supported on Microsoft Windows NT, Windows Me, Windows 98, or Windows 95. In case you aren't using the "Visual Studio Command Prompt" (Start >> Microsoft Visual Studio 2010 >> Visual Studio Tools >> Visual Studio Command Prompt (2010)) you can locate sn.exe at %ProgramFiles%\Microsoft SDKs\Windows\v7.0A\bin\sn.exe

The following command creates a new, random key pair and stores it in keyPair.snk.

sn -k keyPair.snk

The following command stores the key in keyPair.snk in the container MyContainer in the strong name CSP.

sn -i keyPair.snk MyContainer

The following command extracts the public key from keyPair.snk and stores it in publicKey.snk.

sn -p keyPair.snk publicKey.snk

The following command displays the public key and the token for the public key contained in publicKey.snk.

sn -tp publicKey.snk

The following command verifies the assembly MyAsm.dll.

sn -v MyAsm.dll

The following command deletes MyContainer from the default CSP.

sn -d MyContainer

- Using the Assembly Linker(Al.exe)

This tool is automatically installed with Visual Studio and with the Windows SDK. To run the tool, we recommend that you use the Visual Studio Command Prompt or the Windows SDK Command Prompt (CMD Shell). These utilities enable you to run the tool easily, without navigating to the installation folder. For more information, see Visual Studio and Windows SDK Command Prompts.

If you have Visual Studio installed on your computer: On the taskbar, click Start, click All Programs, click Visual Studio, click Visual Studio Tools, and then click Visual Studio Command Prompt. -or- If you have the Windows SDK installed on your computer: On the taskbar, click Start, click All Programs, click the folder for the Windows SDK, and then click Command Prompt (or CMD Shell).

At the command prompt, type the following:

al sources options

- Remarks

All Visual Studio compilers produce assemblies. However, if you have one or more modules (metadata without a manifest), you can use Al.exe to create an assembly with the manifest in a separate file. To install assemblies in the cache, remove assemblies from the cache, or list the contents of the cache, use the Global Assembly Cache Tool (Gacutil.exe).

The following command creates an executable file t2a.exe with an assembly from the t2.netmodule module. The entry point is the Main method in MyClass.

al t2.netmodule /target:exe /out:t2a.exe /main:MyClass.Main

Use Assembly attributes

You can insert the strong name information in the code directly. For this, dependingon wherethe key file is located you can use AssemblyKeyFileAttribute or AssemblyKeyNameAttribute

Use Compiler options :use /keyfile or /delaysign

Safeguarding the key pair from developers is necessary to maintain and guarantee the integrity of the assemblies. The public key is should be accessible, but access to the private key is restricted to only a few individuals. When developing assemblies with strong names, each assembly that references the strong-named target assembly contains the token of the public key used to give the target assembly a strong name. This requires that the public key be available during the development process. You can use delayed or partial signing at build time to reserve space in the portable executable (PE) file for the strong name signature, but defer the actual signing until some later stage (typically just before shipping the assembly). You can use /keyfile or /delaysign in C# and VB.NET

References

http://msdn.microsoft.com/en-us/library/wd40t7ad(v=vs.80).aspx

http://msdn.microsoft.com/en-us/library/c405shex(v=vs.110).aspx

http://msdn.microsoft.com/en-us/library/k5b5tt23(v=vs.80).aspx

http://msdn.microsoft.com/en-us/library/t07a3dye(v=vs.80).aspx

4.14.7.1.1 Round Tripping

Round Tripping

Round Tripping is a revese engineering technique that allow us to decompile an assembly from a certain application. Ildasm.exe can be used for this purpose, and ILAsm is used to recompiled the assembly. The MSIL Disassembler(Ildasm.exe) is a companion tool to the MSIL Assembler (Ilasm.exe). Ildasm.exe takes a portable executable (PE) file that contains Microsoft intermediate language (MSIL) code and creates a text file suitable as input to Ilasm.exe. This tool is automatically installed with Visual Studio and with the Windows SDK.

The importance of Obfuscation

As mentioned before , Round Tripping is indeed a technique used to reverse engineer assemblies. Therefore, if you want to avoid your assemblies being reversed engineered or even worse, that the code is victim of malicious manipulation using the Ildasm and Ilasm tools, then its is advisable to apply it. There are different kinds of products that can be used for this purpose such as DeepSea, Crypto or Dotfuscator.

4.14.7.1.2 How to prevent Round tripping

The importance of Obfuscation

As mentioned before , Round Tripping is indeed a technique used to reverse engineer assemblies. Therefore, if you want to avoid your assemblies being reversed engineered or even worse, that the code is victim of malicious manipulation using the ildasm and Ilasm tools, then its is advisable to apply it. There are different kinds of products that can be used for this purpose such as DeepSea, Crypto or Dotfuscator

Using Obfuscation

The most effective technique used to avoid reverse engineering and tampering of assemblies is the use of Obfuscation. Visual Studio contains a version of Dotfuscator. This program is accessible by choosing on the VS menu, Tools: Dotfuscator(Community Edition menu command). Note: This tools are not available in Express versions

To obfuscate your assemblies :

• Build the project in VS Studio

•Tools—> Dotfuscator Community Edition

• A screen prompts asking for which project type, choose 'Creat New Project' and click OK

• On the Input tab of the Dotfuscator interface, click 'Browse and Add assembly to list'

Browse for the compiled application

4.14.7.2 Setting the right Configurations

Introduction

Securing resources in ASP.NET applications is a combination of configuration settings in the Web.config file but also, its important to remember that the IIS configurations play also a big part on this. It's an integrated approach which provides a total framework of security. The following highlights the most important aspects of ASP.NET configuration settings within the web.config file. For a total overview see chapter ASP.NET security

Type of validation	Control to use	Description
Required entry	RequiredFieldValidator	Ensures that the user does not skip an entry.
Comparison to a value	CompareValidator	Compares a user's entry against a constant value, against the value of another control (using a comparison operator such as less than, equal, or greater than), or for a specific data type.
Range checking	RangeValidator	Checks that a user's entry is between specified lower and upper boundaries. You can check ranges within pairs of numbers, alphabetic characters, and dates.
Pattern matching	RegularExpressionValidator	Checks that the entry matches a pattern defined by a regular expression. This type of validation enables you to check for predictable sequences of characters, such as those in e-mail addresses, telephone numbers, postal codes, and so on.
User-defined	CustomValidator	Checks the user's entry using validation logic that you write yourself. This type of validation enables you to check for values derived at run time.

(https://www.owasp.org/index.php/CRV2_FrameworkSpecIssuesASPNet)

Secure Configuration Values

Sensitive Information saved in config files should be encrypted. Encryption keys stored in the machineKey element for example or connection strings with username and passwords to login to database.

Lock ASP.NET Configuration settings

You can lock configuration settings in ASP.NET configuration files (Web.config files) by adding an allowOverride attribute to a location element

Configure directories using Location Settings

Through the <location> element you can establish settings for specific folders and files. The Path attribute is used to specify the file or subdirectory. This is done in the Web.config file example:

```
<location path="." >
  <section1 .../>
  <section2 ... />
 </location>
 <location path="Default Web Site" >
  <section1 … />
  <section2 … />
 </location
 <location path="Default Web Site/MyApplication/Admin/xyz.html" >
  <section1 ... />
  <section2 ... />
 </location>
```

Configure exceptions for Error Code handling

Showing and handling the correct error code when a user sends a bad request or invalid parameters is an important configuration subject. Logging these errors are also an excellent help when analyzing potential attacks to the application.

It is possible to configure these errors in the code or in the Web.Config file

The HttpException method Describes an exception that occurred during the processing of HTTP requests. For example:

```
if (string.IsNullOrEmpty(Request["id"]))
    throw new HttpException(400, "Bad request");
```

or in the Web.config file:

```
<configuration>
 <system.web>
  <customErrors mode="On" defaultRedirect="ErrorPage.html"
                redirectMode="ResponseRewrite">
    <error statusCode="400" redirect="BadRequest.html" />
    <error statusCode="404" redirect="FileNotFound.html" />
  </customErrors>
 </system.web>
</configuration>
```

Input validation

Anything coming from external sources can be consider as input in a web application. Not only the user inserting data through a web form, but also data retrieved from a web service or database, also headers sent from the browsers fall under this concept. A way of defining when input is safe can be done through outlining a trust boundary.

Defining what is known as trust boundary can help us to visualize all possible untrusted inputs. One of those are user input.ASP.NET has different types of validations depending on the level of control to be applied. By default, web pages code is validated against malicious users. The following is a list types of validations used (MSDN, 2013):

References

MSDN, 2013 "Securing ASP.NET Configurations" available at http://msdn.microsoft.com/en-us/library/ms178699%28v=vs.100%29.aspx (Last Viewed, 25th July 2013)

4.14.7.3 Authentication Options

.NET Authentication Controls

In the .NET, there are Authentication tags in the configuration file. The <authentication> element configures the authentication mode that your applications use.

<authentication>

The appropriate authentication mode depends on how your application or Web service has been designed. The default Machine.config setting applies a secure Windows authentication default as shown below.

authentication Attributes:mode="[Windows|Forms|Passport|None]"

<authentication mode="Windows" />

Forms Authentication Guidelines

To use Forms authentication, set mode=“Forms” on the <authentication> element. Next, configure Forms authentication using the child <forms> element. The following fragment shows a secure

<forms> authentication element configuration:

```
<authentication mode="Forms">
 <forms loginUrl="Restricted\login.aspx" Login page in an SSL protected folder
  protection="All" Privacy and integrity
  requireSSL="true" Prevents cookie being sent over http
  timeout="10" Limited session lifetime
  name="AppNameCookie" Unique per-application name
  path="/FormsAuth" and path
  slidingExpiration="true" > Sliding session lifetime
 </forms>
</authentication>
```

Use the following recommendations to improve Forms authentication security:

• Partition your Web site.

• Set protection=“All”.

• Use small cookie time-out values.

• Consider using a fixed expiration period.

• Use SSL with Forms authentication.

• If you do not use SSL, set slidingExpiration = “false”.

• Do not use the <credentials> element on production servers.

• Configure the <machineKey> element.

• Use unique cookie names and paths.

Classic ASP

For classic ASP pages, authentication is usually performed manually by including the user information in session variables after validation against a DB, so you can look for something like:

Session ("UserId") = UserName

Session ("Roles") = UserRoles

4.14.7.4 Code Review for Managed Code - .Net 1.0 and up

Code Review Manage Code

.NET Managed code is less vulnerable to common vulnerabilities found in unmanaged code such as Buffer Overflows and memory corruption however there could be issues in the code that can affect performance and security. The following is a summary of the recommended practices to look for during the code review. Also, it is worth mentioning some tools that can make the work easier on this part and they can help you understand and pin point flaws in your code

Code Access Security

This supports the execution of semi-trusted code, preventing several forms of security threats. The following is a summary of possible vulnerabilities due to improper use of Code Access security:

(MSDN, 2013)

- Declarative security

Use declarative security instead of imperative whenever possible. Example of declarative syntax(MSDN[2], 2013):

```
[MyPermission(SecurityAction.Demand, Unrestricted = true)]
 public class MyClass
 {
  public MyClass()
  {
     //The constructor is protected by the security call.
  }
  public void MyMethod()
  {
     //This method is protected by the security call.
  }
  public void YourMethod()
  {
     //This method is protected by the security call.
  }
}
```

Unmanaged Code

Even though C# is a strong type language, it is possible to use unmanaged code calls by using the 'unsafe' code. "Check that any class that uses an unmanaged resource, such as a database

connection across method calls, implements the IDisposable interface. If the semantics of the object are such that a Close method is more logical than a Dispose method, provide a Close method in addition to Dispose".

Exception handling

Manage code should use exception handling for security purposes among other reasons. Make sure that you follow these recommendations:

• Avoid exception handling in loops, use try/catch block if it is necessary.

• Identify code that swallows exceptions

• Use exceptions handling for unexpected conditions and not just to control the flow in the application

Tools

- FxCop

FxCop is an analysis tool that analyses binary assemblies, not source code. The tool has a predefined set of rules and it is possible to configure and extend them. Some of the available rules regarding security are(CodePlex, 2010):

EnableEventValidationShouldBeTrue

Verifies if the EnableEventValidation directive is disabled on a certain page

ValidateRequestShouldBeEnabled

Verifies if the ValidateRequest directive is disabled on a certain page.

ViewStateEncryptionModeShouldBeAlways

Verifies if the ViewStateEncryptionMode directive is not set to Never on a certain page.

EnableViewStateMacShouldBeTrue

Verifies if the EnableViewStateMac directive is not set to false on a certain page.

EnableViewStateShouldBeTrue

Verifies if the EnableViewState directive is not set to false on a certain page.

ViewStateUserKeyShouldBeUsed

Verifies if the Page.ViewStateUserKey is being used in the application to prevent CSRF.

DebugCompilationMustBeDisabled

Verifies that debug compilation is turned off. This eliminates potential performance and security issues related to debug code enabled and additional extensive error messages being returned.

CustomErrorPageShouldBeSpecified

Verifies that the CustomErrors section is configured to have a default URL for redirecting uses in case of error.

FormAuthenticationShouldNotContainFormAuthenticationCredentials

Verifies that no credentials are specified under the form authentication configuration.

EnableCrossAppRedirectsShouldBeTrue

Verifies that system.web.authentication.forms enableCrossAppRedirects is set to true. The settings indicate if the user should be redirected to another application url after the authentication process. If the setting is false, the authentication process will not allow redirection to another application or host. This helps prevent an attacker to force the user to be redirected to another site during the authentication process. This attack is commonly called Open redirect and is used mostly during phishing attacks.

FormAuthenticationProtectionShouldBeAll

Verifies that the protection attribute on the system.web.authentication.forms protection is

set to All which specifies that the application use both data validation and encryption to \
help protect the authentication cookie.

FormAuthenticationRequireSSLShouldBeTrue

Verifies that the requireSSL attribute on the system.web.authentication.forms configuration element is set to True which forces the authentication cookie to specify the secure attribute. This directs the browser to only provide the cookie over SSL.

FormAuthenticationSlidingExpirationShouldBeFalse

Verifies that system.web.authentication.forms slidingExpiration is set to false when the site is being served over HTTP. This will force the authentication cookie to have a fixed timeout value instead of being refreshed by each request. Since the cookie will traverse over clear text network and could potentially be intercepted, having a fixed timeout value on the cookie will limit the amount of time the cookie can be replayed. If the cookie is being sent only over HTTPS, it is less likely to be intercepted and having the slidingExpiration setting to True will cause the timeout to be refreshed after each request which gives a better user experience.

HttpCookiesHttpOnlyCookiesShouldBeTrue

Verifies that the system.web.httpCookies httpOnlyCookies configuration setting is set to True which forces all cookies to be sent with the HttpOnly attribute.

HttpCookiesRequireSSLShouldBeTrue

Verifies that the system.web.httpCookies requireSSL configuration is set to True which forces all cookies to be sent with the secure attribute. This indicates the browser to only provide the cookie over SSL.

TraceShouldBeDisabled

Verifies that the system.web.trace enabled setting is set to false which disables tracing. It is recommended to disable tracing on production servers to make sure that an attacker cannot gain information from the trace about your application. Trace information can help an attacker probe and compromise your application.

AnonymousAccessIsEnabled

Looks in the web.config file to see if the authorization section allows anonymous access.

RoleManagerCookieProtectionShouldBeAll

Verifies that the system.web.rolemanager cookieProtection is set to All which enforces the cookie to be both encrypted and validated by the server.

RoleManagerCookieRequireSSLShouldBeTrue

Verifies that the system.web.rolemanager cookieRequireSSL attribute is set to True which forces the role manager cookie to specify the secure attribute. This directs the browser to only provide the cookie over SSL.

RoleManagerCookieSlidingExpirationShouldBeTrue

Verifies that the system.web.rolemanager cookieSlidingExpiration is set to false when the site is being served over HTTP. This will force the authentication cookie to have a fixed timeout value instead of being refreshed by each request. Since the cookie will traverse over clear text network and could potentially be intercepted, having a fixed timeout value on the cookie will limit the amount of time the cookie can be replayed. If the cookie is being sent only over HTTPS, it is less likely to be intercepted and having the slidingExpiration setting to True will cause the timeout to be refreshed after each request which gives a better user experience.

PagesEnableViewStateMacShouldBeTrue

Verifies that the viewstate mac is enabled.

PagesEnableEventValidationMustBeTrue

Verifies that event validation is enabled.

HttpRuntimeEnableHeaderCheckingShouldBeTrue

Verifies that the system.web.httpRuntime enableHeaderChecking attribute is set to true. The setting indicates whether ASP.NET should check the request header for potential injection attacks. If an attack is detected, ASP.NET responds with an error. This forces ASP.NET to apply the ValidateRequest protection to headers sent by the client. If an attack is detected the application throws HttpRequestValidationException.

PagesValidateRequestShouldBeEnabled

Verify that validateRequest is enabled.

PagesViewStateEncryptionModeShouldBeAlways

Verifies that the viewstate encryption mode is not configured to never encrypt.

CustomErrorsModeShouldBeOn

Verifies that the system.web.customErrors mode is set to On or RemoteOnly. This disable detailed error message returned by ASP.NET to remote users.

MarkVerbHandlersWithValidateAntiforgeryToken

Verifies that ValidateAntiforgeryTokenAttribute is used to protect against potential CSRF attacks against ASP.NET MVC applications.

- OWASP O2 platform(VisualStudio C# REPL - O2 Platform (v5.1))

The O2 platform represents a new paradigm for how to perform, document and distribute Web Application security reviews. O2 is designed to Automate Application Security Knowledge and Workflows and to Allow non-security experts to access and consume Security Knowledge. We

strongly recommend it to .NET developers since this tool helps you identify current security issues in the framework.(Cruz, 2013)

O2 Platform is also available as a VisualStudio Extension which you can download from VisualStudio Gallery (see VisualStudio C# REPL - O2 Platform) or directly using VisualStudio's Extension Manager: http://visualstudiogallery.msdn.microsoft.com/295fa0f6-37d1-49a3-b51d-ea4741905dc2

References

MSDN 2013 "Code Access Security" available at http://msdn.microsoft.com/en-us/library/33tceax8.aspx (Last viewed 25th July, 2013)

MSDN[2], 2013, "Declarative Security" available at http://msdn.microsoft.com/en-us/library/kaacwy28.aspx (Last viewed 25th July, 2013)

CodePlex, 2010 "Fxcop ASP.NET Security Rules" Available at http://fxcopaspnetsecurity.codeplex.com/ (Last viewed 25th July, 2013)

Cruz, D. 2013 OWASP O2 Platform, available at https://www.owasp.org/index.php/OWASP_O2_Platform (Last viewed 25th July, 2013)

4.14.7.5 Using OWASP Top 10 as your guideline

Using OWASP TOP 10 as your guideline

The OWASP TOP 10 (https://www.owasp.org/index.php/Top_10_2013-Top_10)is a detailed list of the highest security risks web application faces. It help us identify the most critical security threats facing organizations. Performing a Code review efficiently requires using a model or framework that help us identify these issues quickly. Consequently, OWASP TOP 10 is one of these guides that provides us with the necessary information to implement proper Code Review.

Applying OWASP TOP 10 to ASP.NET code review

The following table contains OWASP TOP 10 - 2013 guideline and how you can apply this during your code review

OWASP TOP 10 risk	Description	What to look for in the code
A1 Injection	Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.	SQL queries are parameterized and that any input used in a SQL query is validated. Look for implementation of Parameter collections If using Stored procedures that GRANT proper permissions and avoids using unfiltered content from user If using LDAP services , check that clear text passwords and authentication is not used in the code
A2 Broken Authentication and Session Management	Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities.	No use of encryption to save passwords Clear-text credentials in web.config files Clear connectionstrings without encryption Proper implementation of <Authentication> en <Authorization> property in web.config files Never use "none" for the <protection> attribute within <Forms> property in web.config file Form element Encryption property: ASP.NET encrypts the cookie, but is not validate it. This may leave your application open to attack. Form element Validation: ASP.NET validates the cookie, but does not encrypt it. This can uncover information to an attacker. Look for failure to limit database access and inadequate separation of privileges Connectionstrings without setting a password such as Server=.\SQLExpress;AttachDbFilename=|DataDirectory|database.mdf;Trusted_Connection=Yes;
A3 Cross-Site Scripting (XSS)	XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect	Check for implementations of RequestValidator method proper escape of characters Implementation of Microsoft Anti-XSS library If using MVC, make sure you implement

	the user to malicious sites	HtmlEncode method
A4 Insecure Direct Object References	A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.	Filtering Requests and URL rewriting in IIS configurations Check configuration of Trust levels and Code Access security permission Check Authorization and Authentication settings
A5-Security Misconfiguration	Security misconfiguration can happen at any level of an application stack, including the platform, web server, application server, database, framework, and custom code. Developers and system administrators need to work together to ensure that the entire stack is configured properly. Automated scanners are useful for detecting missing patches, misconfigurations, use of default accounts, unnecessary services, etc.	**IIS configurations:** Proper authorization and authentication configuration in web.config,Machine config , ApplicationHost.config **SQL (server) code:** GRANT permissions for executing store procedures Privilege of application SQL account to authenticate into database **Protect assemblies:** Implementation of Obfuscation for assemblies Prevention of Round tripping Always check the MD5 hashes of the .NET Framework assemblies to prevent the possibility of rootkits in the framework. . Checking the MD5 hashes will prevent using altered assemblies on a server or client machine. See File:Presentation - .NET Framework Rootkits - Backdoors Inside Your Framework.ppt
A6-Sensitive Data Exposure	The most common flaw is simply not encrypting sensitive data. When crypto is employed, weak key generation and management, and weak algorithm usage is common, particularly weak password hashing techniques. Browser weaknesses are very common and easy to detect, but hard to exploit on a large scale. External attackers have difficulty detecting server side flaws due to limited access and they are also usually hard to exploit.	Encryption of sensitive data in config files Salting passwords Proper implementation of Secure Encryption algorithms Look for hard-coded secrets in code by looking for variable names such as "key", "password", "pwd", "secret", "hash", and "salt"

A7-Missing Function Level Access Control	Applications do not always protect application functions properly. Sometimes, function level protection is managed via configuration, and the system is misconfigured. Sometimes, developers must include the proper code checks, and they forget. Detecting such flaws is easy. The hardest part is identifying which pages (URLs) or functions exist to attack.	Proper configuration of Machine level/User level RSA key container Proper Trust Levels configurations Use of Strong named assemblies Use of Assembly Attributes Proper configuration of folder/ CRUD access permissions of web content files Search for the use of asserts, link demands, and allowPartiallyTrustedCallersAttribute (APTCA).
A8-Cross-Site Request Forgery (CSRF)	A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.	Use of Anti-forgery Helpers, Html.AntiForgeryToken() and the filter [ValidateAntiForgeryToken] Implementation of Log Out functionality
A9-Using Components with Known Vulnerabilities	Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.	Check version of .NET framework Check for found vulnerabilities in the Framework used on the application If using components such as .NETNuke or NuGEt libraries check for vulnerabilities in these.
A10-Unvalidated Redirects and Forwards	Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.	Simply avoid using redirects and forwards MVC1 and 2 are vulnerable for redirections,therefore proper implementation of validation in returnUrl parameters is essential If user input can't be avoided, ensure that the supplied value is valid, appropriate for the application, and is authorized for the user.

4.14.7.6 Code review for Unsafe Code (C#)

C# Unsafe Code

Even though C# has a strong memory management infrastructure, there will be times when is necessary to direct access memory

• Dealing with existing structures on disk

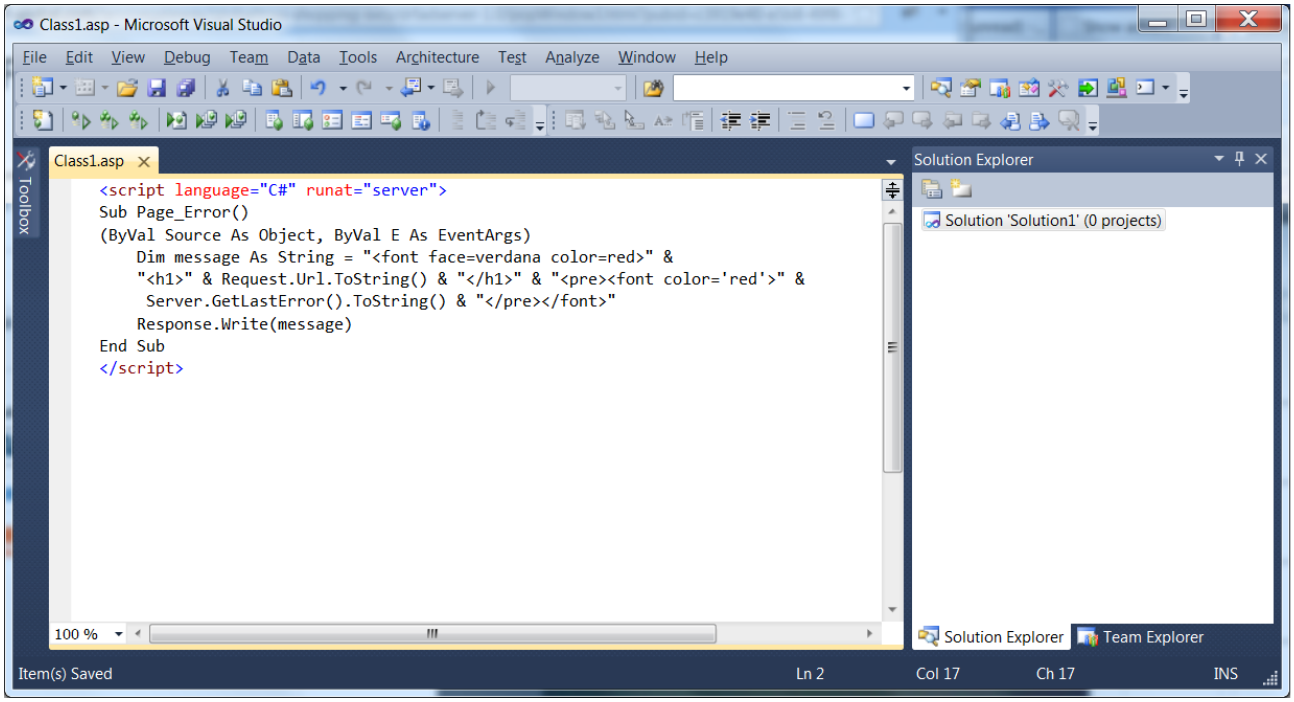
• Advanced COM or Platform Invoke scenarios that involve structures with pointers in them

• Performance-critical code (Microsoft, 2009)

Microsoft strongly discourages the use of the unsafe code when this is not necessary. It is clear that even when using unsafe code might improve performance in the program, the risks might overcome the benefits. Definitely, this is no area for inexperienced programmers.

Unsafe is used by declaring the “unsafe” keyword in the program code. For example:

```
class UnsafeTest {
  // Unsafe method: takes pointer to int:
  unsafe static void SquarePtrParam(int* p)
  {
    *p *= *p;
  }
  unsafe static void Main()
  {
    int i = 5;
    // Unsafe method: uses address-of operator (&):
    SquarePtrParam(&i);
```




```
    Console.WriteLine(i);
  }
}
// Output: 25
```

Risks of using Unsafe Code

Major risk involves

- Buffer overflows
- Unverifiable code
- Pointer errors

References

Microsoft, 2009 , Unsafe Code , available athttp://msdn.microsoft.com/en-us/library/aa288474%28v=VS.71%29.aspx (accessed on 01-07-2013)

4.14.8 PHP Specific Issues (Needs Content)

Lorem Ipsum

4.14.9 Classic ASP

Unlike Java and .NET, classic ASP pages do not have structured error handling in try-catch blocks. Instead they have a specific object called "err". This make error handling in a classic ASP pages hard to do and prone to design errors on error handlers, causing race conditions and information leakage. Also, as ASP uses VBScript (a subtract of Visual Basic), sentences like "On Error GoTo label" are not available.

Vulnerable Patterns for Error Handling

- Page_Error

Page_Error is page level handling which is run on the server side. Below is an example but the error information is a little too informative and hence bad practice.

The text in the example above has a number of issues: Firstly, it redisplay the HTTP request to the user in the form of Request.Url.ToString() Assuming there has been no data validation prior to this point, we are vulnerable to cross site mscripting attacks!! Secondly the error message and stack trace is displayed to the user using Server.GetLastError().ToString() which divulges internal information regarding the application.

4.14.10 C# (Needs Content)

Lorem Ipsum

4.14.11 C/C++ (Needs Content)

Lorem Ipsum

4.14.12 Objective C (Needs Content)

Lorem Ipsum

4.14.13 Java (Needs Content)

Secure configurations in Web.xml

The Web.xml file is the main configuration document responsible for securing configurations in Java Applications. The following section information is based on the article written by Frank Kim(2010) which describes important configuration necessary to protect them.

Configure Custom Error pages

All errors generated by the application, such as 404, 500 etc, must be configured in order to redirect the user to a proper Error page instead of allowing him to see the errors generated by the application. This can serve as a starting point to an attacker to reverse engineer the application and create a specific attack using this information

<error-page>

<error-code>505</error-code>

<location>/error/error.html</location>

</error-page>

Protect data in transit

In order to secure sensitive data, is essential to secure the communication channel and sessions using SSL. Once this has been configured in the server, doesn't mean that it will be automatically be setup in the web application the developer is trying to secure. For this purpose, it is essential to add in the web.xml file the following configuration(Kim, 2010) :

<security-constraint>

...

<user-data-constraint>

<transport-guarantee>CONFIDENTIAL</transport-guarantee>

</user-data-constraint>

</security-constraint>

Configuring proper Authentication and Authorization to directories

Failure to configure proper authentication and authorization of directories, will allow anonymous users to see unprotected files of the web application. Therefore, consider always to set-up proper access controls in the following sections. The following code, for example, makes sure that the 'Accountant' role, is the only one able to access directory "accounting"

<security-constraint>

<web-resource-collection>

<web-resource-name>accounting</web-resource-name>

<url-pattern>/accounting/*</url-pattern>

…

</web-resource-collection>

<auth-constraint>

<role-name>accountant</role-name>

</auth-constraint>

</security-constraint>

Configure http methods

Allow only the necessary http methods to execute in the application, such as the case of GET and POST requests. If the methods are not overtly listed are by default allowed. This will allow an attacker to bypass the web.xml configuration. By removing <http-method> elements from the web.xml and this will offer the proper security.

Use Secure Flag

Make sure that the cookie is created using the seucre flag, otherwise exposes the session cookie to hijacking.

<session-config>

<cookie-config>

<secure>true</secure>

</cookie-config>

</session-config>

Setting a time out session

Undefined time out sessions allows hackers to execute CRSF attacks and hijacking the session make sure that <session-timeout> property is set to a time (in minutes)

References

Fran Kim, 2010 "Seven Security (Mis)Configurations in Java web.xml Files" available at http://software-security.sans.org/blog/2010/08/11/security-misconfigurations-java-webxml-files/ accessed on 4rd October 2013

4.14.14 Android (Needs Content)

Lorem Ipsum

4.14.15 Coldfusion (Needs Content)

Lorem Ipsum

4.14.16 CodeIgniter (Needs Content)

Lorem Ipsum

5 Security code review for Agile development

Some definitions about Agile

The Agile name is an umbrella for quite a lot of practices that range from programming, to testing, to project management and everything in between. There are many flavors of agile, perhaps as many as practitioners. It is like an heterogeneous reference framework and you are free to use what you want.

Agile has some key practices that could affect the way the code is reviewed. First, when the review is done and then, the code itself.

Agile Development is well suited for code review, as two of its practices are "pair programming" and "peer review". AD incorporates code review in their self, in what traditionally was seem as another phase.

- Life Cycle

Agile blurs the difference between developing and testing, and so does with code review. It is not an external activity. Agile tries to keep the code testing and review as near as possible to the development phase, there is no such thing as the develop, test, code review cycle.

It is a common practice to define short development cycles. At the end of each one, all the code must be production quality code. It can be incomplete, but it must add some value. That affects the review process as it must be continuous.

- Pair Programming

This technique is quite controversial, but when it is adopted it has the benefits of making better code, as there is one programmer supervising cooperatively the other one's work. If both programmers know this guide, they will apply it continuously.

- Peer Review

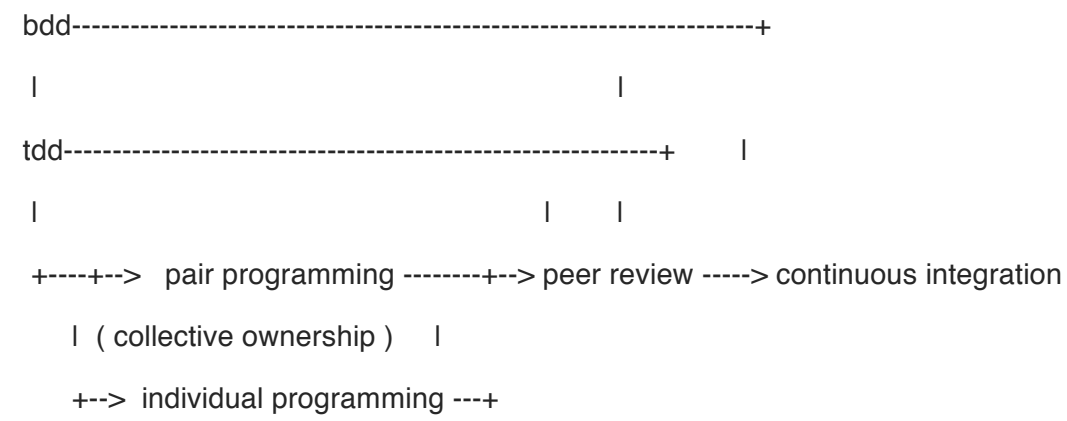
This one is enforced by the usage of tools like Jenkins that ask another user for a code review before committing to the versioning system.

The role of testers...

- Clean Code and "Smells"

The agile community is very committed to code quality...

The Agile Ecosystem



Code Reviewing an Agile Project

If you are going to review an Agile Team project code, the best thing that you can do is give this guide to that Team as early as possible and most of your work will be done for you. Or better yet, integrate to the Team.

Code review must be done at least at the end of every user story and be very fast in order to not introduce delays and detect any error as soon as possible. It's better to do that in every commit to the code repository. That is the reason that peer review is the preferred method for agile code review. Nevertheless, as security review requires some extra specialization, perhaps the best way is to integrate the security code reviewer with the Team.

Tests

Agile projects tend to use a lot of automated testing, in order to review an Agile Project, you will have to extend the review to the tests.

Some guide taken from ...

- Are there enough tests?
- Is the code covered by the tests? Code coverage measure main value is to find unused code.
- Are there the trivial tests? They are as needed as any other test.

- Are there commented out tests? Commented out tests means that some one made a test that the code could not pass or take a long time to run.
- Are boundary conditions tested? These are the tests around maximum and minimum values or near a change in a condition.
- Are bugs exhaustively tested? That is, is an off-by-one bug is found, are there boundary tests for that condition?
- Are the test automatic? If the tests require manual intervention, they will not be run.
- Do the tests conform to F.I.R.S.T.?

Fast: a slow test is a candidate for removal, as it slows down the "make test, implement, test, refactor" cycle.

Independent: one test can not depend on the execution of another one, the order should not matter.

Repeatable: one test should give always the same result in any environment if nothing has changed in the code.

Self-validating: the result of a test should be Pass or Fail, nothing else. There should not be any manual intervention needed.

Timely: the test should be written before the code. That can not be detected with code review, but you can always see the logs of the versioning system.

Other Agile Resources

- The role of testing

It is so fundamental, that the xDD pervades Agile, test first, test earlier

- Continuous integration

it triggers the tests and often static code analysis too. It makes integration problems arise very quickly.

- The role of automatic static code analysis in the Agile Methodologies

TBD

- The XDD

There are many agile practices related to what drives the development of a project, what they have in common is that they could generate testing code. As a security practicioner, there are two aspects of interest. One, sometimes useful is called "test coverage" and it is automatically calculated. 100% code coverage does not mean a good coverage neither a 60% a bad one. It is very difficult to measure the second aspect, the quality of the test. There is possitive testing, that aims at the added value of every piece of software and negative testing, related to bugs and security.

- Test Driven Development

TDD is the practice of making the test before coding, it is the extremme application of the "test early" principle. The idea is that the code always will be tested as the test predates the code itself. A very important side effect is that it forces to simplify the code to make it testeable. It could be very low level with very isolated components, called "unit testing" and high level, when it tests clusters of interrelated components, called "functional testing".

To read more about code coverage: [1]

- Behavior Driven Development

This practice builds upon TDD, providing an interface to non-specialists users, shaping the tests around full blown scenarios. As TDD, it generates testing code.

- Domain Driven Design

This practice consist of... It does not generate code itself, but the architecture. It is very popular among the Agile people, so it is very important to be familiar with. Perhaps the most useful concept is "ubiquitous language".

- Refactoring

Refactoring is the art of changing the code with out changing its behavior. In order to refactor, there must exists a rich battery of tests.

The problem with refactoring is that thanks to the heavy testing, you can trust that the interface does not change, but behind it, the code can be very volatile. You have to review the code continously, another argument for peer review and automatic static analysis.

References and sources

Clean Code: A handbook for Agile Software Craftsmanship - Robert C. Martin

http://groups.yahoo.com/group/foro-agiles/

http://martinfowler.com/articles/continuousIntegration.html

http://martinfowler.com/bliki/TestCoverage.html

not used http://refactoring.com/

not used http://dddcommunity.org/

6 Code Review Tools

Overview

As discussed in Code Review Guide there are many reasons to automate the process of code reviews within the organization SDLC practices. We won't review all those reasons here again but we would like to share with the reader a list of the tools both commercial and open source. OWASP is vendor natural for that reason the vendors themselves supply the text below unless otherwise stated. OWASP does not endorse commercial or open source tools outside of OWASP own projects.

Commercial Code Review Tools

- Crucible by Atlassian Software

https://www.atlassian.com/software/crucible/overview

- Begin Atlassian supplied description of their Code Review tool

Crucible is Atlassian’s on-premises code review solution for enterprise teams. Crucible makes it easy to review code changes, make comments and record outcomes thoroughly and efficiently. It encourages developers to carry out more code reviews – improving code quality and fostering collaboration. It is code review made easy for Subversion, CVS, Perforce and other systems.

The flexible code review process allows you to configure your reviews based on workflows or participants. Whether used to perform ad-hoc reviews or in a formal process, Crucible removes the administrative overhead and enables distributive teams to work together. As reviews are inherently iterative, Crucible's fully threaded comments let teams discuss code regardless of time and location and provide comments directly on specific source lines and files.

When using Crucible, individuals can create reviews directly from the command line, build quick reviews with cut-and-paste snippets and perform one-click reviews from changesets or issues. These reviews can be carried out before check-ins, ensuring the quality of code going into production. As files are always kept up-to-date, developers do not have to worry they are reviewing code that is outdated. With the added bonus of notifications & reminders, audit trails, and reports, Crucible is here to help you produce the best source code possible.

End Atlassian supplied description of their Code Review tool

Open Source Code Review Tools

http://www.reviewboard.org