Seth & Ken's Excellent Adventures (in Code Review)

October 2018

Background

This document provides readers with a common methodology to follow during security-focused source code reviews. It is meant as a guide and readers are encouraged to perform additional checks to ensure that a thorough security examination of the target is performed. This methodology was designed using OWASP's Code Review Guide as a baseline.

Scope

This methodology described in this document can be applied to code reviews of multiple types, including:

- Web Applications
- Client-Side Applications
- Mobile Applications
- Web Services

This document and methodology do NOT address issues related dynamic security testing of an application, network, or database. Security practitioners are encouraged to perform additional tests to ensure that proper security controls are enforced as expected.

Requirements

The methodology assumes that reviewers have a fundamental level of knowledge about application security principles, the provided source language (e.g. Java, C#, PHP, JavaScript) or framework (e.g. Node, Spring, .NET MVC), and means for identifying security issues in source code. At a minimum, a reviewer should possess the following skills and knowledge:

- Familiarity with OWASP Top 10 Most Critical Web Application Security Risks, including testing and mitigating risks of such. Ideally, the reviewer will have familiarity with the principals and topics outlined in the OWASP Code Review Guide.
- Experience building applications in the target language and framework, or experience reviewing source of similar applications.
- Knowledge of common programming paradigms (model-view-controllers) and communication protocols, such as HTTP and SSL/TLS.

Approach

This document details the activities performed during a security code review. The purpose of a security code review is to integrate security into developed software early in the software development lifecycle (SDLC) to decrease the total cost of building security into a product. To assess the security of a code base, reviews depend upon a number of tools and techniques, which are detailed in this document.

In addition to the methodology described in this document, automated static analysis tools specifically designed to identify security defects in source code can be used to supplement the security review. Supported languages and frameworks of the target source is supported, both commercial and open source static analysis tools should be used to inform the manual source code review process.

Static analysis scanners are usually limited because they can only detect easy to define vulnerabilities. Business logic flaws and authorization issues are typically hard to identify programmatically, meaning that the reviewer should pay special attention to these classes of vulnerabilities.

Reviewers can use static analysis tools to aid in manual testing, but additional source code checks are primarily tester-driven. This manual review has the following advantages:

- Manual reviews produce few false positives; while they may be assisted by static analysis tools, experienced reviewers are able to identify and account for mitigating controls that would be missed by automated scanning alone. -
- Manual reviewers are able to recognize security flaws in complex business logic and application flows that would not be identified by automated tools.

Manual testing has one significant disadvantage:

- Human testers are much slower than automated scripts.

A source code review is, by necessity, more than simply running an automated scanner and reviewing source. While important, these techniques must comprise a more holistic approach to assessing risks posed by the application and involve direct interactions with developers and application owners as reviewers seek to understand the purpose and processes surrounding the targeted application.

The source code review methodology will require reviewers to engage with application owners before and during all phases of the assessment. This will better ensure that reviewers have a proper understanding of the application prior to examining it. The methodology will also require various levels of testing, including interrogatory means, automated testing, and manual testing.

Source Code Review

Not every check outlined here will be relevant for the source code under review. It is left to the discretion of the reviewers and management to determine which vulnerabilities will be pertinent and those that are not. Conversely, this is not an exhaustive list of checks, and the tester can perform additional checks and/or analysis if it's deemed valuable. This methodology assumes that the reviewer has sufficient levels of experience with the targeted source code language and framework or peers with such experience to guide their efforts.

At a high level, the source code framework can be distilled to the following steps:

1- Determine application purpose
2- Map the application
3- Brainstorm risks to the application
4- Build list of review items
5- Perform review
6- Report

Taken a step deeper, the recommended approach includes:

- Application Overview and Risk Assessment
- Information Gathering
- Authentication Review
- Authorization Review
- Audit Review
- Injection Review – Input Validation & Output Encoding
- Cryptographic Review
- Configuration Review
- Reporting and Retesting

Each section in this document will introduce high-level concepts related to the topic, recommended sample sets of checks, and, in some instances, links to more detailed testing checks. The Additional Resources will provide links to granular checklists as well.

Application Overview and Risk Assessment

Overview

Reviewer must understand the application's purpose and risk in order to ascertain vulnerabilities and plan out level of effort and scrutiny when performing a code review. Moreover, a fuller understanding of the application will facilitate the development of a threat model that can be used to tailor future tests that are applied.

Guidance

Work with application contacts, including owner, project manager, and developers in this phase. Use project documentation, interviews, and artifacts include configuration files and relevant source files to gather the following information. This will enable the reviewer to validate that any access or authorization for users is appropriate given the use case that the application will have.

Checklist

- Architecture overview
  The reviewer should be familiar with each component of the application's source, including the client interactions, deployment plans, development framework, and supporting data stores. This must also include descriptions of data flows and protocols in use.
- Data sensitivity
  The reviewer needs to understand what data is being collected, stored, and transmitted throughout the application.
- Logging and monitoring capabilities
  Do the various source code libraries and functions generate audit logs that can be reviewed for suspicious activity or to support non-repudiation of a transaction? How will security logs be monitored on an ongoing basis? Are security logs sent to an appropriate external log aggregation system (e.g. Splunk) to support event correlation and long-term storage according to log retention?
- Authentication model
  Applications should use standard industry-supported methods for user and service authentication Where business requirements are not supported by established authentication solutions, business owners may be required to seek a formal security policy exception and document the business justification for the custom authentication mechanism. Security reviewers should take additional care to review any such

customized authentication solution. In addition, the reviewer should determine if multiple sessions are permitted for a user, either by design or unintentionally, and what the lockout policy is, even in the case that single-sign-on (SSO) is being used.

- Authorization levels and methods
  Applications should enforce fine-grained access control for granting roles and privileges. A reviewer needs to understand the different roles in order to validate permission checks within the source.
- Patching process
  The lack of proactive patching on the part of an application owner may expose an application or its underlying components (e.g. outdated middleware versions) to vulnerabilities before they can be detected and reported by the Paranoids.

Information Gathering

Overview

Reviewer maps out application exposed endpoints and functionality (also known as data sources), externally dependent services (also known as data sinks), and relevant data flows for each test procedure to ensure that vulnerabilities can be identified along relevant paths.

Guidance

Begin the formal technical portion of the review through initial source code review, dependency analysis, and automated testing. by enumerating the application's source for data flow sources and sinks. A source is defined as any data provided to or created within the application and a sink is anywhere it sends data to another portion of the code, service, application, library, or component that is not defined within the provided code.

Checklist

- Review application architecture and design documentation, including user and role-based privileges, site partitioning, layered security, and data classifications
- Identify all third-party libraries, dependencies, and components outside of the provided source code.
- Run the code through a static analysis tool, if available, for an initial map of the application and possible security issues.
- Build the source, which will ensure you have all necessary components, dependencies and libraries.
- Document all application sources, including available endpoints, user-input, configuration files, and exposed functions.
- Identify framework and language configuration files, whether yaml, xml, config or otherwise.
- Identify application sinks, where data is sent by the source code to another service, API, component, or library. This includes places where data is reflected back to a client through HTML, JSON, or other methods.

- Review source code for hard-coded keys, passwords, URLs or other sensitive data. Static analysis tools such as HPE Fortify or Veracode can help in this process.

  Developers often hard-code values into source and configuration files for accessing backend services, databases, or interacting with cloud providers. These can usually be identified by searching the codebase for terms like `password`, `username`, `database`, `KEY`, etc. The reviewer should use their intuition in identifying the structure a developer uses to create these strings and keep an eye out as they review sensitive code paths.

- Review source code for interesting comments.
  Comments embedded in source code are often placed by development tools or developers as they debug code or add references for others. If these comments are not removed prior to production deployment, then they may provide information to an attacker about the code and functionality, templates used in the development of the page, and perhaps names/usernames of the developers. In addition, comments sometimes include previously-vulnerable code that could reintroduced by developers unfamiliar with the issue.
  Each language has different characters for defining comments, so use language knowledge to find any instances.

  Findings at this stage involve recommending the removal of comments, especially those that offer helpful information to potential attackers.

Authentication Review

Overview

Examine the authentication flow of the application, what resources are available to unauthenticated vs. authenticated users, and the process by which a user identifies themselves through credentials, tokens, or other methods.

Guidance

Use the enumeration of available application endpoints to trace authentication flow and functions. Any sensitive or business functionality should redirect to this flow if a user has not been identified. A a part of authentication review, the user registration process should be also be documented and reviewed.

Checklist

- What are the different authentication flows?
    - User Login
    - User Registration
    - Forgot Password
- How are users identified? What information do they have to provide?
    - Username, email, password, 2fa token, etc.
- How is authentication handled on each application endpoint?
    - Sensitive endpoints should require authentication
- Are there any hard-coded accounts?
- Does application allow for easily-guessed, default, or common passwords?
- How are usernames determined, what naming conventions?
    - Does registration allow for easy enumeration?
- Does the application allow for account enumeration through server responses or information disclosure?
    - login/registration/forgot password flows
- Are user credentials protected in the data store using modern password hashing algorithms?
- Are security policies are configurable via environment variables and not hard-coded?
- What standard security frameworks are used?
    - Is there code specific to the application, especially when dealing with password storage?

- How are user management events such as authentication failures, password resets, password changes, account lockout and disabled accounts handled?
- Are suspicious event handling such as multiple failed login attempts, session replay and attempted access to restricted resources handled properly?
- Does the application implement strong password policies?
- Are authentication credentials being passed using technologies that could be cached (HTTP GET, lack of proper cache-control settings)?
- If applicable, are encryption mechanisms in place during authentication for secure communications (TLS, etc)?

User registration checklist:

- Are limits on application access, such as geographical boundaries, enforced properly?
- Is there a manual approval process or is access granted automatically?
    - Can the automated process be abused or bypassed using scripting or brute-forcing?
- In cases where a validation email and link are required, how are the tokens generated?
- Can users elevate their initial access via mass assignment or business-logic bypasses?
- Are files and objects owned by a user removed or archived?

Session Management checklist:

- Are encryption and hashing used properly?
- Do encryption protocols use strong algorithms and industry-standard key lengths?
- Are authentication tokens set with time limits?
- Are cookies security parameters set properly (e.g. Secure, HTTPOnly, path)?
- Are session IDs sent over a secure channel?
- Are session IDs invalidated before a new login is made?
- Are CSRF tokens set for all authentication requests?

Authorization Review

Overview

Reviewer validates that user roles defined in the application separate user boundaries appropriately and that users are registered and provisioned properly. Authorization review validates that the application enforces the rules discovered during the information gathering phase that determines what that user is allowed to do.

Guidance

To identify authorization flaws in application source, the reviewer should note each level of authorization within the application source provided and determine which functions are available to each role. For instance, one user should not be able to edit another user's application profile.

Pay particular attention to test both vertical and horizontal authorization weaknesses. Horizontal authorization flaws allow one user to perform actions within the same role level but on unauthorized data, e.g. a user accessing the private messages of another user. Vertical authorization involves escalating privileges to a higher level of access, e.g. a public user gaining access to administrative functions for resetting passwords for all users.

Checklist

- What are the different authorization functions?
    - Token/User/Role validation?
    - Is this handled by custom framework functionality?
    - Decorators vs. static source code
- Can non-privileged users view, add, or alter accounts?
- Is there functionality to add accounts with higher access levels than their own access?
- How is separation of duties handled?
- Are disabled accounts prevented from accessing content?
- Can password-protected pages be directly accessed without authentication?
- Is it possible to bypass authorization restrictions by accessing content directly (e.g. can a non-privileged user access the administration pages of an application)?
- Can a user escalate privileges through cookie modification, altering form input values and HTTP headers, or by fuzzing URL-based parameters?
- Are there horizontal escalation/Insecure Direct Object References in the source code?
- Are authentication and authorization flows the first logic executed for each request?

- Are authorization checks granular (page and directory level) or per-application?
- Are access to sensitive pages and data denied by default?
- Are users forced to re-assert their credentials for requests that have critical side-effect (account changes, password reset, etc)?
- Do authorization checks have clearly defined roles?
- Can authorization be circumvented by parameter or cookie/token manipulation?
- Are CSRF protections in place and appropriate?

Auditing (Logging) Review

Overview

Reviewer will validate the logging and exception handling of the application source, insuring that no sensitive data is logged, and any logging mechanisms are used in a secure manner.

Guidance

From a security review perspective, any endpoint that performs a state-changing operation or has security implications (authentication, authorization, registration, etc) should be logged for immediate analysis and/or future forensic purposes. Included in this portion of the review are checks to ensure that logs may not be manipulated by malicious user input and are appropriate for each user action.

Checklist

- If an exception occurs, does the application fails securely?
- Do error messages reveal sensitive application or unnecessary execution details?
- Are Component, framework, and system errors displayed to end user?
- Does exception handling that occurs during security sensitive processes release resources safely and roll back any transactions?
- Are relevant user details and system actions logged?
- Is sensitive user input flagged, identified, protected, and not written to the logs?
    - Credit Card #s, Social Security Numbers, Passwords, PII, keys
- Are unexpected errors and inputs logged?
    - Multiple login attempts, invalid logins, unauthorized access attempts
- Are log details should be specific enough to reconstruct events for audit purposes?
- Are logging configuration settings configurable through settings or environment variables and not hard-coded into the source?
- Is User-controlled data validated and/or sanitized before logging to prevent log injection?

Injection Review - Input Validation

Overview

Reviewer insures that all input presented to the application is validated for type, format, and content before being used or stored by the application.

Guidance

Utilize the data sources identified during the information gathering phase for this input validation review. Keep in mind that any data source identified needs to be validated, including session variables, external service calls, configuration files, database calls, or any other untrusted data.

Checklist

- Is all input is validated without exception?
- Do the validation routines check for known good characters and cast to the proper data type (integer, date, etc.)?
- Is the user data validated on the client or server or both (security should not rely solely on client-side validations that may be bypassed)?
- If both client-side and server-side data validation is taking place, are these validations consistent and synchronized?
- Do string input validation use regular expressions?
- Do these regular expressions use blacklists or whitelists?
- What bypasses exist within the regular expressions?
- Does the application validate numeric input by type and reject unexpected input?
- How does the application evaluate and process input length?
- Is a strong separation enforced between data and commands (filtering out injection attacks)?
- Is there separation between data and client-side scripts?
- Is provided data checked for special characters before being passed to SQL, LDAP, XML, OS and third party services?
- For web applications, are often forgotten HTTP request components, including HTTP headers (e.g. referrer) validated?

The OWASP Input Validation Cheatsheet documents can be used as a reference to better ensure thorough validation routines are used.

Injection Review - Output Encoding

Overview

The reviewer insures that the Principle of Layered Security is followed and that all application data sources (user input) is protected against misuse when passed to other application components (data sinks).

Guidance

This portion of the review goes hand in hand with input validation to ensure that data is contextually appropriate before it is passed to other uncontrolled sinks.
Review each sink identified during the enumeration phase of the review for output encoding, including coding libraries, third-party services, storage components, file system interactions or anything sink the application could pass data to.

Checklist

- Do databases interactions use parameterized queries?
- Do input validation functions properly encode or sanitize data for the output context?
- How do framework-provided database ORM functions used?
- Does the source code use potentially-dangerous ORM functions? (.raw, etc)
- What output encoding libraries are used?
- Are output encoding libraries up-to-date and patched?
- Is proper output encoding used for the context of each output location?
- Are output encoding routines dependent on regular expressions? Are there any weaknesses or blind-spots in these expressions?

The OWASP XSS Prevention Cheatsheet can be used as a reference for preventing output encoding flaws in web applications.

Cryptographic Review

Overview

Reviewer analyzes code for encryption flaws, including outdated protocols, custom-developed routines, and misuse.

Guidance

This portion of the review checks the use of cryptographic routines and protocols identified during the information gathering phase. Applications that handle credentials or sensitive information (e.g. API tokens, credit card numbers, social security numbers, etc.) must utilize strong cryptographic techniques to mitigate against the possibility that such information will be disclosed to malicious users or open to modification en-route to the server or the user. Even applications that use encryption may be using insecure versions of the protocol or outdated configurations that a malicious user can abuse.

Test Procedures

- What are the standard encryption libraries are used for?
    - Hashing functions - password hashing, cryptographic signing, etc
    - Encryption functions - data storage, communications
- Do the strength of implemented ciphers meet industry standards?
    - Less than 256-bit encryption
    - MD5/SHA1 for password hashing
    - Any RC4 stream ciphers
    - Certificates with less than 1024-bit length keys
    - All SSL protocol versions
- Are cryptographic private keys, passwords, and secrets properly protected?

Configuration Review

Overview

Reviewer analyzes language, framework, and server configuration included in the code for security flaws.

Guidance

The configuration review of application source is highly specific to the language and framework that is targeted. For example, some frameworks allow for enabling/disabling of administrative functionality via the configuration files, so a reviewer will need to ensure that these functions are access limited or require authentication if enabled.

If unsure of or unfamiliar with the configuration options available for a target framework, consult the specific framework version documentation for guides on weaknesses.

Checklist

- Are any endpoints enabled through configurations properly protected with authentication and authorization?
- Are security protections implemented in framework properly configured?
- Does the target language and framework version have any known security issues?
- Are configuration-controlled security headers implemented according to recommended best practices?

Reporting

The reviewer is responsible for documenting findings in a detailed report. This detailed report should include enough information for a developer to identify and remediate each identified issue. Items that may be included:

- Description
- Applicable File, Function, Variables, Line Numbers
- Risk Severity
- Likelihood of Exploitation
- Ease of Exploitation
- Remediation Approach
- References

Additional Resources

Reviewers can reference the following materials for more detailed guidelines:

- OWASP Code Review Guide - Includes some language-specific best practices for Java, .Net, C, C++.
- Simplicable Secure Code Review Checklist
- Infosec Institute - Secure Code Review: A Practical Approach
- OWASP Input Validation Cheatsheet
- OWASP XSS Prevention Cheatsheet
- Wikipedia - Principle of Layered Security
- Wikipedia - Principle of Least Privilege