Auditing a Web Application

Brad Ruppert

Objectives

- Define why application vulnerabilities exist
- Address Auditing Approach
- Discuss Information Interfaces
- Walk through Auditing Process
- Identify High Risk Vulnerabilities
- Outline Mitigating Solutions

Background

- Applications provide the interface between our most sensitive assets:
  - our data and
  - our users
- Proper controls on information flow are of utmost importance
- Having a web presence is no longer "nice to have;" it's a requirement of most businesses

Applications provide the interface between our most sensitive assets, our data, and our users.

Therefore it is of utmost importance that applications exercise the proper controls when it comes to protecting the confidentiality, integrity, and availability of data.

Root Cause of App Vulnerabilities

- "Security is not my job"
- "I get paid to make it work, not to be secure"
- Businesses are driven by revenue and security often takes a backseat to development
- Lack of security awareness and application security training

Unfortunately developers of most applications are provided objectives based solely on functionality and very little if any security measures are taken into account. After all, if it's not written into their job description, why should it concern them? This in turn produces applications containing multiple vulnerabilities ranging from weaknesses around input validation, error handling, session management, and failure to implement proper access controls. Sometimes it takes an exploited vulnerability resulting in a data breach or application defacement for developers and managers to realize the impact of having these weaknesses in their application. According to the SANS (SysAdmin, Audit, Network, Security) Institute Top 20 Internet Security Attack Targets, "every week hundreds of vulnerabilities are being reported in… web applications, and are being actively exploited. The number of attempted attacks every day for some of the large web hosting farms range from hundreds of thousands to even millions."

Auditing Approach

- Work with the developers to enhance education, rather than fault finding
- Be sensitive to discovering vulnerabilities; this is their baby
- Work with developers, QA, Configuration Management, and Project Managers to ensure everyone has their eyes on security and takes time to review it

Where to Start

- Begin by collecting high level data flow diagrams
- Focus on four primary interfaces
  - User ← → Web
  - Web ← → App
  - App ← → App
  - App ← → Database
- Trace the data down to specific functions that request input or return output and document each function's name, file location, input/output arguments

3 Most Important Aspects

- Data Validation
- Data Validation
- Data Validation

Data validation

Data validation is typically the number one cause for application errors or vulnerabilities.

Proper controls need to be put in place to ensure the data received is what's expected and the data returned to the users is of an expected output as well.

Taking the functions identified in "where to start", examine each of these functions to ensure proper data validation (and if necessary data scrubbing) takes place.

Particularly ensure the proper data type (numeric, alpha, or both) is being checked, proper length (min, max, or exact) is specified, if data is required or not, default value is specified (if applicable), and that functions to clean the data are written.

Along with these high level recommendations, below is a checklist of best practice data validation.

Data Validation

- Data Validation is typically the number one cause for application errors or vulnerabilities
- Ensure the data received is what is expected and the data returned to the users is of an expected output
- Proper data type (numeric, alpha, or both)
- Proper length proper length (min, max, or exact)
- Data is required or not
- Default value is specified (if applicable)

Perform Input Validation for All Inputs (prevent “buffer overflow” errors)
Ensure asynchronous consistency to avoid timing and sequence errors, race conditions, deadlocks, order dependencies, synchronization errors, etc.
Use Commit-or-Rollback semantics for exception handling
Use multitasking and multithreading safely
Set initial values for data
Avoid or eliminate “backdoors”
Avoid or eliminate shell escapes
Validate configuration files before use
Validate command-line parameters before use
Validate environment variables before use
Ensure communication connection queues cannot be exhausted

Data Protection

- Protecting sensitive information within the context of the application
- Many applications store and/or use sensitive information, and proper controls must be in place to protect this data
- Examine all functions requiring user input or providing user output to ensure proper data protection for any sensitive information being utilized
- Best Practice
  - Resource connection strings must be encrypted
  - Sensitive data should never be in code
  - Cookies containing sensitive data should be encrypted
  - Sensitive data should not be passed from page to page on client side

Data protection

Data protection addresses protecting sensitive information within the context of the application.

Many applications store and/or use sensitive information, and proper controls must be in place to protect this data.

Taking the functions identified in “where to start” above, examine each of these functions to ensure proper data protection for any sensitive information being utilized.

Sensitive data such as resource connection strings (e.g., database) must be encrypted both in store and in transmission over public networks

Sensitive or secret data values must never be used in code.

Applications must include a “do not remember me” option to allow no session data to be stored on the client.

All cookies containing sensitive or session-based information must be encrypted or transmitted over an encrypted channel.

Sensitive data must not be passed from page to page in client side processing.

Pages containing sensitive data must only retrieve the sensitive data on demand when it is needed instead of persisting or caching it in memory.

Error Handling

- A big component in application security often overlooked
- Examine the error handling of all functions to ensure the errors are sanitized to the point of providing "need to know" information back to the user
- Production applications should not provide the same error messages as used in development
- Best Practice
  - Fail safe; do not fail open
  - Error logs be synchronized with a time server
  - Audit logs are legally protected; so protect them
  - Reports and search logs should be read-only
  - Applications should not run with administrator privileges so log files cannot be manipulated

Error Handling

Error handling is another big component to ensuring the stability of the application.

Take the functions identified in "Where to Start" and ensure that proper error handling is written to handle exceptions.

For any functions that return output to the user, ensure all error messages provide the user with the minimal amount of information needed to correct the error.

Never display any errors to the user that would reveal information about the system or application itself.

Ensure that your application has a "safe mode" which it can return if something truly unexpected occurs. If all else fails, log the user out and close the browser window. Production code should not be capable of producing debug messages.

Assign log files the highest security protection, providing reassurance that you always have an effective 'black box' recorder if things go wrong.

Simply be aware of different attacks. Take every security violation seriously, always get to the bottom of the cause event log errors and don't just dismiss errors unless you can be completely sure that you know it to be a technical problem.

Communication Security

- All sensitive information transferred between the primary interfaces
  - User ← → Web
  - Web ← → App
  - App ← → App
  - App ← → Database

  should be guarded against disclosure or alteration
- Typically this is handled by encryption as well as authentication
- Best Practice
  - Ensure sensitive or personal data is never be passed in clear text through the URL String
  - Ensure message freshness; even a valid message may present a danger if utilized in a "replay attack"
  - Protect message integrity: Can I trust the caller? Did the caller create the message?
  - Protect message confidentiality
  - Calls to the database should use parameterized stored procedures

Communication Security

Any time sensitive information is transported between interfaces as identified in "Where to Start", it should be protected from disclosure or alteration.

This is typically handled by some means of encryption as well as authentication.

Work with the developer to trace the data flow from the functions identified in "Where to Start" to ensure that proper encryption is taking place.

Authentication

- Authentication is establishing or confirming something (or someone) is who they claim to be
- Authentication of an individual should be validated prior to disclosing or providing any information specific to that individual
- Determine all areas where the user interface changes between public and privately visible information
- Best Practice
  - Re-authenticate user for high value transactions and access to protected areas
  - Authenticate the transaction, not the user
  - With form-based authentication, use a trusted access control component rather than writing your own
  - Use strong authentication (tokens, certificates, etc)

Authentication

Authentication is the act of establishing or confirming something (or someone) is who they claim to be.

Authentication of an individual should be validated prior to disclosing or providing any information specific to that individual.

Examine the interfaces identified in the "Where to Start" section above and determine all areas where the user interface changes between public and privately visible information. Work with the developer to determine when accountability needs to be ascertained and ensure all interfaces between these transitions require authentication.

Authorization

- Authorization is the process used to decide if person or program is allowed to have access to data, functionality or a service
- Once the application is certain a person is who they claim to be (authenticated), now we need to decide what permissions or visibility this user has access to
- Following up on the interfaces identified during the "Authentication" stage, now you want to examine the level of visibility or control the user will have and to what areas of the application
- Best Practice
  - Principle of Least Privilege
  - Only restricted and authorized users must have access to administrative interfaces used to manage site content and configuration
  - Always start Access Control Lists using "deny all" and then adding only those roles and privileges necessary

Authorization

Authorization is the process used to decide if person or program is allowed to have access to data, functionality or a service.

This is usually the follow-up step to authentication. Once the application is certain a person is who they claim to be (authenticated), now we need to decide what permissions or visibility this user has access to. Following up on the interfaces identified during the "Authentication" stage, now you want to examine the level of visibility or control the user will have and to what areas of the application. For larger applications, typically roles (or groups) are defined and users are assigned to a specific role within the application.

Example roles could be: Administrator, HelpDesk, Customer, Reporting Analyst, etc.

Work with the developer to identify the roles specific to the application you are auditing and gather the expected authorization (permissions, visibility, access) given to each of these roles.

Then follow the transaction flow after the authentication process to examine how authorization is determined.

Session Management

- Session management follows authentication and authorization
- Includes all environment variables specific to that user and components used to maintain the user's state
- Sessions should be user specific and time specific as well
- Best Practice
  - Authorization and role data should be stored on server side only
  - Session identifiers should be as strong as possible
  - Query strings should not contain session identifiers that represent authenticated users
  - A session timeout length should be defined

Session Management

After the user has been authenticated and authorized within the application, it will be important to manage the user's session.

This will include all environment variables specific to that user and components used to maintain the user's state.

Therefore if the user has already logged in, they should not have to perform this function again during their session.

Sessions should be user specific and time specific as well.

Work with the developer to ensure that proper session security concerns are addressed by following the process flow after the authorization component.

Make note of how sessions maintain uniqueness. What differentiates one session from another?

Determine how session identifiers are created, stored, carried throughout the application, and destroyed.

Best Practice

Form data should not contain hidden fields – if it is hidden, it probably needs to be protected and only available on the server side. However, hidden fields can (and should) be used for sequence protection and to prevent brute force pharming attacks

Session Identifiers:

A strong session identifier will have all (or most) of the following characteristics:

Very hard to guess

Length long enough to prevent brute force attacks

Not related to any user information

Expires on a pre-set date and cannot be reused/replayed

Always transmitted over a secure path (SSL).

Summary

- Auditing an application can seem like a daunting task, but breaking it up piece by piece makes it achievable
- Identify interfaces and protect them first
- Proper data validation is a huge step to securing an application
- Work with developers to provide better coding practices