ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

The Design Issues in Web Services and Security with Service-Oriented Architecture

[I]R. Kamal Krishnan, [II]Dr. R. Selvam, [III]Dr. R. Prabakaran
[I]Guest Lecturer, [II]Assistant Professor, [III]Professor
[I,II]Dept. of Computer Science, Sri Subramanyaswamy Government Arts College, Tiruttani, Thiruvallur District, Tamil Nadu, India
[III]Dept. of Computer Applications, Arignar Anna Institute of Management Studies and Computer Applications, Pennalur, Kancheepuram District, Tamil Nadu, India

Abstract
A Web service is a software application that can be accessed remotely using XML-based languages. It represents a communication interface offered by the server, through that the clients may require different information. Designing a Web service with security in mind presents developers and architects with an interesting set of challenges. Some are unique to service-oriented architecture and some are similar to the challenges that face enterprise Web application development teams. In this analysed various guideline like Authentication, Authorization, Configuration Management, Exception Management, Message Protection, Message Validation, Message Validation, Sensitive Data, Session Management before deploying the Web service and each of these guidelines is briefly described.

Key words
Components, schemas, client, gateway, effective, Validation

I. Introduction

Web services security requirements also involve credential mediation, and service capabilities and constraints. Web Services have to be suitable for secure communication A Web service is most commonly implemented as a wrapper – that is, as an interface between clients consuming the service and back-end business logic components doing the actual work. A Web service acts as a trust boundary in the application architecture. By its nature, a Web service acts as a gateway between trusted business components and less trusted or un-trusted client components. For this reason, it is impossible to think about the security of a Web service without also thinking about authentication, authorization, protection of sensitive data on the network, and handling potentially malicious input. Each of these areas represents key decisions, will need to make in order to maintain the security of the application. By following security best practices in the design of Web service, can use proven practices to improve the decision-making capabilities and make a cascading positive impact on the overall security of the application. Use the following design guidelines to reduce wasted effort trying to solve security problems for which there are already best practices in place to improve the security of service. Security is a path, not a destination. As analyze of infrastructure and applications, identify potential threats and understand that each threat presents a degree of risk. Security is about risk management and implementing effective countermeasures. One of the most important concepts in security is that effective security is a combination of people, process, and technology.

II. Service Oriented Architecture

Service Oriented Architecture (SOA) is architecture of loosely coupled components that can be distributed across platform, technology and physical topology. Service components can be combined to provide a business process or provide more complex services for a client application. The key attributes of service oriented architecture are:

- **Interoperable.** Components can be interoperable across platform and technology boundaries.
- **Componentized.** Services are exposed as autonomous components that can be versioned and managed independently.
- **Composable.** Services can be composed by an application to perform more complex operations.
- **Message-based interfaces.** Interfaces are defined by message contracts and schemas.
- **Distributable.** Service components can be consumed from the same machine or can be distributed to remote machines.
- **Discoverable.** Services publish their metadata as WSDL so that client applications can discover the interfaces, schemas, and generate a client-side proxy to consume the service.

Services are the preferred communication technique to use across application boundaries, including platform, deployment, and trust boundaries. In this table describes some comparison of service orientation and object orientation.

Table 1:

Object Orientation	Service Orientation
Assumes a homogeneous platform and execution environment.	Assumes a heterogeneous platform and execution environment.
Shares types, not schemas.	Shares schemas, not types.
Assumes cheap, transparent communication.	Assumes variable cost, explicit communication.
Objects are linked: object identity and lifetime are maintained by the infrastructure.	Services are autonomous: security and failure isolation are a must.
Typically requires synchronized deployment of both client and server.	Allows continuous, separate deployment of client and server.

Is easy to conceptualize and thus provides a natural model to follow.	Builds on ideas from component software and distributed objects. Dominant theme is to manage/reduce sharing between services.
Provides no explicit guidelines for state management and ownership.	Owns and maintains state or uses the reference state.
Assumes a predictable sequence, timeframe, and outcome of invocations.	Assumes message-oriented, potentially asynchronous, and long-running communications.
Goal is to transparently use functions and types remotely.	Goal is to provide inter-service isolation and wire interoperability based on standards.

III. Security Architecture and Design Issues for Web Services

During the design phase, it is important to think like an attacker and consider potential vulnerabilities that can impact of service. A clear understanding of attacks and vulnerabilities will put in the right mindset to mitigate potential problems and create a design that is resistant to malicious attack. The following table outlines key problem areas for each category in the Web service security frame.

Table 2 :

Vulnerability category	Potential problem due to bad design
Auditing and logging	Failure to observe signs of intrusion; Inability to prove a user's actions; Difficulties in problem diagnosis
Authentication	Identity spoofing; Password cracking; Elevation of privileges; Unauthorized access
Authorization	Access to confidential or restricted data; Tampering; Execution of unauthorized operations
Configuration management	Unauthorized access to administration interfaces; Unauthorized ability to update configuration data; Unauthorized access to user accounts and account profiles
Exception management	Denial of service (DoS) attacks; Disclosure of sensitive system level details; Elevation of privilege.
Message encryption	Sniffing of confidential data off the network; Stealing users' credentials or session information.
Message replay detection	Replaying user messages to gain unauthorized access to resources or data
Message signing	Tampering with messages on the network without detection. Failure to mutually authenticate allows attacker to send messages as if they were a legitimate user.
Message validation	Messages containing malicious input.; Cross-site scripting or SQL injection attacks on the service or clients that rely on the service.
Sensitive data	Confidential information disclosure and data tampering.
Session management	Session hijacking and/or identity spoofing due to Capture of session ID.

A. Deployment Considerations

During the application design phase, should review corporate security policies and procedures together with the infrastructure on which the application is to be deployed. Frequently, the target environment is rigid, and the application design must reflect its restrictions

- **Identify security policies and procedures**. A security policy determines what applications are allowed to do and what the users of the application are permitted to do.
- **Understand network infrastructure components**. Make sure understand the network structure provided by the target environment, as well as the baseline security requirements of the network in terms of filtering rules, port restrictions, supported protocols, and so on.
- **Identify how firewalls and firewall policies are likely to affect application's design and deployment**. If present, firewalls separating the Internet-facing applications from the internal network, as well as additional firewalls in front of the database, can affect of possible communication ports and consequently authentication options from the Web server to remote application and database servers.
- **Identify protocols, ports, and services**. At the design stage, consider what protocols, ports, and services are allowed to access internal resources from the Web servers in the perimeter network.
- **Communicate assumptions**. Communicate and record any assumptions made about network and application-layer security and which component will handle what task.
- **Analyze deployment topologies**. Application's deployment topology, and whether a remote application tier have, are key considerations that must be incorporated into of design.
- **Consider identity flow**. Also consider identity flow and identify the accounts that will be used for network authentication when the application connects to remote servers.
- **Understand intranet, extranet, and Internet considerations**. Intranet, extranet, and Internet application scenarios each present design challenges. Questions that should consider include:

* How will flow caller identity through multiple application tiers to back-end resources?
* Where will perform authentication?
* Can trust authentication at the front end and then use a trusted connection to access back-end resources?

B. Auditing And Logging

Auditing and logging are used to monitor and record important activities, such as transactions or user management events, on both the client and the service. Ensure that logging design allows for the effective auditing of security-critical operations such as

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

user management events or important business operations such as financial transactions.

Consider the following guidelines:

Audit and Log Access Across Application Tiers-Audit and log access across the tiers of the application for the purpose of non-repudiation. Use a combination of application-level logging and platform auditing features.

Back Up and Analyze Log Files Regularly - There is no point in logging activity if the log files are never analyzed. Log files should be removed from production servers on a regular basis. The frequency of removal depends on the application's level of activity.

Consider Identity Flow - Consider how the application will flow caller identity across multiple application tiers. Here two basic choices:

- Flow the caller's identity at the operating system level by using the Kerberos protocol delegation. This allows to the use operating system–level auditing. The drawback to this approach is that it affects scalability because it means there can be no effective database connection pooling at the middle tier.
- Alternatively, it can flow the caller's identity at the application level and use trusted identities to access back-end resources. With this approach, they have to trust the middle tier, which brings a potential repudiation risk. This should generate audit trails in the middle tier that can be correlated with back-end audit trails.

Do Not Log Sensitive Information - Do not include sensitive information in the log entries. The access rights for the log files may be different than the access rights for sensitive operations and data in the service.

Instrument for Significant Business Operations - Track significant business operations. For example, instrument the application to record access to particularly sensitive methods and business logic.

Instrument for Unusual Activity - Instrument the application and monitor events that might indicate unusual or suspicious activity. This enables them to detect and react to potential problems as early as possible.

Instrument for User Management Events - Instrument the application and monitor user management events such as password resets, password changes, account lockout, user registration, and authentication events.

Know The Baseline - Before deploying the application, audit the log files so what normal application behavior looks like.

Log Key Events - The types of events that should be logged include successful and failed logon attempts, modification of data, retrieval of data, network communications, and administrative functions such as the enabling or disabling of logging.

Protect and Audit Log Files -Protect and audit and log files using Windows access control lists (ACLs), and restrict access to the log files. If the log events to Microsoft SQL Server® or to some custom event sink, use appropriate access controls to limit access to the event data.

Additional Considerations - Also keep in mind the following additional considerations:

- Log application events on a separate, protected server.
- Assign appropriate permissions to the log files.
- Log application events in sufficient detail.
- Use performance counters for high-volume, per-request events.
- Use Log Throttling

C. Authentication

Authentication is the mechanism by which the clients can establish their identity with service using a set of credentials that prove that identity. Protect the user's credentials when they are sent over the network, as well as when they are stored on the client or the server.

Be Able to Disable Accounts

If the system is compromised, being able to deliberately invalidate credentials or disable accounts can prevent additional attacks.

Do Not Send Passwords over the Wire in Plaintext

Plaintext passwords sent over a network are vulnerable to eavesdropping. To address this threat, secure the communication channel; for example, by using Secure Sockets layer (SSL) to encrypt the traffic.

Do Not Store Passwords in User Stores

If verify passwords, it is not necessary to actually store the passwords. Instead, store a one-way hash value and then recomputed the hash using the user-supplied passwords. To mitigate the threat of dictionary attacks against the user store, use strong passwords and incorporate a random salt value with the password.

Protect Authentication Cookies

A stolen authentication cookie is a stolen logon. Protect authentication tickets using encryption and secure communication channels.

Require Strong Passwords

Do not make it easy for attackers to crack passwords. There are many guidelines available, but a general practice is to require a minimum of eight characters and a mixture of uppercase and lowercase characters, numbers, and special characters.

Support Password Expiration Periods

Passwords should not be static and should be changed as part of routine password maintenance through password expiration periods. Consider providing this type of facility during application design.

Use Account Lockout Policies for End-user Accounts

Disable end-user accounts or write events to a log after a set number of failed logon attempts.

D. Authorization

Authorization is the mechanism by which the control the operations and resources an authenticated client can access. Where possible, authenticate the users on the same application tier where authorize the users. Run the application in a least-privileged account and use impersonation to increase privileges only when necessary and for the shortest time possible.

Tie Authentication to Authorization on the Same Tier

Where possible, authenticate the users on the same application tier where authorize the users. The further to separate the time of check (authentication) from the time of use (authorization), the larger window of opportunity the give an attacker to subvert the authorization mechanism.

Consider Authorization Granularity

There are three common authorization models, each with varying degrees of granularity and scalability:

- **The most granular approach relies on impersonation.** Resource access occurs using the security context of the caller. Windows ACLs on the secured resources

determine whether the caller is allowed to access the resource. If the application provides access primarily to user-specific resources, this approach may be valid.

- **The least granular but most scalable approach uses the application's process identity for resource access**. This model is referred to as the trusted subsystem or sometimes as the trusted server model. Although this approach supports database connection pooling, it means that the permissions granted to the application's identity in the database are common, irrespective of the identity of the original caller.
- **The third option is to use a limited set of identities for resource access based on the role membership of the caller**. This is really a hybrid of the two models described earlier.

Know The Authorization Options

Know the authorization options and choose the most appropriate one for the scenario. First decide if they want to use resource-based or role-based authorization. Resource-based authorization uses ACLs on the resource to authorize the original caller. Role-based authorization allows them to authorize access to service operations or resources based on the group a user is in.

Restrict User Access to System-level Resources

System-level resources include files, folders, registry keys, Active Directory objects, database objects, event logs, and so on. Use ACLs to restrict which users can access what resources and the types of operations that they can perform.

Use Least-privileged Accounts

In this might need to create a custom service account to isolate the application from other applications on the same server, or to be able to audit each application separately.

Use Multiple Gatekeepers

On the server side, the can use IP Security Protocol (IPSec) policies to provide host restrictions to restrict server-to-server communication. For example, an IPSec policy might restrict any host apart from a nominated Web server from connecting to a database server.

E. Configuration Management

Security settings, authentication, authorization, logging, and other parameters can be set in configuration files. Encrypt configuration sections that contain sensitive data such as connection strings to the SQL database. Protect access to the configuration settings so that an attacker cannot modify security settings for the service. Consider the following guidelines:

Consider The Key Storage Location

If it need to store keys, choose platform features over rolling the own mechanism. The Data Protection API (DPAPI)– and RSA-protected configuration providers used to encrypt sensitive data in configuration files can use either machine stores or user stores for key storage.

Encrypt Sensitive Sections of Configuration Files

Configuration files may contain sensitive information, such as connection strings to the database. Encrypt sensitive information in the configuration files using the DPAPI provider with the machine-key store. This can use the aspnet_regiis command-line tool to encrypt sections of the configuration file.

Use ACLs to Protect The Configuration Files

Use ACLs to lock the configuration files down and restrict inappropriate access. Modifications to the configuration settings, especially binding options, can have a major impact on the security of the service.

Use Secure Settings for Various Operations of Web Services

Set the configuration options to take advantage of features such as message and transport security, which protect the communication channel between the client and the service.

F. Exception Management

Exception management is the means by which the expose and consume exception information within the service and send it back to the clients. Be careful not to reveal internal application details to the clients as this information could assist an attacker trying to exploit the service. Catch and handle exceptions so that error conditions do not lead to a service crash and a DoS condition for the clients. Fail to a secure state so that an error condition does not result in the application running at higher privilege or accessing resources insecurely. Consider the following guidelines:

Catch Exceptions

Use structured exception handling and catch exception conditions with try/catch blocks. Doing so avoids leaving the application in an inconsistent state that may lead to information disclosure.

Do Not Log Private Data Such as Passwords

Exception handlers often will result in an error log entry. Be careful not to log sensitive information such as passwords, credit card numbers, or privately identifiable information (PII).

Do Not Reveal Sensitive System or Application Information

In the event of a failure, do not expose information that could lead to information disclosure.

Log Detailed Error Messages

Send detailed error messages to the error log. Send minimal information to the consumer of the service or application, such as a generic error message and custom error log ID that subsequently can be mapped to a detailed message in the event logs. Make sure that do not log passwords or other sensitive data.

G. Message Protection

Message protection covers the mechanisms used to protect sensitive data in-transport over the network from unauthorized access or modification. Use message or transport security to protect the messages in transit. Consider the following guidelines:

Use Message Security or Transport Security to Encrypt and Sign The Messages

Use message security or transport security to encrypt the messages on the network. Message security encrypts each individual message to protect sensitive data.

Use Platform-Provided Cryptography

Cryptography is notoriously difficult to develop. The Windows crypto APIs have been proven to be effective. These APIs are implementations of algorithms derived from years of academic research and study.

Use Platform Features for Key Management

Use platform features where possible to avoid managing keys them self

Periodically Change The Keys

In this should change the encryption keys from time to time because a static secret is more likely to be discovered over time.

H. Message Validation

Message validation is used to protect the service from malformed messages & message parameters. Message schemas can be used to validate incoming messages, and custom validators can be used to validate parameter data before the service consumes it. Consider

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

the following guidelines:

Do Not Trust Input

An attacker passing malicious input can attempt SQL injection, cross-site scripting, and other injection attacks that aim to exploit the application's vulnerabilities. Check for known good data and constrain input by validating it for type, length, format, and range.

Verify the Message Payload Against a Schema

If the need to validate parameters, message contracts, or data contracts passed to operations, use schemas to validate the incoming message. Schemas provide a wide range of input validation without the need for custom code or validation routines.

Verify the Message Size, Content, and Character Sets

Validate incoming messages to ensure that they match the expectations regarding size, content, and character encoding.

Filter, Scrub, and Reject Input and Output Before Additional Processing

Filter and reject input before allowing the data to be processed by downstream components. Because malicious input may target the routines that process the input, it is important to detect and reject malformed input early before additional processing occurs.

I. Sensitive Data

Sensitive data refers to confidential information that the service processes, transmits, or stores. Protect sensitive data on the network, in configuration files, in local memory or file storage, and in databases and log files. Ensure that aware of all sensitive information the service transmits or processes. Sensitive data includes user identity and credentials as well as any personally identifiable information (PII) such as social security number. Consider the following guidelines:

- Do not store database connections, passwords, or keys in plaintext.
- Do not store secrets if can avoid it.
- Do not store secrets in code.
- Encrypt sensitive data in configuration files.
- Encrypt sensitive data over the network.
- Retrieve sensitive data on demand.

J. Session Management

Sessions are the means by which an application maintains state full communication with a client over time. Protect the session tokens or identifiers so that an attacker cannot gain access and steal a user's session. Reduce the timeouts on the sessions to lower the chances of an attacker being able to steal a session after a user has finished using the application. Consider the following guidelines:

- Authenticate & authorize access to the session store.
- Avoid storing sensitive data in session stores.
- Reduce session timeouts.
- Secure the channel to the session store.

IV. Conclusion

The Web services describes a standardized way of integrating Web-based applications using the XML, SOAP, WSDL and UDDI open standards over an Internet protocol backbone. It allow organizations to communicate data without intimate knowledge of each other's IT systems behind the firewall. In this discussed various guidelines like Authentication, Authorization, Configuration Management, Exception Management, Message Protection, Message Validation, Message Validation, Sensitive Data, and Session Management before deploying the Web service and each of these guidelines is briefly described. Security is about risk management & implementing effective countermeasures. One of the most important concepts in security is that effective security is a combination of people, process, and technology.

References

[1]. Leonard Richardson, Sam Ruby Web services for the real world
[2]. Ralph Moseley, M.T. Savaliya: Developing Web Applications
[3]. Bates: Web Programming Building Internet Applications
[4]. Chris Bates:Web Programming: Building Internet Applications The Dafydd Stuttard Web Application Hacker's Handbook: Finding and Exploiting Security Flaws
[5]. Bryan Sullivan: Web Application Security, A Beginner's Guid
[6]. http://www.infosec.gov.hk/english/technical/files/webss.pdf
[7]. https://msdn.microsoft.com/en-us/library/ff649737.aspx
[8]. https://wcfsecurityguide.codeplex.com/
*[9]. K.-J. Lin; J.-Y. Chung: Service Oriented Computing and Application*s -Web Services