Developing RESTful Services

Chapter2:

Securing Restful Web Services

Chapter Objectives

In this chapter, we will:

- Discuss techniques for securing RESTful web services
- Examine session management practices
- Manage authentication and authorization
- Explore the details of OAuth for authorization

Chapter Concepts

Session Management

- RESTful services should use session-based authentication
  - Establish a session token via `POST`
  - Or use an API key as a `POST` body argument
- Critical information should not appear in the URL
  - Username
  - Password
  - Session token
  - API key

For more information about RESTful security see the following:
https://www.owasp.org/index.php/REST_Security_Cheat_Sheet

Protect Session State

- RESTful services are designed to be stateless
- Use only a session token or API key
  - To maintain client state in a server-side cache
- To prevent replay exploits
  - Use a time limited key or token

Chapter Concepts

Session Management

Authentication and Authorization

Securing RESTful Web Services

Chapter Summary

RESTful Authentication

- Several techniques to accomplish this
    - Basic authentication
    - HMAC
    - OAuth

Basic Authentication

- The simplest way to manage authentication
  - Use HTTP basic authentication
- Username and password are passed in the HTTP header
  - But **not** encrypted
- Use only with SSL/TLS

HMAC Authentication

- Hash-based Message Authentication (HMAC)
- Sends a hashed version of the password
  - With some other information
    - Such as a timestamp, a nonce, or the md5 hash of the message body
    - To help prevent (or at least detect) any tampering of the message body
- Nonce
  - A number that is used only once
  - To prevent replay attacks

OAuth

- “An open protocol to allow secure authorization in a simple and standard method from web, mobile, and desktop applications.”[1]
- Enables third-party applications to obtain limited access to a web service
  - Known as “secure designated access”
- Example scenario:
  - You can grant ESPN.com the right to access your Facebook profile
  - Without knowing your Facebook password
- OAuth does not share passwords
  - Instead it uses authorization tokens
  - Tokens prove an identity between consumers and service providers

1. https://oauth.net/

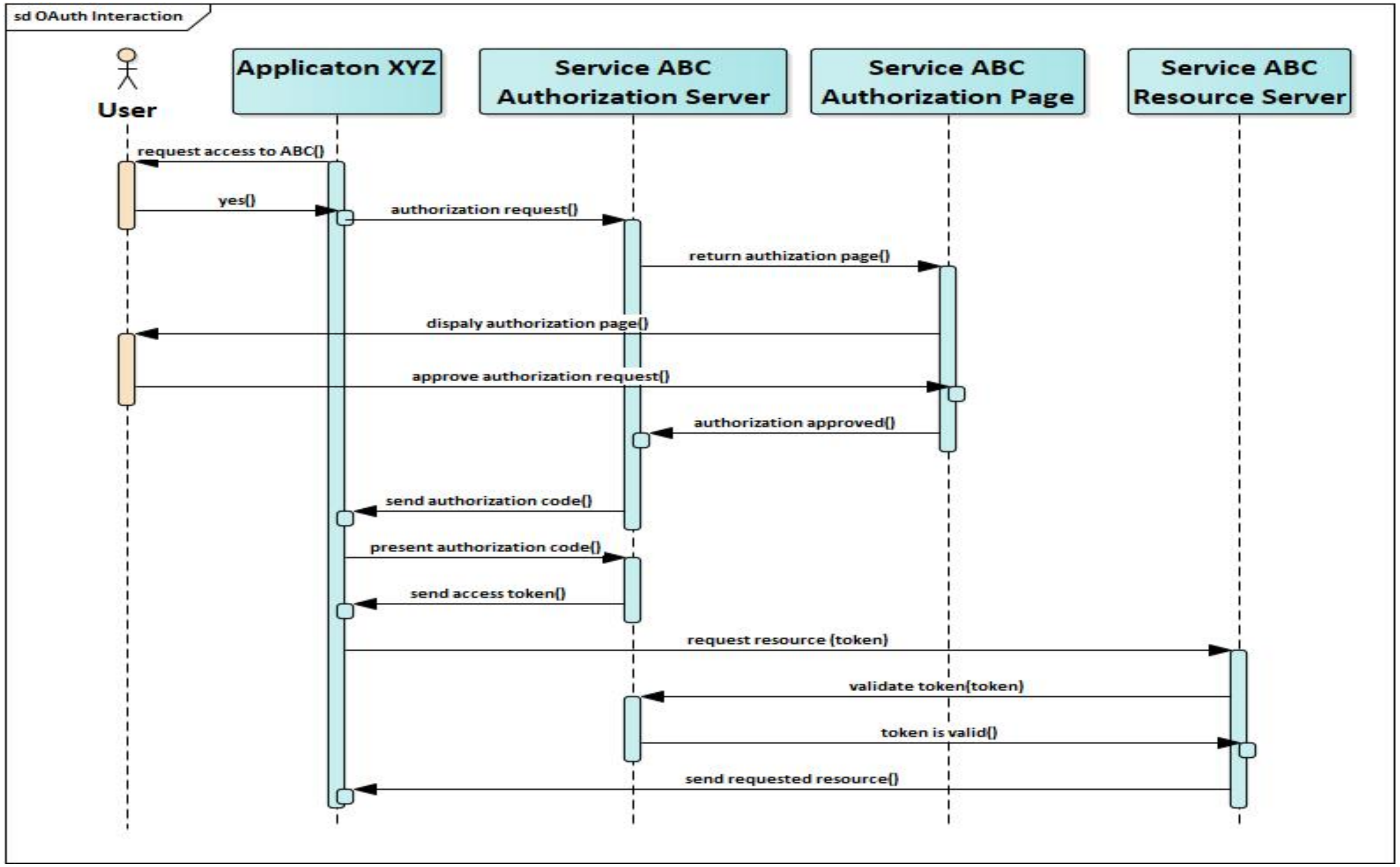
Application Registration

- Before using OAuth, the application must register with the service
  - Using the “developer (API)” portion of the service website
- The following information must be provided:
  - Application name
  - Application website
  - Redirect URI
    - The service will redirect the user to this location after authorization
    - This is the part of your application that will handle authorization codes
- Once the application is registered, the service issues client credentials
  - Client identifier
    - Publicly known string that identifies the application
  - Client secret
    - Used to authenticate the identity of the application to the service API

Authorization Grant

- There are four types of grant types supported by OAuth2
- Authorization Code
  - Used with server-side application
  - Most commonly used
- Implicit
  - Used with mobile applications or web applications that run on the user's device
- Resource Owner Password Credentials
  - Used with trusted applications
  - Like applications owned by the service
- Client Credentials
  - Used with applications API access

OAuth Authorization Code Flow


sd OAuth Interaction
User
Applicaton XYZ
Service ABC Authorization Server
Service ABC Authorization Page
Service ABC Resource Server
request access to ABC()
yes()
authorization request()
return authization page()
dispaly authorization page()
approve authorization request()
authorization approved()
send authorization code()
present authorization code()
send access token()
request resource (token)
validate token(token)
token is valid()
send requested resource()

This diagram is based on RFC 6749, 4.1 https://tools.ietf.org/html/rfc6749#section-4.1

Stepping Through the Authorization Code Flow

1. Application XYZ asks the user for access to resources provided by Service ABC.
    a. The user agrees.
2. Application XYZ sends an authorization request to Service ABC Authorization Server.
    a. Using its authorization endpoint.
3. Service ABC returns an authorization page.
4. The authorization page is displayed to the user.
5. The user approves the requested permissions.
    a. Authorization page form is submitted to the Service ABC Authorization Server.
6. Service ABC Authorization Server issues a short-lived authorization code.
    a. Sent to application XYZ.

Stepping Through the Authorization Code Flow (continued)

7. Application XYZ presents the authorization code to Service ABC Authorization Server.
   a. Sent to Service ABC Authorization Server token endpoint.
8. Service ABC Authorization Server issues an access token for application XYZ.
9. Application XYZ requests the desired resource from Service ABC Resource Server.
   a. Presents the access token.
10. Service ABC Resource Server requests verification of access token.
    a. Sends access token to Service ABC Authorization Server.
11. Service ABC Authorization Server returns confirmation of the access token.
12. Service ABC Resource Server returns the requested resource to application XYZ.

Chapter Concepts

Session Management

Authentication and Authorization

➤ **Securing RESTful Web Services**

Chapter Summary

RESTful Web Security

- The following recommendations are based on the OWASP REST Security Cheat Sheet:
  - HTTPS
  - Security tokens
  - API keys
  - Restrict HTTP methods
  - Input validation
  - Security headers
  - HTTP return codes

For more information about RESTful security see the following:
https://www.owasp.org/index.php/REST_Security_Cheat_Sheet

HTTPS

- Secure RESTful services should provide only HTTPS endpoints
- This protects all data transferred between the client and the service
- For highly sensitive or privileged services, consider using client-side certificates

Security Tokens

- RESTful services can require a security token from the client
  - The token can be used for access control decisions
- JSON Web Tokens are often the preferred format for security tokens
  - https://tools.ietf.org/pdf/rfc7519.pdf
  - A cryptographic signature is the recommended way to protect the integrity of the security token
- Contents of a JSON Web Token:
  - Issuer – is this a trusted issuer?
  - Audience – is the relying party in the target audience for this token?
  - Expiration time – is the token still valid?
  - Not before time – is the token valid yet?

JSON Web Tokens (JWT)

- When to use JSON Web Tokens?
  - Authorization
    - When the user logs in, the JWT will be generated and returned
    - Each subsequent request includes the JWT
  - Information exchange
    - Transmits information securely
    - Signing the JWT with a private key assures the receiver who the sender is
    - Also verifies the content has not been modified
- The contents (header, payload) of a JWT can be decoded and read by anyone
- Encrypt sensitive data in the payload

JSON Web Token Structure

- JSON Web Tokens contain three parts separated by dots (.)
- Header
  - The type of the token (JWT)
  - The signing algorithm (HMAC, SHA256, RSA, …)
- Payload
  - Contains claims about an entity (usually the user)
- Signature
  - Calculated from the encoded header, encoded payload, and a secret
  - Using a specified algorithm

Example JWT

- Header
  - { "alg": "HS256", "typ": "JWT" }
- Payload
  - { "sub": "1234567890", "name": "John Doe", "iat": 1516239022 }
- Signature
  - HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)
- JWT
  - eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.XbPfbIHMI6arZ3Y922BhjWgQzWXcXNrz0ogtVhfEd2o

API Keys

- API keys can reduce the impact of a denial-of-service attack by doing the following:
  - Require an API key for every request to the service
  - Return a 429 HTTP response code if requests are arriving too quickly
  - Revoke the API key if the client violates the usage agreement
- Caution
  - Do not rely exclusively on API keys to protect sensitive, critical, or high-value resource

Restrict HTTP Methods

- Use a whitelist of permitted HTTP methods
  - Only support those on the whitelist
- Reject all requests not on the whitelist with a 405 HTTP response code of “Method not allowed”
- Verify the client is authorized to request the incoming HTTP method on the service

Protect HTTP Methods

- Make sure the incoming request has rights to execute the requested method
- Not everyone should be able to `DELETE` a resource
- Validate the incoming HTTP method
    - Against the session token or API key

Input Validation

- The usual “don’t trust the client” recommendations apply here
- Validate input parameters
- Validate input length, range, and data type
- Reject unexpected or illegal content
- Log validation failures
  - Assume that anyone performing many requests that fail validation rules is not to be trusted

Security Headers

- Send security headers
  - Always send the Content-Type header with the correct type specified
  - This insures the browser interprets the response correctly
- Send an X-Content-Type-Options: nosniff
  - Make sure the browser does not try to detect a different Content-Type
  - This helps to prevent XSS exploits
- Clients should send an X-Frame-Options: deny
  - Protect against drag and drop clickjacking attacks in older browsers

Protect Against Cross-Site Forgery

- Insure that all `PUT`, `POST`, `PATCH`, and `DELETE` requests are protected
  - From cross-site request forgery
- Use a token-based approach
- See this cheat sheet for more detailed information:
  - https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet

Cross-Site Request Forgery

```
<!-- This is embedded in another domain's site -->
<img src="http://target.site.com/add-
user?user=name&grant=admin">
```

- Common approach is to use the `<img>` tag
- This causes the user browser to send a request to a malicious server
- Mitigation techniques include:
  - Short-lived JWTs
  - Special headers added only when they originate from the correct origin
  - Per session cookies
  - Per request tokens
- If JWTs (and session data) are not stored as cookies, CSRF attacks are not possible

HTTP Return Codes

Status code	Message	Description
200	OK	Response to a successful REST API action
201	Created	The request has been fulfilled and resource created. A URI for the created resource is returned in the Location header.
202	Accepted	The request has been accepted for processing, but processing is not yet complete
204	No Content	The request succeeded, but no data is returned in the response
400	Bad Request	The request is malformed, such as message body format error
401	Unauthorized	Wrong or no authentication ID/password provided
403	Forbidden	It's used when the authentication succeeded but authenticated user doesn't have permission to the request resource
404	Not Found	When a non-existent resource is requested
406	Unacceptable	The client presented an unsupported content type in the Accept header
405	Method Not Allowed	The error for an unexpected HTTP method.
413	Payload too large	Use it to signal that the request size exceeded the given size
415	Unsupported Media Type	The requested content type is not supported by the REST service
429	Too Many Requests	The error is used when there may be DOS attack detected or the request is rejected due to rate limiting

Chapter Concepts

Session Management

Authentication and Authorization

Securing RESTful Web Services

➤ **Chapter Summary**

Chapter Summary

In this chapter, we have:

- Discussed techniques for securing RESTful web services
- Examined session management practices
- Managed authentication and authorization
- Explored the details of OAuth for authorization