Chapter 6 Application Security

# Software Assurance Best Practices

Software Development life cycle (SDLC) describes the steps in a model for software development throughout its life cycle.

Image

Phases of SDLC:

1. Planning
2. Requirements: desired functionality.
3. Design
4. Code
5. Testing
6. Training and Transition
7. Operation and Maintenance
8. Decommissioning

## DevSecOps and DevOps

DevOps combines software development and IT.

DevSecOps adds security in DevOps.

## Continuous Integration and Continuous Deployment

Continuous Integration (CI): Is the practice of consistently checking code into a shared repository.

Continuous Deployment (CD): Roll out tested changes into production automatically as soon as they have been tested.

Image

# Designing and Coding for Security

## Secure Coding Practices

Open Worldwide Application Security Project (OWASP) proactive controls:

* Define Security Requirements
* Leverage Security Requirements
* Secure Database Access
* Encode and Escape Data
* Validate All Inputs
* Implement Digital Identity
* Enforce Access Controls
* Protect Data Everywhere
* Implement Security Logging and Monitoring
* Handle All Errors and Exceptions

## API Security

Relis on authentication, authorization, proper data scoping, rate limiting, input filtering, and monitoring and logging.

# Software Security Testing

## Analyzing and Testing Code

### Static Code Analysis

### Dynamic Code Analysis

### Fuzzing

Sending invalid data to an application to test it.

# Injection Vulnerabilities

Supplying code to a web application to have their servers execute it.

## SQL Injection Attacks

Attackers will also do BLIND SQL INJECTIONS, which is they don’t see their results.

2 forms of blind injections:

* Blind Content-Based SQL Injection: Test whether the application is interpreting injected code before attempting to carry out an attack.
* Blind Timing-Based SQL Injection: Testing if there is and how long there is a delay of time between interpreting the SQL Injection and giving a possible output.

## Code Injection Attacks

Adding code to applications.

## Command Injection Attacks

Reaching the victims operating system and running a command.

# Exploiting Authentication Vulnerabilities

## Password Authentication

Ways a password can be discovered:

* Social Engineering
* Eavesdropping on unencrypted networks.
* Obtaining dumped passwords on sites.

## Session Attacks

Stealing an existing authenticated session.

Figure 6.8

## Cookie Stealing and Manipulation

* Eavesdropping on unencrypted networks.
* Installing malware on a user’s browser.
* On-Path Attacks: When a user is on a look-alike website and accepts cookies, the attacker uses the user’s cookies on the actual website.

Session Replay: figure 6.10

To protect cookies, you must be transmitted over encrypted HTTP connections.

NTLM pass-the-hash attack: Attacker gains access to the operating system to harvest a stored NTLM password hash. Then use the hashes to gain administrator access.

## Unvalidated Redirects

Attackers use insecure websites to redirect you to a malicious attack.

Protect: validated redirects (having a list of websites that are approved redirect sites and limits the number of redirects).

# Exploiting Authorization Vulnerabilities

## Insecure Direct Object Reference

When an application does not do an authorization check and the user has access to information that exceeds authority.

## Directory Traversal

When there is a direct path to files from an application and the files are not redirected to a server elsewhere.

## File Inclusion

It is an attack that includes the execution of code within a file.

2 variants:

* Local file inclusion: file in the webserver.
* Remote file inclusion: stored in a remote server.

Webs shells can be used to do the same thing.

## Privilege Escalation

Attacks that make normal users have more privileges.

# Exploiting Web Application Vulnerabilities

## Cross-Site Scripting (XSS)

HTML Injection

## Reflected XSS

Allows reflected input.

## Stored/Persistent XSS

XSS on the web server that continues without the attacker being present.

## Request Forgery

Exploit the relationship and attempt to have users unwittingly execute commands against a remote server.

2 types of forgery:

* Cross-Site Request Forgery (CSRF): exploits the trust between user and remote sites.
* Server-Side Request Forgery (SSRF): Tricking the server to visit the URL.

# Application Security Controls

## Input Validation

Allow listing: developers describe the exact type of input expected.

Deny Listing: a list of potentially malicious inputs that should be blocked.

## Web Application Firewalls (WAF)

WAF are like firewalls but for applications.

WAF is in front of the web server and all traffic goes through it.

Figure 6.14

## Parameterized Queries

Clients send arguments to the server which are inserted and precompiled into a query template.

## Sandboxing

Is a stored procedure practice of running applications on isolated environment.

## Code Security

* Code Signing: Is a way for developers that the code is authentic.
* Code Reuse
* Software Diversity: To avoid single point failure.
* Code Repositories
  + Version control: tracks all the changes
  + Dead code: code that no one is responsible of managing and possibly not know who created it.
* Integrity Measurement: matching code created to code that was approved using hash functions.
* Application Resilience
  + Scalability
  + Elasticity

# Secure Coding Practices

* Source Code Comments
* Error Handling
* Hard-coded Credentials
  + Developer backdoors
  + Developers create access credentials to other services.
* Package Monitoring
* Memory Management
* Resource Exhaustion
  + Can cause memory leaks.
* Pointer Dereferencing
* Buffer Overflows: Uses memory injection to placing data areas that succeed the allocated memory amount in hopes of overriding the code.
* Race Conditions: when code segments run based on a sequence of events:
  + Time -of-Check (TOC)
  + Time-of-USE (TOU)
  + Target of Evaluation (TOE)
* Unprotected APIs

# Automation and Orchestration

Automation and Scripting Use Cases:

* User provisioning
* Resource provisioning
* Guard rails: enforce policy controls
* Security Groups
* Ticket Creation
* Escalation
* Enabling/disabling services and access
* Continuous integration and testing
* Integrations and APIs

Benefits of Automation and Scripting:

* Time saving and efficient
* Enforcing baselines
* Standardizing infrastructure configuration
* Scaling in a secure manner
* Retaining employees
* Reducing reaction time
* Serving as a workforce multiplier

Other Considerations

* Complexity of Build
* Cost
* Single point of failure
* Technical debt
* Ongoing supportability