The OWASP Top Ten 2021-2022

JIM MANICO   Secure Coding Instructor   *www.manicode.com*

OWASP Top 10 – 2021 Learning Objectives

In this Course you will Learn:

What is OWASP – What are the OWASP Top 10 Risks

For Each of OWASP Top 10 – 2021

Key Concepts and Definition
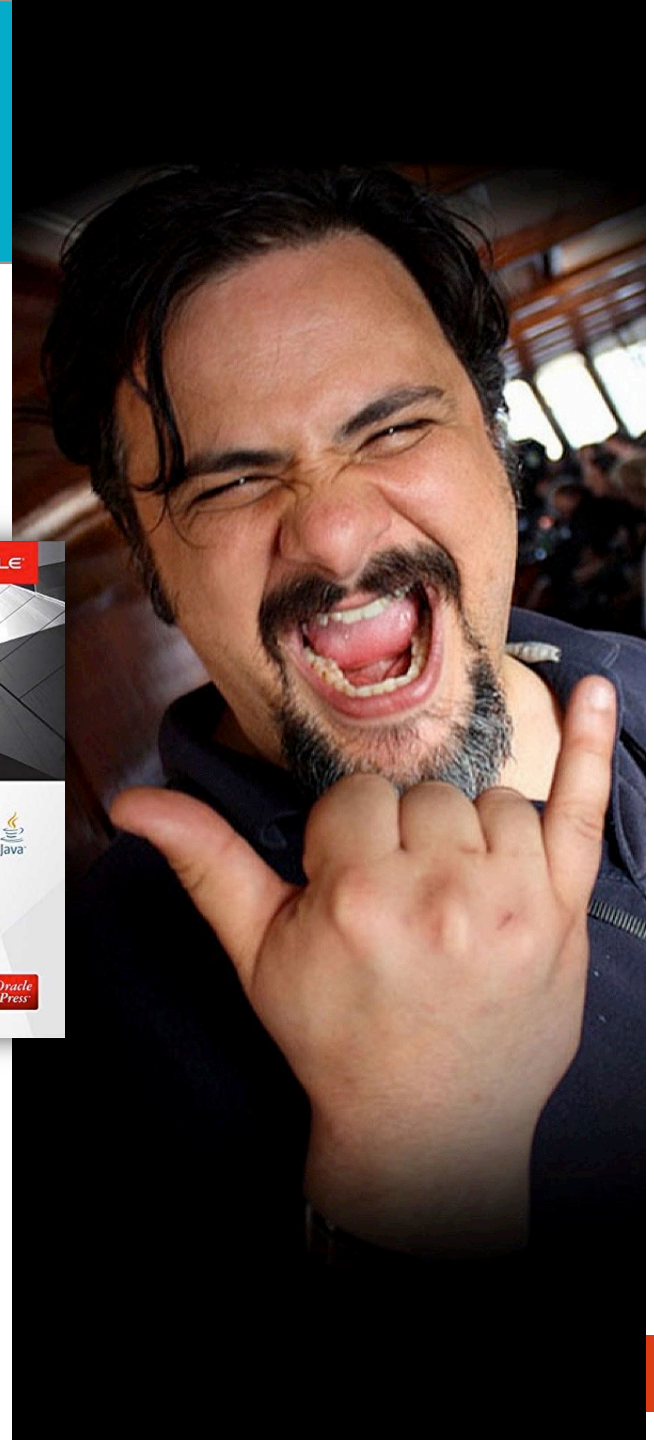
Challenges with this Risk

Examples – Good & Bad Code in Pseudocode

Best Protection Strategies

🐦 @manicode

Former OWASP Global Board Member

- 25+ years of software development experience

- Author – *"Iron-Clad Java, Building Secure Web Applications"*
  - McGraw-Hill/Oracle-Press

- OWASP Project Leader
  - Cheat Sheet Series
  - Java Encoder / HTML Sanitizer
  - Application Security Verification Standard

Iron-Clad Java: Building Secure Web Applications

Best Practices for Secure Java Web Application Development

Jim Manico
August Detlefsen
Contributing Author, Kevin Kenan
Technical Editor, Milton Smith
Oracle Senior Principal Security Product Manager, Java

ORACLE

Oracle Press

What is the OWASP Top Ten?

What is OWASP?

The Open Web Application Security Project (OWASP):

Is a web application security online community – anyone can join

Produces freely-available methods, articles, tools

Is lead by the non-profit OWASP Foundation

- Established as a 501(c) 3 is the US in 2004

- Established as OWASP Europe VZW in Belgium in 2011

- Has a number of key projects and chapters around the world

Brief History of the OWASP Top 10

Is a Flagship Project, first published in 2003

Aims to raise awareness on critical application security risks

Ranks the top 10 application security risks in its year of publication

OWASP Top 10 - 2021 is based on data from over 40 organizations

Previous editions include 2017, 2010, 2007

Is referenced in many standards, such as

- MITRE
- PCI DSS
- Defense Information Systems Agency (DISA-STIG)
- Federal Trade Commission (FTC)

Making the OWASP Top 10 – 2021

Data call – Identifies 8 of the 10 risks

- Organizations asked to contribute their vulnerability data

- Web application vulnerabilities found in various processes

Industry survey – identifies remaining 2 of the 10

- Allows information security practitioners in the front lines to vote

- Catches highest risks that might not be represented in the data

OWASP Top Ten
2021

A1: Broken Access Control

A2: Cryptographic Failures

A3: Injection

A4: Insecure Design

A5: Security Configuration

A6: Vulnerable and Outdated Components

A7: Identity & Authentication Failures

A8: Software and Data Integrity Failure

A9: Security Logging and Monitoring Failures

A10: Server Side Request Forgery

Learning the OWASP Top 10 – 2021

- Key Concepts – Information security terms you need to know

- Definition – The definition using the concepts previously introduced

- Challenges – The root causes behind this risk

- Example – Pseudocode of good example and bad example

- Best Protection Strategies – How to best prevent this risk

Here are our recommendations for when it is appropriate to use the OWASP Top 10:

Use Case	OWASP Top 10 2021	OWASP Application Security Verification Standard
Awareness	Yes	
Training	Entry level	Comprehensive
Design and architecture	Occasionally	Yes
Coding standard	Bare minimum	Yes
Secure Code review	Bare minimum	Yes
Peer review checklist	Bare minimum	Yes
Unit testing	Occasionally	Yes
Integration testing	Occasionally	Yes
Penetration testing	Bare minimum	Yes
Tool support	Bare minimum	Yes
Secure Supply Chain	Occasionally	Yes

A1: Broken Access Control

A01:2021-Broken Access Control moves up from the fifth position; 94% of applications were tested for some form of broken access control. The 34 Common Weakness Enumerations (CWEs) mapped to Broken Access Control had more occurrences in applications than any other category.

https://owasp.org/www-project-top-ten/

2 Key Concepts for
A01:2021 – Broken Access Control

Access Control

- Selectively restricting access

- Features or data require permission to access

Authorization

- Permission to access certain features or data is called authorization

Definition
A01:2021 – Broken Access Control

- **Users of the application can operate outside their defined permissions**

- **This typically leads to unauthorized information or features being processed**

- **Violation of the principle of least privilege or least authority**

Example A01:2021 – Broken Access Control

```
adminReport(param) {
  if (user.isRole("USER"), param)
  {
    //execute admin activity on param
  }
}


adminReport(param) {
  if (user.isAuthorized("ADMIN"), param)
  {
    //execute admin activity on param
  }
}
```

Good code   Bad code   User defined input

Indirect Object References (IDOR)
Horizontal Access Control

Example Feature
https://mail.example.com/message/2356342

This SQL would be vulnerable to tampering
select id,data from messages where messageid = 2356342

Ensure the owner is referenced in the query!
select id,data from messages m where m.id = :one AND (m.owner_id = :two or m.recipient_id = :two)

:one = 2356342
:two = <userid_from_session_or_jwt>

Broken Access Control Challenges

- **Access Control is** difficult to test from automated tools. Your scanning tools **are rarely aware of your custom access control policies.**

- **Access Control is** **difficult for developers to build.** Our frameworks **rarely provide detailed access control functionality.**

Best Protection Strategies
A01:2021 – Broken Access Control

Design access control so all requests must be authorized

Enforce access by activity and only for valid workflow paths, never by role

Build a centralized access control mechanism

Assign permissions to users in the context of data

Refuse access by default, fail securely

Best Practice: Code to the Activity (or Permission)

```
← → C  🗋 view-source

if (user.hasAccess(Feature.ARTICLE_EDIT))
{
  //execute activity
}
```

Code it once, never needs to change again

Implies policy is centralized in some way

Implies policy is persisted in some way

Requires more design/work up front to get right

Access Control Key Concepts I

- **Enforce** access control by an activity or feature, not the role

- **Implement data-contextual access control** to assign permissions to application users in the context of specific data items for horizontal access control requirements

- **Build** a centralized access control mechanism

- **Design** access control so all requests must be authorized

Access Control Key Concepts II

- **Deny** by default, fail securely

- **Server-side trusted data** should drive access control policy decisions

- Be able to change a users entitlements **in real time**

- **Build grouping capability** for users and permissions

- **Build admin screens first** to manage access control policy data

ASVS 4.0.3 Access Control Requirements

https://github.com/OWASP/ASVS/blob/master/4.0/en/0x12-V4-Access-Control.md

V4.1 General Access Control Design

#	Description	L1	L2	L3	CWE
4.1.1	Verify that the application enforces access control rules on a trusted service layer, especially if client-side access control is present and could be bypassed.	✓	✓	✓	602
4.1.2	Verify that all user and data attributes and policy information used by access controls cannot be manipulated by end users unless specifically authorized.	✓	✓	✓	639
4.1.3	Verify that the principle of least privilege exists – users should only be able to access functions, data files, URLs, controllers, services, and other resources, for which they possess specific authorization. This implies protection against spoofing and elevation of privilege. (C7)	✓	✓	✓	285
4.1.4	[DELETED, DUPLICATE OF 4.1.3]				
4.1.5	Verify that access controls fail securely including when an exception occurs. (C10)	✓	✓	✓	285

https://github.com/OWASP/ASVS/blob/master/4.0/en/0x12-V4-Access-Control.md

V4.2 Operation Level Access Control

#	Description	L1	L2	L3	CWE
4.2.1	Verify that sensitive data and APIs are protected against Insecure Direct Object Reference (IDOR) attacks targeting creation, reading, updating and deletion of records, such as creating or updating someone else's record, viewing everyone's records, or deleting all records.	✓	✓	✓	639
4.2.2	Verify that the application or framework enforces a strong anti-CSRF mechanism to protect authenticated functionality, and effective anti-automation or anti-CSRF protects unauthenticated functionality.	✓	✓	✓	352

V4.3 Other Access Control Considerations

#	Description	L1	L2	L3	CWE
4.3.1	Verify administrative interfaces use appropriate multi-factor authentication to prevent unauthorized use.	✓	✓	✓	419
4.3.2	Verify that directory browsing is disabled unless deliberately desired. Additionally, applications should not allow discovery or disclosure of file or directory metadata, such as Thumbs.db, .DS_Store, .git or .svn folders.	✓	✓	✓	548
4.3.3	Verify the application has additional authorization (such as step up or adaptive authentication) for lower value systems, and / or segregation of duties for high value applications to enforce anti-fraud controls as per the risk of application and past fraud.		✓	✓	732

CAUTION

- Good access control is **hard to add to an application late in the lifecycle**

VERIFY

- Automated security tools are poor at verifying access control vulnerabilities since tools are not aware of your access control policy

GUIDANCE

- **https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html**

- **http://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.sp.800-162.pdf**

- **https://github.com/OWASP/ASVS/blob/master/4.0/en/0x12-V4-Access-Control.md**

Summary
A01:2021 – Broken Access Control

Concept

Access Control
- Selectively restricting access
- Features or data require permission to access

Authorization
- Permission to access certain features or data is called authorization

Definition

Users of the application can operate outside their defined permissions

This typically leads to unauthorized information being processed

Violation of the principle of need to know

Example

```
• user.isRole("USER")


• user.isAuthorized("ADMIN")
```

Best Protection Strategies

Design access control so all requests must be authorized
Enforce access by activity and only for valid workflow paths, never by role
Build a centralized access control mechanism
Assign permissions to users in the context of data
Refuse access by default, fail securely

Good code   Bad code   User defined input

A2: Cryptographic Failure

A02:2021-Cryptographic Failures shifts up one position to #2, previously known as A3:2017-Sensitive Data Exposure, which was broad symptom rather than a root cause. The renewed name focuses on failures related to cryptography as it has been implicitly before. This category often leads to sensitive data exposure or system compromise.

https://owasp.org/www-project-top-ten/

2 Key Concepts for A02:2021 – Cryptographic Failures

Cryptography

- The art and science of keeping messages secure

- **Encryption:** An algorithm for transforming messages (**plaintext)** into secure messages (**ciphertext)**, most often using a **key.**

Cryptanalysis

- The art and science of breaking secure messages

- Cryptology = Cryptography + Cryptanalysis

Definition A02:2021 – Cryptographic Failures

- The use of weak, deprecated or incorrect cryptographic algorithms

- Sensitive data transmitted over a network without cryptography

- Insecure certificates, keys and secrets

- Weak creation of random values used for keys or as seeds

Example
A02:2021 – Cryptograpic Failures

```
Cipher cipher =
Cipher.getInstance("DES/CBC/NoPadding");
Cipher.getInstance("DESede/CBC/PKCS5Padding");
Cipher.getInstance("AES/ECB/PKCS5Padding");

Cipher cipher =
Cipher.getInstance("AES/GCM/NoPadding");
```

Good code   Bad code   User defined input

Challenges
A02:2021 – Cryptographic Failures

- Crypto knowledge is a rare commodity because the material to learn cryptography is challenging and difficult

- It is hard to verify the level of security crypto solutions attain

- It takes very senior and sophisticated developer resources

Best Protection Strategies A02:2021 – Cryptographic Failures

Manage keys and secrets properly

Use up to date and strong cryptographic algorithms, protocols and key sizes

Sensitive data requires more protection, so classify them correctly

Instrument encryption for data at rest and in transit

Configure cryptographic protocols well

Transport Layer Protection (HTTPS)

Always Use HTTPS/TLS!

- Use TLS on all connections

- Do not tolerate plaintext communication

- Use HSTS (HTTP Strict Transport Security) and preloading

Key Lifecycle
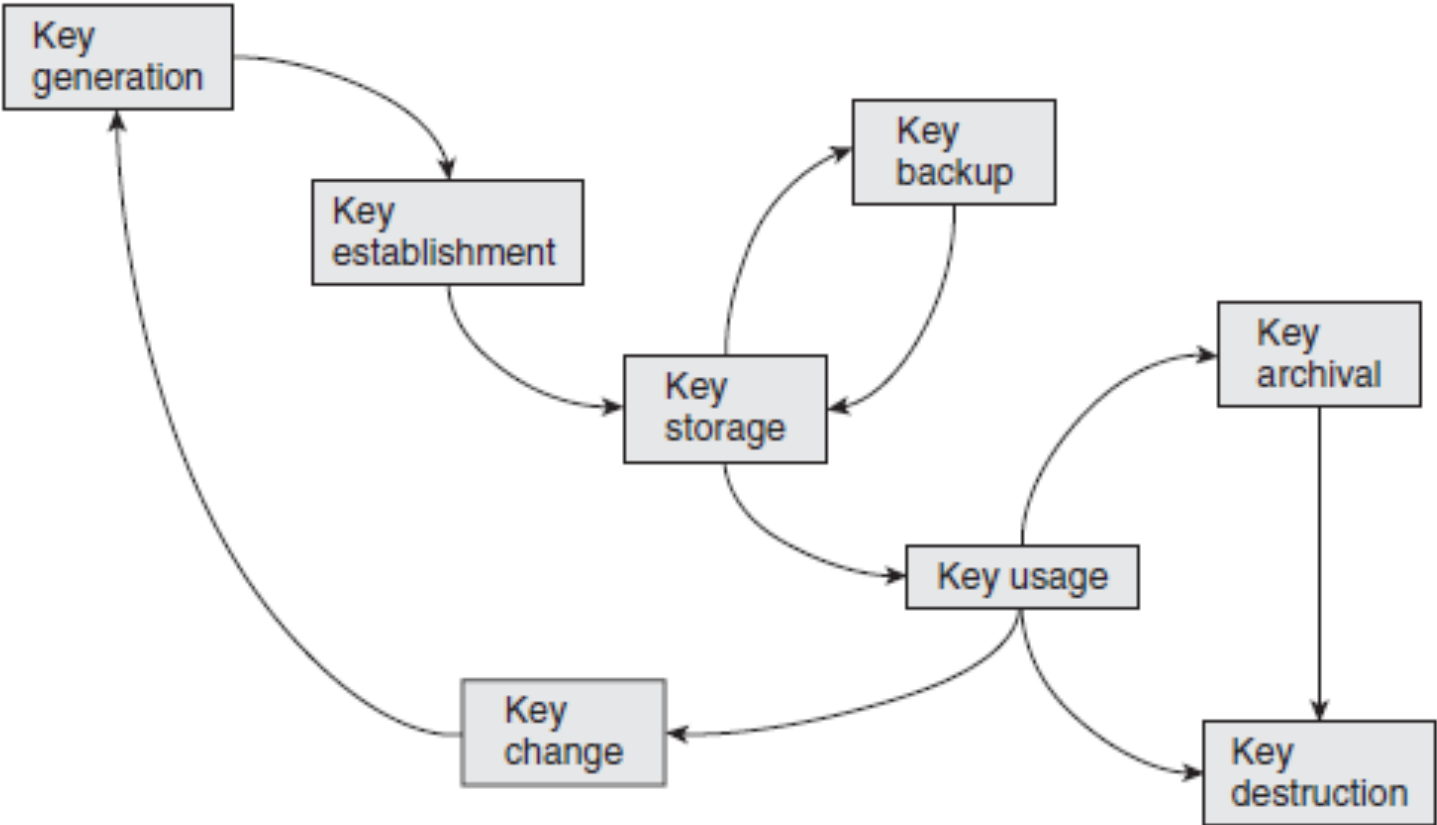
Secrets Management
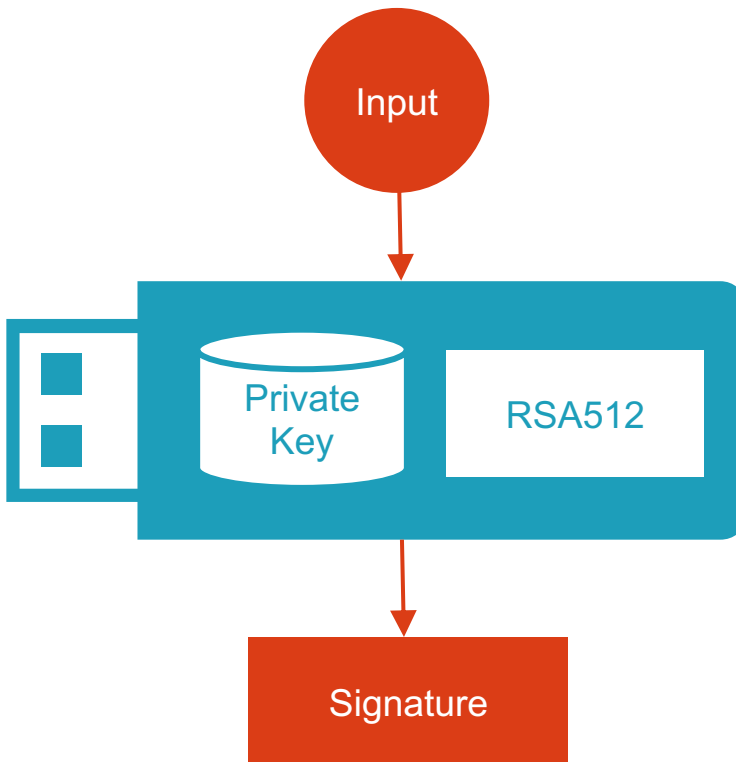
- Private key is stored in the local key database for signature signing only

- Key is not extractable

- Input is the raw data to sign

- RSA512 signature is the output

Advanced Secrets Management

https://docs.microsoft.com/en-us/azure/app-service/app-service-key-vault-references

https://code.cash.app/app-layer-encryption
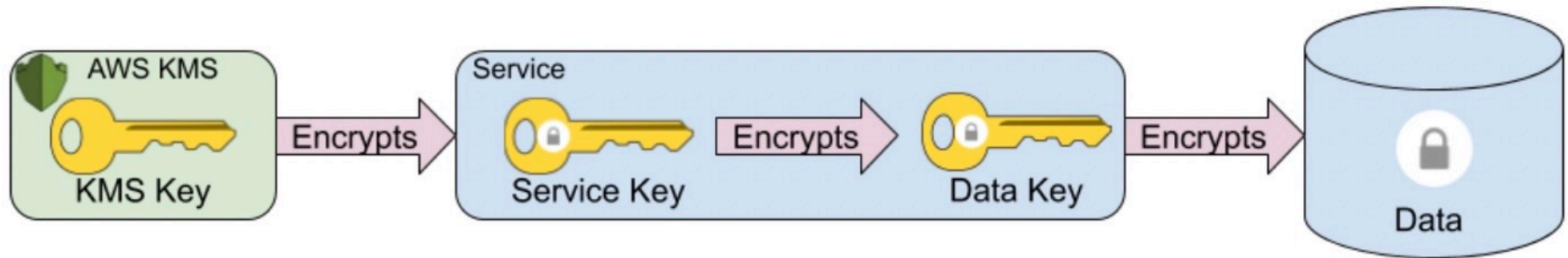
Encrypting data at Rest : Google Tink

https://github.com/google/tink

- A multi-language, cross-platform library that provides cryptographic APIs that are secure, easy to use correctly, and hard(er) to misuse

- Java, Android, C++, Obj-C, Go, and Python are field tested and ready for production

- Integration with Secrets Management

Encrypting data at Rest : Libsodium

https://github.com/jedisct1/libsodium

- A high-security, cross-platform & easy-to-use crypto library

- Modern, easy-to-use software library for encryption, decryption, signatures, password hashing and more

- Supports a variety of compilers and operating systems

CAUTION

– Applied cryptography is difficult

VERIFY

– Bring in senior resources to build, procure and verify your cryptographic implementations, especially at rest

GUIDANCE

– https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html

– https://www.ssllabs.com/

– https://owasp.org/www-project-o-saft/

– https://github.com/drwetter/testssl.sh

– https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html

Summary A02:2021 – Cryptographic Failures

Concept

- Cryptography
The art and science of keeping messages secure
- Encryption: An algorithm for transforming messages (plaintext) into secure messages (ciphertext), most often using a key.
- Cryptanalysis
The art and science of breaking secure messages
Cryptology = Cryptography + Cryptanalysis

Definition

- The use of weak, deprecated or incorrect cryptographic algorithms

- Sensitive data transmitted over a network without cryptography

- Weak creation of random values used for keys or as seeds

Example

```
Cipher cipher =
Cipher.getInstance("DES/CBC/NoPadding");

Cipher cipher =
Cipher.getInstance("AES/GCM/NoPadding");
```

Best Protection Strategies

Manage keys and secrets properly
Use up to date and strong cryptographic algorithms, protocols and key sizes
Sensitive data requires more protection, so classify them correctly
Instrument encryption for data at rest and in transit
Configure cryptographic protocols well

■ Good code ■ Bad code ■ User defined input

A3: Injection
SQL Injection

A03:2021-Injection slides down to the third position. 94% of the applications were tested for some form of injection with a max incidence rate of 19%, an average incidence rate of 3.37%, and the 33 CWEs mapped into this category have the second most occurrences in applications with 274k occurrences. Cross-site Scripting is now part of this category in this edition.

https://owasp.org/www-project-top-ten/

Key Concepts A03:2021 – Injection

- Protocol – Set of rules for exchanging information

- Protocol Encapsulation – Wrapping one set of rules into another

- Example:
  - Web application gives user control of database
  - HTTP as a protocol encapsulates SQL with user given commands

- New malicious commands are added to application hence the term "injection"
  - Injected SQL queries will run under the context of the application account allowing read and/or write access to application data and more

Definition A03:2021 – Injection

An instance where an attacker can supply untrusted data to a web application that is processed by the protocol as a command or query

This changes the execution flow typically leading to:

Stealing data from databases and other data sources

Running malicious operating system commands

Abuse authentication systems

Bypass access control

Many forms of injection depending on the protocol, such as

- SQL Injection
- LDAP Injection
- Operating System (O/S) Command Injection
- Object Query Injection

SQL Injection

Applications that insert untrusted data into database queries via string building allows attackers to execute arbitrary queries against back-end databases

SQL Injection

Injected SQL queries will run under the context of the application account allowing read and/or write access to application data and more

Looks Legit?

`jim'or'1'!='@manicode.com`

HTML5 Email Regular Expressions

The following JavaScript- and Perl-compatible regular expression is an implementation of the above definition.

/^[a-zA-Z0-9.!#$%&'*+\/=?^_`{|}~-]+@[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?(?:\.[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)*$/

Even Valid Data Can Cause Injection

1 `select id,ssn,cc,mmn from customers where email='$email'`

2 `$email = jim'or'1'!='@manicode.com`

3 `select id,ssn,cc,mmn from customers where email='jim'or'1'!='@manicode.com'`

Example
A03:2021 – Injection (Classic SQL Injection)

```
SqlCommand objCommand = new SqlCommand (
    "SELECT id,name FROM user_table WHERE
    username = ' " & Request("NameTextBox.Text") & " ' AND
    password = ' " &     Request("PasswordTextBox.Text")   &
" ' ");


SqlCommand objCommand = new SqlCommand(
    "SELECT id,name FROM user_table WHERE
    Name = @Name AND Password = @Password", objConnection);
objCommand.Parameters.Add("@Name", NameTextBox.Text);
objCommand.Parameters.Add("@Password", PasswordTextBox.Text);
```

Good code	Bad code	User defined input

Challenges
A03:2021 – Injection

- **It is hard enough getting your web application protocols and layers to work together**

- **Limiting what data gets passed where is seen as an additional step**

- **Injection is caused by insufficient user input validation, escaping or parameterization**

- **Injection can be use to circumvent authentication, access control and other defensive layers for data theft.**

Best Protection Strategies A03:2021 – Injection

Validate untrusted data

Encode data where necessary

Configure databases using least privilege principle

Use safe APIs for protocol queries

Sanitize data when parameterization is not available

WARNING:
Some variables cannot be parameterized

```
$dbh->prepare('SELECT name, color,
calories FROM ? WHERE calories < ?
order by ?');
```

- **CAUTION**

  – One SQL Injection can lead to complete data loss so be sure to **parameterize all SQL queries**

- **VERIFY**

  – Code review and static analysis do an exellent job of discovering SQL Injection in your code

- **GUIDANCE**

  – **https://bobby-tables.com/**

  – **https://cheatsheetseries.owasp.org/cheatsheets/Query_Parameterization_Cheat_Sheet.html**

Summary A03:2021 – Injection

Concept

- Protocol – Set of rules for exchanging information
- Protocol Encapsulation – Wrapping one set of rules into another

- Example:
  - Web application gives user control of database
  - HTTP as a protocol encapsulates SQL with user given commands

- New malicious commands are added to application hence the term "injection"

Definition

- An instance where an attacker can supply untrusted data to a web application that is processed by the protocol as a command or query

- This changes the execution flow

Common Forms:
- SQL Injection – LDAP Injection
- Command Injection – Object Query Injection

Example

```
"SELECT id,name FROM user_table WHERE
    username = ' " & Request("NameTextBox.Text")…


"SELECT id,name FROM user_table WHERE
    Name = @Name AND Password = @Password",
    objConnection);
objCommand.Parameters.Add("@Name", NameTextBox.Text);
…
```

Best Protection Strategies

Validate untrusted data
Encode data where necessary
Configure databases using least privilege principle
Use safe APIs for protocol queries
Sanitize data when parameterization is not available

| | Good code | | Bad code | | User defined input |

A3: Injection
Cross Site Scripting (XSS)

Data Type	Context	Defense
String	HTML Body/Attribute	HTML Entity Encode/HTML Attribute Encode
String	JavaScript Variable	JavaScript Hex Encoding
String	GET Parameter	URL Encoding
String	Untrusted URL	URL Validation, Attribute Encoding
String	CSS	CSS Hex Encoding
HTML	Anywhere	HTML Sanitization (Server and Client Side)
Any	DOM	Safe use of JS API's
Untrusted JavaScript	Any	Sandboxing and Deliver from Different Origin
JSON	Embedded	JSON Serialization/Encoding
XSS Standard		Content Security Policy
DOM XSS Standard		Trusted Types

A4: Insecure Design

A04:2021-Insecure Design is a new category for 2021, with a focus on risks related to design flaws. If we genuinely want to "move left" as an industry, we need more threat modeling, secure design patterns and principles, and reference architectures.

https://owasp.org/www-project-top-ten/

3 Key Concepts for A04:2021 – Insecure Design

1. **Architectural flaws**

- Flaws of Omission – Ignoring a threat or security requirement

- Flaws of Commission – Bad design e.g., client-side authentication

2. **Secure Design Patterns**

- Examples include protocol breaks across different network zones

3. **Reference Architectures**

- Examples include detailed technical diagrams, zero trust user access

Definition A04:2021 – Insecure Design

Collection of security flaws that cannot be attributed to, or fixed by implementation

Broad category – captures missing or ineffective controls e.g.,

Architectural flaws of omission, where security requirements have not been provided or are not been followed

Access has not been appropriately restricted

Design flaws have a different root causes to implementation defects

Good implementation cannot fix insecure design

Example A04:2021 – Insecure Design

```
userAccess() {
  if (user.isAuthorized("USER"))
  {
    //authorize user independent of zone
  }
}


userAccess() {
  if (user.isConnectingFromZone(ZONE.SemiTrusted) &&
  user.isAuthorized("USER"))
  {
    //authorize user based on zone
  }
}
```

Challenges A04:2021 – Insecure Design

Not enough time is given to architecture and design

Architectural flaws of omission are often the cause of time constraints

There is no dedicated resource to look at and model the architectural threats

Secure design patterns are often incorrectly customized

It is hard to admit insecure design because the fix is not easy to implement

This can cause a huge waste of engineering time

Often reference architectures are not updated for latest frameworks

Developers want to use the next cool framework or library

Best Protection Strategies
A04:2021 – Insecure Design

Deny by principle, based on policy

Apply known reference architectures

Design using privilege separation

Generate security requirements to counter threats

Use known design patterns

Manage protocols and restrict permissions

Threat Modeling Presentations

Avi Douglen

https://securityweekly.com/shows/threat-modeling-in-appsec-avi-douglen-asw-105/

Tony UV

https://www.youtube.com/watch?v=s21aI-jqIVM

Summary A04:2021 – Insecure Design

Concept

Architectural flaws

Secure Design Patterns
- Examples include protocol breaks across different network zones

Reference Architectures

Definition

Collection of security flaws that cannot be attributed to, or fixed by implementation

Broad category – captures missing or ineffective controls

Design flaws have a different root causes to implementation defects

Example

```
userAccess() {
 if (user.isAuthorized("USER")) {
     //authorize user
 }
 if (user.connectFromZone(ZONE.SemiTrusted)){
     //authorize user
 }
}
```

■ Good code   ■ Bad code   ■ User defined input

Best Protection Strategies

Deny by principle, based on policy
Apply known reference architectures
Design using privilege separation
Generate security requirements to counter threats
Use known design patterns
Manage protocols and restrict permissions

A5: Security Misconfiguration

A05:2021-Security Misconfiguration moves up from #6 in the previous edition; 90% of applications were tested for some form of misconfiguration, with an average incidence rate of 4.5%, and over 208k occurrences of CWEs mapped to this risk category. With more shifts into highly configurable software, it's not surprising to see this category move up. The former category for A4:2017-XML External Entities (XXE) is now part of this risk category.

https://owasp.org/www-project-top-ten/

2 Key Concepts
A05:2021 – Security Misconfiguration

1. **Security Control**

- An information system safeguard or countermeasure

- Designed to protect: Confidentiality, Integrity, or Availability

2. **Misconfiguration that introduces common vulnerabilities**

- Not performing security hardening

- Keeping default settings

Like A4:2021 Insecure Design, also a broad category

- Often referred to as the 'catchall' of the OWASP Top 10

Definition A05:2021 – Security Misconfiguration

Failing to implement the necessary controls to secure the configurations of your application

Configuration puts your systems and data at risk

Vulnerabilities caused due to configuration

Example A05:2021 – Security Misconfiguration

```
<global-exceptions>
  <exception key="global.error.invalidLogin" path=""
 scope="request" type="InvalidLoginException" />
</global-exceptions>

<global-forwards>
  <forward name="sign-in" path="Sign-in.jsp" />
</global-forwards>

<global-forwards>
  <forward name="sign-in" path="/Sign-in.jsp" />
</global-forwards>
```

Good code   Bad code   User defined input

Challenges A05:2021 – Security Misconfiguration

- This topic **can span anything** from password length to file permissions to access control **and more**

- You need to *read the manual* for the framework, deployment environment and **everything in between**

Best Protection Strategies
A05:2021 – Security Misconfiguration

Verify configurations

Assume insecure if you cannot verify

Read existing hardening and security guides

Know your frameworks and libraries

Apply security settings available to you

Study to know enough about your platforms

Know your Framework, Libraries and Production Environment

For frameworks, libraries and production environment:

- Hardening Guide

- Security Guide

- Security Settings

- Secure Deployment

Settings can easily open up major security gaps

e.g. "Open" AWS S3 Buckets

mozilla SSL Configuration Generator

Server Software

- ○ Apache
- ○ AWS ALB
- ○ AWS ELB
- ○ Caddy
- ○ Dovecot
- ○ Exim
- ○ Go
- ○ HAProxy
- ○ Jetty
- ○ lighttpd
- ○ MySQL
- ● nginx
- ○ Oracle HTTP
- ○ Postfix
- ○ PostgreSQL
- ○ ProFTPD
- ○ Redis
- ○ Tomcat
- ○ Traefik

Mozilla Configuration

- ○ **Modern**
  Services with clients that support TLS 1.3 and don't need backward compatibility

- ● **Intermediate**
  General-purpose servers with a variety of clients, recommended for almost all systems

- ○ **Old**
  Compatible with a number of very old clients, and should be used only as a last resort

Environment

Server Version	1.17.7
OpenSSL Version	1.1.1d

Miscellaneous

☑ HTTP Strict Transport Security

This also redirects to HTTPS, if possible

☑ OCSP Stapling

nginx 1.17.7, intermediate config, OpenSSL 1.1.1d

Supports Firefox 27, Android 4.4.2, Chrome 31, Edge, IE 11 on Windows 7, Java 8u31, OpenSSL 1.0.1, Opera 20, and Safari 9

```
# generated 2021-05-12, Mozilla Guideline v5.6, nginx 1.17.7, OpenSSL 1.1.1d, intermediate configuration
# https://ssl-config.mozilla.org/#server=nginx&version=1.17.7&config=intermediate&openssl=1.1.1d&guideline=5.6
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    location / {
        return 301 https://$host$request_uri;
```

You are here: Home > Projects > SSL Server Test > manicode.com

SSL Report: manicode.com (198.199.114.91)

Assessed on: Tue, 11 May 2021 20:29:16 UTC | Hide | Clear cache

Scan Another »

Summary

Overall Rating

A+

Certificate
Protocol Support
Key Exchange
Cipher Strength

0   20   40   60   80   100
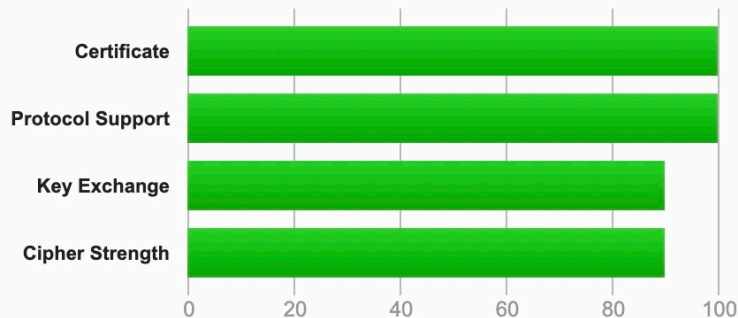
Visit our **documentation page** for more information, configuration guides, and books. Known issues are documented **here**.

This site works only in browsers with SNI support.

This server supports TLS 1.3.

HTTP Strict Transport Security (HSTS) with long duration deployed on this server. **MORE INFO »**

XML EXTERNAL ENTITY PROCESSING

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >
]>
<foo>&xxe;</foo>
```

Configure all of your XML parsers to disable external entity resolution!

https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html

- **CAUTION**
  - This is a huge category that involves everywhere from the OS to the Framework to the App Server and more

- **VERIFY**

  If you can't verify the config assume it's not secure

- **GUIDANCE**
  - Learn the proper settings and *read the manual* for security configuration needs
  - Cloud configuration is especially important and requires proper platform knowledge

Summary A05:2021 – Security Misconfiguration

Concept

Security Control
- An information system safeguard or countermeasure
- Designed to protect: Confidentiality, Integrity, or Availability

Misconfiguration that introduces common vulnerabilities
- Not performing security hardening
- Keeping default settings

Often referred to as the 'catchall' of the OWASP Top 10

Definition

Failing to implement the necessary controls to secure the configurations of your application

Configuration puts your systems and data at risk

Vulnerabilities caused due to configuration

Example

- … path="" … />
- … path="Sign-in.jsp" />

- … path="/Sign-in.jsp" />

Best Protection Strategies

Verify configurations
Assume insecure if you cannot verify
Read existing hardening and security guides
Know your frameworks and libraries
Apply security settings available to you
Study to know enough about your platforms

Good code   Bad code   User defined input

A6: Vulnerable and Outdated Components

A06:2021-Vulnerable and Outdated Components
was previously titled Using Components with Known Vulnerabilities and is #2 in the Top 10 community survey, but also had enough data to make the Top 10 via data analysis. This category moves up from #9 in 2017 and is a known issue that we struggle to test and assess risk.

https://owasp.org/www-project-top-ten/

3 Key Concepts for A06:2021 – Vulnerable and Outdated Components

1. **Software often has**

- Past vulnerable versions using outdated components

2. **Systems and frameworks have a date by when**

- End of life, end of sale, unsupported

3. **Attackers attempt to exploit newly published vulnerabilities**

- Requiring you apply fixes or perform updates

Definition A06:2021 – Vulnerable and Outdated Components

Having in use software components that are vulnerable, unsupported, or out of date

Environments that are not patched in a timely manner

Not knowing the versions of software, you use

Challenges A06:2021 – Vulnerable and Outdated Components

- Patching is a monthly, quarterly, or undefined task, leaving organizations exposed to known vulnerabilities

- Ensuring 3rd party libraries are up to date is often neglected as a process due to time and similar constraints

Best Protection Strategies A06:2021 – Vulnerable and Outdated Components

Check continuously that your libraries are updated and then actually keep them updated

Only obtain components from official trusted sources

Remove unused dependencies

Use only features that are necessary

Stay current with latest vulnerabilities

Third Party Library Security in the NVD Database

NIST

Information Technology Laboratory

NATIONAL VULNERABILITY DATABASE

VULNERABILITIES

🐛 CVE-2021-29447 Detail

Current Description

Wordpress is an open source CMS. A user with the ability to upload files (like an Author) can exploit an XML parsing issue in the Media Library leading to XXE attacks. This requires WordPress installation to be using PHP 8. Access to internal files is possible in a successful XXE attack. This has been patched in WordPress version 5.7.1, along with the older affected versions via a minor release. We strongly recommend you keep auto-updates enabled.

Drupal™

Security advisories

Drupal core   Contributed projects   Public service announcements

Drupal core - Critical - Third-party libraries - SA-CORE-2021-001

Project: Drupal core
Date: 2021–January–20
Security risk: `Critical` 18/25 AC:Complex/A:User/CI:All/II:All/E:Exploit/TD:Uncommon
Vulnerability: Third–party libraries
Description:
The Drupal project uses the pear Archive_Tar library, which has released a security update that impacts Drupal. For more information please see:

- CVE–2020–36193

Exploits may be possible if Drupal is configured to allow `.tar`, `.tar.gz`, `.bz2`, or `.tlz` file uploads and processes them.

Solution:
Install the latest version:

- If you are using Drupal 9.1, update to Drupal 9.1.3.
- If you are using Drupal 9.0, update to Drupal 9.0.11.
- If you are using Drupal 8.9, update to Drupal 8.9.13.
- If you are using Drupal 7, update to Drupal 7.78.

Versions of Drupal 8 prior to 8.9.x are end–of–life and do not receive security coverage.

Disable uploads of `.tar`, `.tar.gz`, `.bz2`, or `.tlz` files to mitigate the vulnerability.

Contact and more information

The Drupal security team can be reached by email at security at drupal.org or via the contact form.

Learn more about the Drupal Security team and their policies, writing secure code for Drupal, and securing your site.

Follow the Drupal Security Team on Twitter @drupalsecurity

Contributing organizations for this advisory

Solathat

Acro Media Inc

Morris Animal Foundation

Acquia

3rd Party Management Tools

OWASP dependency-check
https://owasp.org/www-project-dependency-check/

Maven Security Versions
https://github.com/victims/maven-security-versions

Retire.js (JavaScript 3rd party library analysis)
https://retirejs.github.io/retire.js/

Create PR's for your dependencies automatically
https://dependabot.com/

CAUTION

- Virtually every application has 3rd party library issues because most development teams don't focus on ensuring their libraries are up to date

VERIFY

- Use automation that checks periodically (e.g., every build or check-in) to see if your libraries are out of date and then actually updated them!

GUIDANCE

- **https://owasp.org/www-project-dependency-check/**

Summary A06:2021 – Vulnerable and Outdated Components

Concept

Software often has
- Past vulnerable versions using outdated components

Systems and frameworks have a date by when
- End of life, end of sale, unsupported

Attackers attempt to exploit newly published vulnerabilities
- Requiring you apply fixes or perform updates

Definition

Having in use software components that are vulnerable, unsupported, or out of date

Environments that are not patched in a timely manner

Not knowing the versions of software you use

Example

```
<artifactId>log4j-core</artifactId>
   <version>2.14.1</version>


<artifactId>log4j-core</artifactId>

   <version>2.16.0</version>
   <!-- or newer -->
```

Best Protection Strategies

Check periodically your libraries are updated
Only obtain components from official trusted sources
Remove unused dependencies
Use only features that are necessary
Stay current with latest vulnerabilities

Good code   Bad code   User defined input

A7: Identification and Authentication Failures

A07:2021-Identification and Authentication Failures was previously Broken Authentication and is sliding down from the second position, and now includes CWEs that are more related to identification failures. This category is still an integral part of the Top 10, but the increased availability of standardized frameworks seems to be helping.

https://owasp.org/www-project-top-ten/

3 Key Concepts for A07:2021 – Identification and Authentication Failures

1. **Digital Identity**

- Set of attributes related to a person, organization, application, or device

2. **Identification**

- The act of indicating (showing) one's identity

3. **Authentication**

- Process or action confirming the identity of a user

- The act of verifying (checking) one's identity

Definition A07:2021 – Identification and Authentication Failures

Permit attacks that disclose identity attributes

Allow for authentication controls to be subverted or bypassed

Permit weak or misconfigured attributes (e.g., weak passwords)

Example A07:2021 – Identification and Authentication Failures

```
if (user.equals(username)) {
  if  (pass.equals(password)) {
    response.set("Invalid Password");
  } else {
    response.authoriseUser(user);
  }
}

if (user.equals(username) && pass.equals(password)) {
 response.authoriseUser(user);
else {
 response.set("Invalid Username or Password");
  }
}
```

Good code   Bad code   User defined input

Challenges A07:2021 – Identification and Authentication Failures

- Weakest point is the point of interaction with the user on their identity

- Many attack scenarios including credential stuffing, brute force, session reuse attacks, weak passwords, etc.

High Level Authentication and Session Topics

- Password Binding

- General Authentication Rules

- Credential Storage

- Credential Recovery

- Look-up Recovery Tokens

- Out of Band Authentication

- Session Creation

- Session Termination

- Cookie Based Sessions

- Token Based Sessions

- Federated Authentication

- One Time Passwords

- Service Authentication

How Strong Should Your Digital Identity Solution Be?

Table 5-2 Authenticator Assurance Levels

Authenticator Assurance Level
AAL1: AAL1 provides some assurance that the claimant controls an authenticator registered to the subscriber. AAL1 requires single-factor authentication using a wide range of available authentication technologies. Successful authentication requires that the claimant prove possession and control of the authenticator(s) through a secure authentication protocol.
AAL2: AAL2 provides high confidence that the claimant controls authenticator(s) registered to the subscriber. Proof of possession and control of two different authentication factors is required through a secure authentication protocol. Approved cryptographic techniques are required at AAL2 and above.
AAL3: AAL3 provides very high confidence that the claimant controls authenticator(s) registered to the subscriber. Authentication at AAL3 is based on proof of possession of a key through a cryptographic protocol. AAL3 is like AAL2 but also requires a "hard" cryptographic authenticator that provides verifier impersonation resistance.

https://pages.nist.gov/800-63-3/sp800-63-3.html
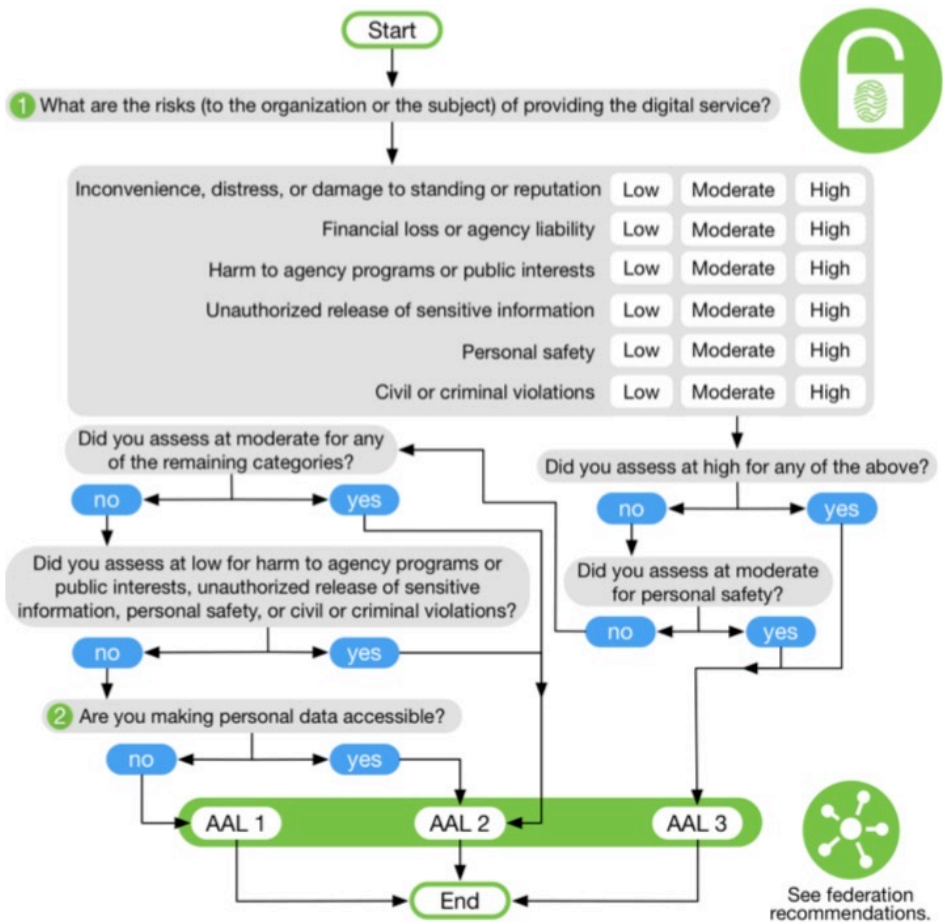
Figure 6-2 Selecting AAL

Modern Password Policy

Should we be limiting characters of a password?

- Limiting password characters to protect against injection is doomed to failure

- Very long passwords can cause DoS

- Minimum 8 char passwords

- Must support up to 64

- No more than 128

Use a Modern Password Policy Scheme

- Consider the password policy and MFA suggestions from the standard NIST SP800-63b

- Do not depend on passwords as a sole credential anytime sensitive data is involved and use MFA

Do not limit the character type of passwords

At least 8 characters and allow up to 64 but no more than 128

Block context-specific passwords like the username or service name

Check against a list of common passwords

Check against a list of breached password

Throttle or otherwise manage brute force attempts

Don't force unnatural password special character rules

Don't use password security questions or hints

No more mandatory password expiration for the sake of it

Force the use of MFA anytime sensitive data is in play

NIST Special Publication 800-63b: Digital AuthN

Password1!

Password
Storage

Configure Password Hashing Functions Correctly

- Use **Argon2id** with a minimum configuration of **15 MiB of memory**, an **iteration count of 2**, and 1 degree of parallelism

- If Argon2id is not available, use **bcrypt** with a **work factor of 10 or more** and with a password limit of 72 bytes

- For legacy systems using **scrypt**, use a minimum CPU/memory cost parameter of (2^16), a minimum block size of 8 (1024 bytes), and a parallelization parameter of 1

- If FIPS-140 compliance is required, use **PBKDF2** with a work factor of 310,000 or more and set with an internal hash function of HMAC-SHA-256

https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

CAUTION

– Authentication and Session Management are very complex layers of software to both build and verify

VERIFY

– There are many requirements to consider for Authentication and Session Management

GUIDANCE

– https://github.com/OWASP/ASVS/blob/master/4.0/en/0x11-V2-Authentication.md

– https://github.com/OWASP/ASVS/blob/master/4.0/en/0x12-V3-Session-management.md

– https://pages.nist.gov/800-63-3/

Best Protection Strategies A07:2021 – Identification and Authentication Failures

Force strong credentials e.g., passwords

Ensure user registration and recovery are hardened

Manage authenticated sessions and tokens

Alert on attacks e.g., brute force

Limit failed login attempts

Enable multi-factor authentication

Summary A07:2021 – Identification and Authentication Failures

Concept
Digital Identity
- Set of attributes related to a person, organization, application, or device

Identification
- The act of indicating (showing) one's identity

Authentication
- Process or action confirming the identity of a user
- The act of verifying (checking) one's identity

Definition
Permit attacks that disclosure identity attributes

Allow for authentication controls to be subverted or bypassed

Permit weak or misconfigured attributes (e.g., weak passwords)

Example
```
if  (pass.equals(password)) {
    response.set("Invalid
Password");

 response.set("Invalid Username or
 Password");
```

Best Protection Strategies
Ensure user registration and recovery are hardened
Limit failed login attempts
Alert administrators on attacks e.g., brute force
Force strong passwords
Implement multi-factor authentication

Good code   Bad code   User defined input

A8: Software and Data Integrity Failures

A08:2021-Software and Data Integrity Failures is a new category for 2021, focusing on making assumptions related to software updates, critical data, and CI/CD pipelines without verifying integrity. One of the highest weighted impacts from Common Vulnerability and Exposures/Common Vulnerability Scoring System (CVE/CVSS) data mapped to the 10 CWEs in this category. A8:2017-Insecure Deserialization is now a part of this larger category.

https://owasp.org/www-project-top-ten/

3 Key Concepts for A08:2021 Software and Data Integrity Failures

1. **Software Integrity Verification – Process of verifying the inclusion of functionality from untrusted software sources**

Applications relying on libraries, modules, resources or plugins received from untrusted sources

Examples of untrusted sources are repositories and Content Delivery Networks (CDNs)

2. **Continuous Integration (CI) – Merging all developers' working software copies to a shared main instance, typically several times a day**

3. **Continuous Delivery (CD) – Releasing software reliably in an automated way without manual intervention**

Definition A08:2021 Software and Data Integrity Failures

Software code that does not prevent the inclusion of functionality from untrusted sources

Downloading or updating source code dependencies from software repositories without performing integrity checks

Deserializing untrusted data or applying updates to a previously trusted application

Example A08:2021 Software and Data Integrity Failures

```
Update() {
    var data = download("https://manicode.com/update.sh");
    exec(data);
}

Update() {
    var data = download("https://manicode.com/update.sh");
    verifyHash(data, "e7de35ebe643d7a$d23fd814639ac420fc2a9b6");
    verifyGitContent(data);
    exec(data);
}
```

Rule of Two

Check your scripts not once ... but twice!

- Be very wary of scripts you down download and run daily
- I'm looking at you DevOps pipelines!

Solution?

- **Verify once via the download hash**
- **Verify a second time via the published repot**

Main Challenges A08:2021 Software and Data Integrity Failures

1. Many solutions auto-update without sufficient integrity protections without immediate solutions

- Attackers are targeting software update mechanisms and this problem is on the rise and widespread

2. Software integrity problems are often challenging to detect

- Keeping your 3rd party components updated (A06),is a significant challenge and is a big part of software and data integrity protection

Best Protection Strategies A08:2021 Software and Data Integrity Failures

Learn enough cryptography to verify integrity of downloads

Ensure all 3rd party software and frameworks are updated

Verify software updates independently, using cryptography

Apply and use digital signatures

Note what software sources you trust

Summary A08:2021 Software and Data Integrity Failures

Concept
- Software Integrity Verification – Process of verifying the inclusion of functionality from untrusted software sources

- Applications relying on libraries, modules, resources or plugins received from untrusted sources

- This can happen most notably during Continuous Integration (CI) and/or Continuous Delivery (CD)

Definition
- Software code that does not prevent the inclusion of functionality from untrusted sources

- Downloading or updating source code dependencies without performing integrity checks

- Deserializing untrusted data or applying updates to a previously trusted application

Example
```
Update() {
    var data = download("https://manicode.com/update.sh");
    exec(data);
}

Update() {
    var data = download("https://manicode.com/update.sh");
    verifyHash(data, "e7de35ebe643d7a$d23fd814639ac420fc2a9b6");
    verifyGitContent(data);
    exec(data);
}
```

Best Protection Strategies
Learn enough cryptography to verify integrity of downloads
Ensure all 3rd party software and frameworks are updated
Verify software updates independently, using cryptography
Apply and use digital signatures
Note what software sources you trust

■ Good code   ■ Bad code   ■ User defined input

A9: Security Logging and Monitoring Failures

A09:2021-Security Logging and Monitoring Failures was previously A10:2017-Insufficient Logging & Monitoring and is added from the Top 10 community survey (#3), moving up from #10 previously. This category is expanded to include more types of failures, is challenging to test for, and isn't well represented in the CVE/CVSS data. However, failures in this category can directly impact visibility, incident alerting, and forensics.

https://owasp.org/www-project-top-ten/

Why Logging?

"...the goal of logging is to be able to alert on specific security events..."

https://cheatsheetseries.owasp.org/cheatsheets/Application_Logging_Vocabulary_Cheat_Sheet.html

3 Key Concepts for A09:2021 – Security Logging and Monitoring Failures

1. Event Logging provides a standard, centralized way of recording important software events

2. Event Monitoring is the process of collecting, analyzing and signaling event occurrences

3. Security Logging and Monitoring focuses on events that can impact the confidenti*ality, integrity or availability of software*

This category is unique in that it is not a specific risk that leads to compromised software

Aids in the accountability, visibility, incident alerting, and forensics and has wide reaching implications to security management of software

Definition A09:2021 –
Security Logging and Monitoring Failures

Auditable events, warnings and errors are not adequately logged

Developers and Security Staff must work together to agree on a security centric logging standard so developers know exactly what events to log

Developers should consider logging labels specific to security

Proper logging infrastructure is necessary in order to securely collect and store logs long term

Example A09:2021 –
Security Logging and Monitoring Failures

```
if (!user.hasAccess("ADMIN_ACTION")) {
  //quietly deny access
}


if (!user.hasAccess("ADMIN_ACTION")) {
  //deny access and log
  log.event(Event.SECURITY, Event.CRITICAL, "User
  attempted to access admin action without permission");
}
```

Good code   Bad code   User defined input

Challenges A09:2021 – Security Logging and Monitoring Failures

- Developers are often unaware of the many security events that need to be logged

  - Appropriate alerting thresholds and response escalation processes are not in place or effective.

  - This can lead to applications that cannot detect, escalate, or alert for attacks or suspicious activity

- Security Operations Centre (SOC) teams often do not onboard application-level logging correctly

Best Protection Strategies A09:2021 – Security Logging and Monitoring Failures

Build a secure logging infrastructure for collection and storage of logs long term

Ensure all authentication and access control events, both failed and successful, are logged

Standardize machine-readable formats for events and alerts

Test incident response based on logging events

What To Log

- Authentication Events

- Access Control Events

- Rate Limiting Events

- File Upload Events

- Input Validation Events

- Malicious Behavior Events

- Permission Changes

- Sensitive Data Changes

- Sequence Errors

- Session Management Errors

- System Events

- User Management

https://cheatsheetseries.owasp.org/cheatsheets/Logging_Vocabulary_Cheat_Sheet.html

CAUTION

- Be sure developers and security teams work together to ensure good security logging

VERIFY

- Verify that proper security events are getting logged and consumed properly by your SOC teams

GUIDANCE

- **https://cheatsheetseries.owasp.org/cheatsheets/Application_Logging_Vocabulary_Cheat_Sheet.html**

- **https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html**

Summary A09:2021 – Security Logging and Monitoring Failures

Concept

- Event Logging provides a standard, centralized way of recording important software events

- Event Monitoring is the process of collecting, analyzing and signaling event occurrences

- Security Logging and Monitoring focuses on events that can impact the confidentiality, integrity or availability of software

Definition

- Auditable events, warnings and errors are not adequately logged

- Developers should consider logging labels specific to security
- Proper logging infrastructure is necessary here to securely collect and store logs long term

Example

```
if (!user.hasAccess("ADMIN_ACTION")) {
    //quietly deny access
}

if (!user.hasAccess("ADMIN_ACTION")) {
    //deny access and log
    log.event(Event.SECURITY, Event.CRITICAL, "User attempted to access
    admin action without permission");
}
```

Best Protection Strategies

Build a secure logging infrastructure for collection and storage of logs long term
Ensure all authentication and access control events, both failed and successful, are logged
Standardize machine-readable formats for events and alerts
Test incident response based on logging events

🟦 Good code 🟧 Bad code ⬛ User defined input
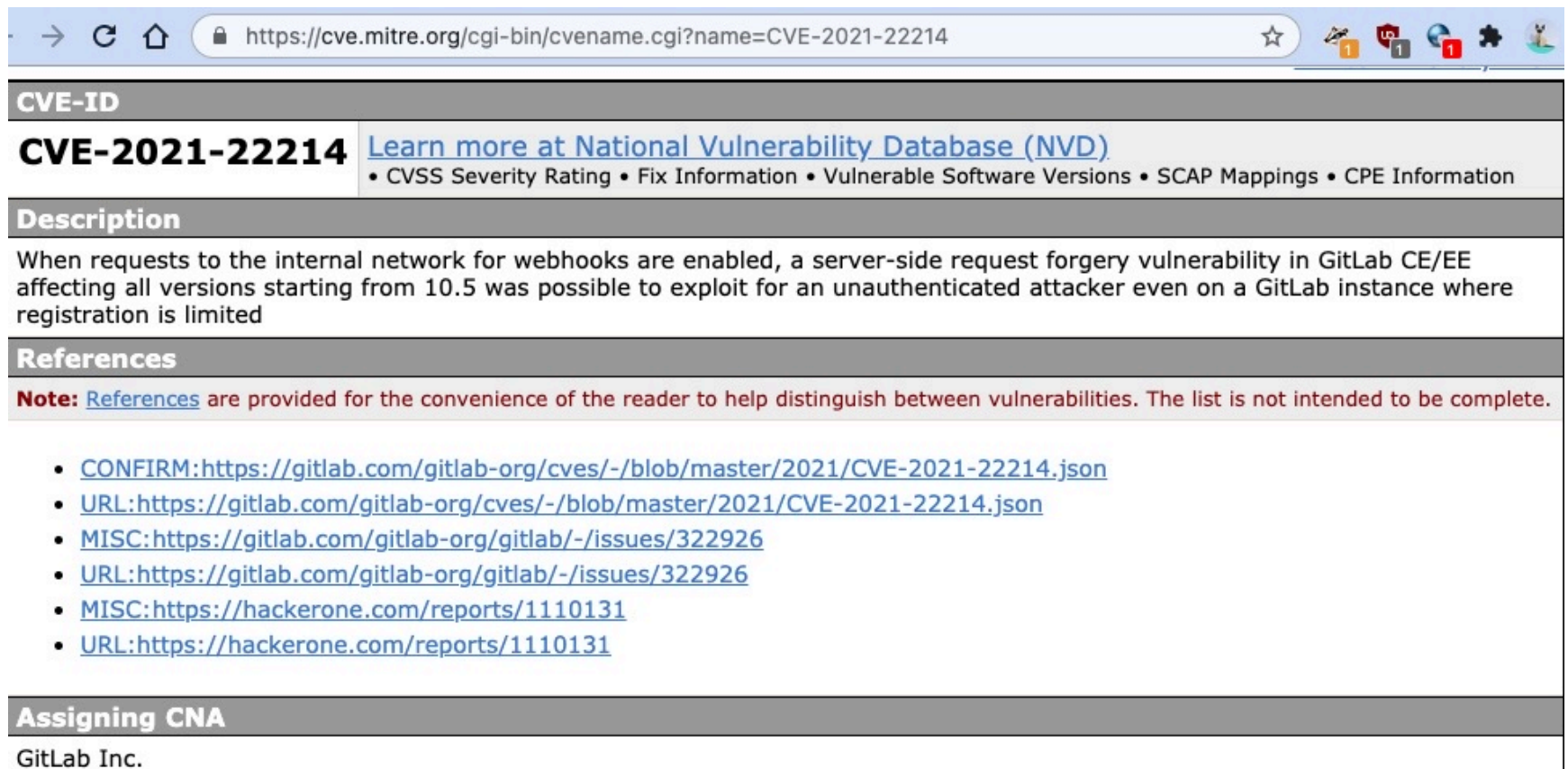
A10: Server Side Request Forgery (SSRF)

A10:2021-Server-Side Request Forgery is added from the Top 10 community survey (#1). The data shows a relatively low incidence rate with above average testing coverage, along with above-average ratings for Exploit and Impact potential. This category represents the scenario where the security community members are telling us this is important, even though it's not illustrated in the data at this time.

https://owasp.org/www-project-top-ten/

SSRF At GitLab

→ C ⌂ 🔒 https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-22214 ☆

CVE-ID

CVE-2021-22214 — Learn more at National Vulnerability Database (NVD)
• CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information

Description

When requests to the internal network for webhooks are enabled, a server-side request forgery vulnerability in GitLab CE/EE affecting all versions starting from 10.5 was possible to exploit for an unauthenticated attacker even on a GitLab instance where registration is limited

References

Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

- CONFIRM:https://gitlab.com/gitlab-org/cves/-/blob/master/2021/CVE-2021-22214.json
- URL:https://gitlab.com/gitlab-org/cves/-/blob/master/2021/CVE-2021-22214.json
- MISC:https://gitlab.com/gitlab-org/gitlab/-/issues/322926
- URL:https://gitlab.com/gitlab-org/gitlab/-/issues/322926
- MISC:https://hackerone.com/reports/1110131
- URL:https://hackerone.com/reports/1110131

Assigning CNA

GitLab Inc.

EXPLOIT DATABASE

Microsoft Exchange 2019 - SSRF to Arbitrary File Write (Proxylogon) (PoC)

EDB-ID:	CVE:
49637	2021-27065
	2021-26855

EDB Verified: ✕

Author:	Type:
TESTANULL	WEBAPPS

Exploit: ⬇ / {}

Platform:	Date:
WINDOWS	2021-03-11

Vulnerable App:

Exploit Title: Microsoft Exchange 2019 - SSRF to Arbitrary File Write (Proxylogon)

4 Key Concepts for A10:2021 – Server-Side Request Forgery

1. **Server-Side** refers to programs and operations that run on the server

2. **Request Forgery** means crafting a web request that appears legitimate, but contains malicious input

3. **Cross-Site Request Forgery (CSRF)** forces a user to execute unwanted actions, while authenticated on a web application

4. **One-click attack** involves sending a malicious URL to an authenticated user that executes an action they do not approve

   ▪ One-click attack resulting in the transfer of funds

Definition A10:2021 – Server-Side Request Forgery

Attack that forces a server to make a request to an unexpected resource

Can lead to a wide variety of critical impacts including loss of data, privilege escalation, and more

A common vulnerability in N-tiered webservices and microservices

Example A10:2021 – Server-Side Request Forgery

```
addToPage(String url) {
  if (Validator.isValidURL(url)) {
      return fetchContent(url);
  }
}


addToPage(String url) {
   if (Validator.isValidURL(url)) {
      if (url.domain == "manicode.com") {
         return fetchContent(url);
      }
   }
}
```

Good code   Bad code   User defined input

SSRF In The Real World
August '19

Capital One hack highlights SSRF concerns for AWS

Infosec pros warn of server-side request forgery vulnerabilities in AWS following the Capital One data breach, which may have revealed an issue regarding the AWS metadata service.

Rob Wright
News Director

Chris Kanaracus
Senior News Writer

Published: 05 Aug 2019

https://searchsecurity.techtarget.com/news/252467901/Capital-One-hack-highlights-SSRF-concerns-for-AWS

1. Accessing the credentials using the SSRF bug

- The attacker seems to have accessed the AWS credentials for a role called `ISRM-WAF-Role` via the endpoint
  `http://169.254.169.254/latest/meta-data/iam/security-credentials/ISRM-WAF-Role` using the SSRF bug.

For example, if the vulnerable application was at `http://example.com` and the SSRF existed in a GET variable called `url`, then the exploitation was possible as

```
curl http://example.com/?url=http://169.254.169.254/latest/meta-data/iam/security-credentials/ISRM-WAF-Role
```

https://blog.appsecco.com/an-ssrf-privileged-aws-keys-and-the-capital-one-breach-4c3c2cded3af

Challenges A10:2021 – Server-Side Request Forgery

- Traditional code and dynamic scanning tools struggle to find *Server-Side Request Forgery* accurately

- Passing URLs and IP addresses is very common (e.g., for logging purposes)

Best Protection Strategies A10:2021 – Server-Side Request Forgery

Validate origin of URLs and IPs when parameters

Ensure authentication and access control on APIs

Setup URL encoding for untrusted parameters

Test and limit network service access with network controls

SSRF Detailed Defense Summary

- Great session management on internal/intranet APIs

- Great access control on internal/intranet APIs

- When URL's are a parameter that the server then acts upon do strong URL Validation

- ***Avoid taking URLs as a full parameter that the server then acts on***

- Building URLs safely with URL Encoding of Parameters

- Limit services with network controls

- Microsegmentation

URLs to backend REST APIs are built with concatenation instead of URIBuilder (Prepared URI)

- Most publically exposed REST APIs turn around and invoke internal REST APIs using URLConnections, Apache HttpClient or other REST clients. If user input is directly concatenated into the URL used to make the backend REST request then the application could be vulnerable to Extended HPPP.

A ──→ Pub REST API ←──→ B ──→ Internal REST API ←──→ DB
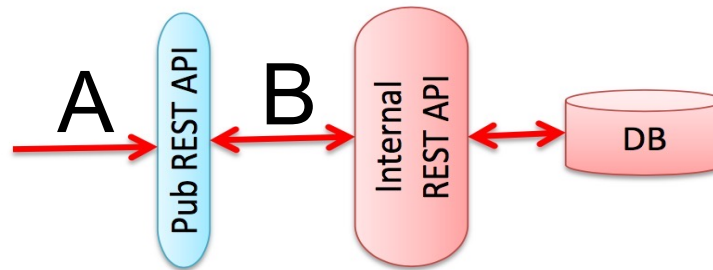
A var = request.getParameter("data");

B new URL("https://internal/data/" + var)

https://someserver/search?data=23

var = request.getParameter("data");
new URL("https://internal/data/" + var)

../../../admin/report/global

https://internal/admin/report/global

../../../admin/report/global

%2e%2e%2f%2e%2e%2f%2e%2e%2f%61%64%6d%69%6e%2f%72%65%70%6f%72%74%2f%67%6c%6f%62%61%6c

new
URL("https://internal/data/" +
encodeForURIPath(var))

new
URL("https://internal?data=+
encodeForURIParam(var))

URLs to backend REST APIs are built with concatenation instead of URIBuilder (Prepared URI)

- Most publically exposed REST APIs turn around and invoke internal REST APIs using URLConnections, Apache HttpClient or other REST clients. If user input is directly concatenated into the URL used to make the backend REST request then the application could be vulnerable to Extended HPPP.
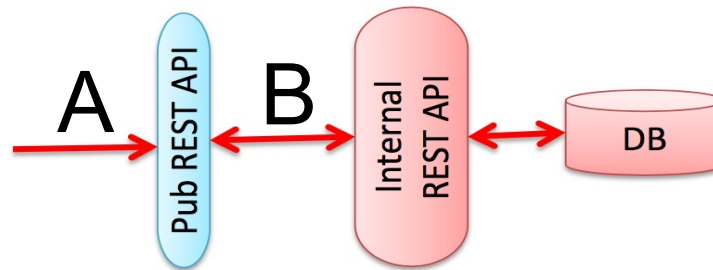
A var = request.getParameter("data");

B new URL(https://internal/data/ + URLEncode(var))

Summary A10:2021 – Server-Side Request Forgery

Concept

- Server-Side refers to programs and operations that run on the server

- Request Forgery means crafting a web request that appears legitimate, but contains malicious input

- Cross-Site Request Forgery (CSRF) forces a user to execute unwanted actions, while authenticated on a web application

Definition

- Attack that forces a server to make a request to an unexpected resource

- Can lead to a wide variety of critical impacts including loss of data, privilege escalation, and more.

- A common vulnerability in N-tiered webservices and microservices

Example

```
addToPage(String url) {
   if (Validator.isValidURL(url)) {
       return fetchContent(url);

addToPage(String url) {
   if (url.domain == "manicode.com") {
       return fetchContent(url);
```

Best Protection Strategies

Validate origin for URLs and IPs when parameters

Ensure authentication and access control on APIs

Setup URL encoding for untrusted parameters

Test and limit network service access with network controls

| Good code | Bad code | User defined input |

Conclusion

Develop Secure Code

- **Use OWASP's Application Security Verification Standard (ASVS) for more comprehensive secure coding requirements**

- **https://owasp.org/www-project-application-security-verification-standard/**

- **Follow the guidance in OWASP's Cheatsheet Series**

- **https://cheatsheetseries.owasp.org/**

- **Use standard security components and frameworks that are a fit for your organization**

Test Continously For Security

- **Automate as much security testing as you can. Consider OWASP ZAP and Dependency Check.**

- **https://owasp.org/www-project-zap/**

- **https://owasp.org/www-project-dependency-check/**

- **Review your applications manually following the OWASP Testing Guide**

- **https://owasp.org/www-project-web-security-testing-guide/**

OWASP Top 10 – 2021 Key InfoSec Concepts

01 Broken Access Control	Access Control – Authorization
02 Cryptographic Failures	Cryptography – Encryption – Cryptanalysis
03 Injection	Protocol Encapsulation – Injecting Commands
04 Insecure Design	Architectural Flaws – Secure Design Patterns
05 Security Misconfiguration	Security Controls – Misconfiguration Vulnerabilities
06 Vulnerable and Outdated Components	End of Life – End of Sale – Unsupported Software
07 Identification and Authentication Failures	Digital Identity – Identification
08 Software and Data Integrity Failures	Software Integrity Verification – (CI) & (CD)
09 Security Logging and Monitoring Failures	Event Logging – Event Monitoring
10 Server-Side Request Forgery	Request Forgery – One Clicks Attacks (CSRF)

OWASP Top 10 – 2021 Definition 1-Liners

01 Broken Access Control	Users Operate Outside Given Permissions
02 Cryptographic Failures	Use of Weak or Incorrect Cryptographic Algorithms
03 Injection	Run Unauthorized Protocols or Commands
04 Insecure Design	Security Flaws not Fixable by Implementation
05 Security Misconfiguration	Vulnerabilities Caused due to Misconfiguration
06 Vulnerable and Outdated Components	Vulnerable, Unsupported or Out of Date Software
07 Identification and Authentication Failures	Permit Attacks that Disclose Identity Attributes
08 Software and Data Integrity Failures	Prevent Inclusion of Functionality from Untrusted
09 Security Logging and Monitoring Failures	Events, Warnings, Errors not Adequately Logged
10 Server-Side Request Forgery	Server forced to make Malicious Outbound Request

OWASP Top 10 – 2021 Examples (Good & Bad)

01 Broken Access Control	`if (user.isAuthorized("ADMIN"))`
02 Cryptographic Failures	`Cipher.getInstance("DESede/CBC/PKCS5Padding");`
03 Injection	`… WHERE username = ' " & Request("username")`
04 Insecure Design	`if(user.isConnectingFromZone(ZONE.SemiTrusted)`
05 Security Misconfiguration	`<fwd name="signin" path="Signin.jsp" />`
06 Vulnerable and Outdated Components	`<artifactId>log4j-core</..><version>2.16.0</…`
07 Identification and Authentication Failures	`response.set("Invalid Username or Password");`
08 Software and Data Integrity Failures	`verifyGitContentWithHash(data);`
09 Security Logging and Monitoring Failures	`log.event("attempted admin access");`
10 Server-Side Request Forgery	`if (Validator.isValidURL(url)) {`

■ Good code   ■ Bad code   ■ User defined input

OWASP Top 10 – 2021 Best Protection Strategies

01 Broken Access Control		**DEBAR**	**D**esign **E**nforce **B**uild **A**ssign **R**efuse
02 Cryptographic Failures		**MUSIC**	**M**anage **U**se **S**ensitive **I**nstrument **C**onfigure
03 Injection		**VECUS**	**V**alidate **E**ncode **C**onfigure **U**se **S**anitize
04 Insecure Design		**DADGUM**	**D**eny **A**pply **D**esign **G**enerate **U**se **M**anage
05 Security Misconfiguration		**VARKAS**	**V**erify **A**ssume **R**ead **K**now **A**pply **S**tudy
06 Vulnerable and Outdated Components		**CORUS**	**C**heck **O**nly **R**emove **U**se **S**tay
07 Identification and Authentication Failures		**FEMALE**	**F**orce **E**nsure **M**anage **A**lert **L**imit **E**nable
08 Software and Data Integrity Failures		**LEVAN**	**L**earn **E**nsure **V**erify **A**pply **N**ote
09 Security Logging and Monitoring Failures		**BEST**	**B**uild **E**nsure **S**tandardize **T**est
10 Server-Side Request Forgery		**VEST**	**V**alidate **E**nsure **S**etup **T**est

It Has Been A Pleasure!

Jim Manico
jim@manicode.com