# APPLICATION SECURITY AWARENESS

## OWASP Top 10

**A01:2021** – **01** – Broken Access Control

**A02:2021** – **02** – Cryptographic Failures

**A03:2021** – **03** – Injection

**A04:2021** – **04** – Insecure Design

**A05:2021** – **05** – Security Misconfiguration

**A06:2021** – **06** – Vulnerable and Outdated Components

**A07:2021** – **07** – Identification and Authentication Failures

**A08:2021** – **08** – Software and Data Integrity Failures

**A09:2021** – **09** – Security Logging and Monitoring Failures

**A10:2021** – **10** – Server-Side Request Forgery (SSRF)

"The OWASP Top Ten is a standard awareness document for developers and web application security professionals. It represents a broad consensus about the most critical security risks to web applications. It was started in 2003 to help organizations and developers with a starting point for secure development. Over the years it's grown into a pseudo standard that is used as a baseline for compliance, education, and vendor tools."

OWASP

# APPLICATION SECURITY AWARENESS

## OWASP Top 10

**A01:2021**
**Broken Access Control**

A02:2021
Cryptographic Failures

A03:2021
Injection

A04:2021
Insecure Design

A05:2021
Security Misconfiguration

A06:2021
Vulnerable and Outdated Components

A07:2021
Identification and Authentication Failures

A08:2021
Software and Data Integrity Failures

A09:2021
Security Logging and Monitoring Failures

A10:2021
Server-Side Request Forgery (SSRF)

Access Control manages or constrains what a user has access to and what actions they can perform, if not implemented correctly this can lead to confidentiality and integrity (modification or deletion of data) not being maintained. Special attention should therefore be paid when implementing Access Control in your applications.

Example:
An attacker instead of navigating through the application, simply types the URL for the admin area of the site that should requires Admin rights to be accessed.

`https://example.com/app/getappInfo` or `https://example.com/app/admin_getappInfo`

If the unauthenticated or non-admin user can access either page, it's a flaw. They should either be redirected to the login page or denied access.

To avoid this, two fundamental security principles should be employed in the design:

1. Secure by Default - by default access is denied and must be granted
2. Complete Mediation - every access by a subject to an object is authorized each time

**For additional information or assistance please talk to your Application Security Team or Security Architect who will be happy to help you.**

# **APPLICATION SECURITY** AWARENESS

●●●●●●●●●●

## OWASP Top 10

A01:2021
Broken Access Control

**A02:2021**
**Cryptographic Failures**

A03:2021
Injection

A04:2021
Insecure Design

A05:2021
Security Misconfiguration

A06:2021
Vulnerable and Outdated Components

A07:2021
Identification and Authentication Failures

A08:2021
Software and Data Integrity Failures

A09:2021
Security Logging and Monitoring Failures

A10:2021
Server-Side Request Forgery (SSRF)

Encryption is used to protect data conserving confidentiality, in transit, at rest and in theory when in use (still very much an evolving area - homomorphic).

*“The first thing is to determine the protection needs of data in transit and at rest. For example, passwords, credit card numbers, health records, personal information, and business secrets require extra protection, mainly if that data alls under privacy laws, e.g., EU's General Data Protection Regulation (GDPR), or regulations, e.g., financial data protection such as PCI Data Security Standard (PCI DSS)”*

Use encryption where necessary paying attention that the cryptographic algorithms are not weak or broken, otherwise an attacker may be able to brute force the decryption and discover the data being protected. As cryptographic algorithms do get broken as the performance of computers increase, it is recommended that cryptographic agility be used in designing the software to allow for migration to newer algorithms at some point in the future.

If proper key management isn't in place then it may be easy for an attacker to compromise a key and the compromised key could then be used to perform a man in the middle attack allowing an adversary to access sensitive information in transit or alternatively they might be able to decrypt sensitive information at rest.

**For additional information or assistance please talk to your Application Security Team or Security Architect who will be happy to help you.**

# APPLICATION SECURITY AWARENESS

## OWASP Top 10

**A01:2021** Broken Access Control

**A02:2021** Cryptographic Failures

**A03:2021** Injection

**A04:2021** Insecure Design

**A05:2021** Security Misconfiguration

**A06:2021** Vulnerable and Outdated Components

**A07:2021** Identification and Authentication Failures

**A08:2021** Software and Data Integrity Failures

**A09:2021** Security Logging and Monitoring Failures

**A10:2021** Server-Side Request Forgery (SSRF)

Injection has been quite a common flaw for some time, there are many different variants of injection that include SQL, NoSQL, OS Command and LDAP Injection amongst others. An example of SQL injection for authentication might be:
You retrieve the username and password from the request

```
username = request.getParameter("username")
```

and

```
password = request.getParameter("password")
```

These values are then used directly in the dynamic construction of an SQL query

```
query = "select * from users where username='" + username + "' and password='" + password + "';"
```

If an attacker were to pass in **' or '1'='1** for both the username and password then this would result in the following query being executed

```
select * from users where username='' or '1'='1' and password='' or '1'='1';"
```

meaning that this would return all the users because **'1'='1'** would always return true for every row.

Using untrusted input in the dynamic construction of queries is not recommended, parameterized queries are the preferred approach. The best defence against injection attacks is input validation using whitelists, although this is not infalable so you should also use other methods. Validation should be performed both on the client but above all on the server in the event that the attacker circumvents the user interface. In addition to validation input sanitization escaping known bad characters is also a good approach.

**For additional information or assistance please talk to your Application Security Team or Security Architect who will be happy to help you.**

OWASP

# **APPLICATION SECURITY** AWARENESS

## OWASP Top 10

**A01:2021**
Broken Access Control

**A02:2021**
Cryptographic Failures

**A03:2021**
Injection

**A04:2021**
**Insecure Design**

**A05:2021**
Security Misconfiguration

**A06:2021**
Vulnerable and
Outdated Components

**A07:2021**
Identification and
Authentication Failures

**A08:2021**
Software and
Data Integrity Failures

**A09:2021**
Security Logging and
Monitoring Failures

**A10:2021**
Server-Side
Request Forgery (SSRF)

*"There is a difference between insecure design and insecure implementation. A secure design can still have implementation defects leading to vulnerabilities that may be exploited. An insecure design cannot be fixed by a perfect implementation as by definition, needed security controls were never created to defend against specific attacks."*

Example:
An insecure design might be a house with an open front (no wall), it doesn't matter if you put the most secure doors available on the back of the house and the most secure windows throughout if the front of the house is open someone could walk right in. To remediate bad design it often means redesigning and rebuilding the system from scratch. The foundations need to be correct.

If the house was designed correctly and had walls on all sides but the doors had locks that are easy to break, this would be an example of an implementation defect and could be remediated once the house was complete.

**For additional information or assistance please talk to your Application Security Team or Security Architect who will be happy to help you.**

# APPLICATION SECURITY AWARENESS

## OWASP Top 10

**A01:2021**
Broken Access Control

**A02:2021**
Cryptographic Failures

**A03:2021**
Injection

**A04:2021**
Insecure Design

**A05:2021**
**Security Misconfiguration**

**A06:2021**
Vulnerable and Outdated Components

**A07:2021**
Identification and Authentication Failures

**A08:2021**
Software and Data Integrity Failures

**A09:2021**
Security Logging and Monitoring Failures

**A10:2021**
Server-Side Request Forgery (SSRF)

Security Misconfigurations can take many forms but are usually because the system hasn't been hardened before go live. To harden a system, you should follow the manufacturers guidelines for hardware and software components.

Remove any software or disable any services that are not required to reduce the attack surface and make sure that default credentials are not being used.

Ensure any debug settings have been removed, log levels set accordingly and that proper exception handling is in place to avoid leaking information to an attacker.

Once you have this configuration in place it is a good practice to create a baseline from it and implement a configuration/change management process to ensure that the baseline image is kept up to date and its state is documented.

Example:
*Configuration UI access from the WAN to your router with default admin credentials could grant an attacker access to your internal network and devices.*

**For additional information or assistance please talk to your Application Security Team or Security Architect who will be happy to help you.**

OWASP

# APPLICATION SECURITY AWARENESS

## OWASP Top 10

- **A01:2021** Broken Access Control
- **A02:2021** Cryptographic Failures
- **A03:2021** Injection
- **A04:2021** Insecure Design
- **A05:2021** Security Misconfiguration
- **A06:2021** **Vulnerable and Outdated Components**
- **A07:2021** Identification and Authentication Failures
- **A08:2021** Software and Data Integrity Failures
- **A09:2021** Security Logging and Monitoring Failures
- **A10:2021** Server-Side Request Forgery (SSRF)

When using third party libraries, it is fundamental that they be kept up to date so that remediations for any known vulnerabilities are propagated down to your projects.

If they are not then an attacker could easily discover or guess that you are using a particular library and try to exploit a known vulnerability.

*A recent example was the log4shell vulnerability that within hours of being made public was being exploited by attackers globally, to take control of systems and breach networks.*

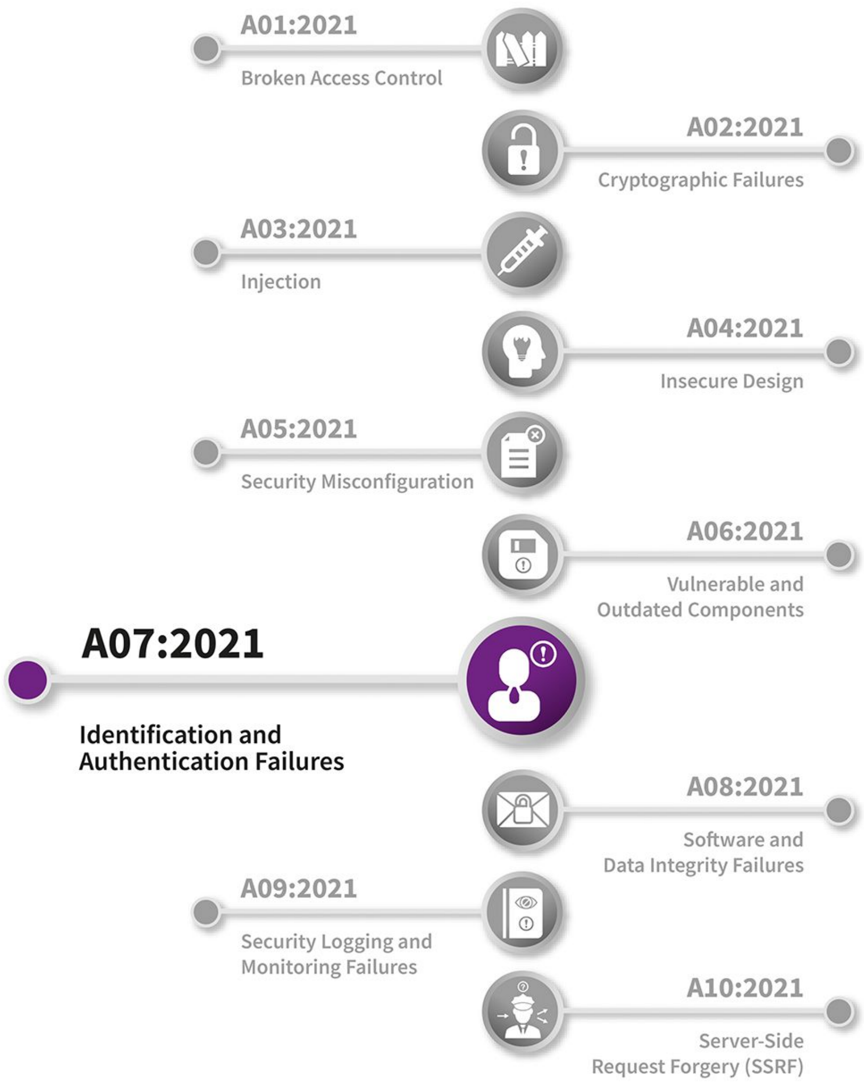
Another reason for upgrading to the latest versions, in particular major versions is that the developers will probably stop supporting older versions and any vulnerabilities that they contain will no longer be remediated.

Software composition analysis tools can be used to alert developers when their dependencies have vulnerabilities, for example OWASP Dependency Track and OWASP Dependency Check.

**For additional information or assistance please talk to your Application Security Team or Security Architect who will be happy to help you.**

# **APPLICATION SECURITY** AWARENESS

## OWASP Top 10



A01:2021
Broken Access Control
A02:2021
Cryptographic Failures
A03:2021
Injection
A04:2021
Insecure Design
A05:2021
Security Misconfiguration
A06:2021
Vulnerable and Outdated Components
A07:2021
Identification and Authentication Failures
A08:2021
Software and Data Integrity Failures
A09:2021
Security Logging and Monitoring Failures
A10:2021
Server-Side Request Forgery (SSRF)

Verifying a users identity is fundamental, so that you can then determine what they can access and you can also create an audit trail of actions they performed. Authentication is the action of verifying with n-factors a users identity. If one of those factors is broken then an attacker could take over the account of a valid user.

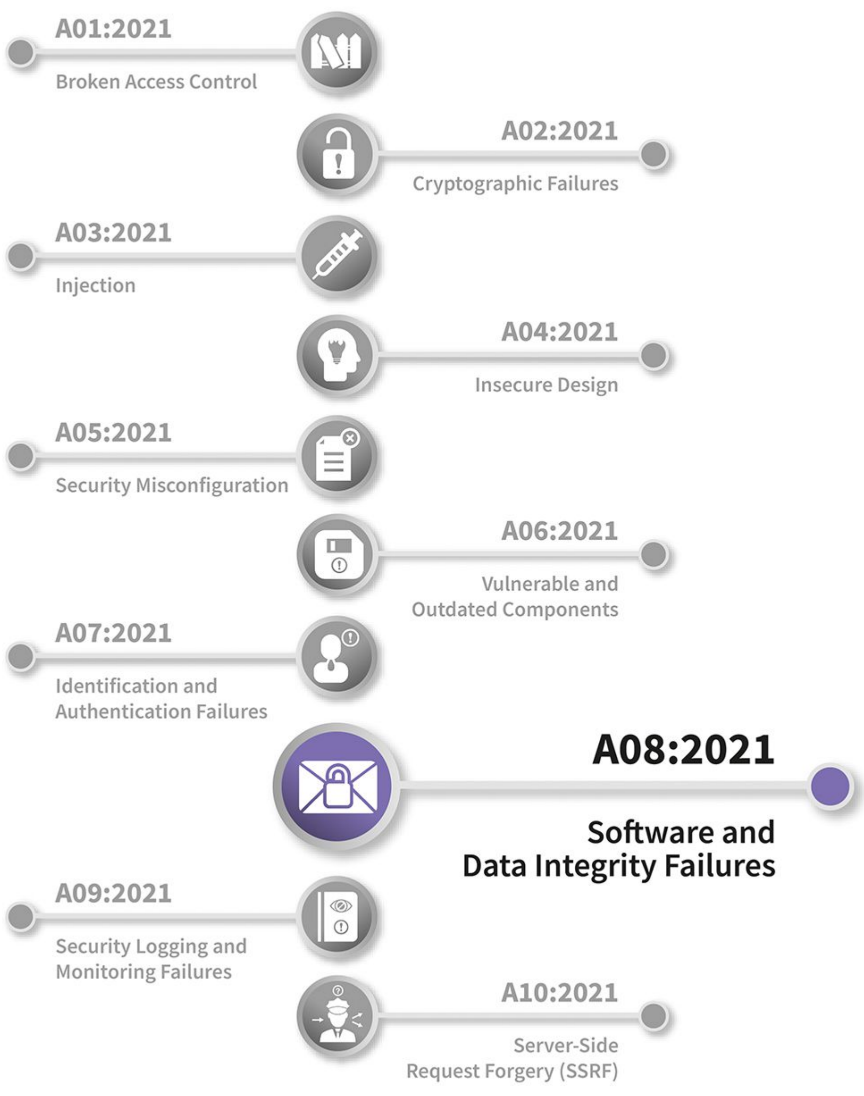
Common failures are caused by:
- Credential Stuffing
- Allowing weak or well-known passwords
- Not blocking when an adversary tries a brute force attack
- Use of ineffective password recovery protection
- Missing or weak cryptography/hashing of stored passwords
- Missing multi-factor authentication
- Session ID exposure in the URL
- Session Fixation
- Sessions not being expired or invalidated

To avoid these issues:
- Block after a number of successive failed login attempts
- Ensure Password retrieval functionality is implemented securely
- Use strong hashing algorithms with salt and do not store the passwords in plain text
- Allow users to enable multi-factor authentication
- Hide session ids, change them after login and invalidate them after logout or after a brief time

**For additional information or assistance please talk to your Application Security Team or Security Architect who will be happy to help you.**

# **APPLICATION SECURITY** AWARENESS

## OWASP Top 10



A01:2021
Broken Access Control
A02:2021
Cryptographic Failures
A03:2021
Injection
A04:2021
Insecure Design
A05:2021
Security Misconfiguration
A06:2021
Vulnerable and Outdated Components
A07:2021
Identification and Authentication Failures
A08:2021
Software and Data Integrity Failures
A09:2021
Security Logging and Monitoring Failures
A10:2021
Server-Side Request Forgery (SSRF)

Many software applications automatically update themselves, if these updates are not digitally signed and subsequently verified during the update process, it may be possible for an attacker to insert malicious code into a download which will then be automatically installed with the next update.

Digitally Signing code, demonstrates the update came from the authors and it hasn't been modified.

This isn't to say that someone working for the authors hasn't inserted malicious code but if the company are vetting their employees and are protecting their network you should feel less at risk with these procedures in place.

*A recent example was the SolarWinds attack which was one of the most far reaching attacks of this type in history.*

Using internal repositories for libraries that are first vetted by security is another way to subvert this type of attack.

**For additional information or assistance please talk to your Application Security Team or Security Architect who will be happy to help you.**

# **APPLICATION SECURITY** AWARENESS

## OWASP Top 10

**A01:2021**
Broken Access Control

**A02:2021**
Cryptographic Failures

**A03:2021**
Injection

**A04:2021**
Insecure Design

**A05:2021**
Security Misconfiguration

**A06:2021**
Vulnerable and Outdated Components

**A07:2021**
Identification and Authentication Failures

**A08:2021**
Software and Data Integrity Failures

## A09:2021

### Security Logging and Monitoring Failures

**A10:2021**
Server-Side Request Forgery (SSRF)

Without logging and monitoring, breaches cannot be detected.
Insufficient logging, detection, monitoring, and active response occurs any time:
- Auditable events, such as logins, failed logins, and high-value transactions, are not logged
- Warnings and errors generate no, inadequate, or unclear log messages
- Logs of applications and APIs are not monitored for suspicious activity
- Logs are only stored locally
- Appropriate alerting thresholds and response escalation processes are not in place or effective
- Penetration testing and scans by dynamic application security testing (DAST) tools (such as OWASP ZAP) do not trigger alerts
- The application cannot detect, escalate, or alert for active attacks in real-time or near real-time

On the flip side, too much logging can:
- Create too much noise
- Be a vector of DOS attacks

So it is necessary to get the balance right and log what is important. Logs should also have proper access control because they may be a source of sensitive information and they should also be centralized, timestamps should be synchronized, and they should be aggregated and correlated in a Security Information and Event Management (SIEM) system where possible.

**For additional information or assistance please talk to your Application Security Team or Security Architect who will be happy to help you.**

# APPLICATION SECURITY AWARENESS

## OWASP Top 10

**A01:2021**
Broken Access Control

**A02:2021**
Cryptographic Failures

**A03:2021**
Injection

**A04:2021**
Insecure Design

**A05:2021**
Security Misconfiguration

**A06:2021**
Vulnerable and
Outdated Components

**A07:2021**
Identification and
Authentication Failures

**A08:2021**
Software and
Data Integrity Failures

**A09:2021**
Security Logging and
Monitoring Failures

**A10:2021**
**Server-Side
Request Forgery (SSRF)**

SSRF flaws occur when web applications fetch remote resources without validating the user supplied URL for example in an Ajax call.

It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network access control list (ACL).

As modern web applications provide end-users with convenient features, fetching a URL becomes a common scenario. As a result and due to modern architectures the incidence of SSRF and severity are increasing.

This could potentially allow an attacker to scan parts of the internal network if not segmented properly, they could also use the FILE schema to read files from the local filesystem, among other types of attack.

There are a number of methods for protected against this, here are a sample few:
- Segment the network to reduce the impact of a successful attack
- Sanitize and valid all untrusted input
- Create allow lists for URL Schemas, ports and destinations

**For additional information or assistance please talk to your Application Security Team or Security Architect who will be happy to help you.**

OWASP