

```

|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|


| 9 | Insecure Data Storage | ```swift
| ```swift
|
| | | func saveData(data: Data)
| | | {
| | |     func
saveData(data: Data) {
|
| | | // Insecure data
storage logic | //
Implement secure data storage logic
|
| | | }
| }
|
| | |
| | |
| | |
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|

| 10 | Insufficient Cryptography | ```swift
| ```swift
|
| | | func encryptData(data:
Data) -> Data { | func
encryptData(data: Data) -> Data {
|
| | | // Insecure
cryptography logic | //
Implement secure cryptography logic
|
| | | }
| }
|
| | |

```

1. Improper Credential Usage:

Non-Compliant Code:

COPY 

```
Non-compliant code with improper credential usage vulnerability
let username = "admin"
let password = "admin123"
```

Description:

The non-compliant code above stores credentials directly within the code. This practice exposes sensitive information, making it vulnerable to unauthorized access if the code is decompiled or if an attacker gains access to the source code.

Compliant Code:

COPY 

```
Compliant code addressing improper credential usage vulnerability
let username = UserDefaults.standard.string(forKey: "username")
let password = KeychainService.loadPassword()
```

Description:

The compliant code addresses the improper credential usage vulnerability by storing credentials securely. It uses the Keychain for password storage, which provides secure and encrypted storage for sensitive information, and UserDefaults for other non-sensitive data.

2. Inadequate Supply Chain Security:

Non-Compliant Code:

```
Non-compliant code with inadequate supply chain security vulnerability  
pod 'InsecureLibrary', '1.0'
```

Description:

The non-compliant code above includes a third-party library (`InsecureLibrary`) without verifying its authenticity or ensuring that it comes from a trusted source. This lack of supply chain security exposes the application to the risk of including malicious or vulnerable components, compromising the overall security of the application.

Compliant Code:

```
Compliant code addressing inadequate  
supply chain security vulnerability  
pod 'SecureLibrary', '1.0'
```

Description:

The compliant code addresses the inadequate supply chain security vulnerability by ensuring that only trusted and validated libraries are included in the project dependencies. It uses a secure library (`SecureLibrary`) from a reputable source, reducing the risk of including malicious or vulnerable components in the application.

3. Insecure Authentication/Authorization:

Non-Compliant Code:

```
Non-compliant code with insecure  
authentication/authorization vulnerability  
func authenticate(username: String, password: String) -> Bool {  
    // Insecure authentication logic
```



```
        return true
    }
}
```

Description:

The non-compliant code above implements insecure authentication logic. It may use weak or outdated authentication methods, such as storing passwords in plain text or using insufficiently strong encryption. Additionally, the authorization logic may lack proper validation, allowing unauthorized access to sensitive functionalities.

Compliant Code:

```
Compliant code addressing insecure
authentication/authorization vulnerability
func authenticate(username: String, password: String) -> Bool {
    // Perform secure authentication using hashed passwords, multi-
    factor authentication, etc.
    return false // Placeholder return value
}
```

COPY 

Description:

The compliant code addresses the insecure authentication/authorization vulnerability by implementing secure authentication and authorization mechanisms. This may include using hashed passwords, multi-factor authentication, role-based access control, or other industry-standard practices to enhance the security of the application.

4. Insufficient Input/Output Validation:

Non-Compliant Code:

```
Non-compliant code with insufficient input validation vulnerability
func validateUsername(username: String) -> Bool {
```

COPY 


```
// Insecure input validation logic
return true
}
```

Description:

The non-compliant code above implements insufficient input validation logic. It may fail to adequately validate user inputs, leaving the application vulnerable to various attacks such as SQL injection, cross-site scripting (XSS), or command injection.

Compliant Code:

```
Compliant code addressing insufficient input validation vulnerability
func validateUsername(username: String) -> Bool {
    // Implement secure input validation logic to prevent common
    vulnerabilities
    return false // Placeholder return value
}
```

COPY 

Description:

The compliant code addresses the insufficient input validation vulnerability by implementing secure input validation logic. This may include sanitizing and validating user inputs to ensure they adhere to expected formats and do not contain malicious content, thereby reducing the risk of exploitation.

5. Insecure Communication:

Non-Compliant Code:

```
Non-compliant code with insecure communication vulnerability
func sendData(data: Data) {
```

COPY 

```
// Insecure communication logic using HTTP  
}
```

Description:

The non-compliant code above communicates data over an insecure channel (HTTP) without encryption or proper security measures. This exposes sensitive information to interception or tampering by malicious actors, compromising the confidentiality and integrity of the data.

Compliant Code:

```
Compliant code addressing insecure communication vulnerability  
func sendData(data: Data) {  
    // Implement secure communication logic using HTTPS with TLS  
    encryption  
}
```

COPY 

Description:

The compliant code addresses the insecure communication vulnerability by using HTTPS with TLS encryption for data transmission. This ensures that data is securely encrypted during transit, protecting it from interception or tampering by attackers and maintaining the confidentiality and integrity of the communication.

6. Inadequate Privacy Controls:

Non-Compliant Code:

```
Non-compliant code with inadequate privacy controls vulnerability  
func trackUserLocation() {  
    // Inadequate privacy control logic  
}
```

COPY 

Description:

The non-compliant code above lacks proper privacy controls for tracking user location. It may collect or share location data without user consent or fail to provide adequate options for users to control their privacy settings, potentially violating user privacy rights and exposing sensitive information.

Compliant Code:

COPY 

```
Compliant code addressing inadequate privacy controls vulnerability
func trackUserLocation() {
    // Implement privacy controls to obtain user consent and provide
    options to opt out
}
```

Description:

The compliant code addresses the inadequate privacy controls vulnerability by implementing privacy controls for tracking user location. This may include obtaining explicit user consent before collecting location data, providing clear privacy policies, and offering options for users to opt out or control their privacy settings.

Devops

DevSecOps

secure coding

Application Security

Published on



DevSecOpsGuides

 Add blog description

MORE ARTICLES

RR Reza Rashidi



Attacking AWS

As businesses increasingly migrate their operations to Amazon Web Services (AWS), the significance o...

RR Reza Rashidi



Attacking Android

In this comprehensive guide, we delve into the world of Android security from an offensive perspecti...

RR Reza Rashidi



Attacking iOS

In this comprehensive guide, we delve into the world of iOS security from an offensive perspective, ...