Introduction to the OWASP Top 10 – 2021

Risk A4: Insecure Design

Key Concepts

Key Concept 1: Architectural Flaws

- Flaws of Omission: ignoring a threat or security requirement
- Flaws of Commission: bad design, e.g. client-side authorization
- Flaws of Realization: design is good, but errors in implementation

Key Concept 2: Secure Design Patterns

- Patterns in code that eliminate vulnerabilities: e.g. the pattern of privilege separation which limits the amount of code running with elevated privileges

Key Concept 3: Reference Architectures

- Template solutions to combine the software you are using: e.g. Java Platform, Enterprise Edition (Java EE), the reference architecture restricts calls to the database from the web layer

Additional Information

There are 3 main categories of Architectural Flaws. An example of a flaw of omission (where a security requirement or threat is ignored) is storing a password in a file is a security architecture flaw of omission. This is because the architect assumes attackers would not have access to the file system.

An example of a flaw of commission (flaw of bad design) is a client/server software that performs authentication on the client instead of the server. This allows server-side authentication to be bypassed using a modified client that skips the check.

An example of a flaw of realization (where the design is good, but errors exist in the implementation) is you require a File Transfer Protocol (FTP) service listening on your internet facing infrastructure. FTP is an unencrypted protocol and therefore considered insecure.

An example of a secure design pattern (a pattern in code that eliminates vulnerabilities) is the pattern of privilege separation. This pattern limits the amount of code running with elevated privileges across your web application or API.

An example of reference architecture (a set of template solutions to combine the software you are using) is in the Java Platform, Enterprise Edition (Java EE), where the reference architecture restricts calls to the database from the web layer.

Definition

The collection of security flaws within the web application that cannot be attributed to or fixed by implementation.

- Broad category: captures missing or ineffective controls
  - Architectural flaws of omission, where security requirements have not been provided or threats are not considered
  - Access has not been appropriately restricted

- Design flaws have a different root causes to implementation defects
  - Good implementation cannot fix insecure design

Additional Information

The definition of A4 of 2021 – Insecure Design – is design that has flaws which cannot be attributed to or fixed by changing the implementation. Insecure design looks at the risks associated with flaws in design and architecture.

The reason why this is a broad category is compounded by the fact that you can have many different types of architectural flaws, where security requirements have not been provided or threats have not been considered. Another example of insecure design is access within your web application or API that has not been appropriately restricted.

Good implementation cannot fix insecure design. The way to fix insecure design is to go back to the drawing board and see what was missed and why.

Example

Bad Example

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

Good Example

```
userAccess() {
 if (user.isConnectingFromZone(ZONE.SemiTrusted))
 {
 //authorize user based on zone
```

```
  }
}
```

Additional Information

For our good and bad example, assume our web application has different network zones that we trust. An authenticated user is given access based on where they are connecting from.

As a bad example, of what not to do, somewhere within the code we only check if the user is authorized. This creates an insecure design risk for our application because we are authorizing the user based only on their entitlements and not considering what network zone they are connecting from.

As a good example, we have sat down with the network team and defined our network zones. We have then entered that design within the user access model of our application. One of the zones we have defined is a semi-trusted zone.

Based on where the user is connecting from, we provide them access to a particular set of resources. With this example we have shown the difference between zoned user access and defined a set of levels of trust for our application. Where we have not implemented this and only authorize access based on entitlements, we have created an insecure design risk.

Challenges

- 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
  - The fix is not easy to implement
  - Huge time expenses or starting over

- Reference architectures are not updated for latest frameworks
  - Developers want to use the latest framework or library which might not be compatible with the security architecture of your web app or API

Additional Information

We now understand how insecure design happens, let us look at why it is a common issue.

To add to the fact that secure design patterns are often incorrectly customized, when you have insecure design, it is hard to admit it, because the fix will not be easy to implement.

The latest cool frameworks or libraries might not be compatible with the existing security architecture of your web application or API.

Best Protection Strategies

DADGUM

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

Additional Information

Above you have some of the best protection strategies against A4 of the Top 10 – Insecure Design.