Business logic vulnerabilities

Business logic vulnerabilities are flaws in the design and implementation of an application that allow an attacker to elicit unintended behavior. This potentially enables attackers to manipulate legitimate functionality to achieve a malicious goal. These flaws are generally the result of failing to anticipate unusual application states that may occur and, consequently, failing to handle them safely.

One of the main purposes of business logic is to enforce the rules and constraints that were defined when designing the application or functionality. Broadly speaking, the business rules dictate how the application should react when a given scenario occurs. This includes preventing users from doing things that will have a negative impact on the business or that simply don't make sense.

Identifying them often requires a certain amount of human knowledge, such as an understanding of the business domain or what goals an attacker might have in a given context. This makes them difficult to detect using automated vulnerability scanners.

How do business logic vulnerabilities arise?

Business logic vulnerabilities often arise because the design and development teams make flawed assumptions about how users will interact with the application.

Logic flaws are particularly common in overly complicated systems that even the development team themselves do not fully understand. To avoid logic flaws, developers need to understand the application as a whole. This includes being aware of how different functions can be combined in unexpected ways. Developers working on large code bases may not have an intimate understanding of how all areas of the application work.

Someone working on one component could make flawed assumptions about how another component works and, as a result, inadvertently introduce serious logic flaws.

What is the impact of business logic vulnerabilities?

Fundamentally, the impact of any logic flaw depends on what functionality it is related to. If the flaw is in the authentication mechanism, for example, this could have a serious impact on your overall security. Attackers could potentially exploit this for privilege escalation, or to bypass authentication entirely, gaining access to sensitive data and functionality.

Flawed logic in financial transactions can obviously lead to massive losses for the business through stolen funds, fraud, and so on.

You should also note that even though logic flaws may not allow an attacker to benefit directly, they could still allow a malicious party to damage the business in some way.

Examples of business logic vulnerabilities

Excessive trust in client-side controls

A fundamentally flawed assumption is that users will only interact with the application via the provided web interface. This is especially dangerous because it leads to the further assumption that client-side validation will prevent users from supplying malicious input

Failing to handle unconventional input

One aim of the application logic is to restrict user input to values that adhere to the business rules. For example, the application may be designed to accept arbitrary values of a certain data type, but the logic determines whether or not this value is acceptable from the perspective of the business

To implement rules like this, developers need to anticipate all possible scenarios and incorporate ways to handle them into the application logic. In other words, they need to tell the application whether it should allow a given input and how it should react based on various conditions. If there is no explicit logic for handling a given case, this can lead to unexpected and potentially exploitable behavior.

Making flawed assumptions about user behavior

One of the most common root causes of logic vulnerabilities is making flawed assumptions about user behavior. This can lead to a wide range of issues where developers have not considered potentially dangerous scenarios that violate these assumptions

Trusted users won't always remain trustworthy:

Applications may appear to be secure because they implement seemingly robust measures to enforce the business rules. Unfortunately, some applications make the mistake of assuming that, having passed these strict controls initially, the user and their data can be trusted indefinitely. This can result in relatively lax enforcement of the same controls from that point on.

Users won't always supply mandatory input

Many transactions rely on predefined workflows consisting of a sequence of steps. The web interface will typically guide users through this process, taking them to the next step of the workflow each time they complete the current one. However, attackers won't necessarily adhere to this intended sequence. Failing to account for this possibility can lead to dangerous flaws that may be relatively simple to exploit.

Using tools like Burp Proxy and Repeater, once an attacker has seen a request, they can replay it at will and use forced browsing to perform any interactions with the server in any order they want. This allows them to complete different actions while the application is in an unexpected state.

Note that this kind of testing will often cause exceptions because expected variables have null or uninitialized values. Arriving at a location in a partly defined or inconsistent state is also likely to cause the application to complain. In this case, be sure to pay close attention to any error messages or debug information that you encounter.

Domain-specific flaws

you will encounter logic flaws that are specific to the business domain or the purpose of the site.

The discounting functionality of online shops is a classic attack surface when hunting for logic flaws. This can be a potential gold mine for an attacker, with all kinds of basic logic flaws occurring in the way discounts are applied.

To identify these vulnerabilities, you need to think carefully about what objectives an attacker might have and try to find different ways of achieving this using the provided functionality. This may require a certain level of domain-specific knowledge in order to understand what might be advantageous in a given context.

Providing an encryption oracle

Dangerous scenarios can occur when user-controllable input is encrypted and the resulting ciphertext is then made available to the user in some way. This kind of input is sometimes known as an "encryption oracle".An attacker can use this input to encrypt arbitrary data using the correct algorithm and asymmetric key.

This becomes dangerous when there are other user-controllable inputs in the application that expect data encrypted with the same algorithm. In this case, an attacker could potentially use the encryption oracle to generate valid, encrypted input and then pass it into other sensitive functions

How to prevent business logic vulnerabilities

- Make sure developers and testers understand the domain that the application serves
- Avoid making implicit assumptions about user behavior or the behavior of other parts of the application

- Maintain clear design documents and data flows for all transactions and workflows, noting any assumptions that are made at each stage.
- Write code as clearly as possible. If it's difficult to understand what is supposed to happen, it will be difficult to spot any logic flaws. Ideally, well-written code shouldn't need documentation to understand it. In unavoidably complex cases, producing clear documentation is crucial to ensure that other developers and testers know what assumptions are being made and exactly what the expected behavior is.
- Note any references to other code that uses each component. Think about any side-effects of these dependencies if a malicious party were to manipulate them in an unusual way.