

WHAT ARE BUSINESS LOGIC VULNERABILITIES?

Business Logic Vulnerabilities are flaws in the design and implementation of an application that allows an attacker to elicit unintended behavior.

EXAMPLE 1 - CHANGE ANOTHER USERS PASSWORD

FUNCTIONALITY

The application has a password change for end users and administrators..

- End users need to fill out the username, existing password, new password and confirm new password fields.
- Administrators only need to fill out the username, new password and confirm new password fields.

ASSUMPTION

The client-side interface presented to users and administrators is different but the password change is controlled for both users by the same function.

CODE

```
String existingPassword = request.getParameter("existingPassword");
if (null == existingPassword) {
    trace("Old password not supplied, must be an administrator");
    return true;
}
else
{
    trace("Verifying user's old password");
    ...
}
```

ATTACK

A regular user submits a request to change another users password by simply not supplying the existing password.

EXAMPLE 2 - BYPASS CHECKOUT FUNCTIONALITY

FUNCTIONALITY

The application has a "place an order" functionality that follows the following stages:

- Browse the product catalog and add items to the shopping basket.
- Return to the shopping basket and finalize the order.
- Enter the payment.
- Enter delivery information.

ASSUMPTION

The developers assumed that users would always access the stages in the intended sequence.

ATTACK

The user proceeds directly from stage 2 to stage 4, finalizing the order for delivery without paying for the order.

- Browse the product catalog and add items to the shopping basket.
- Return to the shopping basket and finalize the order.
- Enter the payment.
- Enter delivery information.

EXAMPLE 3 - BEATING A BUSINESS LIMIT

FUNCTIONALITY

A banking application allows users to transfer funds between bank accounts. As a precaution against fraud, the application prevents users from transferring a value greater than \$10,000.

ASSUMPTION

The developers put a check in the place to ensure that no transaction greater than \$10,000 is allowed to go through.

```
bool CAuthCheck::RequiresApproval(int amount) {
    if (amount <= m_apprThreshold)
        return false;
    else return true; }
...
```

ATTACK

The developers overlooked the possibility that a user would attempt to process a transfer for a negative amount. Any negative number would clear the approval test because it is less than the threshold value.

Therefore, a user who wants to transfer \$20,000 from account A to account B could simply initiate a transfer - \$ 20,000 from account B to account A bypassing the antifraud defense.

EXAMPLE 4 - CHEATING ON BULK DISCOUNTS

FUNCTIONALITY

An e-commerce website allows users to order software products and qualify for bulk discounts if a suitable bundle of items was purchased. The following are the steps involved in the bulk discount functionality.

1. Users adds items in basket
2. If one of the items qualifies for a bulk discount, a discount is applied on the entire cart.
3. User purchases order.

ASSUMPTION

Users will purchase the chosen bundle after the discount is applied

ATTACK

User can exploit this logic flaw by performing the following steps:

1. User adds items in the basket including item that gives the user a bulk discount.
 2. The discount is applied on the entire cart.
 3. User goes back to the cart and removes the item that entitled him to a discount.
 4. Although the item is removed , the discount is still approved, and the user purchases the order at a discounted price.
-

HOW TO FIND AND EXPLOIT BUSINESS LOGIC VULNERABILITIES?

- Map the application. Make note of each and every component in the application and how it operates.
 - If you have access to the code, review the code responsible for each component.
- For each component determine:
 - The potential business flow.

- The assumptions that could have been made by the developers / architects during the design phase.
 - Test each component for all possible use cases that are outside of the intended business flow.
-

HOW TO PREVENT BUSINESS LOGIC VULNERABILITIES

- Ensure that there is proper documentation of the applications design that outlines every assumption that the designer(s) made.
- Mandate that all source code is properly commented and includes the following items:
 - The purpose and intended use of each code component.
 - The assumptions made by each component about anything that is outside of its direct control
 - References to all client-side code that uses the component.
- Write code as clearly as possible.
- Perform security-focused code reviews of the applications design.