Business logic vulnerabilities

Introduction
What is logic?
    Tought experiment
What is a business logic vulnerability
    Examples
    Properties of a logic vulnerability
Development process
    Waterfal model
    V-Model
    Agile

Introduction

There's not a lot of information out there on business logic vulnerabilities. I challenge you to try it, go to google right now and search "business logic vulnerabilities". You will find a very good article on it from portswigger and from owasp but they are very limited and don't explain the concepts very well in my opinion. Today i'm going to talk to you about logic, what it is, how it can go wrong and how can test for logic issues. I do believe logic is something you can train and there are things you can do to help improve this process.

We will go over all of these things and much more, so let's not waste any more time and dive right in!

What is logic?

Our logic is always flawed it's as simple as that. There's nothing wrong with this either, we as humans simple suck at forseeing all the possible issues that could arise. We develop many different risk mitigation strategies but you know as well as me that ruling out all the risk is impossible because we simple can't forsee all the possible variables that play into a situation.

In business situation, we have the same issue. Whether it be banking, a shoe store or a company that sells websites, they all run into the same issues. We all run into

the same issues in our daily lifes. I'm going to give you a simple thought experiment to demonstrate.

Tought experiment

What i have in front of me:

- 1 knife
- 1 slice of bread
- 1 Jar nutella

When i ask the people for instructions on how to make a sandwhich, a few things will get called forward. Before you read on, i want you to think about how you would tell me to make a sandwhich.

I would tell a person of average intelligence to make a sandwhich by putting the knife in the jar and smearing it on the bread but now i want you to imagine explaining this to a person who never held a knife before. Who never opened a jar before. Who never even heard of a sandwhich before. This is why logic issues arise.

We are often using software of which we do have some basic idea's so in a way you can compare us users as caveman with very basic knowledge of tool usage, but suddenly they gave you a tablesaw. Now you have made stone tools yourself and you do know how to use those but this magical thing you've never seen before, that's new to you.

It's the exact same thing with software, we are using this software and even regular users will have trouble and couse unintended behaviour. This behaviour may not always lead to a flaw and definitly not always to a security flaw but it does occur quite often. Much more often than people think.

What is a business logic vulnerability

Now that we know what we understand under the meaning logic, we can also talk about how vulnerabilities arise. In recent times, everything needs to go faster and faster with the coming up of agile development methodologies but these issues have existed for ages! We humans suck at prediciting the consequences of our

actions and even if we could forsee all of our actions, we can't take into account what others will do.

There's inherent risk in working on a software program over multiple weeks, months or even weeks. Everyone who has come into contact with software development will be able to confirm this, the longer a project runs for, the more changes it undergoes. Those changes can be in staff by hiring new members and removing workers that have a lot of knowledge. They can also be in strategy, for example if the project becomes too big for one person to handle, an analyst might be hired and a software tester which do require a change in structure.

This risk can be mitigated but in my opinion, there are a couple of ways a business logic vulnerability can sneak in:

- Most companies have an allergy to documentation it seems. They don't document what they should or they keep it on a workers personal laptop where it is never shared. This means that if a feature needs to be adapted or build upon existing code, the developers and analysts often don't know how the old code works anymore. This is an excellent oppertunity for logic issues! Lacking knowledge of existing features might make it so that some properties of that features have been overlooked, this creates a great entry point for the juicy logic vulnerabilities we are looking for.
- The lack of documentation also means that when teams have to work together or create an integration, its really hard for them to communicate effectively and this may in turn lead to missed properties of the feature.
- The logic is so complex the people working on it get confused themselves, allowing them to make mistakes
- A bad process is in place, allowing these types of issues to slip through several layers of testing

All of these seem logical but they are not, that's why they slipped through the net so it's better if i give you some examples.

Examples

- We have a feature to pay for an order but it's not documented propperly.

- The developers have to edit the system after a year to include mobile banking but since they don't have documentation, they miss a piece of code. It's hidden deep inside the existing code but it allows you to enter 999999999 in the credit card field and it will automatically pay for you throughout the application.
- This has been done by a developer in the original code but has been put in a method that should normally never be reached, he did this for testing purposes.
- With the update of the login system, the developers had no idea what this code did due to it's complexity and they put it the payment procedure, introducing a logic flaw because anyone can now pay by entering 999999999 in the credit card field.

- When you click a link "forgot password" there's basically two systems that will work together.
  - Reset password generation link
  - Mail sending link
- If we enter the following payload

```
POST /resetPass.php
{
"email":"victim@mail.com",
"email":"attacker@mail.com"
}
```

- We can trigger the server that generates the link to generate a link for victim@mail.com but the mail server might only check the last "email" parameter it sees and send it off to attacker@mail.com

---

- Imagine you are a triager at a bug bounty platform, but your system has gotten very complex to triage bugs and you might accidentaly have part of your admin panel exposed to the public without even knowing it because the developers forgot to add authentication to that page.

- There might not be any code reviews going on, which allows for the following statement. The OR should be an AND here:

```
SELECT * FROM users WHERE username='$username' OR password='$password'
```

Properties of a logic vulnerability

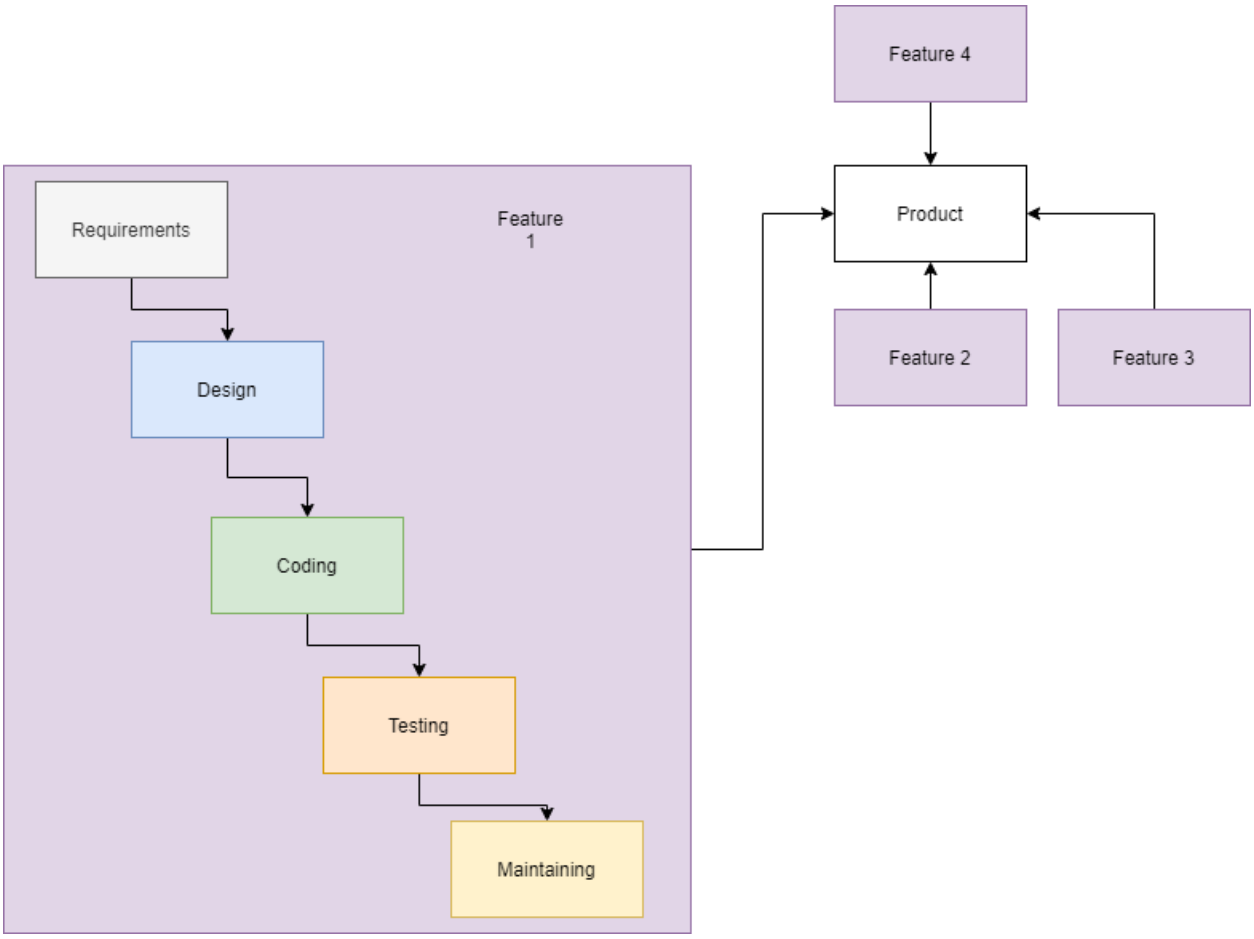
These vulnerabilities can exist in the product for years at a time without being discovered. Business logic vulnerabilities have some inherent properties that allow them to stay undetected for long periods of time.

- You can't detect them with automatic process efficiently. It's really hard to teach a robot logic. 🙂
- These issues are inherently hard to detect manually since they already slipped passed the analyst, developer and tester who could have all caught it.
- They range in severity from low to critical in severity depending on the functionality and how it's impacted

Development process

Let's have a look at the development process because i do think it will allow us to understand these vulnerabilites better. There are several methodologies when it comes to developing software. I'll briefly cover every one of the most prominent methodologies but know that there are a lot more than i am discussing here. It doesn't matter though, they are all weak to business logic. As long as humans work on a program, it's bound to have logic vulnerabilities. They might not be impactful but it's so easy to overlook that i believe we are far from having discovered all of them.

Waterfal model

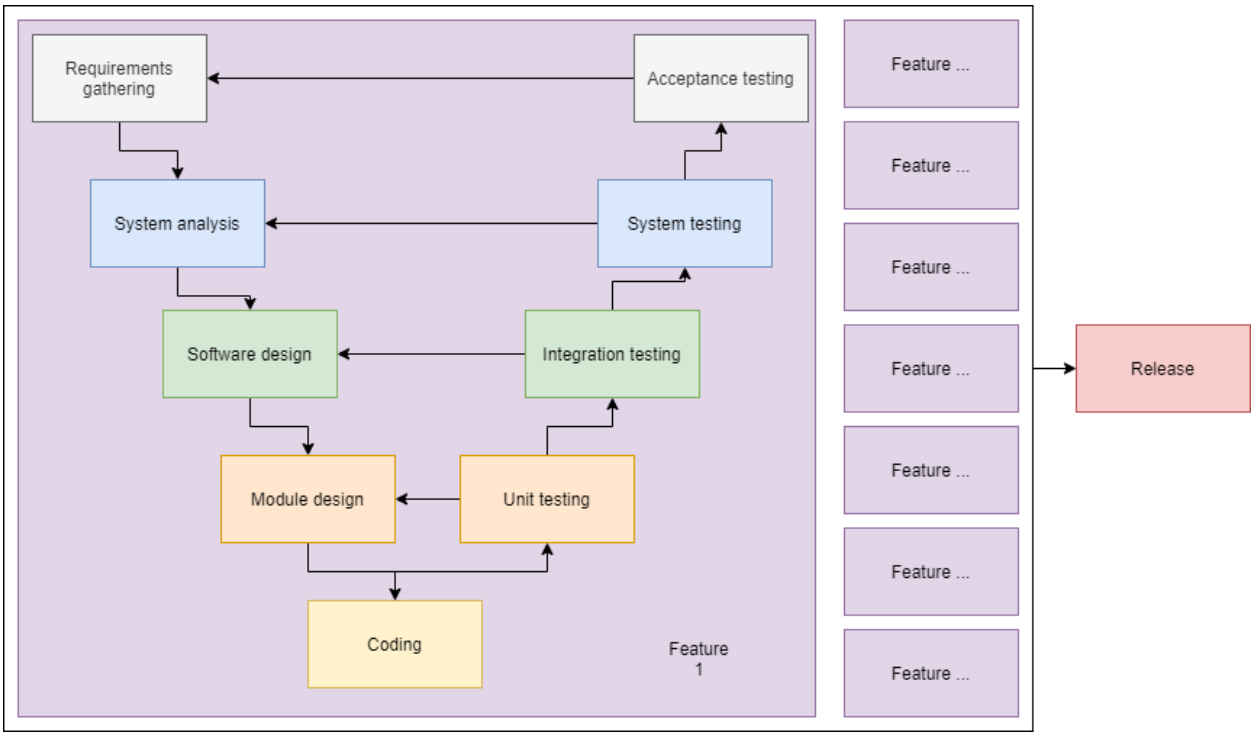


As you can see in the waterfall model, we have several issues that can help us discover logic flaws, since we can now see at what stages they occur. As we can see here, the requirements are created which leads to the design of an application. From there the developers can get to work and create what needs to be created. The testers go to work, trying to destroy what the developers have built. After all checks have been approved, we enter a phase of maintenance, trying to fix any issues that arise with the feature.

After all this is said and done, several features will be stacked upon eachother to build an application. When a new feature has to be built, this is usually not a big issue. Old code is not being touched and new code is being created, usually with insufficient unit tests, and usually without the proper documentation. As the project is young and still full of motivated people that got to start something new, this usually is not a big problem but later down the line when features need to be built upon or refactored, the issues start to arise. This means that whenever we are dealing with a target that seems to bring out major releases every now and

again that we might be talking about the waterfall or V-model. These methodologies share a lot of similarities when it comes to entry points for logic vulnerabilties so we will continue in the next chapter.

V-Model



As we can see in the V-model, every stage of design and coding is tested propperly. This has some advantages and disadvtanges but it appears to have less room for logic flaws. While i agree that the more basic logic flaws can be mitigated by the acceptance testing. There's still a huge surface for business logic vulnerabilities. Above the waterfall model, we also have some extra complexity in there.

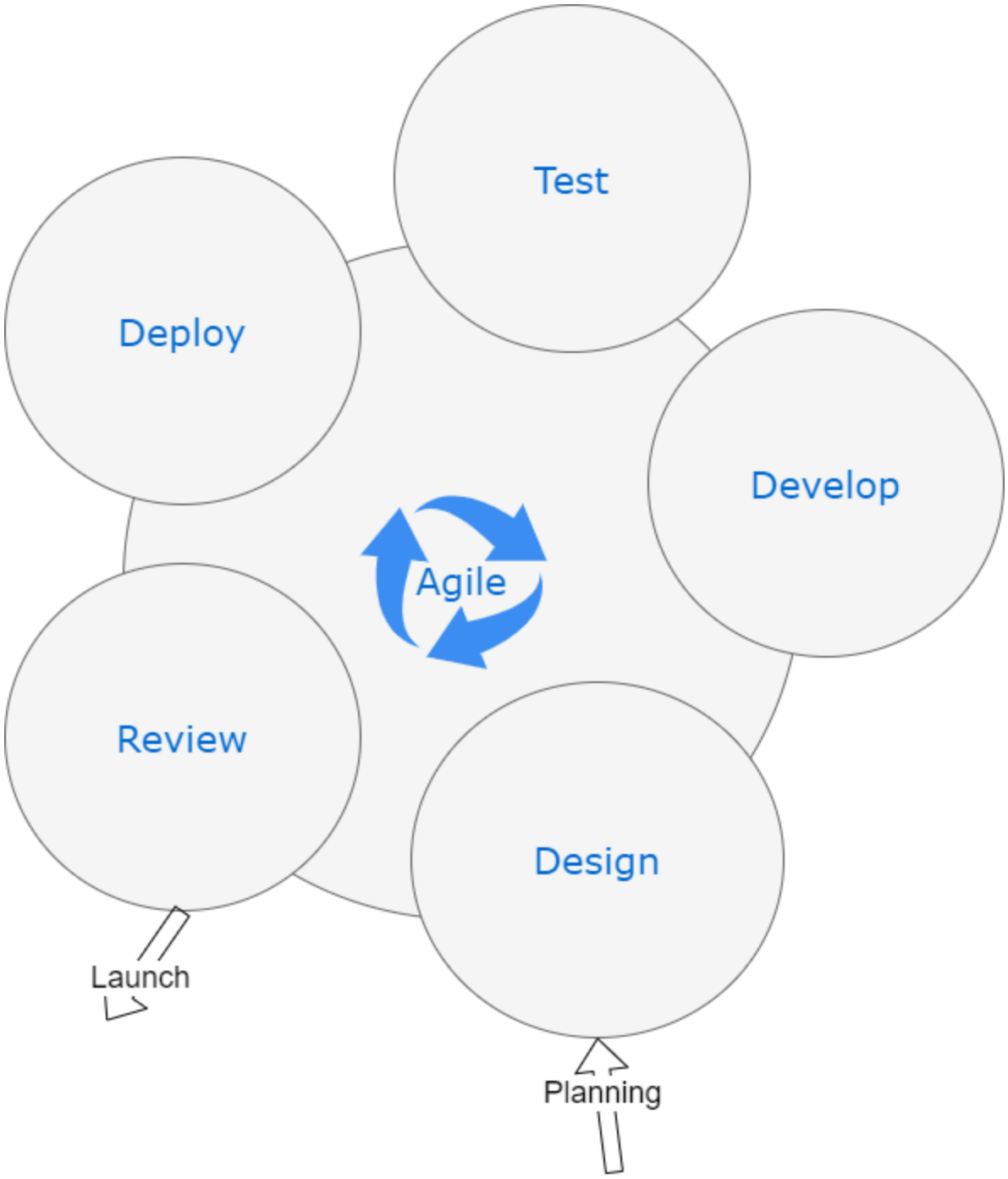
While it might seem the extra complexity may be the cause of a lot of logic vulnerabilities, it's not. In this case the extra complexity makes it so that it's easier to mask business logic vulnerabilities. Extra attention to the process draws away attention from the logic itself and people tend to start focussing more on the existing stories instead of thinking up scenario's that have not been described.

The biggest attack surface for us will be when a new release hits the testing enviornment. This is when all the processes that have been followed and all the

features that have been created come together. We have to test the new features that have been developed and the old features that have been refactored. This is why it's important to subscribe to any release notes if possible. So to summarize:

- Test when new releases hit production
- Test the new features
- Test the refactored features
- Think about how features can interact with one another and if there might be a possible flaw in the interaction
- Try to subscribe to any newsletters if possible

Agile



As you can see, in agile, things move in a cycle, and often a 2 week cycle. This is fast and it allows for the easy introduction of business logic flaws due to either:

- Regression errors
    - A messy developer can easily have an old piece of code on the laptop and commit it, after which it gets smuggles to production in a HUGE code

commit with many files.

    - A developer can easily make a mistake due to the time pressure or missing documentation
- If agile is not being used propperly (which is often the case), the process becomes a lot more prone to business errors
    - User stories are often not chopped up into their smallest workeable tasks, which allows for missed requirements due to higher complexity
    - Sprints are often way too big, taking in a huge scope due to
        - Tasks not being estimated propperly
        - Sprint capacity being overrated
        - Sprints might take way too long, taking in a huge scope

    All of these points make it easier for business logic flaws to appear and we should definitly test:

    - New features
    - Our targets every release cycle (usually 2 weeks)
    - Refactored features

Mitigation

To mitigate all of these topics that we discussed in previous chapters, we have to train our brain to recognize these flaws in logic, take the time to propperly analyze features and document them and write good unit test to prevent regression flaws.

Extra-curriculum activities

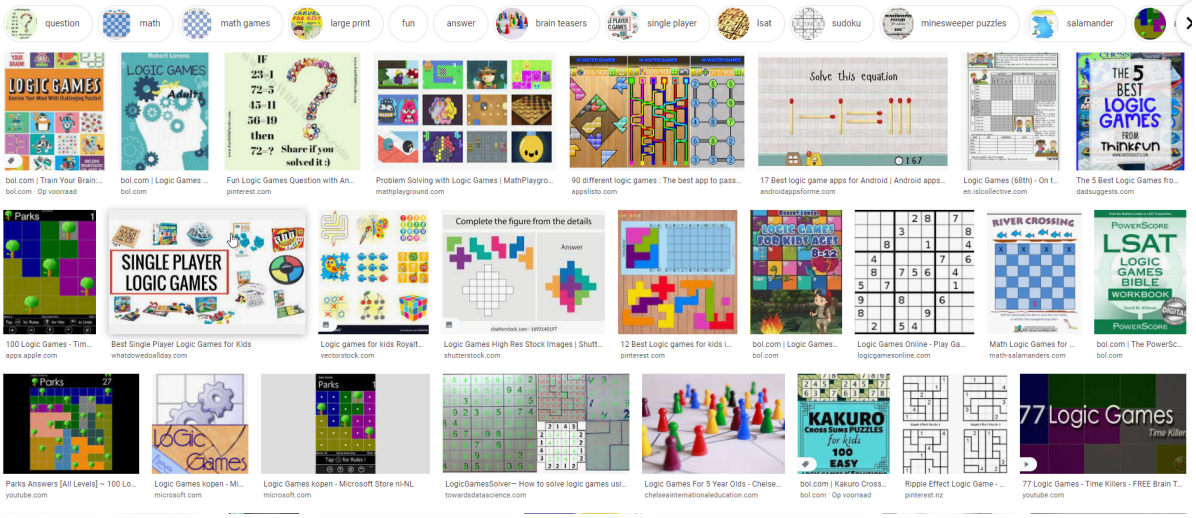
We can train our logic skills easily, there are hunderds of activities you can do to increase your logic skills. My favourite are escape rooms! This is a very fun way to increase your logic thinking skills and it will help you spot those details more easily. A fun and free way to get started is to look up any of the 100's of free escape rooms you can print at home and start playing right away.

Free online escape games

In the time of corona, many escape room creators have been busy creating online games, and many of them have generously made their games available at no charge. I

https://escapethereview.co.uk/free-online-escape-games/#be-the-escape

What's also very useful are these logic games. There are sooooo many of these it's impossible to even count them.



So even though logic seems like a hard skill to train, it seems like enough people are willing to take on the challenge and have developed learning material that can help us. It may seem dumb but yes, we can train our own logic.