# Modelling Risk in Open-Source Software

Róisín Ní Bhriain - 23269640
Sneha Dechamma Mallengada Suresh - 23262168
Course: MCM Computing
Supervisor: Darragh O'Brien

# Introduction

- ➔ Popularity of Open-Source Software has shown a steady increase over time
  - ◆ 3.6 million repos depend on the top 50 open-source projects
- ➔ The use comes with the risk of software vulnerabilities
  - ◆ Attackers can exploit these
- ➔ Prediction of risk can be used to explore risk
  - ◆ Software metrics
  - ◆ Project Activity
  - ◆ Vulnerability Data

# Research Question

Can we provide developers with an effective risk measure when choosing between multiple open-source components?

# Key Terms

- Maven
  - Open-source build automation & project management tool
- Time series analysis
  - Fitted model based on past data can predict future values
- ARIMA
  - Autoregressive integrated moving average
- AutoARIMA
  - Tests parameter combinations to determine the best model

# Background Research

# Vulnerability Propagation

- ➔ Only 1.2% directly use vulnerable code
- ➔ Small packages can affect  375, 607 packages in the Maven ecosystem
  - ◆ jackson-databind, netty-codec-http
  - ◆ CVEs can affect a large number of projects
- ➔ The more dependencies the more complex it is to find vulnerabilities in dependencies

# Project Metadata Analysis

- ➔ Project activity level is an indicator of survival
  - ◆ Number of commits, time-to-fix
  - ◆ Long-time Contributors
- ➔ Less than 50% of abandoned projects find new contributors
- ➔ Standard measurement for commits is:
  - ◆ Number of commits per month

# Vulnerability CVE Data Analysis

- ➔ Predicting number of vulnerabilities per month is useful
- ➔ ARIMA is a very popular method of prediction
  - ◆ Seasonality not a factor
- ➔ Datasets from the NVD were used in every study we analysed
  - ◆ Prediction of CVEs
- ➔ Most models fall flat at the three month mark

# Case Studies

## Log4j Vulnerability

- ➔ Discovered in November 2021 in popular logging library
- ➔ Gateway to gain control of the machine
- ➔ Many projects unaware they were affected

## Heartbleed Vulnerability

- ➔ Discovered in April 2014 in popular cryptography library
- ➔ Receive private information after crafting similar messages
- ➔ Many servers unaware they were affected

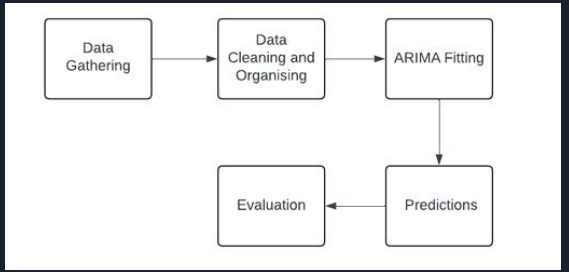
# Methodology

# Data & Dependencies

- ➔ Different GitHub projects to test the algorithm
- ➔ GitHub API for project activity
- ➔ NVD API for vulnerability Data
- ➔ Each Maven Dependency as a node
- ➔ NetworkX graph
- ➔ Keywords and libraries extracted for prediction
- ➔ Colour-coded

| Data Source | Purpose | Type |
|---|---|---|
| GitHub projects | Find Dependencies | Maven-dependency trees |
| GitHub API | Project Meta-data | API |
| NVD API | Vulnerability Prediction | CVE API |

# Predictions




## Project Activity

- GitHub API
  - All available data gathered
- Commits, average time-to-fix issues or both
- AutoARIMA model
- Predicted value for the next month returned

## Vulnerabilities

- NVD API
  - Call made for each keyword
- Number of vulnerabilities per month
- AutoARIMA model
- Predicted value for the next month returned

# Risk Calculations

$$projectActivityScore = (x/numDaysToFixIssues) * 10$$

$$projectActivityScore = (numCommits/x) * 10$$

$$vulnerabilityScore = (x/vulsCountPerMonth) * 10$$

$$overallScore = (projectActivityScore + vulnerabilityScore)/2$$

- ➔ User-decided acceptable levels
- ➔ Options for project activity score
- ➔ Combination of project activity and vulnerability for overall score

| Score | Risk Level |
|---|---|
| <0 | Not Enough Data |
| 0 - 2.5 | Low Risk |
| 2.5-5 | Medium Risk |
| 5-7.5 | High Risk |
| >7.5 | Severe Risk |

# Results

# Example Final Graph

**Key:**

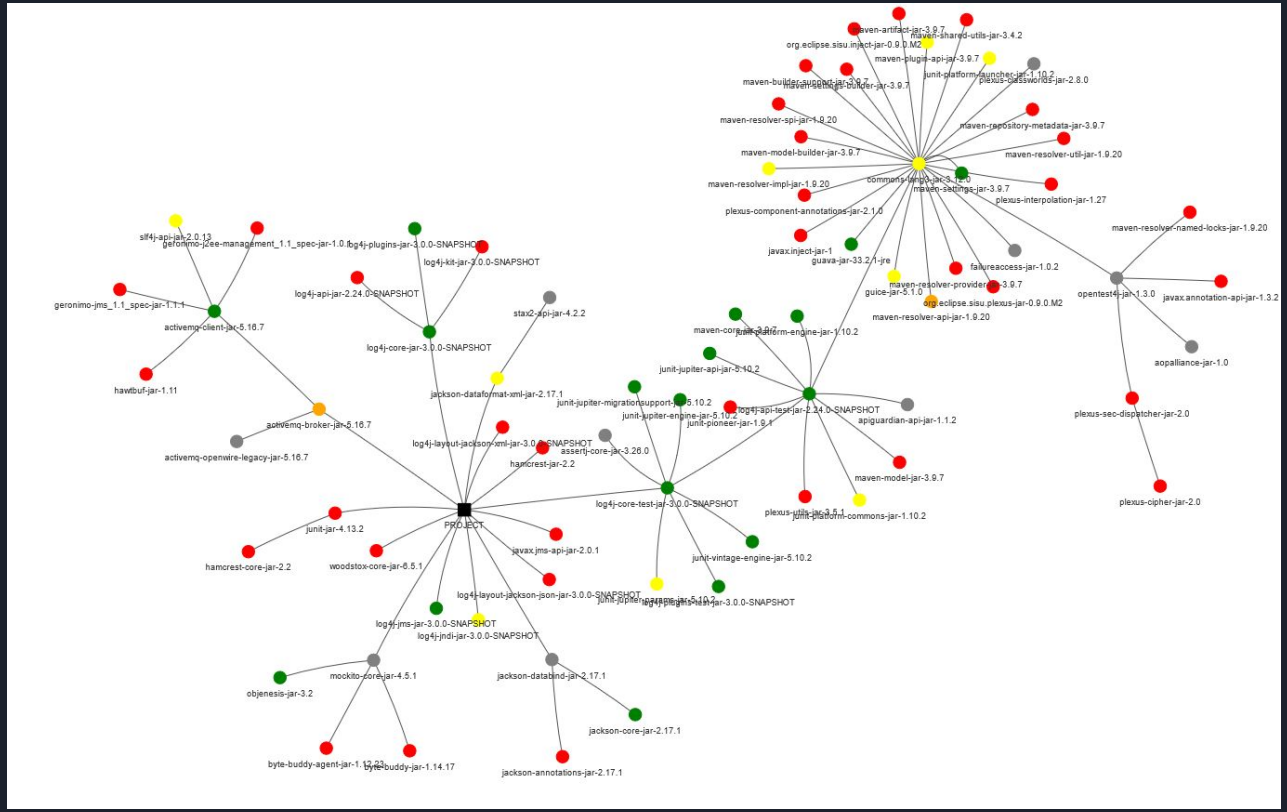
Severe Risk

High Risk

Medium Risk

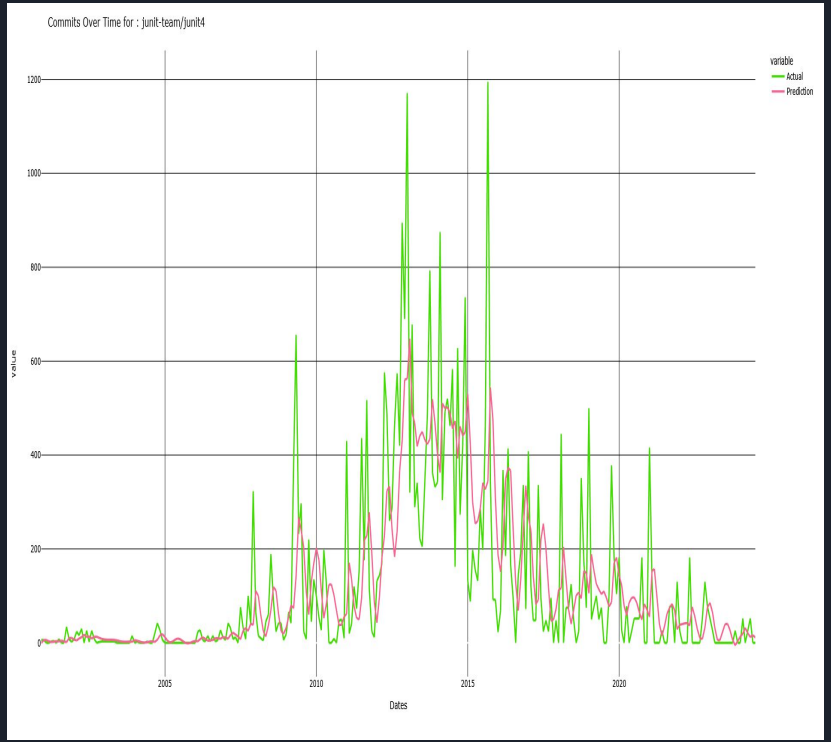
Low Risk

Not Enough Data



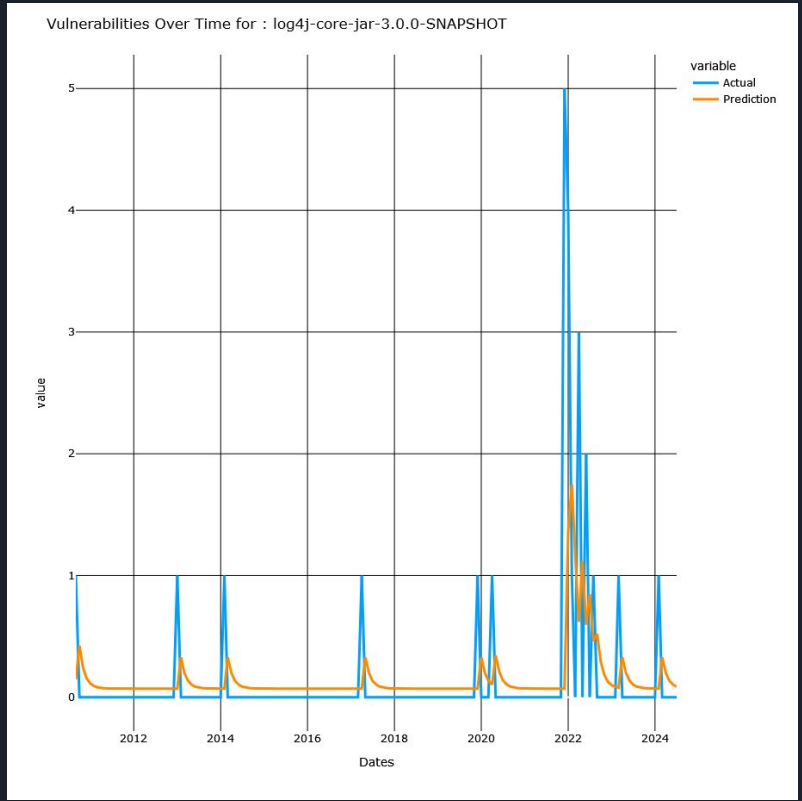
# Example Graph of Project Activity Model

- JUnit4 commits over time
- Clear to see that activity has declined in recent years
- 2012 to 2017 were high commit years
  - At its peak over 200 commits per month
- Fitted values are fairly accurate as can be seen in the graph
- Predicted value for the next month
  - 10.9951



# Example Graph of Vulnerability Data Model

- Log4j vulnerabilities over time
- Clear where many were released in 2021
  - 5 released in one month
- Reasonably accurate fitted values in subsequent months
- Predicted value for the next month
  - 0.07898



# Evaluations

| Prediction | MAE | RMSE |
|---|---|---|
| Commits per month | 123.18 | 213.96 |
| Vulnerabilities per month | 0.145 | 0.476 |

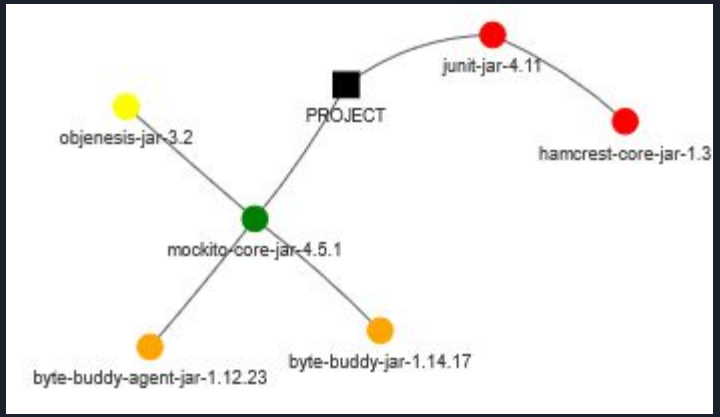
- ➔ Log4j sample project
- ➔ Commonly used metrics
  - ◆ Mean Absolute Error (MAE)
  - ◆ Root Mean Squared Error (RMSE)
- ➔ Commits have larger standard deviation
  - ◆ 152.79
- ➔ Vulnerabilities have lower standard deviation
  - ◆ 0.432

# Technical Challenges

- ➔ Graphing dependencies and dependencies of dependencies etc.
- ➔ Choosing of ARIMA parameters
  - ◆ Decided on AutoARIMA
  - ◆ PACF and ACF graphs
  - ◆ Differencing data until ADF test below 0.05
- ➔ Connecting dependencies to GitHub URLs
  - ◆ Done using a mapping file

# Tool Uses

- ➔ Comparing similar dependencies
  - ◆ JUnit vs **Mockito**
- ➔ Exploring risk in current dependencies
  - ◆ Finding risky modules



# Student Contribution

**Róisín**

- ➔ Gitlab setup + Assigning Issues
- ➔ Literature Review
  - ◆ Writing + read  42 papers
- ➔ Software
  - ◆ All code
- ➔ Practicum Paper
  - ◆ All writing

**Sneha**

- ➔

# Conclusion

- ➔ Growing reliance on OSS
  - ◆ Challenges regarding security
- ➔ Better overview of dependency risk
- ➔ Focused on Maven dependency trees
- ➔ Compromised on accuracy to predict many dependencies
- ➔ Tool could aid developers in
  - ◆ Choosing less risky modules
  - ◆ Discovering already used risky dependencies

# References

Thank You For Listening!

Questions?