ANALYSING THE SECURITY RISKS OF INTEGRATING AI MODELS IN DEVSECOPS PROCESSES

BACKGROUND

- GITHUB COPILOT IS AN AI CODE GENERATOR THAT REDUCES THE AVERAGE DURATION OF CODING PROJECTS BY **55%**
- CONSIDERING ITS RISE IN POPULARITY IN RECENT YEARS, THIS REPORT AIMS TO INVESTIGATE THE AI MODEL'S ABILITY TO RESOLVE SECURITY ISSUES BY SCANNING AND COMPARING PROJECTS BEING SUBJECTED TO MANUAL AND COPILOT-GENERATED FIXES

METHODOLOGY

1. PROJECT CONFIGURATION:
- ESTABLISHED VERSION CONTROL USING GITHUB
- CLONED THE REPOSITORIES THRICE, WITH ONE AS A CONTROL AND THE OTHER TWO FOR FIXES

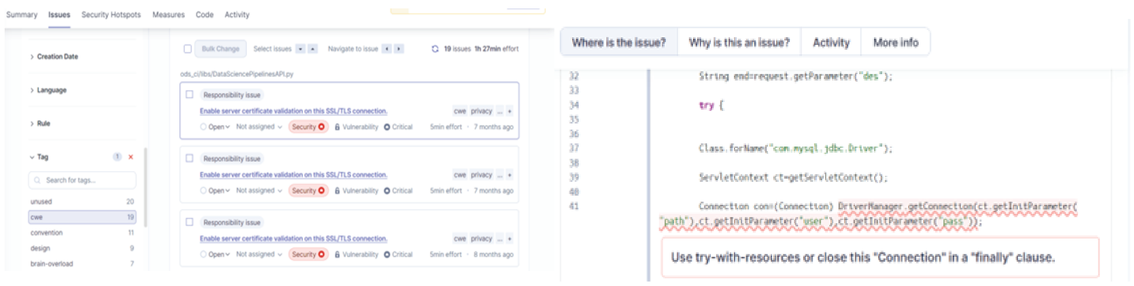
2. SONARCLOUD INTEGRATION:
- INTEGRATED CHOSEN REPOSITORIES WITH VSCODE AND SONARCLOUD
- IDENTIFIED SECURITY ISSUES WITHIN REPOSITORIES

3. FIRST WAVE – MANUAL FIXING
- MANUALLY FIXED CODE WITH SECURITY ISSUES
- SCANNED THE PROJECT AFTER COMMITTING CHANGES

4. SECOND WAVE – COPILOT FIXING:
- INTEGRATED COPILOT INTO VSCODE
- UTILIZED PROMPTS ON VULNERABILITIES IDENTIFIED TO IMPLEMENT SECURITY FIXES
- SCANNED THE PROJECT AFTER COMMITTING THE CHANGES

5. COMPARATIVE ANALYSIS:
- CONDUCTED A COMPARATIVE ANALYSIS OF BOTH METHODS USING DATA IMPORTED FROM SONARCLOUD AND ORGANIZING IT INTO TABLES

SONARCLOUD ANALYSIS



SONARCLOUD'S CATEGORY SYSTEM WAS UTILIZED ALONG WITH ITS IN-DEPTH CODE ANALYSIS TOOLS TO SCAN AND IDENTIFY SECURITY ISSUES

SECURITY METRICS

VULNERABILITIES

VULNERABILITIES ARE DEFINED AS SECTIONS OF CODE THAT CAN BE EXPLOITED BY HACKERS, WITH ALL OF THEM RESULTING IN ONE OR MORE CWES.

SECURITY HOTSPOTS

SECURITY HOTSPOTS ARE DEFINED AS SECTIONS OF SECURITY-SENSITIVE CODE THAT REQUIRE MANUAL REVIEW TO ASSESS WHETHER A VULNERABILITY EXISTS.

CWES

THE CWE IS A CATEGORY SYSTEM FOR SOFTWARE WEAKNESSES AND VULNERABILITIES. CWES ALSO OCCUR IN BUGS AND CODE SMELLS WHICH MAY LEAD TO SECURITY ISSUES IF NOT ADDRESSED ACCORDINGLY.

RESULTS

V_n (2 d.p.)	Project 1	Project 2	Project 3	Project 4	Project 5	Project 6
Initial V_n	1.67	0.83	0.78	13.89	1.36	10.44
V_n after manual fix	0.00	0.00	0.00	0.00	0.00	0.00
% Change	- 100%	- 100%	- 100%	- 100%	- 100%	- 100%
V_n after Copilot fix	0.00	0.27	0.00	0.00	0.43	0.00
% Change	- 100%	- 67.5%	- 100%	- 100%	- 68%	- 100%

TABLE. 6.1. COMPARISON OF NUMBER OF VULNERABILITIES PER 1000 LINES OF CODE (VN), AND PERCENTAGE CHANGE ACROSS 6 PROJECTS BEFORE AND AFTER BOTH METHODS OF

SH_n (2 d.p.)	Project 1	Project 2	Project 3	Project 4	Project 5	Project 6
Initial SH_n	1.33	2.78	1.56	18.52	0.91	20.89
SH_n after manual fix	0.00	0.27	0.00	0.00	0.00	2.61
% Change	- 100%	- 90%	- 100%	- 100%	- 100%	- 88%
SH_n after Copilot fix	0.00	1.08	0.00	0.00	0.43	7.83
% Change	- 100%	- 61%	- 100%	- 100%	- 53%	- 63%

TABLE. 6.2. COMPARISON OF NUMBER OF SECURITY HOTSPOTS PER 1000 LINES OF CODE (SHN), AND PERCENTAGE CHANGE ACROSS THE 6 PROJECTS BEFORE AND AFTER BOTH

CWE_n (2 d.p.)	Project 1	Project 2	Project 3	Project 4	Project 5	Project 6
Initial CWE_n	1.67	2.5	2.47	32.41	1.82	20.89
CWE_n after manual fix	0.00	0.00	0.00	0.00	0.00	0.00
% Change	- 100%	- 100%	- 100%	- 100%	- 100%	- 100%
CWE_n after Copilot fix	0.00	0.28	0.39	0.00	0.43	0.00
% Change	- 100%	- 89%	- 84%	- 100%	- 76%	- 100%

TABLE. 6.3. COMPARISON OF NUMBER OF CWES PER 1000 LINES OF CODE (CWEN), AND PERCENTAGE CHANGE ACROSS THE 6 PROJECTS BEFORE AND AFTER BOTH METHODS OF

ME (3 s.f.)	Project 1	Project 2	Project 3	Project 4	Project 5	Project 6
Manual fixing ME	100%	100%	100%	100%	100%	100%
Copilot fixing ME	100%	67%	100%	100%	75%	100%

TABLE. 6.4. COMPARISON OF MITIGATION EFFECTIVENESS (ME) FOR BOTH METHODS ACROSS THE 6 PROJECTS

EXAMPLE OF COPILOT ERROR

A. ORIGINAL CODE WITH CWE-89: IMPROPER NEUTRALIZATION OF SPECIAL ELEMENTS USED IN AN SQL COMMAND

```
            command.CommandText = sql;
            _context.Database.OpenConnection();

            using var dataReader = command.ExecuteReader();
            dataReader.Read();
            order.OrderId = Convert.ToInt32(dataReader[0]);
        }

        sql = ";\nINSERT INTO OrderDetails (" +
            "OrderId, ProductId, UnitPrice, Quantity, Discount" +
            ") VALUES (@OrderId, @ProductId, @UnitPrice, @Quantity, @Discount)";
        foreach (var (orderDetails, i) in order.OrderDetails.WithIndex())
        {
```

B. CHANGE COPILOT IMPLEMENTED:

```
            command.CommandText = sql;
            _context.Database.OpenConnection();

            using var dataReader = command.ExecuteReader();
            dataReader.Read();
            order.OrderId = Convert.ToInt32(dataReader[0]);
        }

        sql = "INSERT INTO OrderDetails (" +
            "OrderId, ProductId, UnitPrice, Quantity, Discount" +
            ") VALUES (@OrderId, @ProductId, @UnitPrice, @Quantity, @Discount)";
            command.CommandText = sql;
            _context.Database.OpenConnection();

            using var dataReader = command.ExecuteReader();
            dataReader.Read();
            order.OrderId = Convert.ToInt32(dataReader[0]);
        }

        sql = ";\nINSERT INTO OrderDetails (" +
            "OrderId, ProductId, UnitPrice, Quantity, Discount" +
            ") VALUES (@OrderId, @ProductId, @UnitPrice, @Quantity, @Discount)";
        foreach (var (orderDetails, i) in order.OrderDetails.WithIndex())
        {
```

C. COPILOT ISSUE:

THE CHANGE REMOVES THE DIRECT CONSTRUCTION OF SQL QUERIES FROM USER-CONTROLLED DATA, WHICH IS A POSITIVE SECURITY IMPROVEMENT. HOWEVER, OPILOT'S SUBSEQUENT MODIFICATION REINTRODUCES A POTENTIAL SQL INJECTION VULNERABILITY BY NOT USING PARAMETERIZED QUERIES FOR THE 'INSERT INTO ORDERDETAILS' OPERATION. THIS OPENS THE POSSIBILITY OF SQL INJECTION ATTACKS, WHICH ENABLE ATTACKERS TO MANIPULATE DATA

CONCLUSION

IN CONCLUSION:

- MANUAL FIXING IS SUPERIOR TO COPILOT-RECOMMENDED FIXING WHEN IT COMES TO RESOLVING SECURITY ISSUES.
- WHILE COPILOT DOES INDEED DEMONSTRATE EFFICIENCY, IT TENDS EITHER MISUNDERSTAND THE USER'S PROMPT OR OMIT CONTEXT WHEN IT COMES TO ENSURING THE CODE ADHERES TO OTHER SECURITY MEASURES, HENCE SOMETIMES INTRODUCING SECURITY ISSUES
- COPILOT STILL REMAINS A USEFUL TOOL TO PROGRAMMERS ALL OVER THE WORLD, HENCE METICULOUS PROMPT ENGINEERING AND BEST PRACTICES SHOULD BE UTILIZED TO ENSURE THAT COPILOT'S RECOMMENDED CODE ADHERES TO THE BEST SECURITY PRACTICES AND DOES NOT VIOLATE ANY CWE

FUTURE WORK

IN THE FUTURE, WE AIM TO

- SET UP A MORE COMPREHENSIVE DEVSECOPS FRAMEWORK FOR AUTOMATED TESTING AND DATA PROCESSING FOR MORE EXTENSIVE TESTING FOR A LARGE VARIETY OF SECURITY ISSUES IN ACCORDANCE WITH THE ENTIRE CWE LIST
- CONDUCT MORE RESEARCH ON THE COPILOT AI MODEL AND THE GPT MODEL IT IS BASED ON TO GAIN A GREATER UNDERSTANDING ON THE ROOT CAUSE OF PROMINENT SECURITY ISSUES AND HOW WE CAN RESOLVE THEM.

Member:
Yoon Sae Young, Anglo-Chinese School (Independent)
Mentor:
Loh Wan Jing, Defence Science and Technology Agency

YDSP Young Defence Scientists Programme
DSTA Defence Science & Technology Agency
DSO NATIONAL LABORATORIES