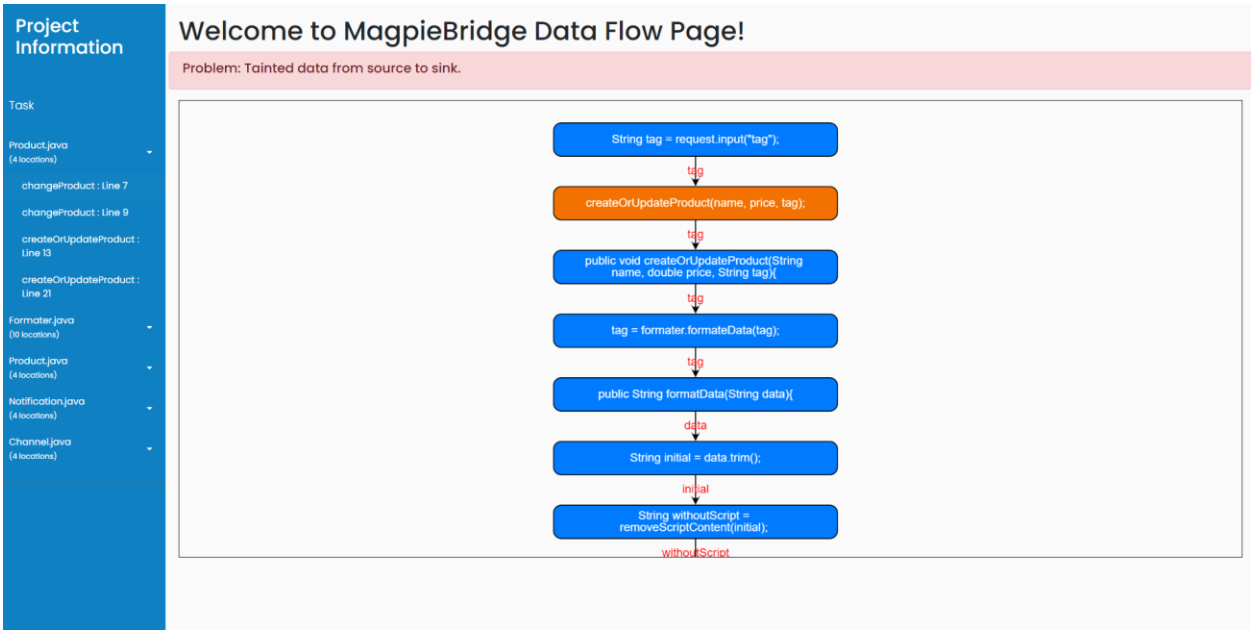
User Study Document of MagpieBridge extension prototype

Introduction:

When we use the static analysis tool to find issues in our program a lot of time it returns false-positive results. To find out whether the issue reported by static analysis is false positive or not we have to check the data flow leading to the point of issue. Generally, this process is done by following the error report and visiting the line of the file. If the error report has lots of file positions, then it becomes very difficult. To improve the process, we are proposing a visualization of the data flow path.

Prototype:

You will be given a prototype that has two views. One view is the error message with data flow visualization. There also a sidebar which provide information about the file and line number for each of the data flow node. It can also have method name with the line number. One can go to the data flow node directly by clicking the sidebar links.



The other view is the view of a code editor which shows the code that the analysis is performed on. Each file of the program has an individual code editor page.

Code

Product.java

```
1  public class Product{
2    public void changeProduct(){
3
4        DataReceiver request = new DataReceiver("post");
5        String name = request.input("name");
6        String price = request.input("price");
7        Stirng tag = request.input("tag");
8
9        createOrUpdateProduct(name, price, tag);
10   }
11
12
13   public void createOrUpdateProduct(String name, double price, String tag){
14       Formater formater = new Formater();
15       Database db = new Database();
16       Table productTable = db.table("product");
17       boolean productExist = productTable.productExist(name);
18
19       name = formater.formateData("name");
20       price = formater.formateData("price");
21       tag = formater.formateData("tag");
22
23       if(! productExist) {
24           String productId = productTable.createNew(name, price, tag);
25       } else {
26           String productId = productTable.getProductByName(name);
```

You can go to the line of code of the code editor directly from the visualization page by clicking the node of the data flow. The line will be marked as selected.

Code

DataReceiver.java

```
1  public class DataReceiver{
2
3    protected String type;
4
5    public DataReceiver(String type){
6        this.type = type;
7    }
8
9    public String input(String name){
10       if(this.type == "POST") {
11           return post(name);
12       } else if(this.type = "GET"){
13           return get(name);
14       }
15   }
16
17   private String post(String name){
18       Post post = new PostRequest();
19       Data values = post.values();
20
21       if(values.contains(name)) {
22           return values.indexValue(name);
23       } else {
24           return null
25       }
26   }
```

Task Overview:

Here you must perform three tasks related to the issue of three different programs. Each task consists of a single data flow visualization page. There is also a code view for each file of the programs. You must identify whether the issue of the task is false positive or not.

Taint Analysis:

Taint analysis is a technique that is mostly used to detect security vulnerabilities. It tracks information flow through a program. Untrusted input and sensitive data are often the information that is tracked in a taint analysis. If any data is collected from a user or some untrusted source then we must ensure that the data is secure to perform the next operations like saving into the database, sending data to another source etc. So, we check the data with different validation methods to make sure the data is secure.

If taint analysis provides some error in the code, we sometimes need to check the path whether there is any checking for data available in the path because the static analysis may not identify the function that is used for validating the data. If we find that there is some checking available for data, then we say that error is a false positive error. That it is giving us an error which is actually not an error.