2020

Enabling Static Program Analysis Using A Graph Database

Jialun Liu
Wright State University

ENABLING STATIC PROGRAM ANALYSIS USING A GRAPH DATABASE

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

by

Jialun Liu

B.E. Engineering in Internet of Things Engineer,
Xi'an University of Art and Science, People's Republic of China,2018

2020
Wright State University

Wright State University
GRADUATE SCHOOL

December 7, 2020

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Jialun Liu ENTITLED ENABLING STATIC PROGRAM ANALYSIS USING A GRAPH DATABASE BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science.

Junjie Zhang, Ph.D.
thesis Director

Mateen M. Rizki, Ph.D.
Chair, Department of Computer
Science and Engineering

Committee on
Final Examination

Junjie Zhang, Ph.D.

Meilin Liu, Ph.D.

Bin Wang, Ph.D.

Barry Milligan, Ph.D.
Interim Dean of the Graduate School

ABSTRACT

Liu, Jialun. M.S., Department of Computer Science and Engineering, Wright State University, 2020. *ENABLING STATIC PROGRAM ANALYSIS USING A GRAPH DATABASE.*

This thesis presents the design, the implementation, and the evaluation of a database-oriented static program analysis engine for the PHP programming language. This engine analyzes PHP programs by representing their semantics using a graph-based data structure, which will be subsequently stored into a graph database. Such scheme will fundamentally facilitate various program analysis tasks such as static taint analysis, visualization, and data mining. Specifically, these complex program analysis tasks can now be translated into built-in declarative graph database operations with rich features. Our engine fundamentally differs from other existing static program analysis systems that mainly leverage intermediate representation (IRs) to perform analysis. Specifically, our engine leverages the graph-based output of the "Uchecker" system; it translates the output into graph files with the form of CSV and then directly inserts them into a graph database. Our engine offers several unique advantages. First, static program analysis tasks could now be implemented using database queries. Second, our engine supports interactive program analysis through the graph database. Third, through our designed query templates, our engine can perform fine-grained program analysis such as data flow analysis on selected variables. We have applied our engine to analyze PHP programs collected from public program repositories such as GitHub and WordPress, where the experimental results have demonstrated the great effectiveness and efficiency of our system.

Contents

List of Figures

List of Tables

Listings

Acknowledgment

I would like to express my sincere appreciation to my advisor, Dr. Junjie Zhang for mentoring me to complete this thesis. I could not have imagined having a better advisor and mentor for my master's program study at Wright State University. Besides my advisor, I would like to thank my thesis committee: Dr. Meilin Liu and Dr. Bin Wang for their insightful comments and suggestions to my thesis work. My sincere thanks also go to everyone who has volunteered time to help me during my graduate thesis work at Wright State University. I would like to express my thank to my families for their support during my academic career. Finally, I would like to thank my girlfriend for supporting me during my master's degree program and her belief in me.

Dedicated to

My Father, Mr. Weiping Liu,

My Mother, Mrs. Yifang Dou, and

My Girlfriend, Miss. Yuanyuan Tong

Abbreviations

AST — Abstract Syntax Tree

IR — Intermediate Representation

OOP — Object Oriented Programming

PHP — PHP Hypertext Preprocessor

Introduction

Static program analysis is to analyze a program without actually executing it. It is widely used in various areas, such as software testing, vulnerability discovery, and bug tracking. It is instrumental in optimizing compilers for producing efficient and safe code, but it could also be used for automatic error detection and other tools to help programmers. Compared with dynamic program analysis methods that need to execute the analysis process, static program analysis methods can analyze the entire program based on its structure and grammar. Therefore, static program analysis can provide a faster and more stable analysis process. A few static program analysis engine have been implemented, where salient examples include Bauhaus Toolkit [1], KLEE [2], Java PathFinder [3] and S2E [4]. Most of these static analysis engines take program codes as input and then represent the program through either intermediate representations (IRs) or binaries [5, 6, 7, 8, 9]. All the analysis would be based on these proposed representations. However, intermediate representation and binaries do not offer mapping to source code directly. Therefore, it probably provides limited information when the analysis process relies on it. Finally, tools and libraries, which generate the intermediate representation and binaries, may not be valid for all the programming languages. To overcome these challenges, we design and implement a database-oriented static program analysis engine, which is based on PHP program. Graph database is a database that uses graph structures for semantic queries with nodes, edges, and properties to represent and store data. Data in the graph database has both the feature of connected graph and the feature of text data. That can address most challenges that im-

plement static program analysis by general intermediate representation. Implement static program analysis using graph database can take following benefits.

- Directed graph representation in graph database: The entire program would be represented as a directed graph and stored in the graph database system. It implies that all the nodes and edges in the graph could represent either row of data represent as text or a visualized graph and display on the graph database server.

- Flexible analysis methods: Compared with the analysis systems based on intermediate representation and binaries, our system could analyze the program based on the database's implementation. A database query could retrieve all the information, and a visualized directed graph could analyze the programs' structure.

Building up this static analysis engine with such features, however, was faced several significant challenges. First of all, a PHP program's source code usually presents more varied information when comparing this program to intermediate representation or binary representation. In order to represent the information correctly, a clear and appropriate database structure is significant. We need to consider how to represent nodes individually, but we also need to consider representing the structure of nodes and edges well. Secondly, compared with a standard database system, a graph database uses "cypher query language" to implement the entire database. Its grammar is not like familiar database query language. Therefore, we need to design transactions for different database operations by the "Cypher query language". After we insert all the data in the graph database, the analysis operation must be designed and implemented based on the database system. Thus, considering the features of the database, we have generalized all the analysis process individually. In order to overcome these challenges systematically, we have made the following contributions.

- Designing nodes and edges structure in database: To represent the directed graph into the database, we designed node structures and attributes based on different nodes. Each node in the database represents an individual variable in the target program.

On the other hand, we also design the edge structure and attributes based on edge types that we defined. These edges would be used to represent the relations between each node in the graph database. These nodes and edge templates could represent the entire program to the directed graph correctly.

- Writing queries for insert data through different methods: Because of the analysis platform's difference, our system needs to have one more step than other analysis engines. That is the insert data process. To insert the directed graph to the graph database, we use graph database driver library, which is based on python, to connect our engine and database system. And then, we design all the insert operation based on different insert methods.

- Develop basic analysis methods and design related implement queries: The analysis process is a significant part of our static analysis system. Compared with other static analysis systems, our system has mainly used different queries to achieve this goal. To implement the analysis process, we designed several different queries for analysis programming in the database system. All of the analysis operations would be implemented by the execution of valid queries.

Our current implementation focuses on the PHP program, one of the most popular programming languages widely used in web application development. It is a dynamic programming language with a dynamic type. In our static analysis engine, we considered all of the core features of the PHP programming language, including **super global variables**, **unary operations**, **binary operations**, **function definitions**, **function calls**, **assignment statements**. We use a different type of nodes to represent these features in our graph database system. Furthermore, to represent the relation between all the PHP program variables, a tree data structure can provide better performance among those data structures that we usually used. Eventually, a tree structure graph that represents a PHP program is stored in the graph database. All the edges can help to form this graph. In this graph, every node

represents one variable, and every edge can represent the relations among nodes. These nodes and edges construct several different paths in the graph. Our analysis, which is for the PHP program, would be completely based on these paths.

To evaluate our static analysis engine system, we have collected $1,377$ plugins from the WordPress plugin repository [10] and 9899 top stars PHP projects from GitHub platform [11], where WordPress and GitHub are both popular open-source content-sharing platforms. These plugins and projects include several different language features and plenty number of codes. Then, We selected eight typical projects for evaluating our system. Experiment results have emphasized that our system can analyze all of the plugins and CMS projects effectively.

The remaining of this thesis is organized as follows. Chapter 3 presents the design of this system and its implementation. The experiment results are presented in Chapter 4 and Chapter 6 will conclude the thesis and talk about our future work.

Related Work

Son and Shmatikov [12] developed a static analysis tool, which is for the analysis of PHP applications. (SAFER-PHP). This analysis tool is the first analysis tool that applies semantic analysis to the program analysis area. It detects the infinite loop trigger bugs and missing authorization vulnerability. It is also the first analysis tool that could support the objected-oriented features, PHP-based web applications. The algorithm is based on inter-procedural algorithms, and the algorithms are based on symbolic execution applied to the semantic security analysis. Their analysis tool parses the PHP source code first and then generates an abstract syntax tree of this PHP code. After that, this analysis tool builds a call graph and a control-flow graph in the entire program. Eventually, the critical variables, which would execute sensitive operations, would be collected from the program call graph and the control-flow graph. A loop, whose termination decide by the external inputs, would be found by the taint analysis. On the other hand, the symbolic execution will check the infinite loop caused by the program. Based on this analysis system, two classes of vulnerabilities, i)denial-of-service ii)authorization missing check, could be detected through this semantic analysis. This analysis tool detects unreported vulnerabilities from the open-source PHP programs.

Ehresmann et al. [13] provided a PHP program analysis and regression testing engine called “PARTE.” This tool could make regression testing for the PHP web applications, which are frequently patched and revised. Comparing with applying the regression testing to the entire PHP program, in their paper, the author tries to utilize his system, “PARTE,” to

identify the affected part of the entire code for the two consecutive versions of the changing code impact analysis. In order to implement an impact analysis, the "PARTE" system uses the Intermediate High Representation and the Abstract Syntax Tree to construct a dependence graph of two consecutive versions for the PHP-based program, which is needed to be analyzed. After that, "PARTE" would identify the code areas' difference between the two consecutive codes. To test the update feature or new functionalities in the affected areas of programs, the author designed new test case generation methods, which generate new executable cases through the string type and the numeric type input value in the program slices. Based on these methods, the "PARTE" could effectively decrease these crucial general test cases and only concentrate on the frequently upgraded web programs' impact areas.

"Uchecker" [14] is also an Abstract-Syntax-Tree-oriented symbolic execution system for detecting known file uploading vulnerabilities. In this system, the entire process of analyzing the PHP program is the first step. The proposed analysis method can detect the restricted file uploading vulnerabilities based on this PHP program analysis process's output. During the analysis process, A proposed PHP programming language interpreter is applied to parse the PHP program. After translating this interpreter, the entire PHP program could be represented as an abstract-syntax-tree, and all the analysis would be based on the proposed abstract syntax tree. Comparing with some other method to analysis program which uses dynamic program analysis, "Uchecker" system can have faster execution time. That is because it applies the idea of the static analysis method to program analysis. All the analysis process is based on the structure and the logic of the program coding. The proposed evaluation of the "Uchecker" system proves its fast speed and better performance, which benefit from static analysis when detecting known file uploading vulnerabilities.

Design and Implementation

Our system is composed of two main phases. The first part would take the PHP program as the input and convert the PHP program to an abstract graph. Eventually, we can get the output of the abstract graph as CSV files from the first part. The second part uses the CSV files as the input, then converts the abstract graph to different data types and inserts them in the database we created. Figure 3.1 represents the architectural overview of the main workflow. In the chapter, we will first introduce the main conceptions of the graph database system we use and the structure of the database we created to store the PHP program. Then we will discuss the implementation of the database system and the insert query. Finally, we would talk about some applications we have developed so far, based on the system we built.

3.1 Heap Graph

A heap graph is a graph-oriented intermediate representation that is generated by the "Uchecker" system [14]. In the system, a PHP program generates an abstract-syntax-tree by interpreting the PHP program's entire statement at first, and the heap graph is the visualization of the generated abstract-syntax-tree. Typically, a heap graph is a tree-based graph. It contains a different type of nodes that are connected by several weighted edges. In our analysis engine, the generated heap graph's format is two CSV files representing node set

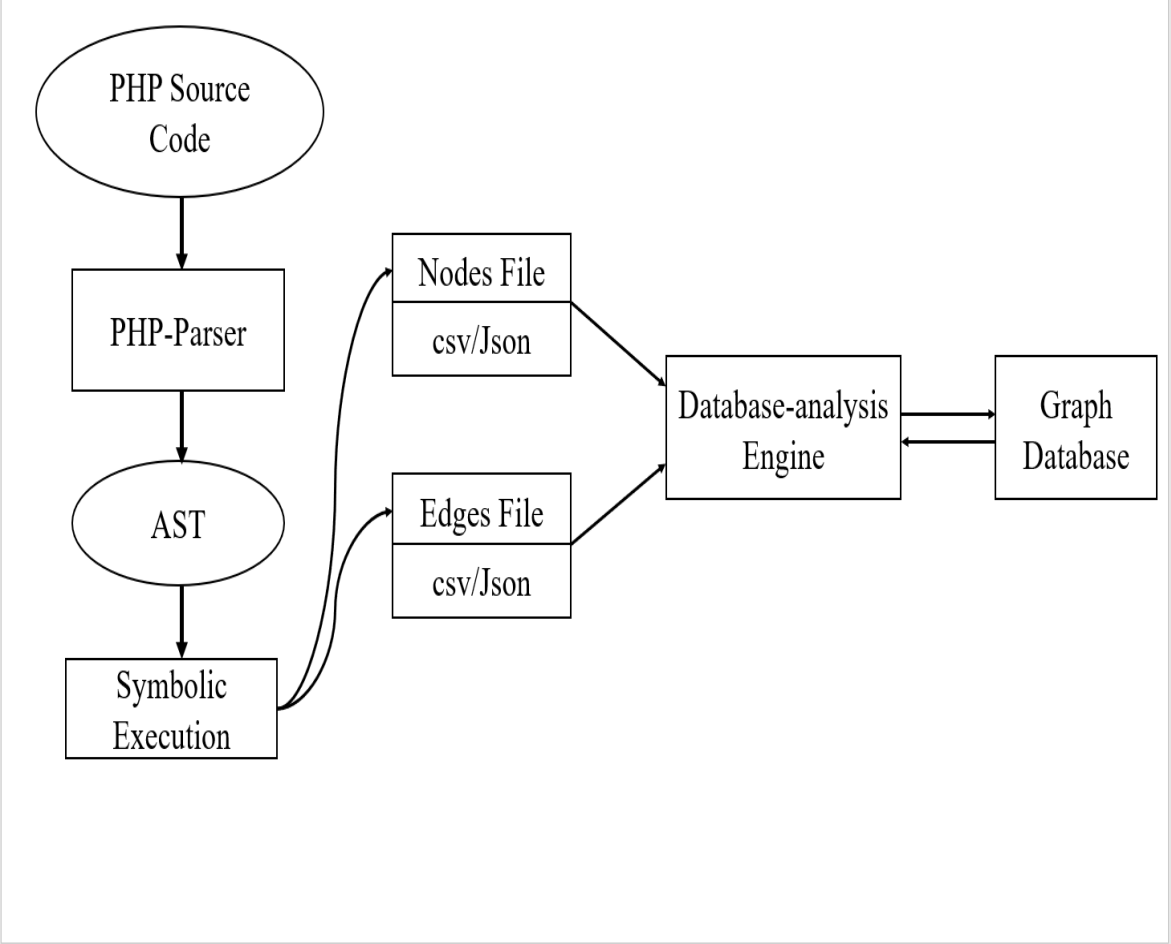

Figure 3.1: System Overview

and edge set. We will insert these sets into our graph database and implement the analysis process towards the graph database.

In this section, we will introduce how the architecture of heap graph representing in our graph database at first. Based on the type of variables in general the PHP program, we would talk about different type of nodes model respectively. Then, we will introduce those edges that connect all the nodes in the graph database.

3.1.1 The Nodes Architecture

Represent each variable and its relationship to the PHP program perfectly is the primary target we aim to achieve in the graph database. For approaching this target, all of the

nodes need to be categorized regarding these variables. In the database, we need to define different types of nodes first and add necessary attributes to express the feature of these nodes. Based on the definition of different types of variables, we define our nodes model into eight categories. It includes “FunctionCall Node”, “Leaf Node”, “Comb Nodes”, “ArrayAccess Node”, “Operation Node”, “Unary Node”, “Array Node” and “Encapsed Node”. We will introduce these nodes’ structures, respectively.

- “FunctionCall Node”: “FunctionCall Node” is used to represent a defined function or used in the PHP program. A node is defined as “FunctionCall Node,” which would have three main attributes. It includes “id,” which is used as an index of this node; “name,” which represents the name of the function in the PHP program,” type,” which represents the type of nodes.

- “Leaf Node”: “Leaf Node” represents the data variables in PHP programs. A node that is defined as a “Leaf Node” has four attributes. It includes “id”, which is used as index of this node; “name”, which is the variable name in the PHP program; “data type”, which is data type of this variable(it has four values, These are “int”,“String”, “Boolean”,“Null”);“type”, which represent the type of nodes.

- “Comb Node”: “Comb Node” is not a variable in the PHP program. It is used to assisting represent a graph in the database. The comb nodes can collect all the values that the variable will probably be and each value’s condition. It just has three attributes, which are “id,” as the index of the node; “name,” as the name of this node; “type,” as the type of this node.

- “Array Node”: An array is a group of data, so we can not represent all of the data in an array, respectively. That would cause a noisy issue when we analyze the program. “Array node” is used to represent an array variable in a PHP program. It has three attributes in the database, “id” is used as the index of this node, “name” represents the name of this variably, and “type” represents the type of this node.

- “ArrayAccess Node”: This type of node connects the array node’s index that the PHP program accessed with the array node. Because the array is a group of data, we use the “array” and “array access” node to represent the array and the accessed data that the program accessed. There are three attributes of this node. It includes “id,” “name,” and “type,” which have the same usage as other nodes.

- “Operation Node”: “Operation Node” represents a computing operation in PHP program, such as “Plus,” “minus,” “or.” It also has three attributes, just like some other nodes. These are “id” which is for index the node, “name” which is for store the name of the variable and “type” represents the type of this node.

- “Unary Node”:“Unary Node” is represented as a unary operation, which is represented as “u-op e,” where “u-op” is a unary operation and “e” is the expression. It has three attributes. It includes “id” ,“name” and “type”.

- “Encapsed Node”:“Encapsed Node” is not a variable in the PHP program. It is used to help represent these variables that is separated by a comma. This node type, just like some other node type, has three main attributes. These are “id” , “name”, and “type”.

Table 3.1: Nodes Model Overview

Nodes types	attributes
ASFuncCallNode	”id”, “name”, ”type”
ASBranchOrNode	”id”, “name”, ” type”
ASBinaryOpNode	”id”, “name”, ”type”
ASUnaryOpNode	”id”, “name”, ”type”
ASArrayAccessNode	”id”, “name”, ”type”
ASArrayNode	”id”, “name”, ”type”
ASSuperGlobalNode	”id”, “name”, ”type”
ASEncapsedNode	”id”, “name”, ”type”
ASLeafNode	”id”, “name”, ”type”, ”data-type”

Table 3.2: Nodes Attributes Overview

Attributes	meaning
”id”	”A unique string that used as index of one node”
”name”	”the name of the variable in program”
”type”	”the type of this node we defined”
”data-type”	”the data type of the node that the variable represnted”

The Table 3.1 and The table 3.2 represents all the nodes type and its attributes. From the list, we can know that most types of nodes only have three main attributes. These are “id,” “name,” and “type”, which are used to represent the index of the node, the name of this variable, and the type of this node, respectively. The “AesLeaf Node” is different from some other nodes. It has another attribute called “type,” which represents the data type of this variable.

3.1.2 The Edge Architecture

The edge model is much simple than these nodes model. To describe each node’s relationship in the database, we define the relation to several different types. All the nodes in the graph have at least one edge to connect to another node. The node attributes are used to describe this node’s feature, and the type of relationship is used to describe the relationship between the two nodes. After all the edges are added to the database, the entire graph could be expressed entirely in our database. We will discuss the type of necessary relation, respectively in the next part.

- “args”: “args” is used to express that the node, which the relation point to, is the function node’s argument. Typically, it exists between a “FunctionCall Node” and a “Leaf Node.”

- “left”,“right”: When a comma in PHP program combines two variables, This structure would be expressed as a “concat node” connecting with two “Leaf node” by “left” and “right” relations.

- “data”: Commonly, a PHP program variable could be assigned a value, which comes from another computing process. Therefore, a data relation is used to show the path for getting this value.

- “ctrl”: A variable can have several different values when some condition is different. The “ctrl” relation is used to express this condition. It can specify the path of this condition.

- “array”,“myindex”: When a variable access an array in a PHP project, These relations could be useful. “myindex” relation points to the variable that presents the index value of the value stored in an array node. Furthermore, the “array” relation points to the array variable from which this variable gets the value.

- “encapsed part”,“encapsed last”:These are two relations to express the “encapsed node.” Commonly, “Encapsed Node” is used to describe the situation in which a comma combines two variables. This structure has two parts, the variable at the left of the comma specified by the “encapsed part” relation. The other part would be specified by “encapsed part” relation.

Table 3.3: Edges Model Overview

Edges types	attributes
Relation	”Edge-type”

Comparing with the node model, the structure of the edge model is much simple. The edge model would only have one attribute used to represent the type of this edge. All types of edges and their attributes are shown in the table 3.3 and the table 3.4.

Table 3.4: Edges Attributes Overview

attributes	meanging
”args”	”represent the arguments of one function node”
”data”	”the data flow of one variable in program”
”left”	”The left of the part that is combined by a comma in program”
”right”	”The right of the part that is combined by a comma in program”
”ctrl”	”the control flow of one variable in program”
”array”	”the array that the variable visited for getting its value”
”myindex”	”the index of an array that a variable visited”
”Encapsed Node”	”one part of a Encapsed node”

3.2 Design of Graph Database

Graph database is a powerful tool we used in our system. Thus, we will first introduce the graph database. The graph database is a database designed to create the relationships between the data as equally important to the data itself. Comparing with a standard database system(like MySQL), it is intended to hold data without constricting it to a pre-defined model. Furthermore, the data is stored and shown how each entity connects with or is related to others. Thus, the graph database can represent the abstract graph as data correctly, especially the relationship between each node in the abstract graph we generated in the first step.

In the graph database, nodes are the entities in the graph. They can hold any number of attributes (key-value pairs) called properties. In order to represent the graph data correctly, so that we could do analysis work in the future, based on two different insert strategies, we defined the node’s label to two categories, including i)common nodes label and ii)specify nodes label. Common nodes label could represent nodes in the abstract graph to the same type node in the database. It only defines one kind of label to represent all the nodes type. Specify labels aim to copy the entire node structure in the abstract graph to the database directly, defining several labels based on their type in the graph. Therefore, using the standard node label would make all the nodes in the database have the same structure. Using a specific node label could make all of the nodes in the abstract graph represent their

specific type in the database, and it also can have its structure.

The Listing 3.1 represents the main structure of the common label. Normally, all the nodes inserted with a common label would use this structure. The Listing 3.2 and The Listing 3.3 represents the specific type of label's structure of "FunctionCall" node and "AsLeaf" Node.

Listing 3.1: Common Label Structure

```
  ``Common labels": [
    "Node"
  ];
  "properties": {
"name": (String type data),
"id": (String type data),
"type": (String type data),
"project_name": (String type data),
"Data_type":(String type data)
  };
}
```

Listing 3.2: Functioncall Node Label Structure

```
  "FunctionCall Node labels": [
    "Node"
  ];
  "properties": {
"name": (String type data),
"id": (String type data),
"type": (String type data),
"project_name": (String type data)
  };
}
```

Listing 3.3: AslLeaf Node Label Structure

```
  "FunctionCall Node labels": [
    "Node"
  ];
  "properties": {
"name": (String type data),
"id": (String type data),
"type": (String type data),
"project_name": (String type data),
```

```
 9  "Data_type":(String type data)
10    };
11  }
```

3.3 Implementation

Our system will insert the abstract graph in the graph database system with two main strategies. The main workflow is to insert nodes to the graph database first, then insert the edges to connect all the nodes in the graph database. In this chapter, we will discuss our system's basic configuration at first, including the basic configuration in the graph database system and the connection set up between the database system and our system. Then we will talk about the two main insert methods one by one. We will also illustrate the main workflow of the entire system and display the final result.

3.3.1 Configuration

There are two main configurations we do before we insert nodes in the database system. It is the configuration in the graph database and the configuration connected between the database and our system.

- The first configuration we do is in the graph database system. We need to create a graph database and database user in the database system at first. We then set up the configuration file parameters to permit import data to the graph database from our graph system and permit it to connect the database through the HTTP address and bolt address from our graph system.

- The second configuration we do is in our graph system. We need to set up the database connection in the graph system and provide the database user's id, password, and database's URL to the connection code we created.

After these two configurations, our graph system could be feasible to work. We need to call the function for creating a connection between our graph system and database when we need to do some operation in the database, and close the connection after we finished the transaction we set up.

The Listing 3.4 represents the configure parameters we added in the configure file. Furthermore, the Listing 3.5 represents the code of creating connection in the graph system.

Listing 3.4: Graph Database Configure File

```
# Bolt connector
dbms.connector.bolt.enabled=true
#dbms.connector.bolt.tls_level=DISABLED
dbms.connector.bolt.listen_address=:11002

# HTTP Connector. There can be zero or one HTTP connectors.
dbms.connector.http.enabled=true
dbms.connector.http.listen_address=:11003

# HTTPS Connector. There can be zero or one HTTPS connectors.
dbms.connector.https.enabled=false

# Name of the service
dbms.windows_service_name=neo4j

# `LOAD CSV` section of the manual for details.
dbms.directories.import=import

# Other Neo4j system properties
apoc.import.file.enabled=true
```

Listing 3.5: Graph Database Driver Connection

```
from neo4j import GraphDatabase

uri=localhost
password=password
user=user

class Example:

    def __init__(self, uri, user, password):
        self.driver = GraphDatabase.driver(uri, auth=(user, ...
            password))

    def close(self):
```

```
        self.driver.close()

if __name__ == "__main__":
    greeter = Example(uri, user, password)
    ***************
    #implement is here
    ***************
    greeter.close()
```

Specifically, the "dbms.connector.bolt.enabled" is used to enable connect the database with bolt address, and we set up the bolt listen address at "11002" through the "dbms.connector.bolt.listen_address". Moreover, we also use the "dbms.connector.http.enabled" to enable connect the database with the HTTP address, and use the "dbms.connector.http.listen_address" to set up the listen address to "11003". The final parameter, "apoc.import.file.enabled", can let the database insert data from outside resource.

The main package we used to create the connection in the python program is "GraphDatabase". To create a connection, we need to use the function called "driver" to create a database driver in the program, then we could operate the graph database by calling the related function through the driver we created. To create a correct driver, we need to provide the correct URL of the database we created, the username and password of the database user, as the "driver" function's parameters. Once we provide the correct information, the driver could be created correctly.

3.3.2 Insert Data with Specify Label

Insert data with a specified label aim to insert the entire abstract graph into the database with different types of nodes and different types of edges, so entire graph data could distinguish the type of each node by different database node labels. Therefore, before we insert the data to the database correctly, we need to distinguish the data type first. Then we will insert these data with a related nodes type label, respectively. After we insert all of the nodes in the graph, we would insert all the edges to connect all the nodes in the database. The process of insert edges is also like the process of insert nodes. All of the edges need to

distinguish types first, then insert edges with related edge type label. After we finish these two processes, the abstract graph succeeds in inserting in the database, and we can access the data through the connection we created in our system.

Listing 3.6: Insert Nodes Queries

```
@staticmethod
    def insert_Function_data(tx, data):
        tx.run("CREATE (n:Function {id:$id,name:$name,type:$type})"
               , id=data["id"],name=data["name"],type=data["type"])

    @staticmethod
    def insert_Leaf_data(tx, data):
        tx.run("CREATE (n:Leaf {id:$id,type:$type,name:$name, ...
            data_type:$data_type})"
               , id = data["id"], type = data["type"],name = ...
                  data["name"], data_type = data["data_type"])

    @staticmethod
    def insert_Operation_data(tx, data):
        tx.run("CREATE (n:Operation {id:$id,type:$type,name:$name})"
               , id=data["id"],type=data["type"],name=data["name"])

    ...
```

Listing 3.7: Insert Edges Queries

```
@staticmethod
    def insert_args_relation(tx, data):
        tx.run("MATCH (m),(n) WHERE m.id=$sid AND n.id=$did "
               " CREATE (m)-[r:args{num:$argnum}]->(n)",
               sid=data["source"],
               "did=data["destination"], argnum=data["attribute"])

    @staticmethod
    def insert_data_relation(tx, data):
        tx.run("MATCH (m),(n) WHERE m.id=$sid AND n.id=$did "
               " CREATE (m)-[r:data]->(n)", sid=data["source"],
               did=data["destination"])

    @staticmethod
    def insert_control_relation(tx, data):
        tx.run("MATCH (m),(n) WHERE m.id=$sid AND n.id=$did "
               " CREATE (m)-[r:control]->(n)", ...
                  sid=data["source"], did=data["destination"])

    '''
```

Listing 3.6 and Listing 3.7 (part of queries) shows the queries of insert nodes and insert edges with a specific label. All nodes and edges would be inserted through different quires, based on the type of nodes and edges. listing 3.8 shows the main algorithm of running database queries through the driver we created. The necessary process is i)read files and get the data. ii) distinguish node and edge type. iii) run the related query to insert the data.

Listing 3.8: Queries Execution

```
    data = self.python_load_json(address)
    edges = self.python_load_json(address)

    with self.driver.session() as session:
        for node in data:

            if node["type"]=="ASFuncCallNode":
                    session.write_transaction(
                    self.insert_Function_data, node)

            elif node["type"]=="ASLeafNode":
                    session.write_transaction(
                    self.insert_Leaf_data, node)

            elif node["type"]=="ASBinaryOpNode":
                    session.write_transaction(
                    self.insert_Operation_data,node)
    ...

        for edge in edges:
            if edge["attribute"][:3]=="arg":
                session.write_transaction(
                self.insert_args_relation, edge)

            elif edge["attribute"]=="data":
                session.write_transaction(
                self.insert_data_relation, edge)

            elif edge["attribute"]=="ctrl":
                session.write_transaction(
                self.insert_control_relation, edge)

    ...
```

3.3.3 Insert Data with Common Label

Generally, Insert data with common labels could make all the data in the database have the same label. All of the nodes and edges would be stored in the database with the same type and same structure. Therefore, we do not need to distinguish the type of data before we insert data into the database, and we also do not need to write different queries for inserting all the nodes and edges. Comparing with insert data with a specific label, insert common label nodes is much more comfortable. We would insert all the nodes with the same label and structure st first., then insert all the edges to connect the nodes in the database. Because all the nodes have the same structure, some nodes' attributes probably do not exist in some other nodes in the abstract graph. We would set those attributes as a default constant value (like "null") to express that these nodes do not have those attributes.

Listing 3.9: Common Label Insert Queries

```
@staticmethod
    def load_csv_nodes(tx, address,package_name):
        tx.run("LOAD CSV WITH HEADERS FROM $address AS line "
               "CREATE (n:Node{id:line.ID,
               name:line.Name,
               type:line.Type,
               Data_type:line.Data_Type,
               project_name:$project_name})"
                , address=address, project_name=package_name)


    @staticmethod
    def load_csv_edges(tx, address):
        tx.run("LOAD CSV WITH HEADERS FROM $address AS line "
               " MATCH (m{id:line.Source_ID}),
               (n{id:line.Destination_ID})"
               " CREATE(m)-[r:relation{type:line.Type}]->(n)"
               , address=address)
```

Listing 3.9 represents the query for inserting nodes and edges with common labels into the database. The database system provides tools to load the CSV file automatically. Therefore, we can load the related CSV file line by line and create related nodes with the label we defined. After that, we will insert edges with the same method of inserting nodes.

Listing 3.10: Common Label Queries Executuon

```
address="Files address"
last_dir= os.path.basename(address)
d=os.path.join(destination, last_dir)
shutil.copytree(address, d)

self.pre_transform_address(address, aDDRESS)
        with self.driver.session() as session:

            node_dir="file:///"+os.path.basename(address)
            +"/"+"nodes.csv"
            package_name = os.path.basename(address)
            session.write_transaction(self.load_csv_nodes
            ,node_dir,package_name)

            edge_dir="file:///"+os.path.basename(address)
            +"/"+"edge.csv"
            session.write_transaction(self.load_csv_edges
            , edge_dir)
```

Listing 3.10 presents the main algorithm of running these defined queries through the driver we created. The primary process is i) move the CSV files to the database import directory. ii) run the query to insert all the nodes through the nodes CSV file. iii) insert all of the edges to connect those nodes, which we inserted previously, by running the related query.

Listing 3.11: A PHP Example

```
<?php

$a = 5;
$b = 6;
$c = $_REQUEST['file'];

if ( $a < 10 ) {
    $b = 10;
}

if ( $a > 4 ) {
    $a = 100;
    $e = 200;
} elseif ( $a > 100 ) {
    $a = 666;
} else {
    $d = "LLLLLL";
}
echo $a;
```

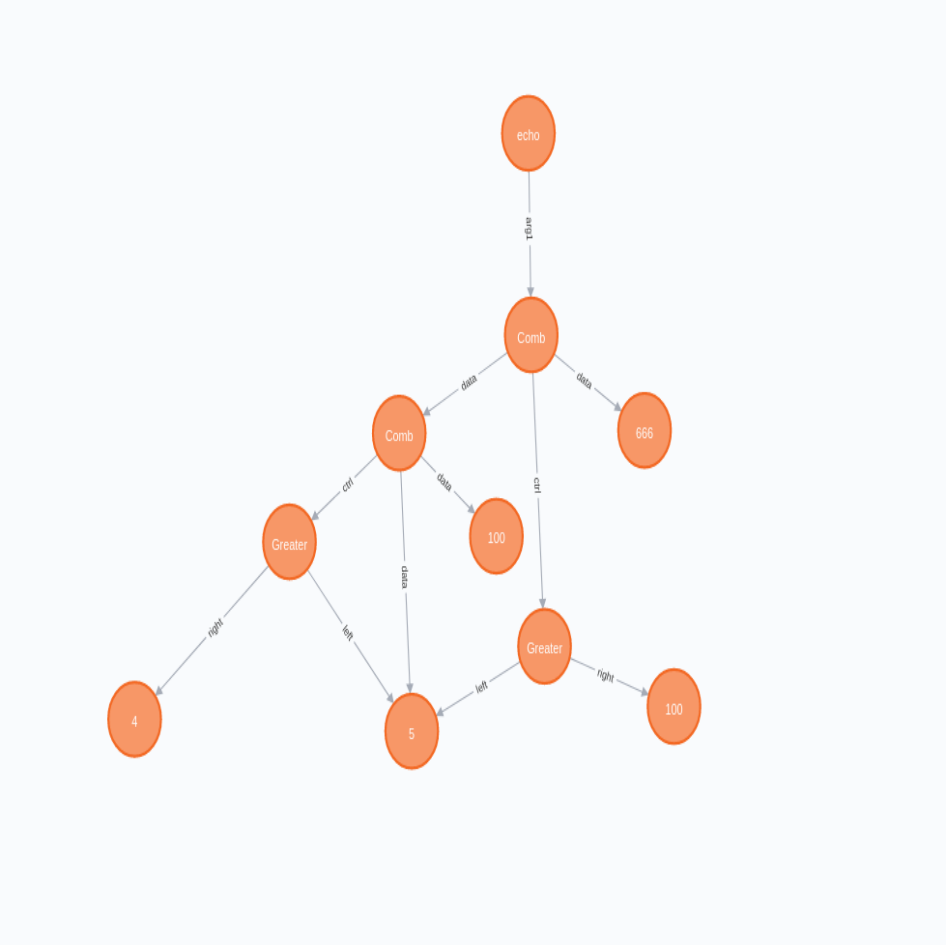

Figure 3.2: Visualized Graph

Listing 3.11 shows an example PHP program that is inserted into the database. Moreover, the figure 3.2 shows the visualization result after we insert the example program in the database.

Evaluation

To evaluate our system performance, we have collected $1,377$ plugins from the WordPress plugin repository [10] and 9899 top stars PHP projects from the GitHub platform. Then, we choose some specific PHP programs from those projects and evaluate our system's performance using some simple projects. These specific programs are filtered from these collected programs for different purposes (such as web application, CMS, plugin). These PHP programs would be the input of our system. We will evaluate the performance of our system, which is based on the output of our system.

4.1 A Running Example

As a simple example, this Listing 4.1 represents a PHP program that would mainly generate two sub-graphs in the directed graph. In these graphs, it has 24 nodes and 24 edges. After this section, we will show how the PHP program graph represents in the graph database system by nodes and edges.

Listing 4.1: PHP Codes Example

```
1 $a = 10;
2 $b = 0;
3 $c = '';
4
5 if ( $a < 0 ) {
6     $c = 'Failure';
7 }
8
9 if ( $a ≥ 0 and $a > 0 ) {
```

```
10      $a = 100;
11      $b = 200;
12      $c = 'Failure';
13
14  } elseif ( !is_null($b) && $a > 10 ) {
15      $a = 666;
16      $c = 'Success!';
17      echo $a;
18  } else {
19      $c = 'Failure';
20  }
21  echo $c;
```

All of the nodes representing all of the variables in this PHP program are shown at the Listing 4.2. And all of the edges that used to connect these nodes were shown in the Listing 4.3.

Listing 4.2: Nodes List

```
1
2  {"name":"SUPER_is_null","id":"FF00000000246156f4000000007e8676c1"
3  ,"project_name":"example-2","type":"ASFuncCallNode"}
4
5  {"name":"Comb","id":"BR00000000246156e6000000007e8676c1"
6  ,"project_name":"example-2","type":"ASBranchOr"}
7
8  {"name":"is_null","id":"FF00000000246156e7000000007e8676c1"
9  ,"project_name":"example-2","type":"ASFuncCallNode"}
10
11 {"name":"echo","id":"FF00000000246156f3000000007e8676c1"
12 ,"project_name":"example-2","type":"ASFuncCallNode"}
13
14 {"name":"0","id":"LL00000000246156e4000000007e8676c1"
15 ,"project_name":"example-2","type":"ASLeafNode","Data_type":"integer"}
16
17 {"name":"null","id":"LL00000000246156e1000000007e8676c1"
18 ,"project_name":"example-2","type":"ASLeafNode","Data_type":"NULL"}
19
20 ...
```

Listing 4.3: Edges list

```
1
2  [{"id":"FF00000000246156f4000000007e8676c1"},{"type":"arg1"}
3  ,{"id":"BR00000000246156e6000000007e8676c1"}]
4
5  ["id":"BR00000000246156e6000000007e8676c1"},{"type": "data"}
```

```
,{"id":"FF00000000246156e7000000007e8676c1"}]

[{"id":"BR00000000246156e6000000007e8676c1"},{"type":"data"}
,{"id":"LL00000000246156e1000000007e8676c1"}]

[{"id":"BR00000000246156e6000000007e8676c1"},{"type":"ctrl"}
,{"id":"LL00000000246156e5000000007e8676c1"}]

[{"id":"FF00000000246156e7000000007e8676c1"},{"type":"arg1"}
,{"id":"LL00000000246156e4000000007e8676c1"}]

[{"id":"FF00000000246156f3000000007e8676c1"},{"type":"arg1"}
,{"id":"LL00000000246156f1000000007e8676c1"}]
...
```

The Generated graph is shown as the figure 4.1. There have two main graphs shown in the entire graph, which is related to the function call that occurs in the program. We will discuss the logic of this program based on the two graphs of the two function call.

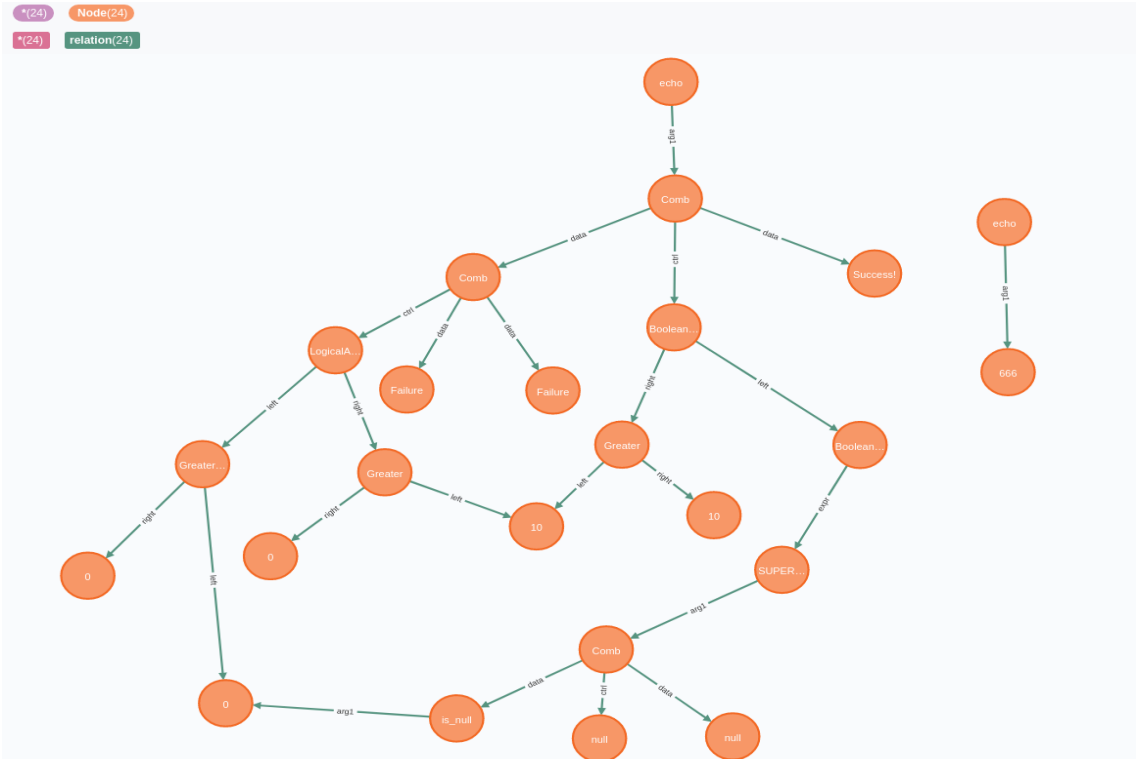

Figure 4.1: Visualized graph for this simple example

Figure 4.2 is one part of the generated graph. It represents one path of one of the program's function call, "echo" here. According to the figure 4.1, we could know that the

result of this “Functioncall” node only depends on one constant value, whose information is shown at the listing 4.4. Therefore, according to the usage of the “echo” function, we could know this “echo” is used to print the value of a constant value, whose value is an integer value ”666”.

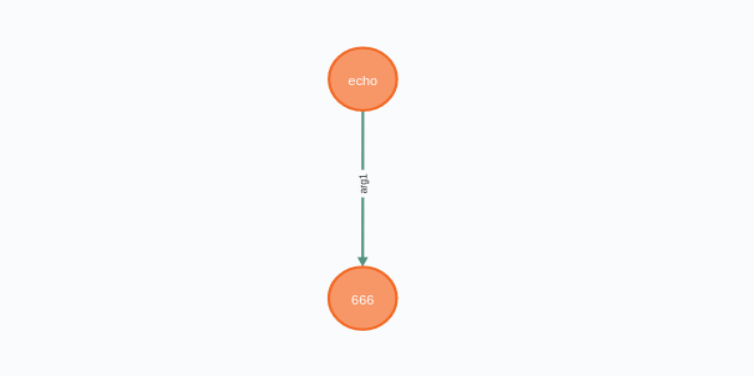

Figure 4.2: Part-1 of The Example Graph

Listing 4.4: The Information of The AsLeafNode

```
    "identity": 139,
    "labels": [
      "Node"
    ],
    "properties": {
  "name": "666",
  "id": "LL00000000246156f1000000007e8676c1",
  "project_name": "example-2",
  "type": "ASLeafNode",
  "Data_type": "integer"
    }
  }
```

Figure 4.3 is another sub-graph that represents another process of the entire function calling in the program, which is “echo.” From the graph that we generated in the database, we can see that the value of the argument of the echo functions has two main possibilities. The first is “Success!” and the second is “Failure.” The condition of each value that occurs is specified at the “ctrl” edges. From the graph, we could know that the value “Success!” occurs because the variable ”b,” represented as a leaf node in the database, is None, and

variable a is larger than integer 10. For another value “failure” occur in the program, the condition is when variable a is greater than integer 0, and variable b is greater than 0.

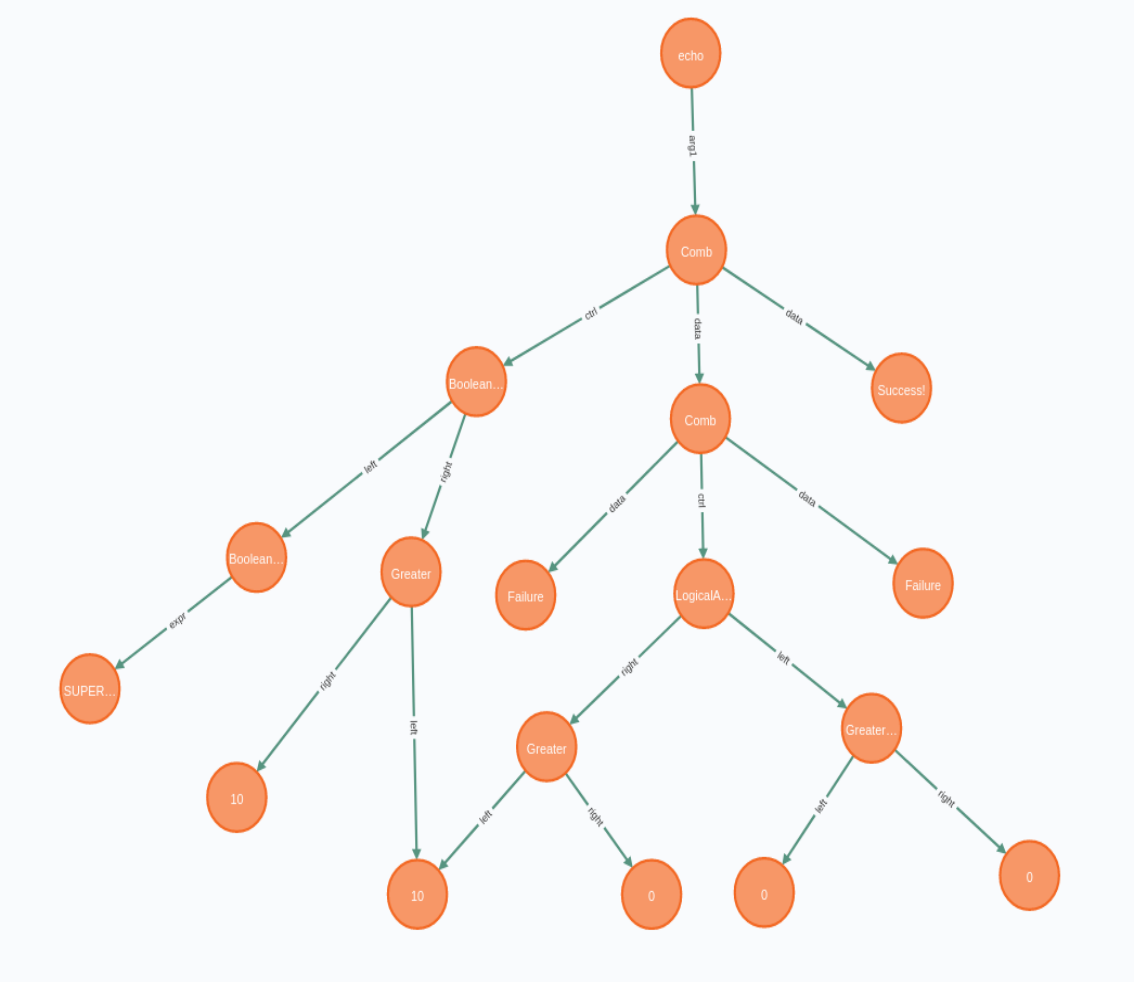

Figure 4.3: Part-2 of The Generated Graph

4.2 Experiments Using Real Examples

We have leveraged real-world samples collected from the PHP code repositories, including the WordPress plugins and the Github CMS projects. Our system can effectively perform static program analysis for all the tested samples. Table 4.1 presents the name of WordPress Plugin, the version, the number of nodes and the number of edges, and the execution time of performing static program analysis.

Table 4.1: Execution Result

Plugin Name	Version	Number of nodes	Number of edges	generate Time(seconds)
easy-form-builder-by-bitware	1.0.0	755	663	1.2
squarecomm	1.1.0	430	410	0.6
http-headers	1.9.4	16029	18634	132.2
woocommerce-checkout-manager	4.2.5	680	650	1.2
N-Upload	4.2.2	1580	1760	4.2
image-gallery-with-slideshow	3.3.0	419	414	0.6
estatik	4.5.3	261	243	0.51
WooCommerce-Catalog-Enquiry	3.0.1	8776	10037	101.7

Application

The primary purpose of we builds this system is to analyze the PHP program statically. Therefore, in this section, we will introduce some simple applications we developed to implement static program analysis. We will discuss three main applications in this section, including, i) the application for checking whether a specific function using in the program. ii)The application for checking the API dependency. iii) The application to implement the taint analysis. iv) The application for checking information leakage existence in the PHP program.

5.1 Function Call Checking

Typically, the most popular PHP programs, can have thousands of code lines, even ten thousand, written in several separate PHP files. Therefore, it is not easy to check whether the PHP program contains specific functions in the entire PHP program. The "Uchecker" can model the PHP program as an abstract syntax tree. However, if we want to make sure whether a function exists in the PHP program through the abstract syntax tree that the "Uchecker" built, we probably need to look up the function in the entire abstract syntax tree. If we do not have a powerful tool, That would be a tough job. Comparing with the searching operation in the "Uchecker" system, to search for some specific elements in the PHP program could be much more comfortable in our system because the entire PHP program model is stored as several rows of data in the database. Thus, this operation could be finished by executing a simple database query.

Listing 5.1: Query for Finding Functions Application

```
@staticmethod
    def find_node(tx,name):
        res=tx.run("MATCH (m{name:$name})"
                   "RETURN m",name=name)
        return res.value()
```

In the query, “MATCH (mname:(name)” is used to match the related node, and then returned the result by ”RETURN m”.

Listing 5.2: Execution of Finding Functions Application

```
def finding_nodes(self,name):


        with self.driver.session() as session:
            res=session.read_transaction(self.find_node,name)
            return res
```

Listing 5.3: Main Code for Finding Function Call Application

```
        driver = GraphDatabase.driver(uri=URL,
        auth=(username, password))
        with self.driver.session() as session:
            session.write_transaction(self.insert_nodes, address)
            session.write_transaction(self.insert_edges, address)
            res=session.read_transaction(self.find_node,name)
            return res
        driver.close()
```

The listing 5.1 shows the query we wrote to find a specific function in the PHP program database. Normally, the query can return the search result of the function name provided in the graph database. The listing 5.2 shows the example code for running the query through the driver we created. Then, close the connection after we commit the query transaction, The entire running process in our system is shown as listing 5.3.

Listing 5.4: A PHP Example Code

```
<?php
function writeMsg() {
```

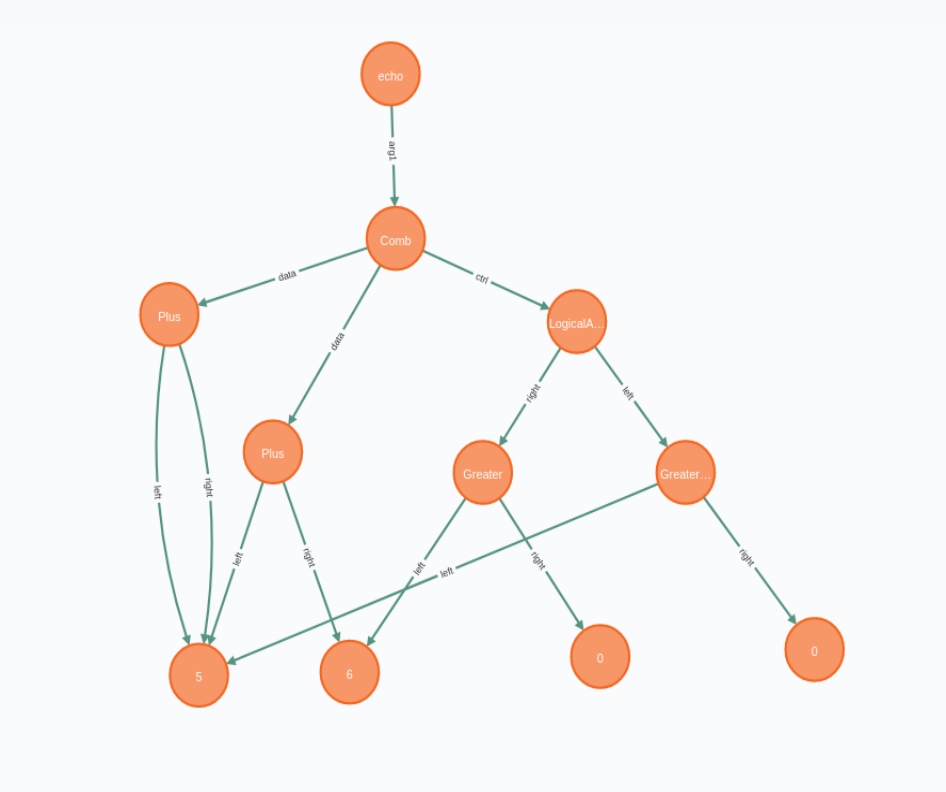

Figure 5.1: Visualized Graph for The PHP Example

```
3    $a = 5;
4    $b = 6;
5    if($a ≥ 0 and $b > 0){
6      $c=$a+$b;
7      }
8    else{
9      $c=$a+$a;
10   }
11   return $c;
12 }
13
14 $res=writeMsg();
15 echo $res;
16 ?>
```

Listing 5.5: Find Function Call Result

```
1  {
2    "identity": 1991,
3    "labels": [
4      "Node"
5    ],
6    "properties": {
```

```
7  "name": "echo",
8  "id": "FF0000000066fe197a000000005946b13c",
9  "project_name": "example-2",
10 "type": "ASFuncCallNode"
11   }
12 }
```

The listing 5.5 represent the result of the application of the example PHP program, which showing at listing 5.4, and the visualized graph shown at figure 5.1. We try to use this application to find whether this program used the “echo” function in its code. The result shows that it does call the function. According to the graph, we could know that the argument of the “echo” function has two possibilities, which are “5+5” and “5+6”. It has been specified in two data flows. Our application proves that our system can implement the searching operation within a graph database system. Finally, the table 5.1 represents the execution time for apply this application to the real PHP example that we used in evaluation chapter.

Table 5.1: Execution time table

Plugin Name	Version	Execution Time(seconds)
easy-form-builder-by-bitware	1.0.0	0.2
squarecomm	1.1.0	0.1
http-headers	1.9.4	0.2
woocommerce-checkout-manager	4.2.5	0.2
N-Upload	4.2.2	0.3
image-gallery-with-slideshow	3.3.0	0.2
estatik	4.5.3	0.5
WooCommerce-Catalog-Enquiry	3.0.1	0.3

5.2 API Dependency

The API Dependency analysis is an effective method for checking whether an API’s output is affected by another API. As long as an API was called in another API’s statement, that implies this API is not independent. It would be affected by the result of another API. While we implement dynamic program analysis on API dependency analysis, the program

needs to be executed first. Then the API dependency could be detected based on the analysis of the program execution process. Therefore, general dynamic analysis probably takes several shortcomings, such as long execution time and fuzzy detection(cannot locate independent API). On the contrary, we implement static program analysis on API dependency. Checking API dependency could be much easier and faster. In the graph that we generated by our analysis engine, all the statements within one function form several paths, and the start point of these paths is the node that represents this function. If another function node appeared in these paths, that implies this start point is not an independent API.

Listing 5.6: API Dependency Query

```
1    def dfs_traversal():
2         CALL gds.graph.create('myGraph', 'Node', 'relation')
3
4         MATCH (first:Node{type:"AsFunctionNode"})
5         MATCH (target:Node{type:"AsFunctionNode"})
6         WITH id(first) as startnode, [id(target)] as targetnodes
7         CALL gds, ...
              alpha.dfs.stream('myGraph',{startNode:startnode, ...
              targetNodes:targetnodes, maxCost:1})
8         YIELD path
9         UNWIND [ n in nodes(path) | n.name ] AS tags
10        RETURN tags
11
12        CALL gds.graph.drop('myGraph')
```

In the query, “CALL gds.graph.create()” use to create a subgraph in the database. We then used “MATCH” syntax to locate all the function nodes as start nodes and all the function nodes as target nodes and used “gds, alpha.dfs.stream” to implement a DFS algorithm to traverse the graph for finding whether there has a path exists. Finally, the result path of traversal would be returned by “UNWIND” and “RETURN” syntax. Delete the subgraph by “gds.graph.drop()”.

Listing 5.7: API Dependency Query Execution

```
1    def traverse_path()
2         with self.driver.session() as session:
3             self.create_sub_graph()
```

```
4                 res=session.read_transaction(self.dfs_traversal)
5                 self.drop_sub_graph()
6             return res
```

The listing 5.6 represents the query for implement the static program analysis to detect the API dependency, and listing 5.8 shows the source code for executing this query. In the query, we mainly use the DFS algorithm to traverse all the paths from the start node we specified. Hence, we can use this query to traverse all the function nodes' paths. Once there has another function node that is appeared in one of those paths. That implies the function node, which is the start point, is not independent in this program. For execute this query, the main code is similar to other applications, and we need to run the query through the driver we created.

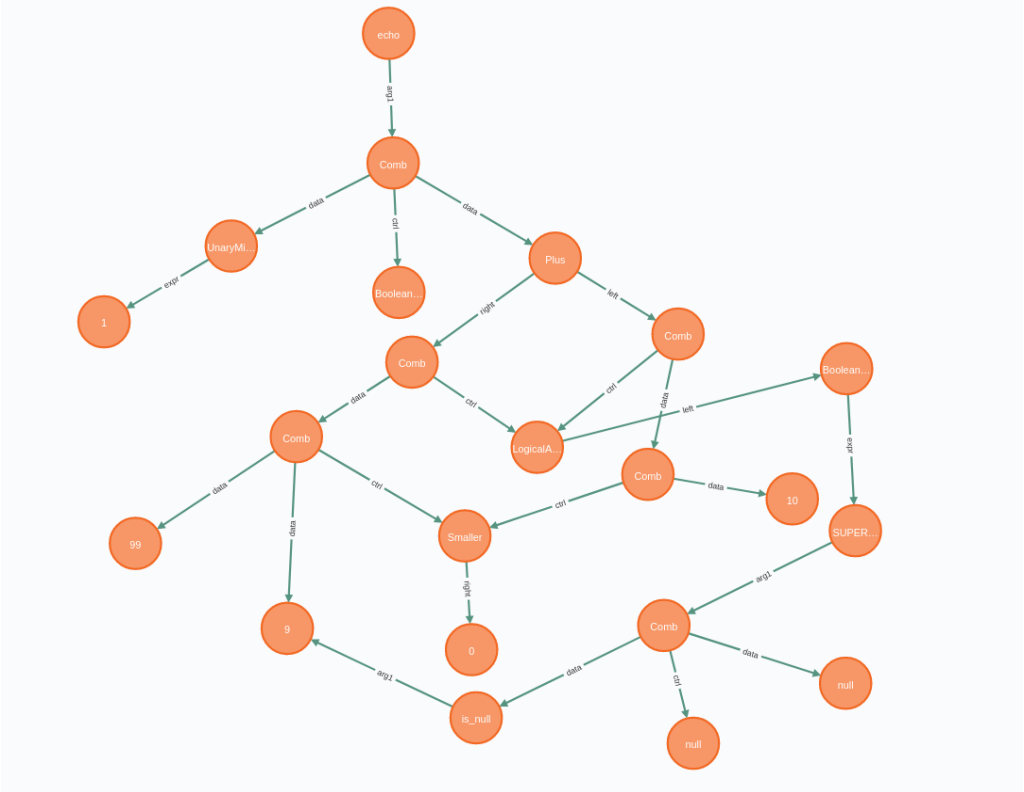

Figure 5.2: API Dependency Example Graph

Listing 5.8: API Dependency Example

```php
<?php
function positive_sum(int $x, int $y){
   $localSum = 0;
   if ($x >0 && $y > 0){
      $localSum = $x + $y;
   } else {
      $localSum = -1;
   }
   return $localSum;
}
function judgement(){
    $a = 5;
    $b = 9;
    $c = 0;

if ( $a < 0 ) {
   $a = 10;
   $b = 99;
} elseif ( !is_null($b) and $a > 10 ) {
   $b = $a;
} else {
   $c = 99999;
    }
$sum=positive_sum($a,$b);
return $sum;
}

$e=judgement();
echo $e;
?>
```

Table 5.2: Execution time table

Plugin Name	Version	Execution Time(seconds)
easy-form-builder-by-bitware	1.0.0	0.4
squarecomm	1.1.0	0.5
http-headers	1.9.4	0.7
woocommerce-checkout-manager	4.2.5	0.4
N-Upload	4.2.2	0.4
image-gallery-with-slideshow	3.3.0	0.3
estatik	4.5.3	0.5
WooCommerce-Catalog-Enquiry	3.0.1	0.7

A PHP example is shown at the listing 5.8. There mainly have three API called in the PHP example. Furthermore, one of them was called in another function. That means the

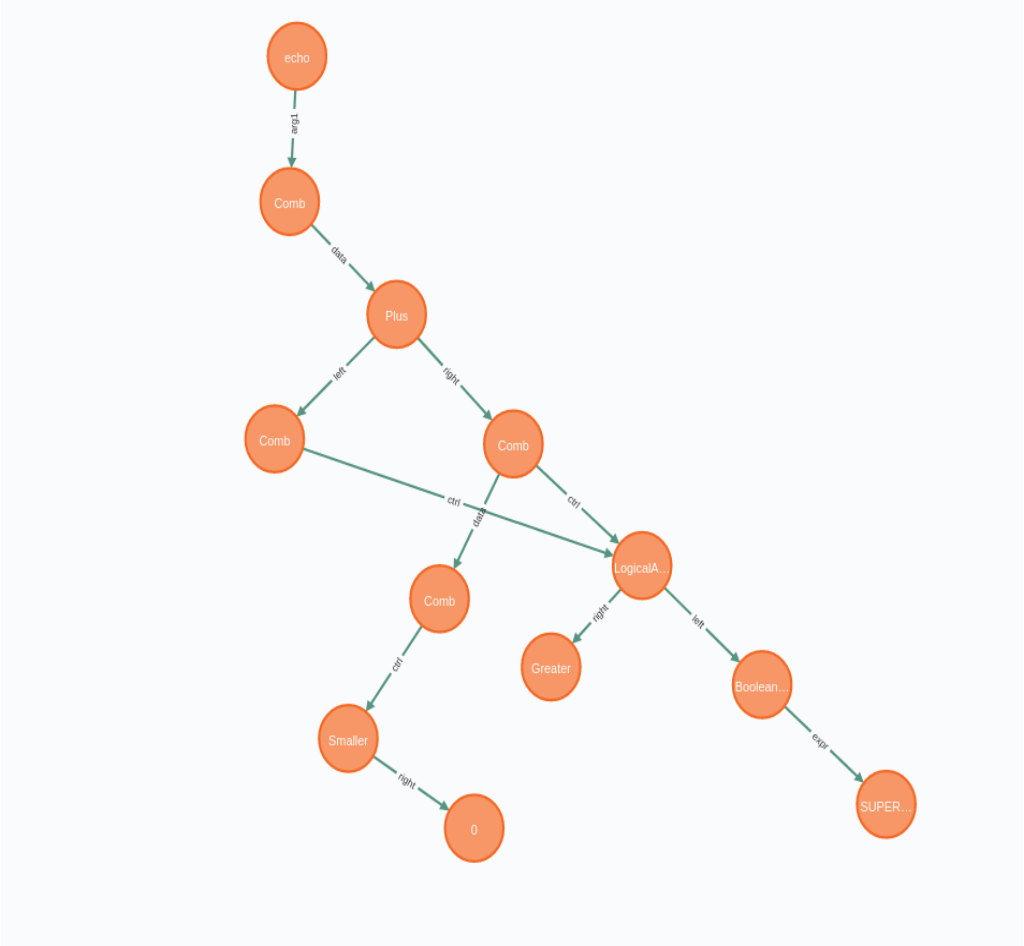

Figure 5.3: API Dependency Example Result

PHP example exists function dependency. The generated graph is shown in the figure 5.2, and the result of our proposed query is shown in the figure 5.3. According to the graph and the result, we could see there have one “FunctionCall” Node appeared in one of another function nodes’ path. That approves our method could detect the function dependency in the PHP program. Finally, the table 5.2 represents the execution time of applying it to real PHP examples.

5.3 Taint Analysis

The taint analysis is a popular method that consists of checking which variables can be modified by the user input. It aims to observe whether function computations are affected

by predefined taint source such as the user input. Typically, taint analysis is based on dynamic program analysis technology. That would have several shortcomings, such as long execution time, complicated configuration, and difficulty tracking the code's vulnerability. We tend to implement a taint analysis using our graph-database-oriented static program analysis system to overcome these shortcomings. That is what this application aims to do. Based on the graph we generated from the PHP program, all of the function calls would be generated as different paths and display in the graph. On the other hand, When PHP developers try to set input data, all the input data would be saved in a super global variable. Therefore, if we want to check whether a computation process is affected by user input, we only need to find whether a super global variable appeared in the paths of a function call node in the graph.

Listing 5.9: Taint Analysis Query

```
1
2       CALL gds.graph.create('myGraph', 'Node', 'relation')//create ...
            subgraph
3
4       MATCH (first:Node{type:"AsFunctionCallNode"})
5       MATCH (target:Node{type:"AsSuperGlobalNode"})
6       WITH id(first) as startnode, [id(target)] as targetnodes
7       CALL gds, alpha.dfs.stream('myGraph',
8       {startNode:startnode,
9       targetNodes:targetnodes, maxCost:1})
10      YIELD path
11      UNWIND [ n in nodes(path) | n.name ] AS tags
12      RETURN tags
13
14      CALL gds.graph.drop('myGraph')
```

In this query, "gds.graph.create()" and "gds.graph.drop()" used to create and delete sub-graph which used for implement the DFS algorithm. "MATCH" syntax used to locate the related node for start nodes and target nodes. "gds, alpha.dfs.stream()" used to implement the DFS algorithm. Furthermore, finally, "UNWIND" and "RETURN" used to return the resulting path of the DFS traversal.

Listing 5.10: Taint Analysis Query Execution

```
    def find_path():
        with self.driver.session() as session:
            self.create_sub_graph()
            res=session.read_transaction(self.dfs_traversal)
            self.drop_sub_graph()
        return res
```

The listing 5.9 shows the query to implement the proposed operation for taint analysis. This query aims to use the DFS algorithm to traverse all the paths that we specified. Naturally, we need to input the start node and stop node as the start point and the stop point of the traverse process. Here, we will set all the function call nodes as the start node and set all the super global node as the stop node. Once there have superglobal variables appeared in these paths, that means the function we set as start node does have a taint problem. The listing 5.10 shows the source code to implement this query, and it does not have any differences comparing with some other applications. The primary strategy is also implemented in the query by the driver we created in our analysis engine.

Listing 5.11: Taint Analysis Example

```
1 <?php
2 $uname = '';
3 if (isset($_GET['uname'])) {
4 $uname = $_GET['username'];
5 }
6 echo $uname ;
7 ?>
```

The Listing 5.11 is a PHP example to present a taint analysis application. In the PHP example, it calls a function “echo” in line 6. It is used to print out the value of the “uname” variable. The generated graph of the PHP example and the result of our proposed query’s execution are illustrated at the figure 5.4 and figure 5.5 respectively. According to the generated graph and the result of this application’s process, we can see a super global

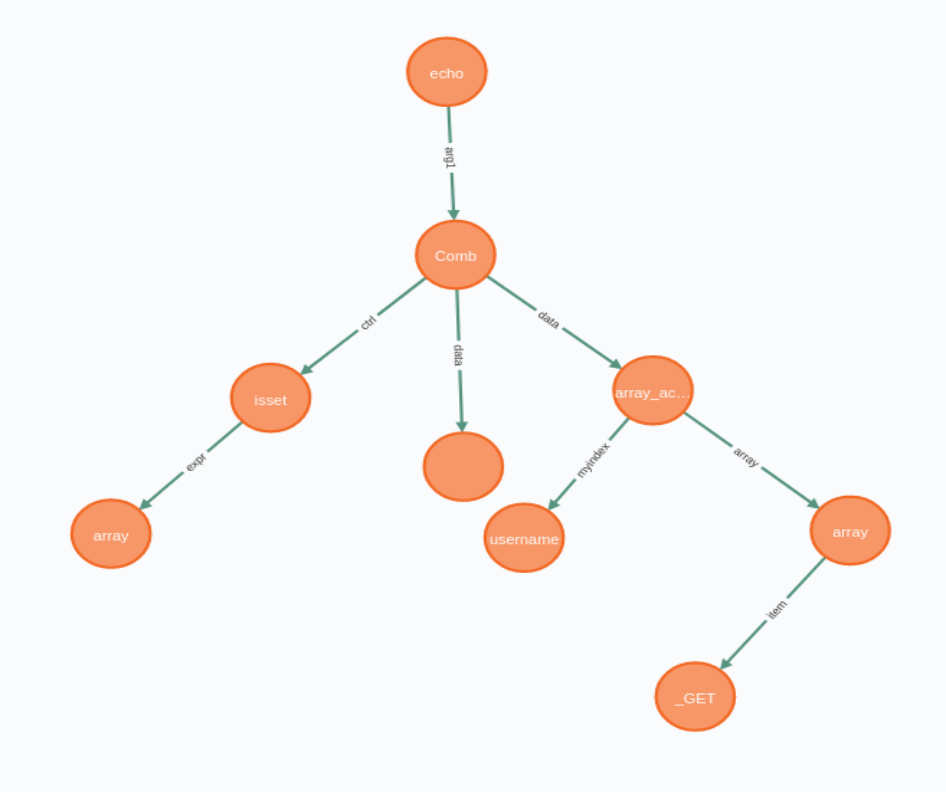

Figure 5.4: Taint Analysis Example Graph

variable that appeared in the data flow of the “As FunctionCall” node “echo.” Therefore, the taint point could be detected by our analysis engine. The table 5.3 represents the execution time of applying this application to the real work PHP examples.

Table 5.3: Execution time table

Plugin Name	Version	Execution Time(seconds)
easy-form-builder-by-bitware	1.0.0	0.2
squarecomm	1.1.0	0.3
http-headers	1.9.4	0.5
woocommerce-checkout-manager	4.2.5	0.2
N-Upload	4.2.2	0.3
image-gallery-with-slideshow	3.3.0	0.5
estatik	4.5.3	0.4
WooCommerce-Catalog-Enquiry	3.0.1	0.3

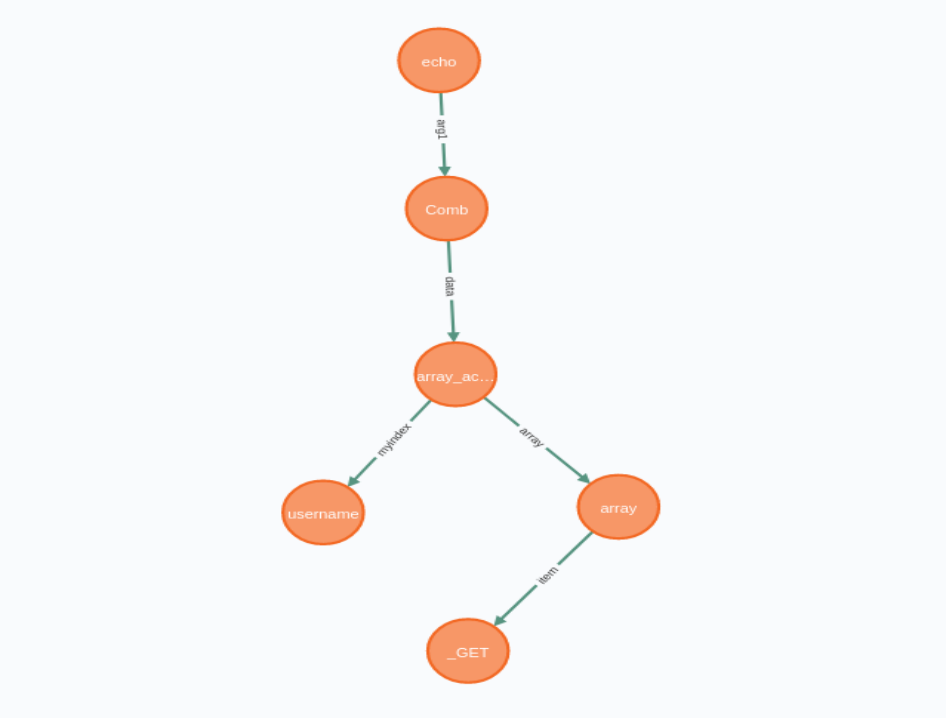

Figure 5.5: Taint Analysis Example Result

5.4 Information Leakage

Information leakage is a category of web vulnerabilities in which revealing information to an unauthorized party. However, there also exist some attackers that develop a malicious application for stealing user's sensitive data. When we access some malicious websites and input personal information, the browser or firewall probably could not detect whether it has the risk of leaking users' data. Therefore, we design and implement an application based on our analysis engine to detect whether a web application exists the risk of information leakage.

Typically, When there has "POST" or "GET" HTTP request exists in a web application and the data that the request approached is from the user's input, the web application probably exists the risk of information leakage. Therefore, we can check whether there have API exists that send "POST" request and the posted data is from the user's input to detect the risk of information leakage in a PHP program. Within the most PHP program,

"curl" is the most common API used to implement HTTP requests. Therefore, we will detect "curl" function to check the information leakage here. Our main target is checking whether it has information leakage risk. If so, return the data that the function accessed and the domain that it send to.

Listing 5.12: Information Leakage Query

```
1
2    @staticmethod
3      def get_post_urls(tx):
4          res=tx.run(
5              "MATCH (post{name:'CURLOPT_POSTFIELDS'})<--
6              (pn{name:'curl_setopt'})-->
7              (m{name:'SUPER_curl_init'})
8              <--(un{name:'curl_setopt'})
9              -->(url{name:'CURLOPT_URL'})"
10             "RETURN pn.id,un.id"
11         )
12         return res.values()
13
14     @staticmethod
15     def finding_global_nodes(tx,start):\\finding data
16         result=tx.run(
17             "MATCH (n{id:$start_id})-[r{type:'arg3'}]->(first) "
18             "WITH id(first) as startnode "
19             "CALL gds, ...
                   alpha.dfs.stream('myGraph',{startNode:startnode}) "
20             "YIELD path "
21             "UNWIND [ n in nodes(path) | n.name ] AS tags "
22             "RETURN tags",start_id=start)
23         return result.values()
24
25     @staticmethod
26     def finding_valid_url(tx,start):\\finding URL
27         result=tx.run(
28             "MATCH (n{id:$start_id})-[r{type:'arg3'}]->(first) "
29             "WITH id(first) as startnode "
30             "CALL gds, ...
                   alpha.dfs.stream('myGraph',{startNode:startnode}) "
31             "YIELD path "
32             "UNWIND [ n in nodes(path) | n] AS tags "
33             "RETURN tags.name,tags.Data_type ", start_id=start)
34         return result.values()
```

The first part of this query using "MATCH" syntax to locate whether there have "curl setopt()" function exists, and it is used to set an HTTP POST request in the project. Both of

the second part and third part implemented DFS traversal. The second part use to traversing the data flow. The target node is "Superglobal Node." If it existed in the path, an information leakage risk probably appeared. The third part is used to find the URL that the request sends to.

Listing 5.13: Information Leakage Execution

```
def information_leakage_workflow(self):
        self.delete_ctrl_node()
        self.create_sub_graph()
        post_curl=self.getting_posts_url_nodes()
        post_wp=self.check_wp_remote_node()
        if (["_POST"] in leaked_data_flow) or (["_GET"] in ...
            leaked_data_flow) or (["_REQUEST"] in leaked_data_flow):
                print(leaked_data_flow)
                print(leaked_url_flow)
                print('-------------')
                break
        self.drop_sub_graph()
```

The listing 5.14 is the queries to implement this application in the graph database. The main workflow has three steps. First of all, we detect whether there have "curl" API exists and place "POST" HTTP request in the PHP program. Normally, if there have a dataflow that can approach a "SuperGlobal Variable" node, that implies an information leakage risk occurs in the project. Therefore, we need to delete all the control flow to reduce noise. We then apply the DFS algorithm to those "curl" function nodes and check whether the function touches the user's input, which represents a superglobal variable node, in their dataflow. Finally, if such a situation occurred in API's statement, That implies it has information leakage risk, and we will return the data it touched and the domain the HTTP request sent to. The code for executing these queries are shown at listing 5.13.

Listing 5.14: Information Leakage Example

```
<?php
function translate_from_baidu ($text, $start, $end, $from = 'zh', ...
    $to = 'en') {

     $url = "http://fanyi.baidu.com/v2transapi";
```

```
5       $data = array (
6           'from' => $from,
7           'to' => $to,
8           'query' => $text
9       );
10      $data = http_build_query ( $data );
11      $ch = curl_init ();
12      curl_setopt ( $ch, CURLOPT_URL, $url );
13      curl_setopt ( $ch, CURLOPT_REFERER, "http://fanyi.baidu.com" );
14      curl_setopt ( $ch, CURLOPT_USERAGENT,
15      'Mozilla/5.0 (Windows NT 6.1; rv:37.0) Gecko/20100101 ...
            Firefox/37.0' );
16      curl_setopt ( $ch, CURLOPT_HEADER, 0 );
17      curl_setopt ( $ch, CURLOPT_POST, 1 );
18      curl_setopt ( $ch, CURLOPT_POSTFIELDS, $data );
19      curl_setopt ( $ch, CURLOPT_RETURNTRANSFER, 1 );
20      curl_setopt ( $ch, CURLOPT_TIMEOUT, 10 );
21      $result = curl_exec ( $ch );
22      curl_close ( $ch );
23      $res_json = json_encode(
24          array_merge(
25              json_decode($result, true),
26              array('start' => $start, 'end' => $end)
27          )
28      );
29      return $res_json;
30  }
31
32  echo translate_from_baidu($_GET['query'],
33  $_GET['start'], $_GET['end'],
34  $_GET['from'],$_GET['to']);
```

Table 5.4: Execution time table

Plugin Name	Version	Execution Time(seconds)
easy-form-builder-by-bitware	1.0.0	0.7
squarecomm	1.1.0	0.5
http-headers	1.9.4	0.9
woocommerce-checkout-manager	4.2.5	0.6
N-Upload	4.2.2	0.8
image-gallery-with-slideshow	3.3.0	0.6
estatik	4.5.3	0.8
WooCommerce-Catalog-Enquiry	3.0.1	0.9

A real PHP example is shown at the listing 5.14, and the generated graph is shown

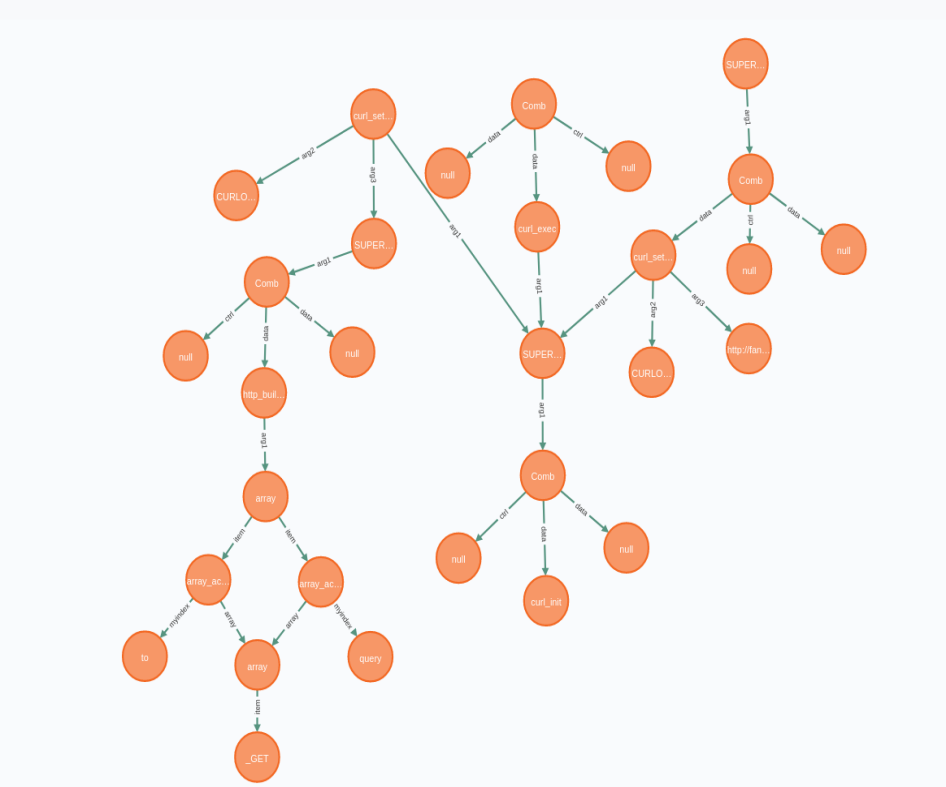

Figure 5.6: Information Leakage Example Graph

at the figure 5.6. In the PHP project, an HTTP "POST" request was set by the "curl setopt" function in line 20, and this request touched the user's data. Therefore, it exists an information leakage risk. Our application's result is shown in the figure 5.7. According to our application result, there have a "SuperGlobal" node exists in the "curl setopt" function node's dataflow, which implies the data of the HTTP "POST" request is from the user's input. From another "curl setopt" function node path, the URL is also shown in its dataflow. All in all, our system correctly detects the information leakage risk from the generated graph. The table 5.4 presents the execution time of this application on real world PHP examples.

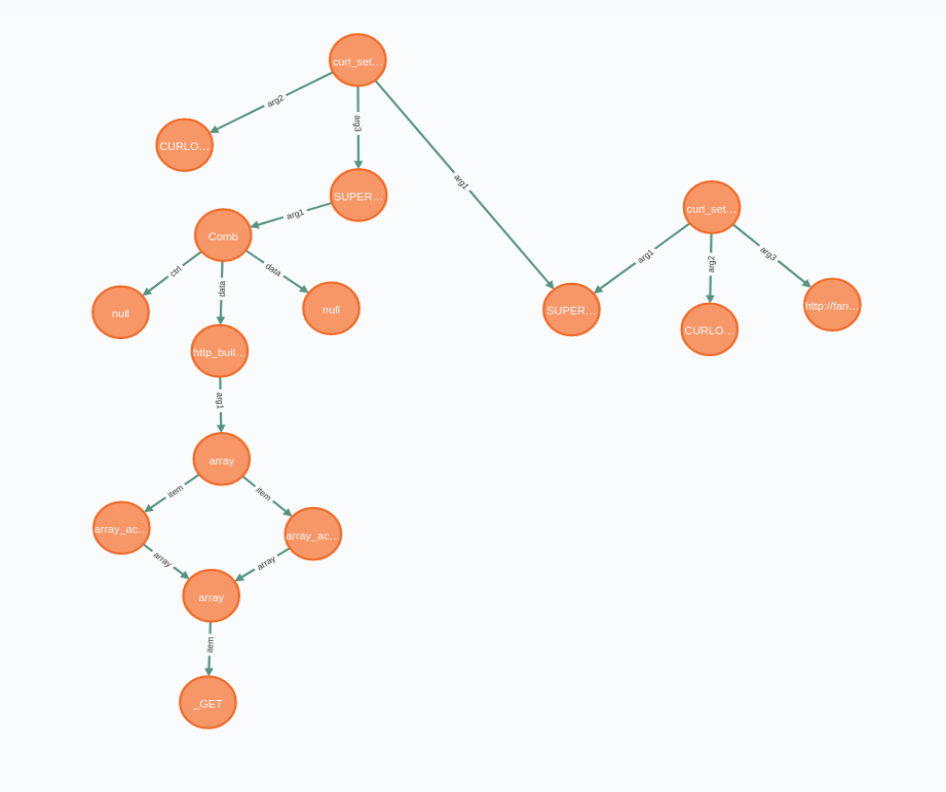

Figure 5.7: Information Leakage Example Result

Conclusion

Our graph-database-oriented static program analysis system, Which inherits the basic definition of symbolic execution from the "Uchecker" system, fundamentally differs from the "Uchecker" program analysis system that mainly takes the PHP program's abstract syntax tree and analysis the PHP program based on the abstract syntax tree. Comparing with the "Uchecker" system that analysis the PHP program on the Abstract syntax tree directly, our system could insert the abstract syntax tree to a graph database we created, and our analysis would be based on the graph data in the graph database. Performing our graph-database-oriented system offers several unique advantages. First of all, the graph database not only could convert the abstract syntax tree to a graph, but it also could be visualized on the server of the graph database. That is much more convenient than "Uchecker" system. Second, Because the entire PHP program has been stored as data in the database, the entire analysis process could be implemented by executing different database queries. That is much easier than the design algorithm to implement the analysis process. Third, The database is much more flexible than "Uchecker" system. In the "Uchecker" system, we need to analyze the program based on the entire abstract syntax tree. That would cause noisy issue when the size of the PHP program is enormous. Compared with that point, our system can delete the PHP program's insignificant part by executing the related query.

Many different adaptations, tests, and optimization have been left for the future due to our future works' lack of time. First of all, there are some ideas that we would have liked to try to optimize our system. It includes design and implements concurrent query

execution, which can save more time on query execution. It could have better performance in execution time and deploy on the cloud platform, which could improve storage capacity to insert more data into the graph database. On the other hand, A more in-depth analysis is also an essential part of our future work. While our system converts PHP programs to different sets of data, our analysis method could be designed more flexibly. Most analysis method we designed is all about the implementation of the database so far. However, data science could be a new area for design analysis method that we can concern. Generally, data science is widely used in data regression and data classification. That also could be applied to our PHP program data set in the future. Therefore, in a new direction, that would be the primary concern in our future work.

Bibliography

[1] H. Kienle R. Koschke J. Czeranski, T. Eisenbarth and D. Simon. Analyzing xfig using the bauhaus tool. *Proceedings Seventh Working Conference on Reverse Engineering*, 3(4):197–199, 2000.

[2] Cristian Cadar, Daniel Dunbar, Dawson R Engler, et al. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *OSDI*, volume 8, pages 209–224, 2008.

[3] Klaus Havelund and Thomas Pressburger. Model checking java programs using java pathfinder. *International Journal on Software Tools for Technology Transfer*, 2(4):366–381, 2000.

[4] Vitaly Chipounov, Volodymyr Kuznetsov, and George Candea. S2e: A platform for in-vivo multi-path analysis of software systems. *ACM SIGPLAN Notices*, 46(3):265–278, 2011.

[5] Tielei Wang, Tao Wei, Zhiqiang Lin, and Wei Zou. Intscope: Automatically detecting integer overflow vulnerability in x86 binary using symbolic execution. In *NDSS*, 2009.

[6] Olivier Coudert, Christian Berthet, and Jean Christophe Madre. Verification of synchronous sequential machines based on symbolic execution. In *International Conference on Computer Aided Verification*, pages 365–373. Springer, 1989.

[7] Sarfraz Khurshid, Corina S Păsăreanu, and Willem Visser. Generalized symbolic execution for model checking and testing. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 553–568. Springer, 2003.

[8] Edward J Schwartz, Thanassis Avgerinos, and David Brumley. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *Security and privacy (SP), 2010 IEEE symposium on*, pages 317–331. IEEE, 2010.

[9] Robert S Boyer, Bernard Elspas, and Karl N Levitt. Select—a formal system for testing and debugging programs by symbolic execution. *ACM SigPlan Notices*, 10(6):234–245, 1975.

[10] WordPress Plugins – plugins extend and expand the functionality of your website. https://wordpress.org/plugins/. Accessed: 2020-05-1.

[11] Github PHP CMS project – top star php-oriented content management system(cms). \OT1\textquotedblrighthttps://github.com/topics/php?o=desc&q=cms&s=stars. Accessed: 2020-10-15.

[12] S. Son and V. Shmatikov. Saferphp: finding semantic vulnerabilities in php applications. *ACM SIGPLAN Conference on Programming Language Design and Implementation*, (8):1–13, 2011.

[13] H. Do A. Marback and N. Ehresmann. An effective regression testing approach for php web applications. *IEEE Fifth International Conference on Software Testing, Verification and Validation*, pages 221–230, 2012.

[14] J. Zhang J. Huang, Y. Li and R. Dai. Uchecker: Automatically detecting php-based unrestricted file upload vulnerabilities. *International Conference on Dependable Systems and Networks(DNS)*, 3(4):581–592, 2019.