上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

PHP-WEB漏洞挖掘

设计文档

小组成员：金子本、王新忠、洪逸杰、张泽挥、杨景犀、钟一鸣

项目名称：PHP-WEB漏洞挖掘

指导老师：黄征

日期：2022年5月27日

目录

第一章 前端设计

1.1 前端设计整体思路

在前端设计上，我们使用的基本技术还是经典的html、css、javascript三件套，其中html负责网页的框架，CSS负责样式的修饰与美化，JavaScript来调整网页效果并与后端进行交互。

我们整个项目由两个网页组成，一个网页是homepage，也就是主页；通过主页，我们能通过单击鼠标跳转到核心功能页面上。在后续的工作中，我们也会在主页上增添更多的页面与功能，例如登陆页面，发现的历史漏洞页面，与其他用户之间的沟通交流页面等。

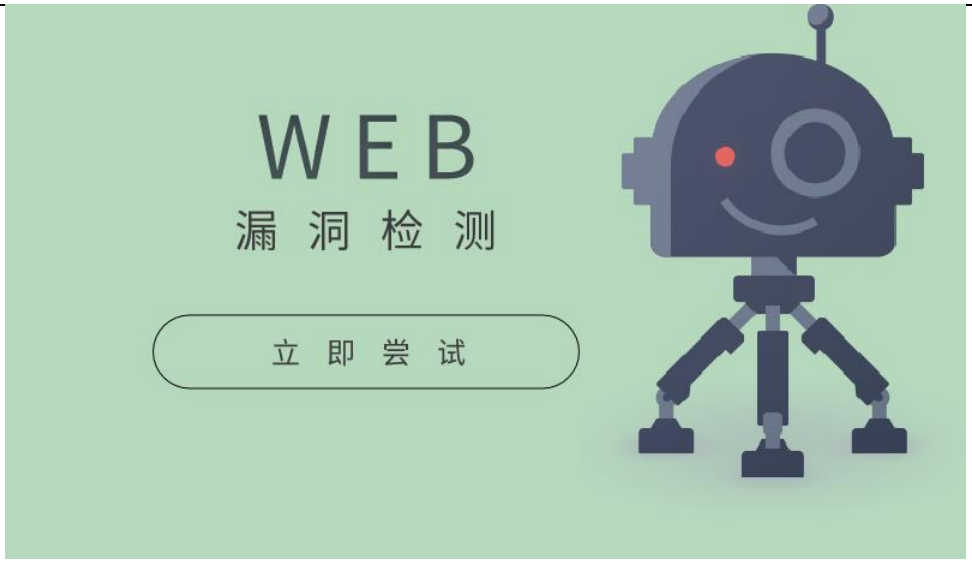
1.2 海报页面设计

在整个网页的设计时，我们主要考虑的是简洁美观。设计海报页面时，因为之前有在sentry.io网站看到过小机器人的图标，而这个机器人元素又与项目本身非常契合。所以我们借用了小机器人的图标作为海报界面最主要的元素。左边配文说明网站的用途。背景颜色使用了看起来清新而美观的淡绿色。

在该页面，我们的html框架由两个分割块组成，在第一个分割块中，我们完成了页面的跳转工作，跳转链接的对象为新插入的封面图片，这样便可以实现点击封面图片便跳转到具体页面的需求。

```
<div class="b1">
    <p  style="text-align: center;
        position: relative;
        padding-left: 300px;
        padding-top:40px ;">
    <span class="anim"></span>
    <a href="index.html" >
    <img src="./img/banner.png" style="display:flex"/>
</a>
</p>
</div>
```

海报的图片设计使用了MasterGo，这是一个免费的设计软件。使用这个工具可以进行团队的协作设计。因为我们只是使用这个平台来进行主要元素的组合，所以使用起来比较简单。图片配文以及机器人元素的添加都是通过这个工具完成。



为了让海报的元素更符合项目使用程序进行自动的漏洞挖掘的特点，我们添加了一个眨眼的动画。小机器人的眨眼动画使用了css的animation属性，使用animation可以创建动画，它可以取代许多网页动画图像、Flash 动画和JavaScript 实现的效果。

```
.anim{
    width: 18px;
    height: 18px;
    border-radius: 50%;
    position: absolute;
    top: 157.8px;
    left: 890.5px;
    transform: translateZ(0px);
    animation: 1s ease 0s infinite normal none running blink-eye;

}
@keyframes blink-eye{
0% {
    background: rgb(224, 62, 47);
    box-shadow: rgb(224 62 47) 0px 0px 10px;
}
50% {
    background: rgb(74, 77, 103);
    box-shadow: none;
}
100% {
    background: rgb(224, 62, 47);
    box-shadow: rgb(224 62 47) 0px 0px 10px;
}
}
```

1.3 核心功能页面设计

在核心功能功能页面的html实现上，我们将其分为三个主要模块来进行实现。他们分别是：上传文件的代码模块、用户手动输入代码的检测模块以及检测结果的输出模块。其主要逻辑功能由jquery实现。在每个div块中，我们定义其具体的

class名、id名，以便后面实现css功能及javascript功能时使用。

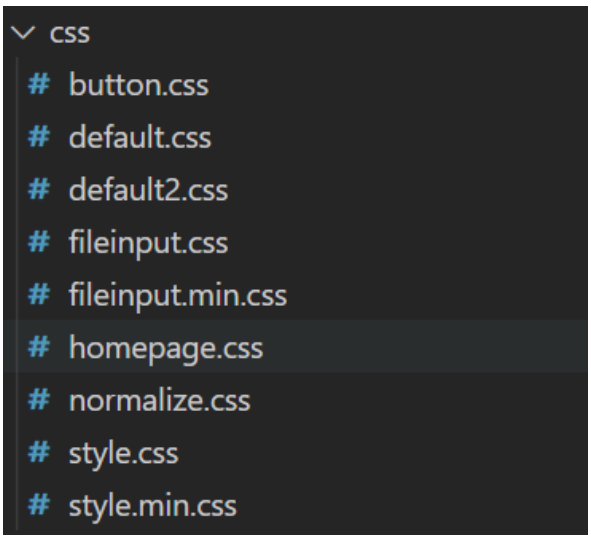
```
<form enctype="multipart/form-data">
    <label>请上传你的代码文件</label>
    <input id="file-zh" name="file-zh[]" type="file" multiple>
    <hr style="border: 2px dotted">

</form>
<hr>
<h4>或者在这里输入你的代码</h4>
<div class="code-submit" style="
    border: 1px solid rgb(21, 20, 22);
    border-radius: 5px;
">
<form enctype="multipart/form-data">
    <textarea id="file-zh-input" name="file-zh-input" rows="14" cols="110" onblur="checkblur1()" mu
    </textarea>
</form>
<div class="submitbtn">
    <button   class="bttn-gradient bttn-md bttn-primary" type = "submit" name = "btn" value = "提交
    id = "btnId" onclick="upload()" >
    提交
   </button>
    </div>
</div>
<br>
<div style="
```

```
    <div calss="submitbtn">
    <button herf="./css.but.css" class="bttn-gradient bttn-md bttn-primary" type = "submit" name =
    id = "btnId" onclick="check()" >
    检测
   </button>
    </div>
   <textarea readonly="readonly" id="test_textcontent" rows="14" cols="110" placeholder="检测结果"
   </div>
</div>

div>
```

我们主要的CSS代码保存在CSS文件夹中，其中fileinput.css文件负责文件上传模块的修饰美化，homepage.css文件负责主页页面的美化与修饰，default.css文件负责全局变量的美化与修饰，例如字体、背景等。button.css文件负责按钮样式的美化修饰。



JavaScript来调整网页效果并与后端进行交互。在此基础上，我们运用了jQuery Ajax，jQuery是一个JavaScript库，旨在简化HTML DOM树的遍历和操作，以及进行事件处理、CSS动画与Ajax。JQueery的应用具体表现为文件上传功能的实现，在实现过程中，我们通过文件id的传递来实现html部分与JQuery部分的分离。

File-zh为文件上传模块的id，在JQuery的实现中，我们定义了上传文件的语言，上传服务器地址并且限制了文件上传的类型。文件上传的具体工作由bootstrap提供的文件上传组件来实现。

```
$('#file-zh').fileinput({
    language: 'zh',
    uploadUrl: 'http://127.0.0.1:8080/uploadCommon',
    allowedFileExtensions : ['jpg', 'png','gif','txt','doc','docx','php'],
});
$('#file-zh-input').val({
    language: 'zh',
    uploadUrl: '#',
    allowedFileExtensions : ['jpg', 'png','gif','txt','doc','docx'],
});
```

在upload函数中，我们通过Ajax来实现与后端服务器之间的上传交互。

```
function upload() {
var code = $("#file-zh-input").val();
$.ajax({
type: "POST",
url: "http://127.0.0.1:8080/upload",
data: {
    "code" : code
},
dataType: "json",
success: function(data) {
    alert("上传成功");
}
```

我们在前端开发过程中还采用了bootstrap插件库中提供的很多套件，Bootstrap是一组用于网站和网框架，包括HTML、CSS及JavaScript的框架，提供字体排印、表单、按钮、导航及其他各种元件及Javascript扩充套件。在本次开发过程中，我们通过bootstrap提供的文件上传组件来实现了从文件系统中获取上传文件的功能。该组件可以兼容Windows操作系统与Linux操作系统。

第二章 后端设计

2.1 后端整体设计思路

前端上传的文件或者输入的代码会存储到后端的一个固定的目录中。前端点击检测后后端就会调用我们写好的脚本，对存储在固定目录中php文件进行分析并输出检测的结果，我们将检测结果打包成json数据包再返回给前端。

2.2 后端框架选择

后端我们选择了Spring Boot这个框架来进行编写：

Spring Boot 是在 Spring 的基础上提供的一套全新的开源框架，简化了 Spring 应用的搭建和开发过程。Spring Boot 去除了大量的 XML 配置文件，简化了复杂的依赖管理。而且集成了大量常用的第三方库配置，使得开发者能够更加专注于业务逻辑，比较方便。Spring Boot具有可以创建独立的Spring应用程序，并且基于其Maven或Gradle插件，可以创建可执行的JARs和WARs；内嵌Tomcat或Jetty等Servlet容器；尽可能自动配置Spring容器；提供准备好的特性，如指标、健康检查和外部化配置等优点。因此我们选用该矿价进行网站后端的搭建。

2.3 具体函数实现

2.3.1 上传文件函数uploadcommon

在这个函数中，我们首先接收到前端传来的MultipartFile类型的数据，这个类型就是前端传来的文件的集合。然后我们选择文件的固定存储位置为：

```
String folder = "/home/test/Desktop/upload/received/";
```

定义好存储位置后，我们结合该位置与用户上传文件的文件名，将其作为我们最终需要存储的文件。新建一个File文件，将路径和文件名信息传进去：

```
File filelocal = new File(folder, Name);
```

最后将MultipartFile中的数据传送到filelocal实现文件的保存并向前端发送确认信息。

```
multipartFile.transferTo(filelocal);
```

2.3.2 输入代码函数upload

在这个函数，我们会接收到前端传来的代码数据。在与前端的同学交流后，他们会以ajax的形式将数据传送回来。后端设计的函数接受的参数为HttpServletRequest request，我们通过这个函数获得前端用户输入的code变量：

```
String code = request.getParameter("code");
```

获得code变量后，我们就和上传文件函数一样的处理，定义存储位置并新建文件。传进来的php代码我们默认存储在default.php中：

```
String path = "/home/test/Desktop/upload/received/";
File writename = new File(path+"default.php");
```

2.3.3 检测函数

前端点击检测会发出请求，后端接收到请求后开始进行检测的工作。后端主要调用提前写好的三个脚本build.sh，start_neo4j.sh，start_pyjoern.sh，其中build.sh就是将php导入图神经网络的模型中确定该文件是否存在漏洞，start_neo4j.sh则是若有漏洞，则生成CPG代码属性图并导入neo4j图数据库中，最后调用start_pyjoern.sh脚本对图数据库进行查询并得到漏洞所在的具体位置。我们会获取最后检测脚本的输出：

```
Process p3=builder3.start();
BufferedReader stdInput = new BufferedReader(new
InputStreamReader(p3.getInputStream()));
```

最后我们将获得的检测结果打包成一个json数据包并传送到前端：

```
result.put("test_textcontent",text);
return result.toString();
```

前端收到数据后刷新显示文本框，至此一次完整的检测过程就完成了。

第三章 图卷积神经网络

在图神经网络的构建中，主要包括代码向量化和图卷积神经网络构建这两个阶段，在代码向量化阶段，主要是将php代码抽象成一种能够在概念上帮助神经网络学习的过程；在图神经网络阶段，主要是提取代码的特征，通过反向传播来学习它们的重要性。

3.1 代码向量化

在我们的设计中，我们使用的数据是文件级别的php代码，也就是我们以整体的形式来分析php文件。在这个过程中我们通过提取解析标记的线性序列以捕获语法的依赖性，还提取了控制流图并将控制流图转换以供输入神经网络。

3.1.1 标记序列

在生成标记序列的过程中，我们使用了phply对php文件进行语义分析，phply是PHP编程语言的解析器，使用PLY编写，PLY是一个针对Python的Lex/yacc风格的解析器生成器工具包。而在使用的过程中，我们将用户定义的函数、变量和常数值等进行抽象标记，并且只保留每个文件中前200个变量的标记。因为一些php产生的漏洞与一些内置的函数有关，所以我们还对一些php函数保留了具体的标记，例如query、exec、strip_tags等函数。除此之外因为注释、php打开和关闭的标签并不会存在漏洞，我们在处理的过程中将它们删去，具体如下图所示。

```
for type in ['SQLi', 'XSS', 'Command injection']:
    no_flaws_cur = 0
    no_cur = 0
    print(type)
    dir = "../SARD/" + type
    tree = etree.parse(dir + "/manifest.xml")
    ignore = {'WHITESPACE', 'OPEN_TAG', 'CLOSE_TAG'}
    for file in tree.findall('testcase/file'):
        p = file.get('path')
        file_data = []

        if not ((type == 'XSS' and p.startswith('CWE_79'))
            or (type == 'SQLi' and p.startswith('CWE_89'))
            or (type == 'Command injection' and p.startswith('CWE_78'))):
            continue
        lexer = phplex.lexer.clone()
        file_path = dir + '/' + p
        code_tokens = []
        codetokens = []
        allvar = []
        with open(file_path, "r") as myfile:
            data = myfile.read()
            lexer.input(data)
            while True:
                tok = lexer.token()
                if not tok:
                    break
                if tok in ignore:
                    continue
                tok1 = sub_tokens.sub_token(tok,None,allvar)
                allvar = list(allvar)
                code_tokens.append(tok1)
                codetokens.append(sub_tokens.sub_token(tok))
```

接下来因为使用的循环门单元需要固定长度的向量作为输入，我们使用scikit-learn中的LabelEncoder根据标记表将每一个标记转换为一个数字。我们还对从每个文件中提取的标志数量进行了限值，因为提取的是文件级别的样本，我们设置的提取数量是3000个标记。如果标记数量超过了3000个，则将多余的部分舍弃，这样的做法虽然会导致相关信息的丢失，但是因为文件长度是不确定，并且3000个标记的数量可以分析大部分php文件的信息，所以这样的做法应该不会导致漏报率的大幅提升；如果php文件中没有那么多的标记数量，我们则将使用0将剩余的标记补充完整。具体的代码如下。

```
tokens = sub_tokens.getReplacedTokens() + reserved + unparsed + [
    # Operators
    'PLUS', 'MINUS', 'MUL', 'DIV', 'MOD', 'AND', 'OR', 'NOT', 'XOR', 'SL',
    'SR', 'BOOLEAN_AND', 'BOOLEAN_OR', 'BOOLEAN_NOT', 'IS_SMALLER',
    'IS_GREATER', 'IS_SMALLER_OR_EQUAL', 'IS_GREATER_OR_EQUAL', 'IS_EQUAL',
    'IS_NOT_EQUAL', 'IS_IDENTICAL', 'IS_NOT_IDENTICAL',

    # Assignment operators
    'EQUALS', 'MUL_EQUAL', 'DIV_EQUAL', 'MOD_EQUAL', 'PLUS_EQUAL',
    'MINUS_EQUAL', 'SL_EQUAL', 'SR_EQUAL', 'AND_EQUAL', 'OR_EQUAL',
    'XOR_EQUAL', 'CONCAT_EQUAL',

    # Increment/decrement
    'INC', 'DEC',

    # Arrows
    'OBJECT_OPERATOR', 'DOUBLE_ARROW', 'DOUBLE_COLON',

    # Delimiters
    'LPAREN', 'RPAREN', 'LBRACKET', 'RBRACKET', 'LBRACE', 'RBRACE', 'DOLLAR',
    'COMMA', 'CONCAT', 'QUESTION', 'COLON', 'SEMI', 'AT', 'NS_SEPARATOR',

    # Casts
    'ARRAY_CAST', 'BINARY_CAST', 'BOOL_CAST', 'DOUBLE_CAST', 'INT_CAST',
    'OBJECT_CAST', 'STRING_CAST', 'UNSET_CAST',
```

```
combined = sql + echo + input + preg +  exec + filtering + html + additionals + casting + ["FUNCTION_CALL"]
def sub_token(token, allfunc=None,allvar=None):
    if token.value in combined :
        return token.value.upper()
    else:
        if allfunc is not None and token.value in allfunc:
            return 'FUNCTION_CALL'
        if allvar is not None and token.type == 'VARIABLE' :
            if token.value not in allvar:
                allvar.append(token.value)
            if allvar.index(token.value) < 10:
                return 'VAR' + str(allvar.index(token.value))
        # elif allvar is not None and token.value in allvar:
        #   print("here")
        #   print(allvar)
        #   print(token.type)
        #   return 'STRING_WITH_VAR'
        # if token.type == "LNUMBER" and RepresentsInt(token.value) and int(token.value) < 10:
        #   return str(int(token.value))
        return token.type


def sub_token_ast(token, allfunc=None):
    if token in combined :
        return token.upper()
    else:
        return 'FunctionCall'
```

```
        if len(codeTokens) < MAX_TOKEN_LEN:
            single_sample_tokens.extend(np.zeros(MAX_TOKEN_LEN-len(codeTokens), dtype=int))
        if len(codeTokens) > MAX_TOKEN_LEN:
            codeTokens = codeTokens[:MAX_TOKEN_LEN]
        if len(codeTokens) == 0:
            continue
```

3.1.2 控制流图

控制流图的生成在语法阶段，我们使用phpjoren将每一个文件解析为一个抽象语法树，再提取出控制流图，在这个过程中我们把每一行代码视作一个节点，转换成数值向量。为了输入到图卷积神经网络，我们将节点整理成一个二维的矩阵，其中每一行包含与行索引对应的节点的特征，接下来我们通过元组向量（i，j）表示CFG边，该向量表示节点i到节点j的一条有向边。

```
    cfg = listener.get_graph()
    edges = [[list(cfg.nodes).index(i),list(cfg.nodes).index(j)] for (i,j) in cfg.edges]
    edges = torch.tensor(edges,  dtype=torch.long)
    allvars = []
    for node in list(cfg.nodes):
        for var in node.stmt_vars:
            if var not in allvars:
                allvars.append(var)
    graph_nodes = [process(node.text,list(allvars)) for node in list(cfg.nodes)]
    flaw = file.find('flaw')
    if flaw is not None:
        if ",".join(codetokens) + str(i) not in data_flaw and ",".join(codetokens) not in data_un:
            data = Data(x=torch.tensor(graph_nodes), edge_index=edges.t().contiguous(),y=i)
            data_un.add(",".join(codetokens))
            data_flaw.add(",".join(codetokens) + str(i))
            # number here as it won't pickle as a Data object otherwise
            data_ls.append([data,1])
            no_flaws += 1
```

3.2 图卷积神经网络

在我们的神经网络架构中，我们首先是输入嵌入层，将上述产生的每个数值输入转换为向量；然后我们将标记向量输入双向门控循环单元，从标记向量中提取特征；最后我们将提取到的CFG图输入到图卷积神经网络中，提取漏洞代码的特征。

3.2.1 嵌入层

在这一阶段我们有两个嵌入层，一个嵌入层用来处理标记的数值，另一个嵌入层用来处理CFG图，这两个嵌入层都使用长度为100的向量，即使用一个函数将一个数值映射到一个1×100维的向量。该函数主要是将字符转为十进制数，如果不够100维，则用0补齐。

```
max_len = 100
def process_char(line):
    chars = list(line)
    ret = []
    i = 0
    for c in chars:
        if i < max_len:
            ret.append(ord(c))
        else:
            break
        i += 1
    if i < max_len:
        for j in range(max_len - i):
            ret.append(0)
    if len(ret) != 100:
        print("error")
        print(len(ret))
        print(ret)
        exit(1)
    return ret
```

3.2.2 GRU

因为导致漏洞的代码在很大程度上依赖于php的语法以及它们的上下文，而大多数的标记都携带着关于标记序列中下一个标记的信息，该信息流一直传播到代码语句结束，我们使用GRU学习整个源代码中的代码标记的双向序列来利用这些信息流。

```
class PhpNet(nn.Module):
    def __init__(self, EMBED_SIZE, EMBED_DIM, LSTM_SIZE,
                 LSTM_LAYER, LSTM_BIDIR, FC_DROP, FC_OUT):
        super(PhpNet, self).__init__()

        self.embed = nn.Embedding(num_embeddings=EMBED_SIZE,
                                  embedding_dim=EMBED_DIM)
        # self.multi = nn.MultiheadAttention(EMBED_DIM, 5)
        self.lstm1 = nn.GRU(input_size=EMBED_DIM,
                            hidden_size=LSTM_SIZE,
                            num_layers=LSTM_LAYER,
                            batch_first=True,
                            bidirectional=LSTM_BIDIR)
```

3.2.3 图卷积神经网络

最后我们使用图卷积神经网络来提取CFG中的特征，它能够将依赖关系嵌入到我们的模型中，并通过反向传播来学习它们的重要性。在神经网络的选择上，我们使用了三层GCN每一层后面跟着一层边池。然后我们取图卷积层的输出，并使用最大池化将其变平。最后我们结合GCN和GRU的输出向量，将它们输入线性分类层，再通过ReLU激活函数得到每个类的概率向量，用来判别文件是否存在漏洞。

```
710  class PhpNetGraphTokensCombineFileLevel(nn.Module):
711      def __init__(self, VOCAB_SIZE_graph, VOCAB_SIZE_tokens):
712          super(PhpNetGraphTokensCombineFileLevel, self).__init__()
713          self.embed1 = nn.Embedding(num_embeddings=VOCAB_SIZE_graph,
714                                     embedding_dim=100)
715          self.conv1 = GCNConv(2000,2000)
716          self.pool1 = EdgePooling(2000)
717          self.conv2 = GCNConv(2000, 4000)
718          self.pool2 = EdgePooling(4000)
719          self.conv3 = GCNConv(4000, 4000)
720          self.pool3 = EdgePooling(4000)
721
722          self.embed = nn.Embedding(num_embeddings=VOCAB_SIZE_tokens,
723                                    embedding_dim=100)
724          self.lstm1 = nn.GRU(input_size=100,
725                              hidden_size=200,
726                              num_layers=3,
727                              batch_first=True,
728                              bidirectional=True)
729
730          self.lin1 = nn.Linear(5200, 1000)
731          self.lin2 = nn.Linear(1000, 500)
732          self.lin3 = nn.Linear(500, 4)
```

第四章 静态代码分析

在通过神经网络分辨后，有漏洞的文件将被送入静态代码分析，查找具体有漏洞的位置。在静态代码分析阶段，主要是构建代码属性图，然后通过代码属性图后向遍历来查找具体的污点路径。

4.1 代码属性图

我们通过构建一个图形结构来表示可能的执行路径，具体来说该模型的建立是基于代码属性图。代码属性图整合了抽象语法树、控制流图和程序依赖图，其组成部分主要包括：

1) 节点及其类型。节点代表程序的组成部分，包含有底层语言的函数、变量、控制结构等。
2) 有向边及标签。节点之间的边代表它们之间的关系，而标签是关系的说明。
3) 键值对。节点带有键值对，即属性，键取决于节点的类型。比如一个函数至少会有一个方法名，而一个局部变量至少有名字和类型。

代码属性图的生成效果如下。

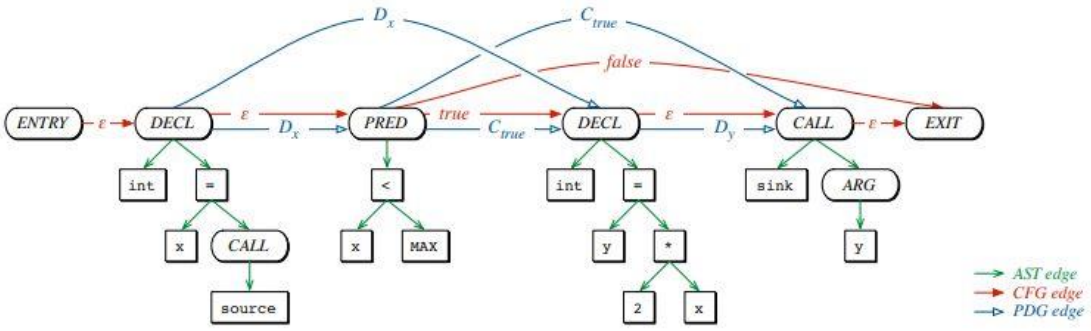
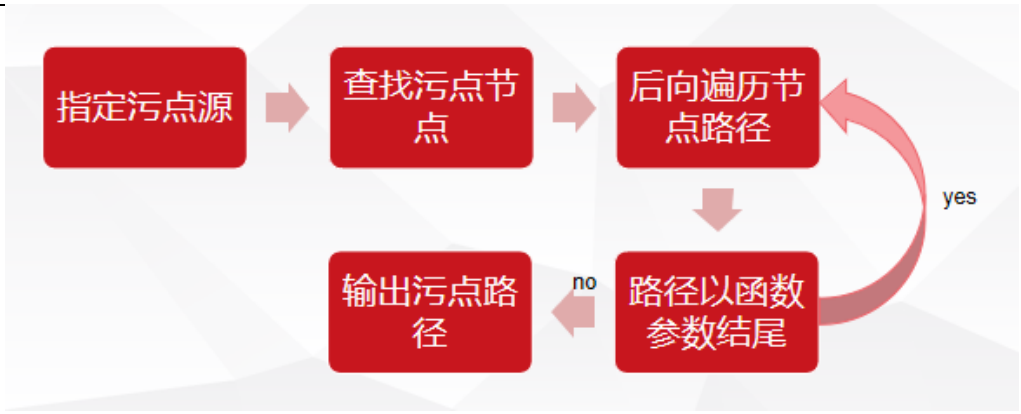


Fig. 4: Code property graph for the code sample given in Figure 1.

4.2 查找污点路径

在污点路径的查找方面，我们首先指定源节点和污点源，然后在代码属性图中查询源节点和污点源之间路径。例如sql注入漏洞，源节点$_REQUEST['id']，污点函数为mysqli_query等，我们查找包含这些函数的节点，通过后向遍历找到源节点到污点源之间的路径。需要注意的是，如果最终查找的路径以函数结尾，说明该函数也被污染，需要继续遍历该节点，否则输出污点路径。




```
sql_query_funcs = ["mysql_query", "mysqli_query", "pg_query", "sqlite_query"]

repairs = ["md5", "addslashes", "mysqli_real_escape_string", "mysql_escape_string"]

Paths = []

VulnerablePaths = g.V().filter{sql_query_funcs.contains(it.code)  && isCallExpression(it.nameToCall().next())}.as('sink')
.callexpressions().as('sloop').statements().inE('REACHES').outV.loop('sloop')
{ it.loops < 10  }{ it.object.containsLowSource().toList() != []}.path().toList()


for ( vpaths in VulnerablePaths){
    vlen = vpaths.size()
    p = []

    for (int i = 1; i < vlen ; i+=2) {
        node = vpaths[i];
        if ( i+1 < vlen){
            source_var = vpaths[i+1].var;
        }
        else{
            source_var = vpaths[i].match {
                            it.type == "AST_VAR" &&
                            it.containsLowSource().toList().size() > 0 }.varToName().toList()[0]
        }

```