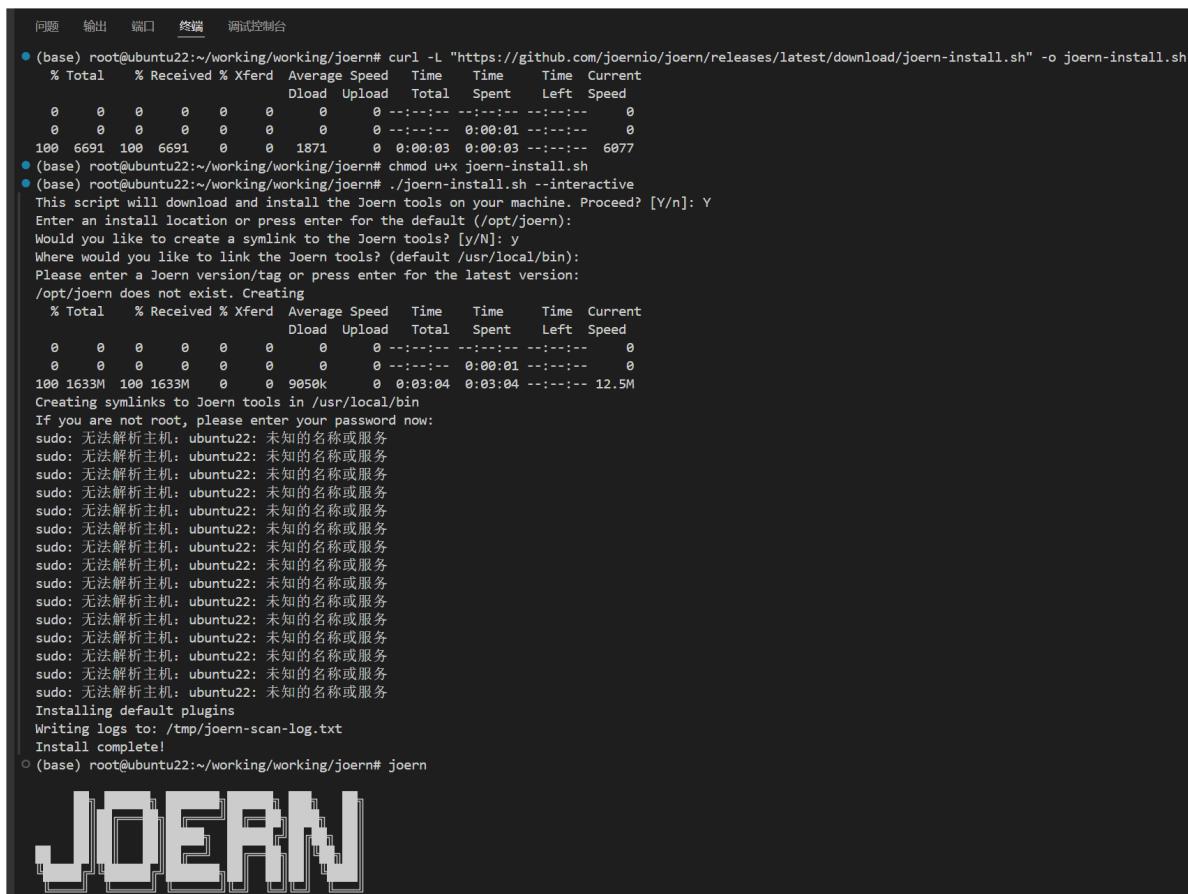


一、joern常用语句

1.Joern的安装

使用joern官网 (<https://docs.joern.io/installation/>) 提供的命令即可安装Joern:

```
mkdir joern && cd joern # optional
curl -L "https://github.com/joernio/joern/releases/latest/download/joern-install.sh" -o joern-install.sh
chmod u+x joern-install.sh
./joern-install.sh --interactive
```



```
问题   输出   端口   终端   调试控制台
● (base) root@ubuntu22:~/working/working/joern# curl -L "https://github.com/joernio/joern/releases/latest/download/joern-install.sh" -o joern-install.sh
% Total    % Received % Xferd  Average Speed   Time   Time  Current
          Dload  Upload   Total Spent  Left  Speed
0     0    0    0    0    0    0 0:00:01 0:00:03 0:00:03 0:00:03 0
0     0    0    0    0    0    0 0:00:01 0:00:03 0:00:03 0:00:03 0
100 6691 100 6691 0    0 1871 0 0:00:03 0:00:03 0:00:03 0:00:03 6077
● (base) root@ubuntu22:~/working/working/joern# chmod u+x joern-install.sh
● (base) root@ubuntu22:~/working/working/joern# ./joern-install.sh --interactive
This script will download and install the Joern tools on your machine. Proceed? [Y/n]: Y
Enter an install location or press enter for the default (/opt/joern):
Would you like to create a symlink to the Joern tools? [y/N]: y
Where would you like to link the Joern tools? (default /usr/local/bin):
Please enter a Joern version/tag or press enter for the latest version:
/opt/joern does not exist. Creating
% Total    % Received % Xferd  Average Speed   Time   Time  Current
          Dload  Upload   Total Spent  Left  Speed
0     0    0    0    0    0    0 0:00:01 0:00:01 0:00:01 0:00:01 0
0     0    0    0    0    0    0 0:00:01 0:00:01 0:00:01 0:00:01 0
100 1633M 100 1633M 0    0 9050k 0 0:03:04 0:03:04 0:03:04 0:03:04 12.5M
Creating symlinks to Joern tools in /usr/local/bin
If you are not root, please enter your password now:
sudo: 无法解析主机: ubuntu22: 未知的名称或服务
Installing default plugins
Writing logs to: /tmp/joern-scan-log.txt
Install complete!
● (base) root@ubuntu22:~/working/working/joern# joern
JOERN
```

注意：想要安装旧版本，可以直接修改.sh文件中的JOERN_DEFAULT_VERSION变量

2.污点分析过程所用的joern语句

(1) 根据参数名称、文件名称过滤得到待分析的第一条语句

```
cpg.call.filter(node => node.code.contains("category")).filter(node =>
node.location.filename=="settings.php").toJsonPretty

cpg.call.filter(node => node.code.contains("return student.name")).toJsonPretty
cpg.call.filter(node => node.code.contains("$info = ")).toJsonPretty
```

```

cpg.call.filter(node =>(node.code.contains("$content") && node.code.contains("eval"))).filter(node =>
node.location.filename=="include/smarty/libs/sysplugins/smarty_cacheresource_custom.php").filter(node =>
node.location.lineNumber==Some(value = 154)).toJsonPretty

86973:
cpg.call.filter(node =>(node.code.contains("PHPWG_ROOT_PATH") &&
node.code.contains("include_once"))).filter(node => node.location.filename=="comments.php").filter(node =>
node.location.lineNumber==Some(value = 13)).toJsonPretty

836133\838777
cpg.call.filter(node =>(node.code.contains("$themeconf") && node.code.contains("="))).filter(node =>
node.location.filename=="themes/default/themeconf.inc.php").filter(node =>
node.location.lineNumber==Some(value = 2)).toJsonPretty

cpg.call.filter(node =>(node.code.contains("$id") && node.code.contains("print"))).filter(node =>
node.location.filename=="managers.php").filter(node => node.location.lineNumber==Some(value =
454)).toJsonPretty

```

cpg.call.filter(node =>node.code.contains("Student s1 = ")).take(1).toJsonPretty

```

joern> cpg.call.filter(node => node.code.contains("category")).filter(node => node.location.filename=="settings.php").isCall.toJsonPretty
val res3: String = """[
{
  "name": "<operator>.assignment",
  "signature": "",
  "code": "$category = empty($_GET[\"category\"])? \"system\": $_GET[\"category\"]",
  "typeFullName": "ANY",
  "order": 42,
  "methodFullName": "<operator>.assignment",
  "_label": "CALL",
  "dynamicTypeHintFullName": [
    "<operator>.assignment"
  ],
  "dispatchType": "STATIC_DISPATCH",
  "lineNumber": 151,
  "id": 1121006,
  "possibleTypes": [
    ...
  ],
  "argumentIndex": -1
},
]
"""

```

(2) 根据node id查询所有后继节点

```
cpg.call.filter(node => node.id==1121006).dominates.isCall.toJsonPretty
```

```

joern> cpg.call.filter(node => node.id==1121006).dominates.isCall.toJsonPretty
val res4: String = """[
{
  "name": "showSystemSettings",
  "signature": "<unresolvedSignature>(1)",
  "code": "$controller->showSystemSettings($category)",
  "typeFullName": "SettingsController->showSystemSettings.<returnValue>",
  "order": 43,
  "methodFullName": "SettingsController->showSystemSettings",
  "_label": "CALL",
  "dynamicTypeHintFullName": [
    ...
  ],
  "dispatchType": "DYNAMIC_DISPATCH",
  "lineNumber": 152,
  "id": 1121017,
  "possibleTypes": [
    ...
  ],
  "argumentIndex": -1
}
]"""

```

(3) 根据函数名称查找函数

```
cpg.method.fullName("SettingsController->showSystemSettings").toJsonPretty  
cpg.method.filter(node => node.fullName == "SettingsController->showSystemSettings").toJsonPretty  
cpg.method.filter(node => node.fullName == "get_nfilter_request_var").toJsonPretty  
cpg.method.filter(node => node.fullName == "htmlentities").astParent.astParent.toJsonPretty  
(这两语句居然有着一样的作用，推荐使用第2条语句)
```

```
joern> cpg.method.fullName("SettingsController->showSystemSettings").toJsonPretty  
val res21: String = """[  
  {  
    "name": "showSystemSettings",  
    "astParentFullName": "SettingsController",  
    "fullName": "SettingsController->showSystemSettings",  
    "signature": "<unresolvedSignature>(1)",  
    "astParentType": "TYPE_DECL",  
    "_label": "METHOD",  
    "code": "function showSystemSettings(this,$category)",  
    "isExternal": false,  
    "lineNumber": 29,  
    "id": 48734,  
    "order": 3,  
    "filename": "controllers/settings.ctrl.php"  
  }  
]""""
```

(4) 查看一个函数的所有后继节点

```
cpg.method.fullName("SettingsController->showSystemSettings").dominates.isCall.sortBy(node =>  
node.lineNumber).toJsonPretty
```

```
joern> cpg.method.fullName("SettingsController->showSystemSettings").dominates.isCall.sortBy(node => node.lineNumber).code.toJsonPretty  
val res73: String = """[  
  "$category = addslashes($category)",  
  "addslashes($category)",  
  "$this->set(\"list\",$this->__getAllSettings(true,1,$category))",  
  "$this->__getAllSettings(true,1,$category)",  
  "$category == \"system\\\"",  
  "$langCtrler = ",  
  "LanguageController->__construct()",  
  "$tmp0 = LanguageController.<alloc>()",  
  "LanguageController.<alloc>()",  
  "$langList = $langCtrler->__getAllLanguages(\" where translated=1\")",  
  "$langCtrler->__getAllLanguages(\" where translated=1\")",  
  "$this->set(\"langList\",$langList)",  
]""""
```

注意：为了进入函数内部查看函数内部的逻辑，这里不应该使用1121006节点的后继节点1121017来查找其后继节点，而是应该用这个函数来查其后继节点，如果查节点1121017的后继节点，结果并不会进入到该函数内部：

```
joern> cpg.call.filter(node => node.id==1121017).dominates.toJsonPretty  
val res75: String = """[  
  {  
    "parserTypeName": "PhpBreakStmt",  
    "code": "break",  
    "order": 44,  
    "controlStructureType": "BREAK",  
    "_label": "CONTROL_STRUCTURE",  
    "argumentIndex": -1,  
    "lineNumber": 153,  
    "id": 1121020  
  }  
]""""
```

3. 获取赋值语句左边的变量

属性：

```
.target: Left-hand sides of assignments
```

源码：

```
$category = empty($_GET['category']) ? 'system' : $_GET['category'];
```

示例：

```
cpg.call.filter(node => node.id==1121006).assignment.target.toJsonPretty
```

```
joern> cpg.call.filter(node => node.id==1121006).assignment.target.l
val res75: List[io.shiftleft.codepropertygraph.generated.nodes.Expression] = List(
  Identifier(
    id = 1121007L,
    argumentIndex = 1,
    argumentName = None,
    code = "$category",
    columnNumber = None,
    dynamicTypeHintFullName = ArraySeq(),
    lineNumber = Some(value = 151),
    name = "category",
    order = 1,
    possibleTypes = ArraySeq(),
    typeFullName = "ANY"
  )
)
```

注意：左值、右值都可能有多个

```
joern> cpg.call.filter(node => node.code.contains("$value = $_GET[$name] = $_REQUEST[$name]").assignment.target.code.l
val res83: List[String] = List("$value", "$_GET[$name]", "$_REQUEST[$name]")
joern> cpg.call.filter(node => node.code.contains("$value = $_GET[$name] = $_REQUEST[$name]").assignment.source.code.l
val res84: List[String] = List(
  "$_GET[$name] = $_REQUEST[$name] = preg_replace($pattern,\\"\\",$value)",
  "$_REQUEST[$name] = preg_replace($pattern,\\"\\",$value)",
  "preg_replace($pattern,\\"\\",$value)"
)
```

4. 获取赋值语句右边的变量

属性：

```
.source: Right-hand sides of assignments
```

源码：

```
$category = empty($_GET['category']) ? 'system' : $_GET['category'];
```

示例：

```
cpg.call.filter(node => node.id==1121006).assignment.source.toJsonPretty
```

```

joern> cpg.call.filter(node => node.id==1121006).assignment.source.1
val res76: List[io.shiftleft.codepropertygraph.generated.nodes.Expression] = List(
  Call(
    id = 1121008L,
    argumentIndex = 2,
    argumentName = None,
    code = "empty($_GET[\"category\"]) ? \"system\" : $_GET[\"category\"]",
    columnNumber = None,
    dispatchType = "STATIC_DISPATCH",
    dynamicTypeHintFullName = ArraySeq(),
    lineNumber = Some(value = 151),
    methodFullName = "<operator>.conditional",
    name = "<operator>.conditional",
    order = 2,
    possibleTypes = ArraySeq(),
    signature = "",
    typeFullName = "ANY"
  )
)

```

5.按照某一属性对结果进行排序

属性:

```
.sortBy: Sorts the query results by the specified fields
```

源码:

```

# function to show system settings
function showSystemSettings($category='system') {
  $category = addslashes($category);
  $this->set('list', $this->_getAllSettings(true, 1, $category));
  if ($category == 'system') {
    $langCtrler = New LanguageController();
    $langList = $langCtrler->_getAllLanguages(" where translated=1");
    $this->set('langList', $langList);
    $timezoneCtrler = New TimeZoneController();
    $timezoneList = $timezoneCtrler->_getAllTimezones();
    $this->set('timezoneList', $timezoneList);
    $currencyCtrler = new CurrencyController();
    $this->set('currencyList', $currencyCtrler->_getAllCurrency(" and paypal=1 and status=1
and name!= ''"));
    $countryCtrl = new CountryController();
    $this->set('countryList', $countryCtrl->_getAllCountryAsList());
  }
  $this->set('category', $category);
  // if report settings page
  if ($category == 'report') {
    $spTextReport = $this->getLanguageTexts('report', $_SESSION['lang_code']);
    $this->set('spTextReport', $spTextReport);
    $scheduleList = array(
      1 => $_SESSION['text']['label']['Daily'],
      2 => $spTextReport['2 Days'],
      7 => $_SESSION['text']['label']['Weekly'],
      30 => $_SESSION['text']['label']['Monthly'],
    );
    $this->set('scheduleList', $scheduleList);
    $this->render('settings/showreportsettings');
  } else if ($category == 'proxy') {
    $spTextProxy = $this->getLanguageTexts('proxy', $_SESSION['lang_code']);
  }
}

```

```

        $this->set('spTextProxy', $spTextProxy);
        $this->render('settings/showproxysettings');
    } else {
        $spTextPanel = $this->getLanguageTexts('panel', $_SESSION['lang_code']);
        // switch through category
        switch ($category) {
            case "api":
                $this->set('headLabel', $spTextPanel['API Settings']);
                break;
            case "moz":
                $this->set('headLabel', $spTextPanel['MOZ Settings']);
                break;
            case "google":
                $this->set('headLabel', $spTextPanel['Google Settings']);
                break;
            case "dataforseo":
                $this->set('headLabel', $spTextPanel['DataForSEO Settings']);
                break;
            case "mail":
                $this->set('headLabel', $spTextPanel['Mail Settings']);
                break;
            default:
                break;
        }
        $this->render('settings/showsettings');
    }
}

```

示例:

```
cpg.method.name("showSystemSettings").where(_.isExternal(false)).dominates.isCall.sortBy(node =>
node.lineNumber).code.1
```

```

joern> cpg.method.name("showSystemSettings").where(_.isExternal(false)).dominates.isCall.sortBy(node => node.lineNumber).code.1
val res113: List[String] = List(
"$category = addslashes($category)",
"addslashes($category)",
"$this->set(\"list\",$this->_getAllSettings(true,1,$category))",
"$this->_getAllSettings(true,1,$category)",
"$category == \"system\"",
"$langCtrler = ",
"LanguageController->__construct()", 
"$tmp0 = LanguageController.<alloc>()", 
"LanguageController.<alloc>()", 
"$langList = $langCtrler->_getAllLanguages(\" where translated=1\")",
"$langCtrler->_getAllLanguages(\" where translated=1\")",
"$this->set(\"langList\",$langList)",
"$timezoneCtrler = ",
"TimeZoneController->__construct()", 
"$tmp1 = TimeZoneController.<alloc>()", 
"TimeZoneController.<alloc>()", 
"$timezoneList = $timezoneCtrler->_getAllTimezones()", 
"$timezoneCtrler->_getAllTimezones()", 
"$this->set(\"timezoneList\",$timezoneList)",
"$currencyCtrler = ",
"CurrencyController->__construct()", 
"$tmp2 = CurrencyController.<alloc>()", 
"CurrencyController.<alloc>()", 
"""$this->set(\"currencyList\",$currencyCtrler->_getAllCurrency(\" and paypal=1 and status=1 and name!=\'\')\""",
"""$currencyCtrler->_getAllCurrency(\" and paypal=1 and status=1 and name!=\'\')\""",
"$countryCtrl = ",
"CountryController->__construct()", 
"$tmp3 = CountryController.<alloc>()", 
"CountryController.<alloc>()", 
"$this->set(\"countryList\",$countryCtrl->_getAllCountryAsList())",
"$countryCtrl->_getAllCountryAsList()", 
"$this->set(\"category\",$category)",
"$category == \"report\"",
```

6.查询函数的参数

源码

```
function showSystemSettings($category='system') {
```

示例

```
cpg.method.fullName("SettingsController->showSystemSettings").parameter.toJsonPretty
```

```
joern> cpg.method.fullName("SettingsController->showSystemSettings").parameter.toJsonPretty
val res48: String = """[
  {
    "dynamicTypeHintFullName": [
      "SettingsController"
    ],
    "name": "this",
    "evaluationStrategy": "BY_SHARING",
    "isVariadic": false,
    "typeFullName": "SettingsController",
    "order": 0,
    "_label": "METHOD_PARAMETER_IN",
    "possibleTypes": [
      ],
      "index": 0,
      "code": "this",
      "lineNumber": 29,
      "id": 48735
    },
    {
      "dynamicTypeHintFullName": [
        ],
        "name": "category",
        "evaluationStrategy": "BY_VALUE",
        "isVariadic": false,
        "typeFullName": "ANY",
        "order": 1,
        "_label": "METHOD_PARAMETER_IN",
        "possibleTypes": [
          ],
          "index": 1,
          "code": "$category",
          "lineNumber": 29,
          "id": 48736
        }
  ]"""
```

7.查询函数被调用时传入的实参

源码：

```
$name=friendly($_POST['name']);
```

示例：

```
cpg.call.filter(node => node.id==9737).assignment.source.toJsonPretty
cpg.call.filter(node => node.id==9739).argument.toJsonPretty
```

9737对应的是" \$name=friendly(\$_POST['name']); ",9739对应的是" friendly(\$_POST['name']) "。

```

joern> cpg.call.filter(node => node.id==9737).assignment.source.toJsonPretty
val res12: String = """[
  {
    "name": "friendly",
    "signature": "<unresolvedSignature>(1)",
    "code": "friendly($_POST[\"name\"])",
    "typeFullName": "ANY",
    "order": 2,
    "methodFullName": "friendly",
    "_label": "CALL",
    "dynamicTypeHintFullName": [
      "friendly"
    ],
    "dispatchType": "STATIC_DISPATCH",
    "lineNumber": 23,
    "id": 9739,
    "possibleTypes": [
      ],
      "argumentIndex": 2
    }
]"""

```

```

joern> cpg.call.filter(node => node.id==9739).argument.toJsonPretty
val res13: String = """[
  {
    "name": "<operator>.indexAccess",
    "signature": "",
    "code": "$_POST[\"name\"]",
    "typeFullName": "ANY",
    "order": 1,
    "methodFullName": "<operator>.indexAccess",
    "_label": "CALL",
    "dynamicTypeHintFullName": [
      ],
      "dispatchType": "STATIC_DISPATCH",
      "lineNumber": 23,
      "id": 9740,
      "possibleTypes": [
        ],
        "argumentIndex": 1
      }
]"""

```

注意：有些时候查出来的参数可能是一个BLOCK!

源码

```
$template->set_filenames(array('plugins' => 'plugins_installed.tpl'));
```

```

joern> cpg.call.filter(node => node.id==73881).toJsonPretty
val res2: String = """[
  {
    "name": "set_filenames",
    "signature": "<unresolvedSignature>(1)",
    "code": "$template->set_filenames()", 
    "typeFullName": "ANY",
    "order": 47,
    "methodFullName": "<unresolvedNamespace>\$template->set_filenames",
    "_label": "CALL",
    "dynamicTypeHintFullName": [
      ],
      "dispatchType": "DYNAMIC_DISPATCH",
      "lineNumber": 16,
      "id": 73881,
      "possibleTypes": [
        ],
        "argumentIndex": -1
      }
]"""

```

此时可以看到，在Joern解析得到的代码中，这一行的函数调用参数少了参数：array('plugins' => 'plugins_installed.tpl')，我们尝试查询这里传入的参数，结果发现第一个参数变为了BLOCK：

```
cpg.call.filter(node => node.id==73881).argument.toJsonPretty
```

```
joern> cpg.call.filter(node => node.id==73881).argument.toJsonPretty
val res1: String = """[
  {
    "dynamicTypeHintFullName": [
      ],
      "name": "template",
      "code": "$template",
      "typeFullName": "ANY",
      "order": 1,
      "_label": "IDENTIFIER",
      "lineNumber": 16,
      "id": 73882,
      "possibleTypes": [
        ],
        "argumentIndex": 0
    },
    {
      "dynamicTypeHintFullName": [
        ],
        "code": "<empty>",
        "typeFullName": "ANY",
        "order": 2,
        "_label": "BLOCK",
        "lineNumber": 16,
        "id": 73883,
        "possibleTypes": [
          ],
          "argumentIndex": 1
    }
]""""
```

我们找一下这个参数具体的位置：

```
joern> cpg.call.filter(node => node.code.contains("plugins_installed.tpl")).filter(node => node.location.filename=="admin/plugins_installed.php").toJsonPretty
val res11: String = """[
  {
    "name": "<operator>.assignment",
    "signature": "",
    "code": "$tmp0[\"plugins\"] = \"plugins_installed.tpl\\\"",
    "typeFullName": "ANY",
    "order": 1,
    "methodFullName": "<operator>.assignment",
    "_label": "CALL",
    "dynamicTypeHintFullName": [
      ],
      "dispatchType": "STATIC_DISPATCH",
      "lineNumber": 16,
      "id": 73884,
      "possibleTypes": [
        ],
        "argumentIndex": -1
    }
]"""
joern> cpg.call.filter(node => (node.code.contains("plugins_installed.tpl") && node.code.contains("array"))).filter(node => node.location.filename=="admin/plugins_installed.php").toJsonPretty
val res14: String = """[
  ]"""
```

可以看到，包含了array的参数已经不在了，猜测这里出现错误的原因在于Joern将array变量建模为了BLOCK。

8. 查询实参被哪些call site调用（部分到整体）

源码：

```
$name=friendly($_POST['name']);
```

现在想要看 \$_POST['name'] 被哪些call site调用过，\$_POST['name'] 对应的node id为9740：

```
joern> cpg.call.filter(node => node.id==9740).toJsonPretty
val res25: String = """[
  {
    "name": "<operator>.indexAccess",
    "signature": "",
    "code": "$_POST[\"name\"]",
    "typeFullName": "ANY",
    "order": 1,
    "methodFullName": "<operator>.indexAccess",
    "_label": "CALL",
    "dynamicTypeHintFullName": [
      ],
    "dispatchType": "STATIC_DISPATCH",
    "lineNumber": 23,
    "id": 9740,
    "possibleTypes": [
      ],
    "argumentIndex": 1
  }
]"""
```

逐次往上分析：

(1) 找到 `friendly($_POST['name'])`

```
cpg.call.filter(node => node.id==9740).astParent.toJsonPretty
```

```
joern> cpg.call.filter(node => node.id==9740).astParent.toJsonPretty
val res26: String = """[
  {
    "name": "friendly",
    "signature": "<unresolvedSignature>(1)",
    "code": "friendly($_POST[\"name\"])",
    "typeFullName": "ANY",
    "order": 2,
    "methodFullName": "friendly",
    "_label": "CALL",
    "dynamicTypeHintFullName": [
      "friendly"
    ],
    "dispatchType": "STATIC_DISPATCH",
    "lineNumber": 23,
    "id": 9739,
    "possibleTypes": [
      ],
    "argumentIndex": 2
  }
]"""
```

(2) 找到 `$name=friendly($_POST['name'])`

```
cpg.call.filter(node => node.id==9739).astParent.toJsonPretty
```

```
joern> cpg.call.filter(node => node.id==9739).astParent.toJsonPretty
val res27: String = """[
  {
    "name": "<operator>.assignment",
    "signature": "",
    "code": "$name = friendly($_POST[\"name\"])",
    "typeFullName": "ANY",
    "order": 4,
    "methodFullName": "<operator>.assignment",
    "_label": "CALL",
    "dynamicTypeHintFullName": [
      ],
      "dispatchType": "STATIC_DISPATCH",
      "lineNumber": 23,
      "id": 9737,
      "possibleTypes": [
        ],
        "argumentIndex": -1
    }
]""""
```

(3) 找到 `<empty>`，此时认为已经不能找到更上一级的调用点了。

```
cpg.call.filter(node => node.id==9737).astParent.toJsonPretty
```

```
joern> cpg.call.filter(node => node.id==9737).astParent.toJsonPretty
val res28: String = """[
  {
    "dynamicTypeHintFullName": [
      ],
      "code": "<empty>",
      "typeFullName": "ANY",
      "order": 2,
      "_label": "BLOCK",
      "lineNumber": 18,
      "id": 9721,
      "possibleTypes": [
        ],
        "argumentIndex": -1
    }
]""""
```

9.查询一条语句的子语句（整体到部分）

源码：

```
$result=register_patient($gender,$age,$serial,$name,$contact,$email,$weight,$profession,$ref_contact,$address);
```

现在想要看这一行代码可以被拆分为哪些子语句：

```

joern> cpg.call.filter(node => node.id==9779).toJsonPretty
val res48: String = """[
{
  "name": "<operator>.assignment",
  "signature": "",
  "code": "$result = register_patient($gender,$age,$serial,$name,$contact,$email,$weight,$profession,$ref_contact,$address)",
  "typeFullName": "ANY",
  "order": 1,
  "methodFullName": "<operator>.assignment",
  "_label": "CALL",
  "dynamicTypeHintFullName": [
    ],
    "dispatchType": "STATIC_DISPATCH",
    "lineNumber": 31,
    "id": 9779,
    "possibleTypes": [
      ],
      "argumentIndex": -1
  }
]"""

```

逐次往下分析：

(1) 找到 \$result 和

```

register_patient($gender, $age, $serial, $name, $contact, $email, $weight, $profession, $ref_contact, $address)

```

```
cpg.call.filter(node => node.id==9779).astChildren.toJsonPretty
```

```

joern> cpg.call.filter(node => node.id==9779).astChildren.toJsonPretty
val res49: String = """[
{
  "dynamicTypeHintFullName": [
    ],
    "name": "result",
    "code": "$result",
    "typeFullName": "bool",
    "order": 1,
    "_label": "IDENTIFIER",
    "lineNumber": 31,
    "id": 9780,
    "possibleTypes": [
      ],
      "argumentIndex": 1
  },
  {
    "name": "register_patient",
    "signature": "<unresolvedSignature>(10)",
    "code": "register_patient($gender,$age,$serial,$name,$contact,$email,$weight,$profession,$ref_contact,$address)",
    "typeFullName": "ANY",
    "order": 2,
    "methodFullName": "register_patient",
    "_label": "CALL",
    "dynamicTypeHintFullName": [
      "register_patient"
    ],
    "dispatchType": "STATIC_DISPATCH",
    "lineNumber": 31,
    "id": 9781,
    "possibleTypes": [
      ],
      "argumentIndex": 2
  }
]"""

```

(2) 找到 register_patient 函数的实参

```
cpg.call.filter(node => node.id==9781).astChildren.toJsonPretty
```

```
joern> cpg.call.filter(node => node.id==9781).astChildren.toJsonPretty
val res50: String = """[
  {
    "dynamicTypeHintFullName": [
      ],
      "name": "gender",
      "code": "$gender",
      "typeFullName": "ANY",
      "order": 1,
      "_label": "IDENTIFIER",
      "lineNumber": 31,
      "id": 9782,
      "possibleTypes": [
        ],
        "argumentIndex": 1
  },
  {
    "dynamicTypeHintFullName": [
      ],
      "name": "age",
      "code": "$age",
      "typeFullName": "ANY",
      "order": 2,
      "_label": "IDENTIFIER",
      "lineNumber": 31,
      "id": 9783,
      "possibleTypes": [
        ],
        "argumentIndex": 2
  }
]"""
```

(3) 找到 `<empty>`，此时认为已经不能找到更上一级的调用点了。

```
cpg.call.filter(node => node.id==9782).astChildren.toJsonPretty
```

```
joern> cpg.call.filter(node => node.id==9782).astChildren.toJsonPretty
val res51: String = """"[

]""""
```

10.argument 和 parameter 的区别

argument表示实际传给某一个函数的实参；而parameter表示定义函数时的形参。

argument一般在cpg.call语句中使用；而parameter一般在cpg.method语句中使用。

11.查询指定字段

注意：指定字段前，要确保你查出来的结果中含有这些字段，否则会报错。

```
cpg.call.map( x=> (x.node.id, x.node.code)).toJsonPretty
cpg.call.filter(node => node.id==9737).dominates.isCall.map( x=> (x.node.id, x.node.code)).toJsonPretty
cpg.method.filter(node => node.fullName.contains("printf")).map( x=> (x.fullName)).toJsonPretty
```

```
joern> cpg.call.filter(node => node.id==9737).dominates.isCall.map( x=> (x.node.id, x.node.code)).toJsonPretty
val res30: String = """[
  {
    "_1":9797,
    "_2":"echo <div class='alert alert-danger' role='alert'>Please fill out all required fields!</div>"
  },
  {
    "_1":9824,
    "_2":"print(</script>)"
  },
  {
    "_1":9819,
    "_2":"print(` self.location=\`register-report.php?id=\` . $result . \`&success\``;`)"
  },
  {
    "_1":9820,
    "_2":"` self.location=\`register-report.php?id=\` . $result . \`&success\``;`"
  },
  {
    "_1":9817,
    "_2":"print(<script>)"
  },
]
```

12.joern污点分析的使用

(1) 案例一

源码

```
import os

def source(x):
    temp_x = x
    temp_y = temp_x
    temp_z = temp_y
    os.system(temp_z)
```

定义source、sink

```
def source = cpg.method.name("source").parameter
def sink = cpg.call.name("system").argument
```

```
sink.reachableBy(source).l
```

```

joern> sink.reachableBy(source).l
val res122: List[io.shiftleft.codepropertygraph.generated.nodes.MethodParameterIn] = List(
  MethodParameterIn(
    id = 23L,
    closureBindingId = None,
    code = "x",
    columnNumber = Some(value = 12),
    dynamicTypeHintFullName = ArraySeq(),
    evaluationStrategy = "BY_SHARING",
    index = 1,
    isVariadic = false,
    lineNumber = Some(value = 3),
    name = "x",
    order = 1,
    possibleTypes = ArraySeq(),
    typeFullName = "ANY"
  ),
  MethodParameterIn(
    id = 23L,
    closureBindingId = None,
    code = "x",
    columnNumber = Some(value = 12),
    dynamicTypeHintFullName = ArraySeq(),
    evaluationStrategy = "BY_SHARING",
    index = 1,
    isVariadic = false,
    lineNumber = Some(value = 3),
    name = "x",
    order = 1,
    possibleTypes = ArraySeq(),
    typeFullName = "ANY"
  )
)

```

sink.reachableByFlows(source).p

```

joern> sink.reachableByFlows(source).p
val res130: List[String] = List(
  """
    |

| nodeType          | tracked   | line | method | file |
|-------------------|-----------|------|--------|------|
| MethodParameterIn | source(x) | 3    | source |      |


    |"""
  ,
  """
    |

| nodeType          | tracked   | line | method | file |
|-------------------|-----------|------|--------|------|
| MethodParameterIn | source(x) | 3    | source |      |


    |
  )

```

(2) 案例二

源码

```
import os

def source(x):
    temp_x = input()
    temp_y = temp_x
    temp_z = temp_y
    os.system(temp_z)
```

定义source、sink

```
def source = cpg.call.name("input")
def sink = cpg.call.name("system").argument
```

```
sink.reachableBy(source).l
```

```
joern> sink.reachableBy(source).l
val res135: List[io.shiftleft.codepropertygraph.generated.nodes.Call] = List(
  Call(
    id = 29L,
    argumentIndex = 2,
    argumentName = None,
    code = "input()",  
columnNumber = Some(value = 14),
    dispatchType = "DYNAMIC_DISPATCH",
    dynamicTypeHintFullName = ArraySeq(),
    lineNumber = Some(value = 4),
    methodFullName = "__builtin.input",
    name = "input",
    order = 2,
    possibleTypes = ArraySeq(),
    signature = "",
    typeFullName = "ANY"
  ),
  Call(
    id = 29L,
    argumentIndex = 2,
    argumentName = None,
    code = "input()",  
columnNumber = Some(value = 14),
    dispatchType = "DYNAMIC_DISPATCH",
    dynamicTypeHintFullName = ArraySeq(),
    lineNumber = Some(value = 4),
    methodFullName = "__builtin.input",
    name = "input",
    order = 2,
    possibleTypes = ArraySeq(),
    signature = "",
    typeFullName = "ANY"
  )
)
```

```
sink.reachableByFlows(source).p
```

```

joern> sink.reachableByFlows(source).p
val res134: List[String] = List(
    """
    |

| nodeType | tracked | line | method | file |
|----------|---------|------|--------|------|
| Call     | input() | 4    | source |      |


    |"""
    |,
    """
    |

| nodeType | tracked | line | method | file |
|----------|---------|------|--------|------|
| Call     | input() | 4    | source |      |


    |"""
)

```

注意：这里两处都是call，如果定义source时使用了method，结果如下：

```

joern> def source = cpg.method.name("input")
def source: Iterator[io.shiftleft.codepropertygraph.generated.nodes.Method]

joern> def sink = cpg.call.name("system").argument
def sink: Iterator[io.shiftleft.codepropertygraph.generated.nodes.Expression]

joern> sink.reachableBy(source).l
val res136: List[io.shiftleft.codepropertygraph.generated.nodes.Method] = List()

joern> sink.reachableByFlows(source).p
val res137: List[String] = List()

```

(3) 案例三

源码

```

import os

_GET = []

def source(x):
    temp_x = _GET["category"]
    temp_y = temp_x
    temp_z = temp_y
    os.system(temp_z)

```

定义source、sink

```

def source = cpg.call.name("<operator>.indexAccess").filter(node => node.code.contains("_GET"))
def sink = cpg.call.name("system").argument

```

```

sink.reachableBy(source).l

```

```

joern> sink.reachableBy(source).l
val res150: List[io.shiftleft.codepropertygraph.generated.nodes.Call] = List(
  Call(
    id = 30L,
    argumentIndex = 2,
    argumentName = None,
    code = "_GET[\"category\"]",
    columnNumber = Some(value = 14),
    dispatchType = "STATIC_DISPATCH",
    dynamicTypeHintFullName = ArraySeq(),
    lineNumber = Some(value = 6),
    methodFullName = "<operator>.indexAccess",
    name = "<operator>.indexAccess",
    order = 2,
    possibleTypes = ArraySeq(),
    signature = "",
    typeFullName = "ANY"
  ),
  Call(
    id = 30L,
    argumentIndex = 2,
    argumentName = None,
    code = "_GET[\"category\"]",
    columnNumber = Some(value = 14),
    dispatchType = "STATIC_DISPATCH",
    dynamicTypeHintFullName = ArraySeq(),
    lineNumber = Some(value = 6),
    methodFullName = "<operator>.indexAccess",
    name = "<operator>.indexAccess",
    order = 2,
    possibleTypes = ArraySeq(),
    signature = "",
    typeFullName = "ANY"
  )
)

```

sink.reachableByFlows(source).p

nodeType	tracked	line	method	file
Call	temp_x = _GET["category"]	6	source	
Identifier	temp_x = _GET["category"]	6	source	
Identifier	temp_y = temp_x	7	source	
Identifier	temp_y = temp_x	7	source	
Identifier	temp_z = temp_y	8	source	
Identifier	temp_z = temp_y	8	source	
Identifier	os.system(temp_z)	9	source	
....				
....				
nodeType	tracked	line	method	file
Call	temp_x = _GET["category"]	6	source	
Identifier	temp_x = _GET["category"]	6	source	
Identifier	temp_y = temp_x	7	source	
Identifier	temp_y = temp_x	7	source	
Identifier	temp_z = temp_y	8	source	
Identifier	temp_z = temp_y	8	source	
Identifier	os.system(temp_z)	9	source	
Identifier	os.system(temp_z)	9	source	
....				

(4) 案例四

源码

```

import os

def source():
    x = input()

```

```

        return x

def sink(text):
    os.system(text)

def main(x):
    temp_x = source()
    temp_y = temp_x
    temp_z = temp_y
    sink(temp_z)

```

示例：

```

joern> def source = cpg.call.name("source")
def source: Iterator[io.shiftleft.codepropertygraph.generated.nodes.Call]

joern> def sink = cpg.call.name("sink").argument
def sink: Iterator[io.shiftleft.codepropertygraph.generated.nodes.Expression]

joern> sink.reachableByFlows(source).p
val res25: List[String] = List(
    """
    
```

nodeType	tracked	line	method	file
Call	source()	11	main	
Identifier	temp_x = source()	11	main	
Identifier	temp_y = temp_x	12	main	
Identifier	temp_y = temp_x	12	main	
Identifier	temp_z = temp_y	13	main	
Identifier	temp_z = temp_y	13	main	
Identifier	sink(temp_z)	14	main	
			

)

(5) 案例五

源码：

/home/devdata/repos/seopanel_seo-panel/settings.php

```

settings.php X showsettings.ctp.php
repos > seopanel_seo-panel > settings.php > ...
147
148     break;
149
150     default:
151         $category = empty($_GET['category']) ? 'system' : $_GET['category'];
152         $controller->showSystemSettings($category);      sendtogeoo, 13年前 * seo panel version 3.0.0
153         break;
154     }
155 }
156 ?>

```

/home/devdata/repos/seopanel_seo-panel/themes/classic/views/settings/showsettings.ctp.php

[

```
The file path at the top is 'repos > seopanel\\_seo-panel > themes > classic > views > settings > showsettings.ctp.php'."/>

```

settings.php X showsettings.ctp.php X
repos > seopanel_seo-panel > themes > classic > views > settings > showsettings.ctp.php > form#updateSettings > table.actionSec > td > a.actionbut
47 <form id="updateSettings">
48 <table class="list">
49 <tr>
50 <td>
51 <table class="actionSec">
52 <tr>
53 <td style="padding-top: 6px;text-align:right;">
54 <a onclick="scriptDoLoad\('settings.php?category=<?php echo \$category?>', 'content', 'layout=ajax'\) href='javascript:void\(0\);' class='actionbut'><?php echo \$spText\['button'\]\['Cancel'\]?>
55 <?php \$actFun = SP_DEMO ? "alertDemoMsg\(\)" : "confirmSubmit\('settings.php', 'updateSettings', 'content'\)"; ?>
56 <a onclick=<?php echo \$actFun?> href='javascript:void\(0\);' class='actionbut'><?php echo \$spText\['button'\]\['Proceed'\]?>
57 </td>
58 </tr>
59 </table>
60 </td>
61 </tr>
62 </table>
63 </form>

```](javascript:void(0);)

```

def source = cpg.call.name("<operator>.indexAccess").filter(node =>
node.code.contains("_GET[\"category\"]"))
def sink = cpg.call.name("echo").argument
sink.reachableBy(source).l

```

在这个案例中没有结果：

```

joern> def source = cpg.call.name("<operator>.indexAccess").filter(node => node.code.contains("_GET[\"category\"]"))
def source: Iterator[io.shiftleft.codepropertygraph.generated.nodes.Call]

joern> def sink = cpg.call.name("echo").argument
def sink: Iterator[io.shiftleft.codepropertygraph.generated.nodes.Expression]

joern> sink.reachableBy(source).l
val res3: List[io.shiftleft.codepropertygraph.generated.nodes.Call] = List()

```

## 13.查询一个类的父类

---

源码：

```

<?php
class SettingsController extends Controller{
 var $layout = 'ajax';

 # function to show system settings
 function showSystemSettings($category='system') {
 $category = addslashes($category);
 $this->set('list', $this->_getAllSettings(true, 1, $category));
 if ($category == 'system') {
 $langCtrlr = New LanguageController();
 $langList = $langCtrlr->_getAllLanguages(" where translated=1");
 $this->set('langList', $langList);
 $timezoneCtrlr = New TimeZoneController();
 $timezoneList = $timezoneCtrlr->_getAllTimezones();
 $this->set('timezoneList', $timezoneList);
 $currencyCtrlr = new CurrencyController();
 $this->set('currencyList', $currencyCtrlr->_getAllCurrency(" and paypal=1 and status=1
and name!= ''"));
 $countryCtrl = new CountryController();
 $this->set('countryList', $countryCtrl->_getAllCountryAsList());
 }
 $this->set('category', $category);

 // if report settings page
 if ($category == 'report') {
 $spTextReport = $this->getLanguageTexts('report', $_SESSION['lang_code']);
 $this->set('spTextReport', $spTextReport);
 $scheduleList = array(
 1 => $_SESSION['text']['label']['Daily'],
 2 => $spTextReport['2 Days'],
 7 => $_SESSION['text']['label']['Weekly'],
 30 => $_SESSION['text']['label']['Monthly'],
);
 $this->set('scheduleList', $scheduleList);
 $this->render('settings/showreportsettings');
 } else if ($category == 'proxy') {
 $spTextProxy = $this->getLanguageTexts('proxy', $_SESSION['lang_code']);
 }
 }
}

```

```

 $this->set('spTextProxy', $spTextProxy);
 $this->render('settings/showproxysettings');
 } else {
 $spTextPanel = $this->getLanguageTexts('panel', $_SESSION['lang_code']);

 // switch through category
 switch ($category) {
 case "api":
 $this->set('headLabel', $spTextPanel['API Settings']);
 break;
 case "moz":
 $this->set('headLabel', $spTextPanel['MOZ Settings']);
 break;
 case "google":
 $this->set('headLabel', $spTextPanel['Google Settings']);
 break;
 case "dataforseo":
 $this->set('headLabel', $spTextPanel['DataForSEO Settings']);
 break;
 case "mail":
 $this->set('headLabel', $spTextPanel['Mail Settings']);
 break;
 default:
 break;
 }
 }
 $this->render('settings/showsettings');
}
}
?>

```

## 示例

```

cpg.typeDecl.name("SettingsController").inheritsFromTypeFullName.toJsonPretty
cpg.typeDecl.filter(node => node.name=="SettingsController").inheritsFromTypeFullName.toJsonPretty

```

```

joern> cpg.typeDecl.name("SettingsController").toJsonPretty
val res16: String = """[
 {
 "inheritsFromTypeFullName": [
 "Controller"
],
 "name": "SettingsController",
 "astParentFullName": "controllers/settings.ctrl.php:<global>",
 "fullName": "SettingsController",
 "astParentType": "METHOD",
 "_label": "TYPE_DECL",
 "code": "class SettingsController extends Controller",
 "isExternal": false,
 "lineNumber": 24,
 "id": 48720,
 "order": 1,
 "filename": "controllers/settings.ctrl.php"
 }
]"""

joern> cpg.typeDecl.name("SettingsController").inheritsFromTypeFullName.toJsonPretty
val res17: String = """[
 "Controller"
]"""

```

还可以接着查父类的父类：

```

joern> cpg.typeDecl.name("Controller").toJsonPretty
val res19: String = """[
 {
 "inheritsFromTypeFullName": [
 "Seopanel"
],
 "name": "Controller",
 "astParentFullName": "libs/controller.class.php:<global>",
 "fullName": "Controller",
 "astParentType": "METHOD",
 "_label": "TYPE_DECL",
 "code": "class Controller extends Seopanel",
 "isExternal": false,
 "lineNumber": 24,
 "id": 89690,
 "order": 1,
 "filename": "libs/controller.class.php"
 }
]"""

joern> cpg.typeDecl.name("Seopanel").toJsonPretty
val res20: String = """[
 {
 "inheritsFromTypeFullName": [
],
 "name": "Seopanel",
 "astParentFullName": "libs/seopanel.class.php:<global>",
 "fullName": "Seopanel",
 "astParentType": "METHOD",
 "_label": "TYPE_DECL",
 "code": "class Seopanel",
 "isExternal": false,
 "lineNumber": 24,
 "id": 1110182,
 "order": 1,
 "filename": "libs/seopanel.class.php"
 }
]"""

```

## 14. 查找CFG子节点

查找CFG子节点的函数不是cfgNext，而是\_cfgOut函数。

源码：

```

def main():
 a = 5
 b = 20
 x = 10
 if x < 10:
 print("x < 10")
 else:
 print("x >= 10")
 c = 30
 d = a + b + c

```

示例：

查找 `a = 5` 的CFG子节点

```
joern> cpg.call.filter(node => node.id==35).toJsonPretty
val res112: String = """[
 {
 "name": "<operator>.assignment",
 "signature": "",
 "code": "a = 5",
 "typeFullName": "ANY",
 "columnNumber": 5,
 "order": 1,
 "methodFullName": "<operator>.assignment",
 "_label": "CALL",
 "dynamicTypeHintFullName": [
],
 "dispatchType": "STATIC_DISPATCH",
 "lineNumber": 4,
 "id": 35,
 "possibleTypes": [
],
 "argumentIndex": -1
 }
]""""
```

```
cpg.all.filter(node => node.id==35).cfgOut.toJsonPretty
```

```
joern> cpg.all.filter(node => node.id==35).cfgOut.toJsonPretty
val res113: String = """[
 {
 "dynamicTypeHintFullName": [
],
 "name": "b",
 "code": "b",
 "typeFullName": "__builtin.int",
 "columnNumber": 5,
 "order": 1,
 "_label": "IDENTIFIER",
 "lineNumber": 5,
 "id": 37,
 "possibleTypes": [
],
 "argumentIndex": 1
 }
]""""
```

**注意：**这里cpg节点类型要使用 `all` 类型，这样做是为了找节点子节点过程中，不会受到节点本身类型的影响。

## 15.查找创建对象语句

源码：

```
$langCtrlr = New LanguageController();
$langList = $langCtrlr->_getAllLanguages(" where translated=1");
```

示例：

```
cpg.call.filter(node => node.id==48764).assignment.source.toJsonPretty
```

```

joern> cpg.call.filter(node => node.id==48764).assignment.source.toJsonPretty
val res45: String = """[
 {
 "name": "__getAllLanguages",
 "signature": "<unresolvedSignature>(1)",
 "code": "$langCtrlr->__getAllLanguages(\" where translated=1\")",
 "typeFullName": "LanguageController->__getAllLanguages.<returnValue>",
 "order": 2,
 "methodFullName": "LanguageController->__getAllLanguages",
 "_label": "CALL",
 "dynamicTypeHintFullName": [
 "LanguageController->__getAllLanguages"
],
 "dispatchType": "DYNAMIC_DISPATCH",
 "lineNumber": 35,
 "id": 48766,
 "possibleTypes": [
],
 "argumentIndex": 2
 }
]"""

```

cpg.call.filter(node => node.id==48755).assignment.source.toJsonPretty

```

joern> cpg.call.filter(node => node.id==48755).assignment.source.toJsonPretty
val res46: String = """[
 {
 "dynamicTypeHintFullName": [
],
 "code": "",
 "typeFullName": "ANY",
 "order": 2,
 "_label": "BLOCK",
 "lineNumber": 34,
 "id": 48757,
 "possibleTypes": [
],
 "argumentIndex": 2
 },
 {
 "name": "<operator>.alloc",
 "signature": "",
 "code": "LanguageController.<alloc>()",
 "typeFullName": "LanguageController",
 "order": 2,
 "methodFullName": "<operator>.alloc",
 "_label": "CALL",
 "dynamicTypeHintFullName": [
],
 "dispatchType": "STATIC_DISPATCH",
 "lineNumber": 34,
 "id": 48760,
 "possibleTypes": [
],
 "argumentIndex": 2
 }
]"""

```

源码：

```

class student():
 def __init__(self):
 self.name = None
 self.age = None

 def main():
 s1 = student() # 112
 s2 = s1 # 115

```

示例：

```
cpg.call.filter(node => node.id==112).assignment.source.toJsonPretty
```

```
joern> cpg.call.filter(node => node.id==112).assignment.source.toJsonPretty
val res18: String = """[
{
 "name": "student",
 "signature": "",
 "code": "student()",
 "typeFullName": "ANY",
 "columnNumber": 10,
 "order": 2,
 "methodFullName": ":<module>.student",
 "_label": "CALL",
 "dynamicTypeHintFullName": [
],
 "dispatchType": "DYNAMIC_DISPATCH",
 "lineNumber": 7,
 "id": 110,
 "possibleTypes": [
],
 "argumentIndex": 2
}
]"""
```

总结：

创建对象语句是赋值语句，但其source与普通的赋值语句具有一定的区别。

TODO：还需要观察，看看这个BLOCK node是不是必然出现。

## 16.查看函数返回值

```
cpg.method.filter(node => node.fullName == ":<module>.student").methodReturn.toJsonPretty
cpg.method.filter(node => node.fullName == "set_request_var").methodReturn.toJsonPretty
```

```
joern> cpg.method.fullName(":<module>.student").methodReturn.toJsonPretty
val res16: String = """[
{
 "dynamicTypeHintFullName": [
],
 "evaluationStrategy": "BY_VALUE",
 "code": "RET",
 "typeFullName": "ANY",
 "order": 2,
 "_label": "METHOD_RETURN",
 "possibleTypes": [
],
 "id": 183
}
]"""
```

## 17.查看函数紧接着的CFG Node

源码：

```

function to show system settings
function showSystemSettings($category='system') {
 $category = addslashes($category);
 $this->set('list', $this->__getAllSettings(true, 1, $category));
}

```

示例：

```

cpg.method.filter(node => node.fullName == "SettingsController->showSystemSettings").toJsonPretty
cpg.method.filter(node => node.fullName == "SettingsController->render").toJsonPretty

```

```

joern> cpg.method.fullName("SettingsController->render").toJsonPretty
val res14: String = """
]"""

joern> cpg.method.filter(node => node.fullName == "SettingsController->showSystemSettings").toJsonPretty
val res15: String = """
{
 "name": "showSystemSettings",
 "astParentFullName": "SettingsController",
 "fullName": "SettingsController->showSystemSettings",
 "signature": "<unresolvedSignature>(1)",
 "astParentType": "TYPE_DECL",
 "_label": "METHOD",
 "code": "function showSystemSettings(this,$category)",
 "isExternal": false,
 "lineNumber": 29,
 "id": 48734,
 "order": 3,
 "filename": "controllers/settings.ctrl.php"
}
]"""

```

现在要查找SettingsController->showSystemSettings紧接着的CFG Node：

```

cpg.method.filter(node => node.fullName == "SettingsController-
>showSystemSettings").cfgNext.toJsonPretty
cpg.method.filter(node => node.fullName == ":<module>.student.
<returnValue>.find_min").cfgNext.toJsonPretty

```

```

joern> cpg.method.fullName("SettingsController->render").toJsonPretty
val res14: String = """
]"""

joern> cpg.method.filter(node => node.fullName == "SettingsController->showSystemSettings").toJsonPretty
val res15: String = """
{
 "name": "showSystemSettings",
 "astParentFullName": "SettingsController",
 "fullName": "SettingsController->showSystemSettings",
 "signature": "<unresolvedSignature>(1)",
 "astParentType": "TYPE_DECL",
 "_label": "METHOD",
 "code": "function showSystemSettings(this,$category)",
 "isExternal": false,
 "lineNumber": 29,
 "id": 48734,
 "order": 3,
 "filename": "controllers/settings.ctrl.php"
}
]"""

```

似乎也可以使用：

```

cpg.method.filter(node => node.fullName == "SettingsController-
>showSystemSettings")._cfg0ut.toJsonPretty
cpg.method.filter(node => node.fullName == "Student.hello:void(Student)")._cfg0ut.toJsonPretty

```

```

joern> cpg.method.filter(node => node.fullName == "SettingsController->showSystemSettings")._cfgOut.toJsonPretty
val res17: String = """[
 {
 "dynamicTypeHintFullName": [
],
 "name": "category",
 "code": "$category",
 "typeFullName": "addslashes.<returnValue>",
 "order": 1,
 "_label": "IDENTIFIER",
 "lineNumber": 30,
 "id": 48739,
 "possibleTypes": [
],
 "argumentIndex": 1
 }
]"""

```

特殊情况：

当这个函数有返回值时，用原本的fullname找不到后继节点，这时候应该去除fullname中的  
`<returnValue>`，例如：

源码：

```

class student():
 def __init__(self):
 self.name = None
 self.age = None

 def find_min(self, a: int, b: int, c: int, d: int):
 min1 = min(a, b)
 min2 = min(min1, c)
 min3 = min(min2, d)
 return min3

 def find_max(a: int, b: int, c: int, d: int):
 max1 = max(a, b)
 max2 = max(max1, c)
 max3 = max(max2, d)
 return max3

 def main():
 s1 = student() # 212
 x = s1.find_min(a=10, b=20, c=30, d=40) # 223
 y = find_max(a=10, b=20, c=30, d=40) # 231
 print(f"x={x}") # 237
 print(f"y={y}") # 243

main()

```

示例：

```

cpg.method.filter(node => node.fullName == ":<module>.student.
<returnValue>.find_min").cfgNext.toJsonPretty

```

```

joern> cpg.method.filter(node => node.fullName == ":<module>.student.<returnValue>.find_min").cfgNext.toJsonPretty
val res23: String = """[
]"""

```

```

cpg.method.filter(node => node.fullName == ":<module>.student.find_min").cfgNext.toJsonPretty

```

```
joern> cpg.method.filter(node => node.fullName == ":<module>.student.find_min").cfgNext.toJsonPretty
val res24: String = """[
{
 "dynamicTypeHintFullName": [
],
 "name": "min1",
 "code": "min1",
 "typeFullName": "__builtin.min.<returnValue>",
 "columnNumber": 9,
 "order": 1,
 "_label": "IDENTIFIER",
 "lineNumber": 7,
 "id": 71,
 "possibleTypes": [
],
 "argumentIndex": 1
}
]"""
```

## 18.查看IF语句的条件与分支

```
cpg.ifBlock.condition.toJsonPretty
```

```
joern> cpg.ifBlock.condition.toJsonPretty
val res24: String = """[
{
 "name": "<operator>.greaterThan",
 "signature": "",
 "code": "a > b",
 "typeFullName": "ANY",
 "columnNumber": 8,
 "order": 1,
 "methodFullName": "<operator>.greaterThan",
 "_label": "CALL",
 "dynamicTypeHintFullName": [
],
 "dispatchType": "STATIC_DISPATCH",
 "lineNumber": 36,
 "id": 300,
 "possibleTypes": [
],
 "argumentIndex": -1
}
]"""
```

```
cpg.ifBlock.whenTrue.toJsonPretty
```

```
joern> cpg.ifBlock.whenTrue.toJsonPretty
val res23: String = """[
{
 "dynamicTypeHintFullName": [
],
 "code": "print(\"a > b\")",
 "typeFullName": "ANY",
 "columnNumber": 5,
 "order": 2,
 "_label": "BLOCK",
 "lineNumber": 36,
 "id": 305,
 "possibleTypes": [
],
 "argumentIndex": -1
}
]"""
```

```
cpg.ifBlock.whenFalse.toJsonPretty
```

```
joern> cpg.ifBlock.whenFalse.toJsonPretty
val res28: String = """[
 {
 "dynamicTypeHintFullName": [
],
 "code": "print(\"b < a\")",
 "typeFullName": "ANY",
 "columnNumber": 9,
 "order": 3,
 "_label": "BLOCK",
 "lineNumber": 39,
 "id": 309,
 "possibleTypes": [
],
 "argumentIndex": -1
 }
]"""
```

## 19.查看控制语句的条件

源码：

```
if a > b: # 314
 print("a > b") # 317
else:
 print("b < a") # 322
while b > 0: # 326
 b = b - 1
 a = a + 1
```

示例：

```
cpg.controlStructure.toJsonPretty
cpg.controlStructure.filter(node => node.id==24).astChildren.toJsonPretty
查看for循环的初始化、条件和更新3个部分对应的CPG Nodes
cpg.controlStructure.filter(node => node.id==24).astChildren.isCall.toJsonPretty
```

```
joern> cpg.controlStructure.toJsonPretty
val res45: String = """[
 {
 "parserTypeName": "<empty>",
 "code": "if ... : ...",
 "columnNumber": 5,
 "order": 15,
 "controlStructureType": "IF",
 "_label": "CONTROL_STRUCTURE",
 "argumentIndex": -1,
 "lineNumber": 39,
 "id": 324
 },
 {
 "parserTypeName": "<empty>",
 "code": "while ... : ...",
 "columnNumber": 5,
 "order": 16,
 "controlStructureType": "WHILE",
 "_label": "CONTROL_STRUCTURE",
 "argumentIndex": -1,
 "lineNumber": 43,
 "id": 343
 },
 {
 "parserTypeName": "<empty>",
 "code": "...",
 "columnNumber": 5,
 "order": 17,
 "controlStructureType": "DO",
 "_label": "CONTROL_STRUCTURE",
 "argumentIndex": -1,
 "lineNumber": 45,
 "id": 344
 }
]"""
```

```
cpg.controlStructure.filter(node => node.id==324).condition.toJsonPretty
cpg.controlStructure.filter(_.lineNumber==Some(value = 39)).toJsonPretty
```

```
joern> cpg.controlStructure.filter(node => node.id==324).condition.toJsonPretty
val res46: String = """[
 {
 "name": "<operator>.greaterThan",
 "signature": "",
 "code": "a > b",
 "typeFullName": "ANY",
 "columnNumber": 8,
 "order": 1,
 "methodFullName": "<operator>.greaterThan",
 "_label": "CALL",
 "dynamicTypeHintFullName": [
],
 "dispatchType": "STATIC_DISPATCH",
 "lineNumber": 39,
 "id": 320,
 "possibleTypes": [
],
 "argumentIndex": -1
 }
]"""
```

```
cpg.controlStructure.filter(node => node.id==343).condition.toJsonPretty
```

```
joern> cpg.controlStructure.filter(node => node.id==343).condition.toJsonPretty
val res47: String = """[
 {
 "name": "<operator>.greaterThan",
 "signature": "",
 "code": "b > 0",
 "typeFullName": "ANY",
 "columnNumber": 11,
 "order": 1,
 "methodFullName": "<operator>.greaterThan",
 "_label": "CALL",
 "dynamicTypeHintFullName": [
],
 "dispatchType": "STATIC_DISPATCH",
 "lineNumber": 43,
 "id": 332,
 "possibleTypes": [
],
 "argumentIndex": -1
 }
]"""
```

## 20. 判断一条语句是否是控制流

查询该语句后面的一个call语句，看它是否被该语句所控制，用到的命令主要有：

```
cpg.all.filter(node => node.id==35).cfgOut.toJsonPretty
cpg.call.filter(node => node.id==182).controlledBy.toJsonPretty
```

## 21. 寻找一条语句对应的控制语句

方法一：根据行号

```
cpg.controlStructure.filter(_.lineNumber==Some(value = 83)).toJsonPretty
```

```

joern> cpg.controlStructure.filter(_.lineNumber==Some(value = 83)).toJsonPretty
val res86: String = """[
 {
 "parserTypeName": "<empty>",
 "code": "if (day == 1)",
 "columnNumber": 9,
 "order": 59,
 "controlStructureType": "IF",
 "_label": "CONTROL_STRUCTURE",
 "argumentIndex": -1,
 "lineNumber": 83,
 "id": 382
 }
]"""

```

方法二：根据是否含有相应的代码

```
cpg.controlStructure.filter(_.code.contains("day")).toJsonPretty
```

```

joern> cpg.controlStructure.filter(_.code.contains("day")).toJsonPretty
val res87: String = """[
 {
 "parserTypeName": "<empty>",
 "code": "switch(// day:350 switch: 349\nday)",
 "order": 58,
 "controlStructureType": "SWITCH",
 "_label": "CONTROL_STRUCTURE",
 "argumentIndex": -1,
 "id": 349
 },
 {
 "parserTypeName": "<empty>",
 "code": "if (day == 1)",
 "columnNumber": 9,
 "order": 59,
 "controlStructureType": "IF",
 "_label": "CONTROL_STRUCTURE",
 "argumentIndex": -1,
 "lineNumber": 83,
 "id": 382
 },
 {
 "parserTypeName": "<empty>",
 "code": "if (day == 2)",
 "columnNumber": 16,
 "order": 1,
 "controlStructureType": "IF",
 "_label": "CONTROL_STRUCTURE",
 "argumentIndex": -1,
 "lineNumber": 85,
 "id": 393
 },
 {
 "parserTypeName": "<empty>",
 "code": "if (day == 3)",
 "columnNumber": 16,
 "order": 1,
 "controlStructureType": "IF",
 "_label": "CONTROL_STRUCTURE",
 "argumentIndex": -1,
 "lineNumber": 87,
 "id": 404
 }
]"""

```

方法二会检索到许多节点，还需要根据node id进行判断才行。

## 22.判断函数是否是类的函数？

源码：

```

class Student {
 String name;
 int age;
 double[] scores = new double[3];
 HashMap<String, String> myMap = new HashMap<String, String>();

 public void hello(Teacher teacher) {
 System.out.println("Hello! teacher " + teacher.name);
 }

 public void hello(Student student) {
 System.out.println("Hello! student " + student.name);
 }
}

```

```
s1.hello(s3); // 172
```

示例：

```
cpg.call.filter(node => node.id==172).toJsonPretty
```

```
joern> cpg.call.filter(node => node.id==172).toJsonPretty
val res13: String = """[
{
 "name": "hello",
 "signature": "void(Student)",
 "code": "s1.hello(s3)",
 "typeFullName": "void",
 "columnNumber": 9,
 "order": 20,
 "methodFullName": "Student.hello:void(Student)",
 "_label": "CALL",
 "dynamicTypeHintFullName": [
],
 "dispatchType": "DYNAMIC_DISPATCH",
 "lineNumber": 35,
 "id": 172,
 "possibleTypes": [
],
 "argumentIndex": -1
}
]"""
```

```
cpg.method.filter(node => node.fullName == "Student.hello:void(Student)").toJsonPretty
```

```
joern> cpg.method.filter(node => node.fullName == "Student.hello:void(Student)").toJsonPretty
val res14: String = """[
{
 "name": "hello",
 "astParentFullName": "<empty>",
 "fullName": "Student.hello:void(Student)",
 "signature": "void(Student)",
 "astParentType": "<empty>",
 "lineNumberEnd": 19,
 "_label": "METHOD",
 "columnNumberEnd": 5,
 "code": "public void hello(Student student)",
 "isExternal": false,
 "lineNumber": 17,
 "id": 72,
 "columnNumber": 5,
 "order": 6,
 "filename": ""
}
]"""
```

```
cpg.method.filter(node => node.fullName == "Student.hello:void(Student)").astParent.toJsonPretty
```

```
joern> cpg.method.filter(node => node.fullName == "Student.hello:void(Student)").astParent.toJsonPretty
val res15: String = """[
{
 "inheritsFromTypeFullName": [
 "java.lang.Object"
],
 "name": "Student",
 "astParentFullName": "<global>",
 "fullName": "Student",
 "astParentType": "NAMESPACE_BLOCK",
 "_label": "TYPE_DECL",
 "code": "class Student",
 "isExternal": false,
 "lineNumber": 7,
 "id": 17,
 "columnNumber": 1,
 "order": 2,
 "filename": ""
}
]"""
```

根据函数定义的父节点的 `_label` 属性是否为 `"TYPE_DECL"` 即可判断该函数是否是类的函数。

## 23.判断函数是否是普通函数?

源码：

```
def test1():
 return "None"
s2 = test1() # 261
```

示例：

```
cpg.call.filter(node => node.id==261).assignment.source.toJsonPretty
```

```
joern> cpg.call.filter(node => node.id==261).assignment.source.toJsonPretty
val res5: String = """[
{
 "name": "test1",
 "signature": "",
 "code": "test1(')",
 "typeFullName": "__builtin.str",
 "columnNumber": 10,
 "order": 2,
 "methodFullName": "<:module>.test1",
 "_label": "CALL",
 "dynamicTypeHintFullName": [
],
 "dispatchType": "DYNAMIC_DISPATCH",
 "lineNumber": 26,
 "id": 259,
 "possibleTypes": [
],
 "argumentIndex": 2
}
]"""
```

```
cpg.method.filter(node => node.fullName == "<:module>.test1").toJsonPretty
```

```
joern> cpg.method.filter(node => node.fullName == "<:module>.test1").toJsonPretty
val res8: String = """[
{
 "name": "test1",
 "astParentFullName": "<empty>",
 "fullName": "<:module>.test1",
 "signature": "",
 "astParentType": "<empty>",
 "lineNumberEnd": 14,
 "_label": "METHOD",
 "offsetEnd": 238,
 "columnNumberEnd": 0,
 "code": "<empty>",
 "isExternal": false,
 "lineNumber": 12,
 "id": 199,
 "offset": 206,
 "columnNumber": 1,
 "order": 5,
 "filename": ""
}
]"""
```

```
cpg.method.filter(node => node.fullName == "<:module>.test1").astParent.toJsonPretty
```

```
joern> cpg.method.filter(node => node.fullName == ":<module>.test1").astParent.toJsonPretty
val res6: String = """[
 {
 "name": "<module>",
 "astParentFullName": "<empty>",
 "fullName": ":<module>",
 "signature": "",
 "astParentType": "<empty>",
 "lineNumberEnd": 5,
 "_label": "METHOD", ↑
 "columnNumber": 1,
 "code": "<empty>",
 "isExternal": false,
 "lineNumber": 1,
 "id": 7,
 "offset": 0,
 "columnNumberEnd": 5,
 "order": 1,
 "filename": ""
 }
]"""
```

根据函数定义的父节点的 `_label` 属性是否为 `"METHOD"` 即可判断该函数是否是普通函数。

## 24.查找函数的接收者

源码：

```
$this->render('settings/showsettings'); // "id":48971
$blogContent = getCustomizerPage('aboutus'); // "id":49194
```

示例：

```
cpg.call.filter(node => node.id==48971).receiver.toJsonPretty
```

```
joern> cpg.call.filter(node => node.id==48971).receiver.toJsonPretty
val res5: String = """[
 {
 "dynamicTypeHintFullName": [
],
 "name": "this",
 "code": "$this",
 "typeFullName": "SettingsController",
 "order": 1,
 "_label": "IDENTIFIER",
 "lineNumber": 102,
 "id": 48972,
 "possibleTypes": [
],
 "argumentIndex": 0
 }
]"""
```

```
cpg.call.filter(node => node.id==49194).receiver.toJsonPretty
```

```
joern> cpg.call.filter(node => node.id==49194).receiver.toJsonPretty
val res7: String = """[

]"""
```

## 25.绘制CPG

```
cpg.method.name("main").plotDotCpg14
```

```
joern> cpg.method.name("main").plotDotCpg14
joern>
```

## 26.scala语言中的与或非

```
scala中的逻辑或
cpg.call.filter(node => (node.methodFullName == "<operator>.assignmentConcat" || node.methodFullName == "<operator>.concat")).toJsonPretty

也可以使用一个数组来判断
cpg.call.filter(node => Array("<operator>.assignmentConcat",
<operator>.concat").contains(node.methodFullName)).toJsonPretty

scala中的逻辑与
cpg.call.filter(node => Array("<operator>.assignmentConcat",
<operator>.concat").contains(node.methodFullName)).filter(node => (node.code.contains("SELECT") &&
node.code.contains("_GET"))).toJsonPretty

scala中的逻辑非
cpg.call.filter(node => Array("<operator>.assignmentConcat").contains(node.methodFullName)).filter(node => (node.code.contains("<a>") && node.code.contains("$") && (!
node.code.contains("${_SESSION}"))).toJsonPretty

cpg.call.filter(node => (node.code.contains("${_GET}["HTTP_CLIENT_IP"]") ||
node.code.contains("${_POST}["HTTP_CLIENT_IP"]") || node.code.contains("${_COOKIE}["HTTP_CLIENT_IP"]") ||
|| node.code.contains("${_SERVER}["HTTP_CLIENT_IP"]") ||
node.code.contains("${_REQUEST}["HTTP_CLIENT_IP"]") ||
node.code.contains("${_SESSION}["HTTP_CLIENT_IP"]") || node.code.contains("${_FILES}["HTTP_CLIENT_IP"]") ||
|| node.code.contains("${_ENV}["HTTP_CLIENT_IP"]))).map(x=> (x.node.id, x.node.code,
x.node.location.filename, x.node.location.lineNumber, x.node.methodFullName)).toJsonPretty

cpg.call.filter(node => ((node.code.contains("echo") || node.code.contains("print")) &&
node.code.contains("$category") && node.location.filename.contains("settings/showsettings"))).map(x=>
(x.node.id, x.node.code, x.node.location.filename, x.node.location.lineNumber,
x.node.methodFullName)).toJsonPretty
```

## 27.获取函数调用点

```
cpg.method.filter(node => node.fullName == "session->getip").callIn.toJsonPretty
```

```
joern> cpg.method.filter(node => node.fullName == "session->getip").callIn.toJsonPretty
val res18: String = """[
 {
 "name":"getip",
 "signature":"<unresolvedSignature>(0)",
 "code":"getip()",

 "typeFullName":"ANY",
 "order":2,
 "methodFullName":"session->getip",
 "_label":"CALL",
 "dynamicTypeHintFullName": [
 "session->getip"
],
 "dispatchType":"STATIC_DISPATCH",
 "lineNumber":42,
 "id":65369,
 "possibleTypes": [
],
 "argumentIndex":2
 }
]"""
```

```
joern> cpg.method.filter(node => node.fullName == "session->getip").callIn.map(x=> (x.node.id, x.node.code, x.node.location.filename, x.node.location.lineNumber)).toJsonPretty
val res17: String = """[
 {
 "_1":65369,
 "_2":"getip()",

 "_3":"inc/class_session.php",
 "_4":42
 }
]"""
```

## 28. 获取函数被哪些函数调用

```
cpg.method.filter(node => node.fullName == "session->getip").caller.toJsonPretty
cpg.method.filter(node => node.fullName == "session->init").caller.toJsonPretty
```

```
joern> cpg.method.filter(node => node.fullName == "session->getip").caller.toJsonPretty
val res20: String = """[
 {
 "name":"init",
 "astParentFullName":"session",
 "fullName":"session->init",
 "signature":"<unresolvedSignature>(0)",
 "astParentType":"TYPE_DECL",
 "_label":"METHOD",
 "code":"function init(this)",

 "isExternal":false,
 "lineNumber":36,
 "id":65355,
 "order":9,
 "filename":"inc/class_session.php"
 }
]"""
```

## 29. 查看一条语句所属的函数定义节点

```

cpg.call.filter(node => node.id==73597).dominatedBy.isMethod.sortBy(node =>
node.lineNumber).toJsonPretty
cpg.call.filter(node => node.id==65369).dominatedBy.isMethod.sortBy(node =>
node.lineNumber).toJsonPretty
cpg.call.filter(node => node.id==757).dominatedBy.isMethod.sortBy(node => node.lineNumber).toJsonPretty
cpg.call.filter(node => node.id==263).dominatedBy.isMethod.sortBy(node => node.lineNumber).toJsonPretty
cpg.call.filter(node => node.id==127).dominatedBy.isMethod.sortBy(node => node.lineNumber).toJsonPretty
cpg.call.filter(node => node.id==1121006).dominatedBy.isMethod.sortBy(node =>
node.lineNumber).toJsonPretty

cpg.call.filter(node => node.id==211382).dominatedBy.isMethod.sortBy(node =>
node.lineNumber).toJsonPretty
cpg.call.filter(node => node.id==211382).dominatedBy.sortBy(node => node.lineNumber).toJsonPretty

```

```

joern> cpg.call.filter(node => node.id==73597).dominatedBy.isMethod.sortBy(node => node.lineNumber).toJsonPretty
val res24: String = """[
{
 "name": "getip",
 "astParentFullName": "inc/functions.php:<global>",
 "fullName": "getip",
 "signature": "<unresolvedSignature>(0)",
 "astParentType": "METHOD",
 "_label": "METHOD",
 "code": "function getip()",
 "isExternal": false,
 "lineNumber": 1345,
 "id": 73549,
 "order": 47,
 "filename": "inc/functions.php"
}
]"""

```

```

joern> cpg.call.filter(node => node.id==65369).dominatedBy.isMethod.sortBy(node => node.lineNumber).toJsonPretty
val res62: String = """[
{
 "name": "init",
 "astParentFullName": "session",
 "fullName": "session->init",
 "signature": "<unresolvedSignature>(0)",
 "astParentType": "TYPE_DECL",
 "_label": "METHOD",
 "code": "function init(this)",
 "isExternal": false,
 "lineNumber": 36,
 "id": 65355,
 "order": 9,
 "filename": "inc/class_session.php"
}
]"""

```

```

joern> cpg.call.filter(node => node.id==757).dominatedBy.isMethod.sortBy(node => node.lineNumber).toJsonPretty
val res2: String = """[
{
 "name": "makeonoffcode",
 "astParentFullName": "admin/adminfunctions.php:<global>",
 "fullName": "makeonoffcode",
 "signature": "<unresolvedSignature>(3)",
 "astParentType": "METHOD",
 "_label": "METHOD",
 "code": "function makeonoffcode($title,$name,$value)",
 "isExternal": false,
 "lineNumber": 223,
 "id": 750,
 "order": 21,
 "filename": "admin/adminfunctions.php"
}
]"""

```

```
joern> cpg.call.filter(node => node.id==263).dominatedBy.isMethod.sortBy(node => node.lineNumber).toJsonPretty
val res6: String = """[
 {
 "name":"main",
 "astParentFullName":"><empty>",
 "fullName":":<module>.main",
 "signature":"",
 "astParentType":"><empty>",
 "lineNumberEnd":53,
 "_label":"METHOD",
 "offsetEnd":949,
 "columnNumberEnd":0,
 "code":"><empty>",
 "isExternal":false,
 "lineNumber":24,
 "id":256,
 "offset":339,
 "columnNumber":1,
 "order":13,
 "filename": ""
 }
]"""
```

```
joern> cpg.call.filter(node => node.id==127).dominatedBy.isMethod.sortBy(node => node.lineNumber).toJsonPretty
val res9: String = """[
 {
 "name":"main",
 "astParentFullName":"><empty>",
 "fullName":">java_class.main:void(java.lang.String[])",
 "signature":">void(java.lang.String[])",
 "astParentType":"><empty>",
 "lineNumberEnd":95,
 "_label":"METHOD",
 "columnNumberEnd":5,
 "code":">public static void main(String[] args)",
 "isExternal":false,
 "lineNumber":23,
 "id":109,
 "columnNumber":5,
 "order":1,
 "filename": ""
 }
]"""
```

```
joern> cpg.call.filter(node => node.id==1121006).dominatedBy.isMethod.sortBy(node => node.lineNumber).toJsonPretty
val res59: String = """[
 {
 "name":"><global>",
 "astParentFullName":">settings.php:<global>",
 "fullName":">settings.php:<global>",
 "signature":"><unresolvedSignature>(0)",
 "astParentType":">TYPE_DECL",
 "_label":"METHOD",
 "code":">VIRTUAL PUBLIC STATIC function <global>(),
 "isExternal":false,
 "id":1120594,
 "order":1,
 "filename":">settings.php"
 }
]"""
```

## 30.限制查询结果数量 (TODO: 使用该方法优化joern查询速度)

```
cpg.call.filter(node => node.code == "$_SERVER[\"REQUEST_METHOD\"]").take(1).toJsonPretty
```

```

joern> cpg.call.filter(node => node.code == "$_SERVER[\"REQUEST_METHOD\"]").take(1).toJsonPretty
val res5: String = """[
 {
 "name": "<operator>.indexAccess",
 "signature": "",
 "code": "$_SERVER[\"REQUEST_METHOD\"]",
 "typeFullName": "ANY",
 "order": 1,
 "methodFullName": "<operator>.indexAccess",
 "label": "CALL",
 "dynamicTypeHintFullName": [
],
 "dispatchType": "STATIC_DISPATCH",
 "lineNumber": 56,
 "id": 14212,
 "possibleTypes": [
],
 "argumentIndex": 1
 }
]"""

```

## 31.重复执行某一操作

---

源码：

```

for (int i = 0; i < 5; i++) { // "int i = 0": 26, "i < 5": 29, "i++": 32
 a++; // 35
 System.out.println("a1:" + a); // 37
}

```

示例一：

```
cpg.call.filter(node => node.id==35).repeat(_.cfgOut)(_.maxDepth(3)).toJsonPretty
```

```

joern> cpg.call.filter(node => node.id==35).repeat(_.cfgOut)(_.maxDepth(3)).toJsonPretty
val res2: String = """[
 {
 "name": "<operator>.fieldAccess",
 "signature": "",
 "code": "System.out",
 "typeFullName": "java.io.PrintStream",
 "columnNumber": 13,
 "order": 1,
 "methodFullName": "<operator>.fieldAccess",
 "label": "CALL",
 "dynamicTypeHintFullName": [
],
 "dispatchType": "STATIC_DISPATCH",
 "lineNumber": 6,
 "id": 38,
 "possibleTypes": [
],
 "argumentIndex": 0
 }
]"""

```

等价于执行了三次`_cfgOut`方法：

```

joern> cpg.call.filter(node => node.id==35).cfgOut.toJsonPretty
val res3: String = """[
 {
 "dynamicTypeHintFullName": [
],
 "name": "System",
 "code": "System",
 "typeFullName": "java.lang.System",
 "columnNumber": 13,
 "order": 1,
 "_label": "IDENTIFIER",
 "lineNumber": 6,
 "id": 39,
 "possibleTypes": [
],
 "argumentIndex": 1
 }
]"""

```

```

joern> cpg.all.filter(node => node.id==39).cfgOut.toJsonPretty
val res4: String = """[
 {
 "code": "out",
 "columnNumber": 20,
 "order": 2,
 "_label": "FIELD_IDENTIFIER",
 "argumentIndex": 2,
 "canonicalName": "out",
 "lineNumber": 6,
 "id": 40
 }
]"""

```

```

joern> cpg.all.filter(node => node.id==40).cfgOut.toJsonPretty
val res5: String = """[
 {
 "name": "<operator>.fieldAccess",
 "signature": "",
 "code": "System.out",
 "typeFullName": "java.io.PrintStream",
 "columnNumber": 13,
 "order": 1,
 "methodFullName": "<operator>.fieldAccess",
 "_label": "CALL",
 "dynamicTypeHintFullName": [
],
 "dispatchType": "STATIC_DISPATCH",
 "lineNumber": 6,
 "id": 38,
 "possibleTypes": [
],
 "argumentIndex": 0
 }
]"""

```

**注意：**这里不应该使用**times**方法，因为这个方法已经被废弃了

```

joern> cpg.call.filter(node => node.id==35).repeat(_.cfgOut).times(3).toJsonPretty
1 warning found
-- Deprecation Warning: -----
1 |cpg.call.filter(node => node.id==35).repeat(_.cfgOut)(_.times(3)).toJsonPretty
|-----^
|method times in class Builder is deprecated since 1.153: use `maxDepth` instead - semantically equivalent, while it describes the meaning more precisely
val res1: String = """[
 {
 "name": "<operator>.fieldAccess",
 "signature": "",
 "code": "System.out",
 "typeFullName": "java.io.PrintStream",
 "columnNumber": 13,
 "order": 1,
 "methodFullName": "<operator>.fieldAccess",
 "_label": "CALL",
 "dynamicTypeHintFullName": [
],
 "dispatchType": "STATIC_DISPATCH",
 "lineNumber": 6,
 "id": 38,
 "possibleTypes": [
],
 "argumentIndex": 0
 }
]"""

```

## 示例二：

```
cpg.call.filter(node => node.id==21).repeat(_.cfgNext)(_.until(_.isCall)).toJsonPretty
```

```

joern> cpg.call.filter(node => node.id==21).repeat(_.cfgNext)().until(_.isCall)).toJsonPretty
val res11: String = """[{"name":<operator>.assignment", "signature": "", "code": "int i = 0", "typeFullName": "int", "columnNumber": 18, "order": 2, "methodFullName": <operator>.assignment", "label": "CALL", "dynamicTypeHintFullName": [{"dispatchType": "STATIC_DISPATCH", "lineNumber": 4, "id": 26, "possibleTypes": [{"argumentIndex": -1}]}]}]"""

```

**注意：**这里不能使用`_cfgOut`方法，原因不清楚。

```

joern> cpg.call.filter(node => node.id==63).repeat(_.cfgOut)().until(_.isCall)).toJsonPretty
-- [E008] Not Found Error: -----
1 |cpg.call.filter(node => node.id==63).repeat(_.cfgOut)().until(_.isCall)).toJsonPretty
| value isCall is not a member of Iterator[io.shiftleft.codepropertygraph.generated.nodes.StoredNode], but could be made available as an extension method.
| The following import might make progress towards fixing the problem:
| import sourcecode.Text.generate
1 error found

```

## 32.查找语句所属控制结构

源码：

```

if (array_key_exists('Action', $_POST) && $_POST['Action'] == 'Retrieve' && !empty($_POST['Event'])) {

} elseif (array_key_exists('Action', $_GET) && $_GET['Action'] == 'List' && !empty($_GET['Event'])) {
 $sSQL = 'SELECT * FROM events_event WHERE event_type = '.$_GET['Event']. ' ORDER BY event_start';
// 70479
 $spageTitle = gettext('All Events of Type').': '.$_GET['Type'];
} else {
 $sSQL = 'SELECT * FROM events_event ORDER BY event_start';
}

```

**示例一：**

```
cpg.call.filter(node => node.id==70479).controlledBy.toJsonPretty
```

```

joern> cpg.call.filter(node => node.id==70479).controlledBy.toJsonPretty
val res2: String = """[
 {
 "name": "<operator>.logicalAnd",
 "signature": "",
 "code": "array_key_exists(\"Action\",$_POST) && $_POST[\"Action\"] == \"Retrieve\" && !empty($_POST[\"Event\"])",
 "typeFullName": "ANY",
 "order": 1,
 "methodFullName": "<operator>.logicalAnd",
 "_label": "CALL",
 "dynamicTypeHintFullName": [
],
 "dispatchType": "STATIC_DISPATCH",
 "lineNumber": 18,
 "id": 70344,
 "possibleTypes": [
],
 "argumentIndex": -1
 },
 {
 "name": "<operator>.logicalAnd",
 "signature": "",
 "code": "array_key_exists(\"Action\",$_GET) && $_GET[\"Action\"] == \"List\" && !empty($_GET[\"Event\"])",
 "typeFullName": "ANY",
 "order": 1,
 "methodFullName": "<operator>.logicalAnd",
 "_label": "CALL",
 "dynamicTypeHintFullName": [
],
 "dispatchType": "STATIC_DISPATCH",
 "lineNumber": 52,
 "id": 70463,
 "possibleTypes": [
],
 "argumentIndex": -1
 }
]"""

```

## 33.如何判断语句处于控制结构的哪一分支?

对于处于控制结构之下的一条语句，例如**IF**控制结构，这条语句可能处于**True/False**分支，为了探索该语句所处的具体分支，可以采用如下语句：

源码：

```

if (array_key_exists('Action', $_POST) && $_POST['Action'] == 'Retrieve' && !empty($_POST['Event'])) {

} elseif (array_key_exists('Action', $_GET) && $_GET['Action'] == 'List' && !empty($_GET['Event'])) {
 $sSQL = 'SELECT * FROM events_event WHERE event_type = '.$_GET['Event']. ORDER BY event_start';
// 70479
 $sPageTitle = gettext('All Events of Type').': '.$_GET['Type'];
} else {
 $sSQL = 'SELECT * FROM events_event ORDER BY event_start';
}

```

### (1) 首先需要找到这一条语句所属的控制结构语句

```

cpg.call.filter(node => node.id==70479).controlledBy.isCall.map(x=> (x.node.id,
x.node.code)).toJsonPretty
cpg.call.filter(node => node.id==70479).controlledBy.isCall.map(x=> (x.node.id,
x.node.code)).toJsonPretty

```

```

joern> cpg.call.filter(node => node.id==70479).controlledBy.isCall.map(x=> (x.node.id, x.node.code)).toJsonPretty
val res28: String = """[
 {
 "_1":70344,
 "_2":"array_key_exists(\"Action\",$_POST) && $_POST[\"Action\"] == \"Retrieve\" && !empty($_POST[\"Event\"])"
 },
 {
 "_1":70463,
 "_2":"array_key_exists(\"Action\",$_GET) && $_GET[\"Action\"] == \"List\" && !empty($_GET[\"Event\"])"
 }
]"""

```

**注意：**调用isCall方法的目的在于可以调用map方法，但是如果要采用这种方法，控制结构的条件必须是一个Call语句。在下面这个例子中，就不应该采用此方法。

源码：

```

switch(day) { // day:350 switch: 349
 case 1: //
 System.out.println("Monday"); // 393
 break; //
 case 2: //
 System.out.println("Tuesday"); // 401
 break; //
 case 3: //
 System.out.println("Wednesday"); // 409
 break; //
 default: //
 System.out.println("Invalid day"); // 416
}

```

```

joern> cpg.call.filter(node => node.id==393).controlledBy.toJsonPretty
val res31: String = """[
 {
 "dynamicTypeHintFullName": [
],
 "name": "day",
 "code": "day",
 "typeFullName": "int",
 "columnNumber": 16,
 "order": 1,
 "_label": "IDENTIFIER",
 "lineNumber": 80,
 "id": 389,
 "possibleTypes": [
],
 "argumentIndex": -1
 }
]"""

joern> cpg.call.filter(node => node.id==393).controlledBy.isCall.map(x=> (x.node.id, x.node.code)).toJsonPretty
val res32: String = """[
]"""

```

## (2) 找到控制结构语句的子分支语句

```

cpg.call.filter(node => node.id==70344).repeat(_.cfgNext)(_.until(_.isCall)).toJsonPretty
cpg.call.filter(node => node.id==70344).repeat(_.cfgNext)(_.until(_.isCall)).isCall.map(x=> (x.node.id,
x.node.code)).toJsonPretty
cpg.call.filter(node => node.id==70482).repeat(_.cfgNext)(_.until(_.isCall)).isCall.map(x=> (x.node.id,
x.node.code)).toJsonPretty

```

```

jоern> cpg.call.filter(node => node.id==70344).repeat(_.cfgNext)().until(_.isCall)).isCall.map(x=> (x.node.id, x.node.code)).toJsonPretty
val res25: String = """[
 {
 "_1":70362,
 "_2":$_POST["Choice"]
 },
 {
 "_1":70465,
 "_2":"array_key_exists("Action",$_GET)"
 }
]"""

```

### (3) 检查两个分支子语句所主导的语句中有没有目标语句

```

// 检查True分支
cpg.call.filter(node => node.id==70362).dominates.filter(node => node.id==70479).toJsonPretty
cpg.call.filter(node => node.id==70362).dominates.isCall.filter(node => node.id==70479).map(x=>
(x.node.id, x.node.code)).toJsonPretty

// 检查False分支
cpg.call.filter(node => node.id==70465).dominates.filter(node => node.id==70479).toJsonPretty
cpg.call.filter(node => node.id==70465).dominates.isCall.filter(node => node.id==70479).map(x=>
(x.node.id, x.node.code)).toJsonPretty

```

```

jоern> cpg.call.filter(node => node.id==70362).dominates.isCall.filter(node => node.id==70479).map(x=> (x.node.id, x.node.code)).toJsonPretty
val res24: String = """[

]"""


```

```

jоern> cpg.call.filter(node => node.id==70465).dominates.isCall.filter(node => node.id==70479).map(x=> (x.node.id, x.node.code)).toJsonPretty
val res23: String = """[
 {
 "_1":70479,
 "_2":$sSQL = "SELECT * FROM events_event WHERE event_type = " . $_GET["Event"] . " ORDER BY event_start"
 }
]"""

```

### (4) 根据是否有结果生成，就可以判断是否位于对应分支中

## 34.查询代码以特殊字符开头的CPG节点

以查找define语句为例，源码

```
define("IN_WS", true);
```

查询语句（比起使用contains查询，这样能够节省后续所需的处理）：

```
cpg.call.filter(node => (node.code.startsWith("define") && node.code.contains(", "))).toJsonPretty
```

```

jоern> cpg.call.filter(node => (node.code.startsWith("define") && node.code.contains(", "))).take(5).toJsonPretty
val res48: String = """[
 {
 "name":"define",
 "signature":<unresolvedSignature>(2),
 "code":"define(\"$PHPWG_ROOT_PATH\", \".\")",
 "typeFullName":"ANY",
 "order":10,
 "methodFullName":"define",
 "_label":"CALL",
 "dynamicTypeHintFullName": [

],
 "dispatchType":"STATIC_DISPATCH",
 "lineNumber":10,
 "id":8,
 "possibleTypes": [
],
 "argumentIndex": -1
 }
]"""

```

## 35.查找影响变量的语句

`_reachingDefIn` 命令用于查找与影响当前节点的所有可达的变量定义，即这些定义可能影响当前节点的变量值。它显示哪些语句定义了该变量，并且在控制流图中可以到达这个节点。

相反，`_reachingDefOut` 命令则显示受当前节点影响的所有可达的变量定义，这些定义在当前节点执行后仍然可以到达。这有助于了解在当前节点之后，哪些定义可能会影响后续的语句和计算。

这两个命令结合使用，可以全面分析变量在代码中的影响链。

源码来源：**piwigo 14.5.0** ./include/smarty/libs/sysplugins/smarty\_Internal\_TestInstall.php

```
class Smarty_Internal_TestInstall
{
 /**
 * diagnose Smarty setup
 * If $errors is specified, the diagnostic report will be appended to the array, rather than being
 * output.
 *
 * @param \Smarty $smarty
 * @param array $errors array to push results into rather than outputting them
 *
 * @return bool status, true if everything is fine, false else
 */
 public static function testInstall(Smarty $smarty, &$errors = null)
 {
 $status = true;
 if ($errors === null) {
 echo "<PRE>\n";
 echo "Smarty Installation test...\n";
 echo "Testing template directory...\n";
 }
 $_stream_resolve_include_path = function_exists('stream_resolve_include_path');
 // test if all registered template_dir are accessible
 foreach ($smarty->getTemplateDir() as $template_dir) {
 $_template_dir = $template_dir;
 $template_dir = realpath($template_dir);
 // resolve include_path or fail existence
 if (!$template_dir) {
 if ($smarty->use_include_path && !preg_match('/^([\\\\\\\\]| [a-zA-Z]:[\\\\\\\\])/',
$_template_dir)) {
 // try PHP include_path
 if ($_stream_resolve_include_path) {
 $template_dir = stream_resolve_include_path($_template_dir);
 } else {
 $template_dir = $smarty->ext->_getIncludePath-
 >getIncludePath($_template_dir, null, $smarty);
 }
 if ($template_dir !== false) {
 if ($errors === null) {
 echo "$template_dir is OK.\n";
 }
 continue;
 } else {
 $status = false;
 $message =
 }
 }
 }
 }
 }
}
```

```

 "FAILED: $_template_dir does not exist (and couldn't be found
in include_path either)"; // CPG ID:
 if ($errors === null) {
 echo $message . ".\n"; // CPG ID:
 } else {
 $errors['template_dir'] = $message;
 }
 continue;
}
} else {
 $status = false;
 $message = "FAILED: $_template_dir does not exist"; // CPG ID:
 if ($errors === null) {
 echo $message . ".\n"; // CPG ID:
 } else {
 $errors['template_dir'] = $message;
 }
 continue;
}
if (!is_dir($template_dir)) {
 $status = false;
 $message = "FAILED: $template_dir is not a directory"; // CPG ID:
 if ($errors === null) {
 echo $message . ".\n"; // CPG ID:
 } else {
 $errors['template_dir'] = $message;
 }
} elseif (!is_readable($template_dir)) {
 $status = false;
 $message = "FAILED: $template_dir is not readable"; // CPG ID:
 if ($errors === null) {
 echo $message . ".\n"; // CPG ID:313348
 } else {
 $errors['template_dir'] = $message;
 }
} else {
 if ($errors === null) {
 echo "$template_dir is OK.\n";
 }
}
}

```

我们重点关注 `$message` 的处理流程：第89行（CPG ID:313348）将 `$message` 打印输出，这里可能会造成XSS漏洞，因此我们从此处开始分析：

```
joern> cpg.all.filter(node => node.id==313348).toJsonPretty
val res31: String = """[
 {
 "name": "echo",
 "signature": "",
 "code": "echo $message . \".\\n\"",
 "typeFullName": "ANY",
 "order": 1,
 "methodFullName": "echo",
 "_label": "CALL",
 "dynamicTypeHintFullName": [
],
 "dispatchType": "STATIC_DISPATCH",
 "lineNumber": 89,
 "id": 313348,
 "possibleTypes": [
],
 "argumentIndex": -1
 }
]"""
```

我们想要知道这个Sink函数的实际参数是什么，为此我们可以使用 `astChildren` 函数（后面发现用 `_reachingDefIn` 也一样能获取）来获取：

```
joern> cpg.call.filter(node => node.id==313348).astChildren.toJsonPretty
val res37: String = """[
 {
 "name": "<operator>.concat",
 "signature": "",
 "code": "$message . \".\\n\"",
 "typeFullName": "ANY",
 "order": 1,
 "methodFullName": "<operator>.concat",
 "_label": "CALL",
 "dynamicTypeHintFullName": [
],
 "dispatchType": "STATIC_DISPATCH",
 "lineNumber": 89,
 "id": 313349,
 "possibleTypes": [
],
 "argumentIndex": 1
 }
]"""
```

```
joern> cpg.call.filter(node => node.id==313349).astChildren.toJsonPretty
val res36: String = """[
 {
 "dynamicTypeHintFullName": [
],
 "name": "message",
 "code": "$message",
 "typeFullName": "ANY",
 "order": 1,
 "_label": "IDENTIFIER",
 "lineNumber": 89,
 "id": 313350,
 "possibleTypes": [
],
 "argumentIndex": 1
 },
 {
 "dynamicTypeHintFullName": [
],
 "code": "\n",
 "typeFullName": "string",
 "order": 2,
 "_label": "LITERAL",
 "lineNumber": 89,
 "id": 313351,
 "possibleTypes": [
],
 "argumentIndex": 2
 }
]""""
```

从ast结构能够发现传入的参数受到89行 \$message 变量的影响 (CPG ID: 313350) , 为此, 我们需要分析影响此变量的具体语句, 此时, 我们使用 `_reachingDefIn` 函数进行如下查找:

所用查询语句 (注意: 要用all类型) :

```
cpg.all.filter(node => node.id==313350)._reachingDefIn.toJsonPretty
```

```

joern> cpg.all.filter(node => node.id==313350)._reachingDefIn.toJsonPretty
val res38: String = """[
 {
 "name": "testInstall",
 "astParentFullName": "Smarty_Internal_TestInstall",
 "fullName": "Smarty_Internal_TestInstall::testInstall",
 "signature": "<unresolvedSignature>(2)",
 "astParentType": "TYPE_DECL",
 "_label": "METHOD",
 "code": "PUBLIC STATIC function testInstall($smarty,&$errors)",
 "isExternal": false,
 "lineNumber": 28,
 "id": 313149,
 "order": 2,
 "filename": "include/smarty/libs/sysplugins/smarty_internal_testinstall.php"
 },
 {
 "dynamicTypeHintFullName": [
],
 "name": "message",
 "code": "$message",
 "typeFullName": "ANY",
 "order": 1,
 "_label": "IDENTIFIER",
 "lineNumber": 87,
 "id": 313338,
 "possibleTypes": [
],
 "argumentIndex": 1
 },
 {
 "dynamicTypeHintFullName": [
],
 "code": ".\\n\"",
 "typeFullName": "string",
 "order": 2,
 "_label": "LITERAL",
 "lineNumber": 89,
 "id": 313351,
 "possibleTypes": [
],
 "argumentIndex": 2
 }
]

```

我们需要忽略这些 `_label` 为 `LITERAL` 和 `METHOD` 的节点，继续向上处理其余节点：

```

joern> cpg.all.filter(node => node.id==313338)._reachingDefIn.toJsonPretty
val res58: String = """[
 {
 "name": "encaps",
 "signature": "",
 "code": "\"FAILED: \\" . $template_dir . \\" is not readable\\\"",
 "typeFullName": "string",
 "order": 2,
 "methodFullName": "encaps",
 "_label": "CALL",
 "dynamicTypeHintFullName": [
],
 "dispatchType": "STATIC_DISPATCH",
 "lineNumber": 87,
 "id": 313339,
 "possibleTypes": [
],
 "argumentIndex": 2
 }
]"""

```

```

joern> cpg.call.filter(node => node.id==313339).astChildren.toJsonPretty
val res59: String = """[
 {
 "dynamicTypeHintFullName": [
],
 "code": "\"FAILED: \"",
 "typeFullName": "string",
 "order": 1,
 "_label": "LITERAL",
 "lineNumber": 87,
 "id": 313340,
 "possibleTypes": [
],
 "argumentIndex": 1
 },
 {
 "dynamicTypeHintFullName": [
],
 "name": "template_dir",
 "code": "$template_dir",
 "typeFullName": "ANY",
 "order": 2,
 "_label": "IDENTIFIER",
 "lineNumber": 87,
 "id": 313341, -----^
 "possibleTypes": [
],
 "argumentIndex": 2
 },
 {
 "dynamicTypeHintFullName": [
],
 "code": "\" is not readable\"",
 "typeFullName": "string",
 "order": 3,
 "_label": "LITERAL",
 "lineNumber": 87,
 "id": 313342,
 "possibleTypes": [
],
 "argumentIndex": 2
 }
]

```

```

joern> cpg.all.filter(node => node.id==313341)._reachingDefIn.toJsonPretty
val res1: String = """[
 {
 "name": "testInstall",
 "astParentFullName": "Smarty_Internal_TestInstall",
 "fullName": "Smarty_Internal_TestInstall::testInstall",
 "signature": "<unresolvedSignature>(2)",
 "astParentType": "TYPE_DECL",
 "_label": "METHOD",
 "code": "PUBLIC STATIC function testInstall($smarty,&$errors)",
 "isExternal": false,
 "lineNumber": 28,
 "id": 313149,
 "order": 2,
 "filename": "include/smarty/libs/sysplugins/smarty_internal_testinstall.php"
 },
 {
 "dynamicTypeHintFullName": [
],
 "name": "template_dir",
 "code": "$template_dir",
 "typeFullName": "ANY",
 "order": 1,
 "_label": "IDENTIFIER",
 "lineNumber": 85,
 "id": 313332, -----^
 "possibleTypes": [
],
 "argumentIndex": 1
 }
]

```

```
joern> cpg.all.filter(node => node.id==313332)._reachingDefIn.toJsonPretty
val res2: String = """[
 {
 "name": "testInstall",
 "astParentFullName": "Smarty_Internal_TestInstall",
 "fullName": "Smarty_Internal_TestInstall::testInstall",
 "signature": "<unresolvedSignature>(2)",
 "astParentType": "TYPE_DECL",
 "_label": "METHOD",
 "code": "PUBLIC STATIC function testInstall($smarty,&$errors)",
 "isExternal": false,
 "lineNumber": 28,
 "id": 313149,
 "order": 2,
 "filename": "include/smarty/libs/sysplugins/smarty_internal_testinstall.php"
 },
 {
 "dynamicTypeHintFullName": [
],
 "name": "template_dir",
 "code": "$template_dir",
 "typeFullName": "ANY",
 "order": 1,
 "_label": "IDENTIFIER",
 "lineNumber": 77,
 "id": 313302, ←
 "possibleTypes": [
],
 "argumentIndex": 1
 }
]"""

```

```
joern> cpg.all.filter(node => node.id==313302)._reachingDefIn.toJsonPretty
val res3: String = """[
 {
 "name": "testInstall",
 "astParentFullName": "Smarty_Internal_TestInstall",
 "fullName": "Smarty_Internal_TestInstall::testInstall",
 "signature": "<unresolvedSignature>(2)",
 "astParentType": "TYPE_DECL",
 "_label": "METHOD",
 "code": "PUBLIC STATIC function testInstall($smarty,&$errors)",
 "isExternal": false,
 "lineNumber": 28,
 "id": 313149,
 "order": 2,
 "filename": "include/smarty/libs/sysplugins/smarty_internal_testinstall.php"
 },
 {
 "dynamicTypeHintFullName": [
],
 "name": "template_dir",
 "code": "$template_dir",
 "typeFullName": "ANY",
 "order": 1,
 "_label": "IDENTIFIER",
 "lineNumber": 42,
 "id": 313201, ←
 "possibleTypes": [
],
 "argumentIndex": 1
 }
]"""

```

```

joern> cpg.all.filter(node => node.id==313201)._reachingDefIn.toJsonPretty
val res4: String = """[
 {
 "name": "testInstall",
 "astParentFullName": "Smarty_Internal_TestInstall",
 "fullName": "Smarty_Internal_TestInstall::testInstall",
 "signature": "<unresolvedSignature>(2)",
 "astParentType": "TYPE_DECL",
 "_label": "METHOD",
 "code": "PUBLIC STATIC function testInstall($smarty,&$errors)",
 "isExternal": false,
 "lineNumber": 28,
 "id": 313149,
 "order": 2,
 "filename": "include/smarty/libs/sysplugins/smarty_internal_testinstall.php"
 },
 {
 "dynamicTypeHintFullName": [
],
 "name": "template_dir",
 "code": "$template_dir",
 "typeFullName": "ANY",
 "order": 1,
 "_label": "IDENTIFIER",
 "lineNumber": 40,
 "id": 313196,
 "possibleTypes": [
],
 "argumentIndex": 1
 }
]
"""

```

```

joern> cpg.all.filter(node => node.id==313196)._reachingDefIn.toJsonPretty
val res5: String = """[
 {
 "name": "realpath",
 "signature": "<unresolvedSignature>(1)",
 "code": " realpath($template_dir)",
 "typeFullName": "ANY",
 "order": 2,
 "methodFullName": " realpath",
 "_label": "CALL",
 "dynamicTypeHintFullName": [
],
 "dispatchType": "STATIC_DISPATCH",
 "lineNumber": 40,
 "id": 313197,
 "possibleTypes": [
],
 "argumentIndex": 2
 }
]
"""

```

当到达第40行的  `realpath($template_dir)` (CPG ID:313197) 时，已经没有继续分析的必要了，因为  `realpath` 内置函数返回的内容不可能被污染。此时可以认为89行 `$message` 变量不是污点数据。因此可以避免分析此处。画个图来展示一下具体分析流程：

```

38 foreach ($smarty->getTemplateDir() as $template_dir) {
39 $_template_dir = $template_dir;
40 $template_dir = realpath($template_dir);
41 // resolve include_path or fail existence
42 if (!$template_dir) {
43 if ($smarty->use_include_path && !preg_match('/^([\\\\\\]|[a-zA-Z]:[\\\\\\\\\\])/', $_template_dir)) {
44 try PHP include_path
45 if ($stream_resolve_include_path) {
46 $template_dir = stream_resolve_include_path($_template_dir);
47 } else {
48 $template_dir = $smarty->ext->_getIncludePath->getIncludePath($_template_dir, null, $smarty);
49
50 if ($template_dir === false) {
51 if ($errors === null) {
52 echo "$template_dir is OK.\n";
53 }
54 continue;
55 } else {
56 $status = false;
57 $message =
58 "FAILED: $_template_dir does not exist (and couldn't be found in include_path either)";
59 if ($errors === null) {
60 echo $message . ".\n";
61 } else {
62 $errors['template_dir'] = $message;
63 }
64 continue;
65 }
66 } else {
67 $status = false;
68 $message = "FAILED: $_template_dir does not exist";
69 if ($errors === null) {
70 echo $message . ".\n";
71 } else {
72 $errors['template_dir'] = $message;
73 }
74 continue;
75 }
76 }
77 if (!is_dir($template_dir)) {
78 $status = false;
79 $message = "FAILED: $template_dir is not a directory";
80 if ($errors === null) {
81 echo $message . ".\n";
82 } else {
83 $errors['template_dir'] = $message;
84 }
85 } elseif (!is_readable($template_dir)) {
86 $status = false;
87 $message = "FAILED: $template_dir is not readable";
88 if ($errors === null) {
89 echo $message . ".\n";

```

[- hide -](#) [+ show +](#)



需要注意一个问题：为什么46、48两行没有被查出来呢？我们仔细检查代码发现第54行有一个 continue语句，可以结束后续语句的运行，因此导致46、48两行的定义不能影响89行的 \$message 变量。

## 36. 处理PHP Array变量

源码：

```

<?php
$person = array(
 "name" => "John",
 "age" => 30,
 "city" => "New York"
);
echo $person[0];
$info = array("John", "age", "city", 30); // CPG ID: 31
echo $info[3];
?>

```

对于赋值语句 `$info = array("John", "age", "city", 30);`, 其右侧是一个**array**对象 (和python的字典差不多, 但是更加灵活)。从原始的CPG Node不太好直接获取array的相关信息:

```
joern> cpg.call.filter(node => node.code.contains("$info = ")).toJsonPretty
val rest: String = """[
 {
 "name": "<operator>.assignment",
 "signature": "",
 "code": "$info = ",
 "typeFullName": "ANY",
 "order": 7,
 "methodFullName": "<operator>.assignment",
 "_label": "CALL",
 "dynamicTypeHintFullName": [
],
 "dispatchType": "STATIC_DISPATCH",
 "lineNumber": 8,
 "id": 31,
 "possibleTypes": [
],
 "argumentIndex": -1
 }
]"""
```

(1) 我们首先通过检查其右值 (source) 是否含有BLOCK节点来推断是否可能含有array:

```
cpg.call.filter(node => node.id==31).assignment.source.toJsonPretty
```

```
joern> cpg.call.filter(node => node.id==31).assignment.source.toJsonPretty
val res2: String = """[
 {
 "dynamicTypeHintFullName": [
],
 "code": "<empty>",
 "typeFullName": "ANY",
 "order": 2,
 "_label": "BLOCK", ←
 "lineNumber": 8,
 "id": 33,
 "possibleTypes": [
],
 "argumentIndex": 2
 },
 {
 "dynamicTypeHintFullName": [
],
 "name": "tmp1",
 "code": "$tmp1",
 "typeFullName": "array",
 "order": 5,
 "_label": "IDENTIFIER",
 "lineNumber": 8,
 "id": 54,
 "possibleTypes": [
],
 "argumentIndex": -1
 }
]"""
```

(2) 检查BLOCK节点的 `_cfgIn` 是否为array对象 (检查 `typeFullName` 属性) :

```
cpg.all.filter(node => node.id==33)._cfgIn.toJsonPretty
```

```
joern> cpg.all.filter(node => node.id==33)._cfgIn.toJsonPretty
val res4: String = """[
 {
 "dynamicTypeHintFullName": [
],
 "name": "tmp1",
 "code": "$tmp1",
 "typeFullName": "array",
 "order": 5,
 "_label": "IDENTIFIER",
 "lineNumber": 8,
 "id": 54,
 "possibleTypes": [
],
 "argumentIndex": -1
 }
]"""
```

(3) 访问BLOCK节点的 `AST Children`, 获得所有赋值语句, 对其处理, 并转换为 `PHPArray Stmt`:

```
cpg.all.filter(node => node.id==33)._astOut.toJsonPretty
```

```
joern> cpg.all.filter(node => node.id==33)._astOut.toJsonPretty
val res5: String = """[
 {
 "name": "<operator>.assignment",
 "signature": "",
 "code": "$tmp1[0] = \"John\"",
 "typeFullName": "ANY",
 "order": 1,
 "methodFullName": "<operator>.assignment",
 "_label": "CALL",
 "dynamicTypeHintFullName": [
],
 "dispatchType": "STATIC_DISPATCH",
 "lineNumber": 8,
 "id": 34,
 "possibleTypes": [
],
 "argumentIndex": -1
 },
 {
 "name": "<operator>.assignment",
 "signature": "",
 "code": "$tmp1[1] = \"age\"",
 "typeFullName": "ANY",
 "order": 2,
 "methodFullName": "<operator>.assignment",
 "_label": "CALL",
 "dynamicTypeHintFullName": [
],
 "dispatchType": "STATIC_DISPATCH",
 "lineNumber": 8,
 "id": 39,
 "possibleTypes": [
],
 "argumentIndex": -1
 }
],
```

具体处理逻辑不赘述了，对于两种类型的array（下标是整数、下标是字符串）的处理逻辑是相同的。

## 37.Joern解析PHP时存在的问题

### (1) 处理array问题

Joern对array处理时会发生很多混乱的问题，例如在处理 [piwigo/tools/triggers\\_list.php](#) 时，  
\$core = array(... 中含有许多的array操作，Joern将它们错误识别为了赋值语句的target和source，最终生成的赋值语句的左值、右值都有1247个。

```
<?php
$core = array(
array(
 'name' => 'allow_increment_element_hit_count',
 'type' => 'trigger_change',
 'vars' => array('bool', 'content_not_set'),
 'files' => array('picture.php'),
),
array(
 'name' => 'batch_manager_perform_filters',
 'type' => 'trigger_change',
```

```
'vars' => array('array', 'filter_sets', 'array', 'bulk_manager_filter'),
'files' => array('admin\\batch_manager.php'),
'infos' => 'New in 2.7',
)...
cpg.call.filter(node => node.id==843673).toJsonPretty
```

```
joern> cpg.call.filter(node => node.id==843673).toJsonPretty
val res6: String = """[
 {
 "name": "<operator>.assignment",
 "signature": "",
 "code": "$core = ",
 "typeFullName": "ANY",
 "order": 447,
 "methodFullName": "<operator>.assignment",
 "_label": "CALL",
 "dynamicTypeHintFullName": [

],
 "dispatchType": "STATIC_DISPATCH",
 "lineNumber": 2,
 "id": 843673,
 "possibleTypes": [

],
 "argumentIndex": -1
 }
]"""
```

```
cpg.call.filter(node => node.id==843673).assignment.target.take(2).toJsonPretty
```

```
joern> cpg.call.filter(node => node.id==843673).assignment.target.take(2).toJsonPretty
val res7: String = """[
 {
 "dynamicTypeHintFullName": [

],
 "name": "core",
 "code": "$core",
 "typeFullName": "ANY",
 "order": 1,
 "_label": "IDENTIFIER",
 "lineNumber": 2,
 "id": 843674,
 "possibleTypes": [

],
 "argumentIndex": 1
 },
 {
 "name": "<operator>.indexAccess",
 "signature": "",
 "code": "$tmp0[0]",
 "typeFullName": "ANY",
 "order": 1,
 "methodFullName": "<operator>.indexAccess",
 "_label": "CALL",
 "dynamicTypeHintFullName": [

],
 "dispatchType": "STATIC_DISPATCH",
 "lineNumber": 3,
 "id": 843677,
 "possibleTypes": [

],
 "argumentIndex": 1
 }
]"""
```

日志文件中保存的此赋值语句的String:

```

1507 | LValues:
31374 | LValues[1245]:
31380 | operands:
31391 | |
31392 | |
31393 | | node_type: Literal
31394 | | type: int
31395 | | value: 0
31396 | |
31397 | |
31398 | LValues[1246]:
31399 | |
31400 | | node_type: Operation
31401 | | cpg_id: 850339
31402 | | code: $tmp436['infos']
31403 | | operator: <operator>.indexAccess
31404 | | operands:
31405 | | | operands[0]:
31406 | | | |
31407 | | | | node_type: Variable
31408 | | | | cpg_id: 850340
31409 | | | | code: $tmp436
31410 | | | | type: array
31411 | | | | identifier: $tmp436
31412 | | | | value: None
31413 | | | | signature: <[Variable]: array: $tmp436>
31414 | |
31415 | | operands[1]:
31416 | |
31417 | | | node_type: Literal
31418 | | | type: string
31419 | | | value: infos
31420 | |
31421 | |
31422 | RValue:
31423 | |
31424 | | node_type: Literal
31425 | | type: string
31426 | | value: New in 2.6.2.
31427 | |
31428 | |

```

## (2) 两种filter语句存在差异

源码：

```

class Student {
 String name;
 int age;

 public void hello(Teacher teacher) {
 System.out.println("Hello! teacher " + teacher.name);
 }

 public void hello(Student student) {
 System.out.println("Hello! student " + student.name);
 }
}

```

示例一：

```
cpg.method.filter(node => node.fullName == "Student.hello:void(Teacher)").toJsonPretty
```

```

joern> cpg.method.filter(node => node.fullName == "Student.hello:void(Teacher)").toJsonPretty
val res11: String = """[
 {
 "name": "hello",
 "astParentFullName": "<empty>",
 "fullName": "Student.hello:void(Teacher)",
 "signature": "void(Teacher)",
 "astParentType": "<empty>",
 "lineNumberEnd": 11,
 "_label": "METHOD",
 "columnNumberEnd": 5,
 "code": "public void hello(Teacher teacher)",
 "isExternal": false,
 "lineNumber": 9,
 "id": 54,
 "columnNumber": 5,
 "order": 3,
 "filename": ""
 }
]"""

```

示例二：

```
cpg.method.fullName("Student.hello:void(Teacher)").toJsonPretty
```

```

joern> cpg.method.fullName("Student.hello:void(Teacher)").toJsonPretty
val res12: String = """[

]"""

```

为什么两条语句执行有不同的结果？

下面这两条语句执行也有区别

```
cpg.controlStructure.filter(_.lineNumber == Some(value = 39)).toJsonPretty
cpg.controlStructure.filter(node => node.location.lineNumber == Some(value = 39)).toJsonPretty
```

```

joern> cpg.controlStructure.filter(_.lineNumber == Some(value = 39)).toJsonPretty
val res81: String = """[
 {
 "parserTypeName": "<empty>",
 "code": "if (name == \"liaoshuang\")",
 "columnNumber": 9,
 "order": 26,
 "controlStructureType": "IF",
 "_label": "CONTROL_STRUCTURE",
 "argumentIndex": -1,
 "lineNumber": 39,
 "id": 188
 }
]"""

joern> cpg.controlStructure.filter(node => node.location.lineNumber == Some(value = 39)).toJsonPretty
val res82: String = """[
]"""

```

### (3) 函数调用点获取失败

在Joern的官方文档中给出了callIn这个方法：

Traversal Steps

| Traversals | Description                                       | Example                                |
|------------|---------------------------------------------------|----------------------------------------|
| .call      | All call-sites in the code                        | cpg.call.name.1                        |
| .callOut   | Return the outgoing call-sites for a given method | cpg.method.name("main").callOut.name.1 |
| .callIn    | Return the call-sites of a given method           | cpg.method.name("exit").callIn.code.1  |

在一些简短的代码中测试发现，这个方法可以用来找到函数的调用点。

示例一：

```
cpg.method.filter(node => node.fullName == "Student.hello:void(Teacher)").callIn.toJsonPretty
```

```
joern> cpg.method.filter(node => node.fullName == "Student.hello:void(Teacher)").callIn.toJsonPretty
val res28: String = """[
 {
 "name": "hello",
 "signature": "void(Teacher)",
 "code": "si.hello(ti1)", // 目标代码行
 "typeName": "void",
 "columnNumber": 9,
 "order": 21,
 "methodFullName": "Student.hello:void(Teacher)",
 "label": "CALL",
 "dynamicTypeHintFullName": [
],
 "dispatchType": "DYNAMIC_DISPATCH",
 "lineNumber": 36,
 "id": 175,
 "possibleTypes": [
],
 "argumentIndex": -1
 }
]"""
```

但是在不同文件中存在函数调用时，使用callIn方法就行不通了，例如CVE-2007-1963。

示例二：

```
cpg.method.filter(node => node.method.fullName == "session->init").callIn.toJsonPretty
```

```
joern> cpg.method.filter(node => node.method.fullName == "session->init").callIn.toJsonPretty
val res38: String = """"[
]""""
```

这个函数是存在调用点的，例如：

### (1) archive/global.php

```
global.php x
repos > mybb_mybb > archive > global.php
88
89 // Initialise session
90 $session = new session();
91 $session->init(); // 目标代码行
92
93 if(!$mybb->settings['bblanguage'])
94 {
95 | $mybb->settings['bblanguage'] = "english";
96 }
97 $lang->setLanguage($mybb->settings['bblanguage']);
```

### (2) global.php

```

class_session.php 6 global.php 9+ ×
global.php > ...
29 {
30 | $mybb->trigger_generic_error("install_directory");
31 |
32 | Dennis Tsang, 19年前 • Started on new feature bar
33 //
34 // Create this users session
35 //
36 if(isset($nosession[$mybb->input['action']])){
37 {
38 | define("NO_ONLINE", 1);
39 }
40 require "./inc/class_session.php";
41 $session = new session;
42 $session->init(); ←
43
44 $plugins->run_hooks("global_start");
45

```

joern> cpg.call.filter(node => node.code.contains("\$session->init()")).filter(node => node.location.filename=="global.php").isCall.toJsonPretty

```

val res0: String = """[
{
 "name": "init",
 "signature": "<unresolvedSignature>(0)",
 "code": "$session->init()",
 "typeFullName": "session->init.<returnValue>",
 "order": 65,
 "methodFullName": "session->init",
 "_label": "CALL",
 "dynamicTypeHintFullName": [
 "session->init"
],
 "dispatchType": "DYNAMIC_DISPATCH",
 "lineNumber": 42,
 "id": 57286,
 "possibleTypes": [
 ...
],
 "argumentIndex": -1
}
]"""

```

一些函数是能够找到其函数调用点的，例如CVE-2007-1963中的 `session->getip()` 函数示例三：

```
cpg.method.filter(node => node.fullName == "session->getip").callIn.toJsonPretty
```

joern> cpg.method.filter(node => node.fullName == "session->getip").callIn.toJsonPretty

```

val res37: String = """[
{
 "name": "getip",
 "signature": "<unresolvedSignature>(0)",
 "code": "getip()",
 "typeFullName": "ANY",
 "order": 2,
 "methodFullName": "session->getip",
 "_label": "CALL",
 "dynamicTypeHintFullName": [
 "session->getip"
],
 "dispatchType": "STATIC_DISPATCH",
 "lineNumber": 42,
 "id": 65369,
 "possibleTypes": [
 ...
],
 "argumentIndex": 2
}
]"""

```

**可能的解决方法：**对于找不到`callIn`的函数，检查其是否是类的函数，随后找到所有定义该类的语句，根据类的名称进行搜索，例如 `session->init` 对应的类为 `class session`，那么我们首先找到定义该类的语句，其左值就是对象实例的名称，根据该名称+函数名称（`session->getip`）就能再找到对应的函数调用点了。

现在的临时解决方案：使用**methodFullName**进行过滤，查找所有函数调用点。

```
cpg.call.filter(node => node.methodFullName == "session->init").map(x=> (x.node.id, x.node.code, x.node.location.filename, x.node.location.lineNumber)).toJsonPretty
```

```

joern> cpg.call.filter(node => node.methodFullName == "session->init").map(x=> (x.node.id, x.node.code, x.node.location.filename, x.node.location.lineNumber)).toJsonPretty
val res87: String = """[
 {
 "_1":48753,
 "_2":"$session->init()",
 "_3":"archive/global.php",
 "_4":91
 },
 {
 "_1":57286,
 "_2":"$session->init()",
 "_3":"global.php",
 "_4":42
 }
]"""

```

## (4) global函数被错误识别

在CVE-2007-1963中 inc\functions.php 的 getip() 函数能够获取用户输入，这个函数可以被其它PHP调用：

```

1348 function getip() {
1349 global $_SERVER;
1350 if($_SERVER['HTTP_X_FORWARDED_FOR'])
1351 {
1352 if(preg_match_all("#[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}#s", $_SERVER['HTTP_X_FORWARDED_FOR'], $addresses))
1353 {
1354 while(list($key, $val) = each($addresses[0]))
1355 {
1356 if(!preg_match("#^(10|172\.\d{1,2}|192\.\d{1,2})\.\d{1,3}$", $val))
1357 {
1358 $ip = $val;
1359 break;
1360 }
1361 }
1362 }
1363 }
1364 if(!$ip)
1365 {
1366 if($_SERVER['HTTP_CLIENT_IP'])
1367 {
1368 $ip = $_SERVER['HTTP_CLIENT_IP'];
1369 }
1370 else
1371 {
1372 $ip = $_SERVER['REMOTE_ADDR'];
1373 }
1374 }
1375 return $ip;
1376 }

```

通过 callIn 接口或者 methodFullName 过滤两种方式，都能得到它的调用点。

```

joern> cpg.call.filter(node => node.methodFullName == "getip").map(x=> (x.node.id, x.node.code, x.node.location.filename, x.node.location.lineNumber)).toJsonPretty
val res88: String = """[
 {
 "_1":2526,
 "_2":"getip()",
 "_3":"admin/adminfunctions.php",
 "_4":711
 },
 {
 "_1":16541,
 "_2":"getip()",
 "_3":"admin/global.php",
 "_4":55
 },
 {
 "_1":16778,
 "_2":"getip()",
 "_3":"admin/global.php",
 "_4":110
 }
]"""

```

但是，在类的函数中如果调用了此函数，这两种方法都不能找到对应的调用点，例如在 inc\class\_session.php 中就存在这样一个调用点：

```

36 function init()
37 {
38 global $ipaddress, $db, $mybb, $noonline;
39 //
40 // Get our visitors IP
41 //
42 $this->ipaddress = $ipaddress = getip();
43 //
44 //
45 // User-agent
46 //
47 $this->useragent = $_SERVER['HTTP_USER_AGENT'];
48 if(strlen($this->useragent) > 100)
49 {
50 $this->useragent = substr($this->useragent, 0, 100);
51 }
52

```

我们检查这个函数，发现其CPG节点：

```

jоern> cpg.call.filter(node => node.id==65367).assignment.source.toJsonPretty
val res91: String = """[
{
 "name": "getip",
 "signature": "<unresolvedSignature>(0)",
 "code": "getip()",
 "typeFullName": "ANY",
 "order": 2,
 "methodFullName": "session->getip",
 "_label": "CALL",
 "dynamicTypeHintFullName": [
 "session->getip"
],
 "dispatchType": "STATIC_DISPATCH",
 "lineNumber": 42,
 "id": 65369,
 "possibleTypes": [
],
 "argumentIndex": 2
}
]"""

```

其 “methodFullName” 居然变为了 “session->getip”，但检查并未在 **session** 类中发现它自己独立实现了这个方法。通过 “session->getip” 是可以查找到这个调用点的。

```

jоern> ccpg.call.filter(node => node.methodFullName == "session->getip").map(x=> (x.node.id, x.node.code, x.node.location.filename, x.node.location.lineNumber)).toJsonPretty
val res94: String = """[
{
 "_1": 65369,
 "_2": "getip()",
 "_3": "inc/class_session.php",
 "_4": 42
}
]"""

```

比较 “session->getip” 和 “getip” 的CPG节点：

```
joern> cpg.method.filter(node => node.fullName == "session->getip").toJsonPretty
val res92: String = """[
 {
 "name": "getip",
 "astParentFullName": "<global>",
 "fullName": "session->getip",
 "signature": "<unresolvedSignature>(0)",
 "astParentType": "NAMESPACE_BLOCK",
 "_label": "METHOD",
 "code": "<empty>",
 "isExternal": true,
 "id": 168599,
 "order": 0,
 "filename": "<empty>"
 }
]"""
```

```
joern> cpg.method.filter(node => node.fullName == "getip").toJsonPretty
val res93: String = """[
 {
 "name": "getip",
 "astParentFullName": "inc/functions.php:<global>",
 "fullName": "getip",
 "signature": "<unresolvedSignature>(0)",
 "astParentType": "METHOD",
 "_label": "METHOD",
 "code": "function getip()",
 "isExternal": false,
 "lineNumber": 1345,
 "id": 73549,
 "order": 47,
 "filename": "inc/functions.php"
 }
]"""
```

这两个节点的相同点很明显，它们有着同样的**name**属性，差异之处有许多。另外，这两个函数的内容不一致！

```
joern> cpg.method.filter(node => node.fullName == "session->getip")._cfgOut.toJsonPretty
val res97: String = """[
 {
 "dynamicTypeHintFullName": [
],
 "evaluationStrategy": "BY_VALUE",
 "code": "RET",
 "typeFullName": "ANY",
 "order": 2,
 "_label": "METHOD_RETURN",
 "possibleTypes": [
],
 "id": 168601
 }
]"""

joern> cpg.method.filter(node => node.fullName == "getip")._cfgOut.toJsonPretty
val res98: String = """[
 {
 "dynamicTypeHintFullName": [
],
 "name": "__SERVER",
 "code": "$__SERVER",
 "typeFullName": "ANY",
 "order": 1,
 "_label": "IDENTIFIER",
 "lineNumber": 1346,
 "id": 73552,
 "possibleTypes": [
],
 "argumentIndex": 1
 }
]"""
```

可以看到，由于 “`session->getip`” 是一个External函数， “`getip`” 是一个非External函数，“`session->getip`” 内部是没有具体的函数内容的，而 “`getip`” 却有函数内部的内容，接着往下找 `_cfgOut` 可以看到其余内容：

```
joern> cpg.all.filter(node => node.id==168601)._cfgOut.toJsonPretty
val res99: String = """[
]"""

joern> cpg.all.filter(node => node.id==73552)._cfgOut.toJsonPretty
val res100: String = """[
{
 "name": "global",
 "signature": "",
 "code": "global $_SERVER",
 "typeFullName": "void",
 "order": 6,
 "methodFullName": "global",
 "_label": "CALL",
 "dynamicTypeHintFullName": [
 "global"
],
 "dispatchType": "STATIC_DISPATCH",
 "lineNumber": 1346,
 "id": 73551,
 "possibleTypes": [
],
 "argumentIndex": -1
}
]"""
```

## (5) 部分文件解析失败

解析domainmod项目（<https://github.com/domainmod/domainmod>）中的 `/home/devdata/repos/domainmod_domainmod/install/go.php` 时，这个文件内容不能获取到，猜测：似乎是Joern在解析整个项目时，未能成功处理此文件，导致它未被加入到CPG中。

```
joern> cpg.call.filter(node => node.location.filename.contains("go.php")).toJsonPretty
val res15: String = """[
]"""
```

## (6) 魔术变量不当处理

Joern将PHP的**魔术变量** `_DIR_`、`_FILE_`、`_METHOD_` 等信息建模为了global变量，解析时容易和 `define` 定义的全局变量弄混，建议对魔术变量单独建模。

## (7) namespace未建模

使用Joern查找代码中含有 `namespace` 的语句时会失败，似乎没有将这些信息记录在CPG中。

源码：

```
<?php
namespace MyNamespace;
echo __NAMESPACE__;
?>
```

查询语句：

```
cpg.call.filter(node => node.code.contains("namespcae= ")).toJsonPretty
```

```
joern> cpg.call.filter(node => node.code.contains("namespcae= ")).toJsonPretty
val res1: String = """[
]"""
```

## (8) 赋值语句解析有待改进

源码：

```
<?php
$a = $b = $c = array(// CPG ID: 7
 "a1" => "111",
 "a2" => "222"
);
echo $a["a1"];
?>
```

对于上面这段代码中的赋值语句，Joern能够将右值的类型赋给左值，但是似乎只赋了最右侧的左值（`$c`）：

```

joern> cpg.call.filter(node => node.id==7).assignment.target.toJsonPretty
val res12: String = """[
 {
 "dynamicTypeHintFullName": [
],
 "name": "a",
 "code": "$a",
 "typeFullName": "ANY",
 "order": 1,
 "_label": "IDENTIFIER",
 "lineNumber": 2,
 "id": 8,
 "possibleTypes": [
],
 "argumentIndex": 1
 },
 {
 "dynamicTypeHintFullName": [
],
 "name": "b",
 "code": "$b",
 "typeFullName": "ANY",
 "order": 1,
 "_label": "IDENTIFIER",
 "lineNumber": 2,
 "id": 10,
 "possibleTypes": [
],
 "argumentIndex": 1
 },
 {
 "dynamicTypeHintFullName": [
],
 "name": "c",
 "code": "$c",
 "typeFullName": "array",
 "order": 1,
 "_label": "IDENTIFIER",
 "lineNumber": 2,
 "id": 12,
 "possibleTypes": [
],
 "argumentIndex": 1
 }
]
]
]
}

```

可以看到 \$c 的类型已经被赋为了array，但是 \$a、\$b 的类型仍然没有改变。

## (9) 查询次数过多导致Joern服务崩溃

```

$chars['in'] = chr(128).chr(131).chr(138).chr(142).chr(154).chr(158)
 .chr(159).chr(162).chr(165).chr(181).chr(192).chr(193).chr(194)
 .chr(195).chr(196).chr(197).chr(199).chr(200).chr(201).chr(202)
 .chr(203).chr(204).chr(205).chr(206).chr(207).chr(209).chr(210)
 .chr(211).chr(212).chr(213).chr(214).chr(216).chr(217).chr(218)
 .chr(219).chr(220).chr(221).chr(224).chr(225).chr(226).chr(227)
 .chr(228).chr(229).chr(231).chr(232).chr(233).chr(234).chr(235)
 .chr(236).chr(237).chr(238).chr(239).chr(241).chr(242).chr(243)
 .chr(244).chr(245).chr(246).chr(248).chr(249).chr(250).chr(251)
 .chr(252).chr(253).chr(255);

```

使用joern.py时，在处理到这种操作数非常多的操作时常常会发生崩溃。

暂时还没有大规模测试这种崩溃是不是由于处理过多操作数引起的，但是在重复测试中发现：Joern查询次数过多时必然会发生错误，基本查询2600次以上时就会出错（多次测试均出现此问题）：

The screenshot shows a terminal session on SSH (SSH: 61.170.32.3) displaying a series of failed database queries. The output shows numerous repeated error messages indicating failed queries for various identifiers, such as 'res2615', 'res2616', 'res2617', 'res2618', and 'res2619'. Each query is associated with a timestamp (e.g., 145.20, 145.27, 145.33), a class ('dict'), success status ('success'), UUID ('uuid'), and stdout ('stdout'). The queries are identical and involve a 'val' declaration. The right pane of the terminal shows a process named 'python3 static\_an...'. The bottom of the terminal displays the command 'python 3.10.12 (venv: ver)'.

This screenshot shows a second terminal session on SSH (SSH: 61.170.32.3) with similar failed database queries. The errors are labeled 'res2880', 'res2882', 'res2883', 'res2884', 'res2885', and 'res2886'. Each query is timestamped and includes details like 'typeFullname', 'code', 'order', 'label', 'id', and 'possiblities'. The right pane shows a 'python3 static\_an...' process, and the bottom of the terminal shows the command 'python 3.10.12 (venv: ver)'.

## (10) 奇怪的define类型

本问题可以说并不是Joern存在的问题，只是一种设计的思想。在CVE-2018-11404的`./assets/edit/ssl-provider-account.php`的第235行往下的代码如下：

```
235 ?>
236 <?php require_once DIR_INC . '/doctype.inc.php'; ?>
237 <html>
238 <head>
239 <title><?php echo $layout->pageTitle($page_title); ?></title>
240 <?php require_once DIR_INC . '/layout/head-tags.inc.php'; ?>
241 </head>
242 <body class="hold-transition sidebar-mini layout-fixed text-sm select2-red<?php echo $layout->bodyDarkMode(); ?>">
243 <?php require_once DIR_INC . '/layout/header.inc.php'; ?>
244 <?php
245 echo $form->showFormTop('');
```

第236行中的`DIR_INC`是一个宏定义的变量，直观感受是，在CPG中这个变量的类型可能是“IDENTIFIER”或“LITERAL”，但是这个节点的类型却是`<operator>.fieldAccess`，这就导致在实际处理此变量时可能会出现问题。后续发现所有全局变量均被处理为`<operator>.fieldAccess`类型，应该是Joern有意设计的。

```
joern> cpg.all.filter(node => node.id==43373).cfgOut.toJsonPretty
val res4: String = """[
 {
 "name": "<operator>.fieldAccess",
 "signature": "",
 "code": "DIR_INC",
 "typeFullName": "ANY",
 "order": 1,
 "methodFullName": "<operator>.fieldAccess",
 "_label": "CALL",
 "dynamicTypeHintFullName": [
],
 "dispatchType": "STATIC_DISPATCH",
 "lineNumber": 229,
 "id": 43371,
 "possibleTypes": [
],
 "argumentIndex": 1
 }
]""""
```