



平安安全应急响应中心
PINGAN Security Response Center



金融壹账通
ONECONNECT

金融安全 中的攻防对抗

平安SRC线上沙龙系列主题活动 第一期

FINANCE SECURITY
ATTACK AND DEFEND

CodeQL 漏洞挖掘分享

pandaos

个人介绍

- 个人 ID : PandaOS / 无名侠
- 看雪精华 10+
- Syclover & Nu1L 战队
- 研究方向: 二进制
- Blog: panda0s.top
- wx: chenqqcom162

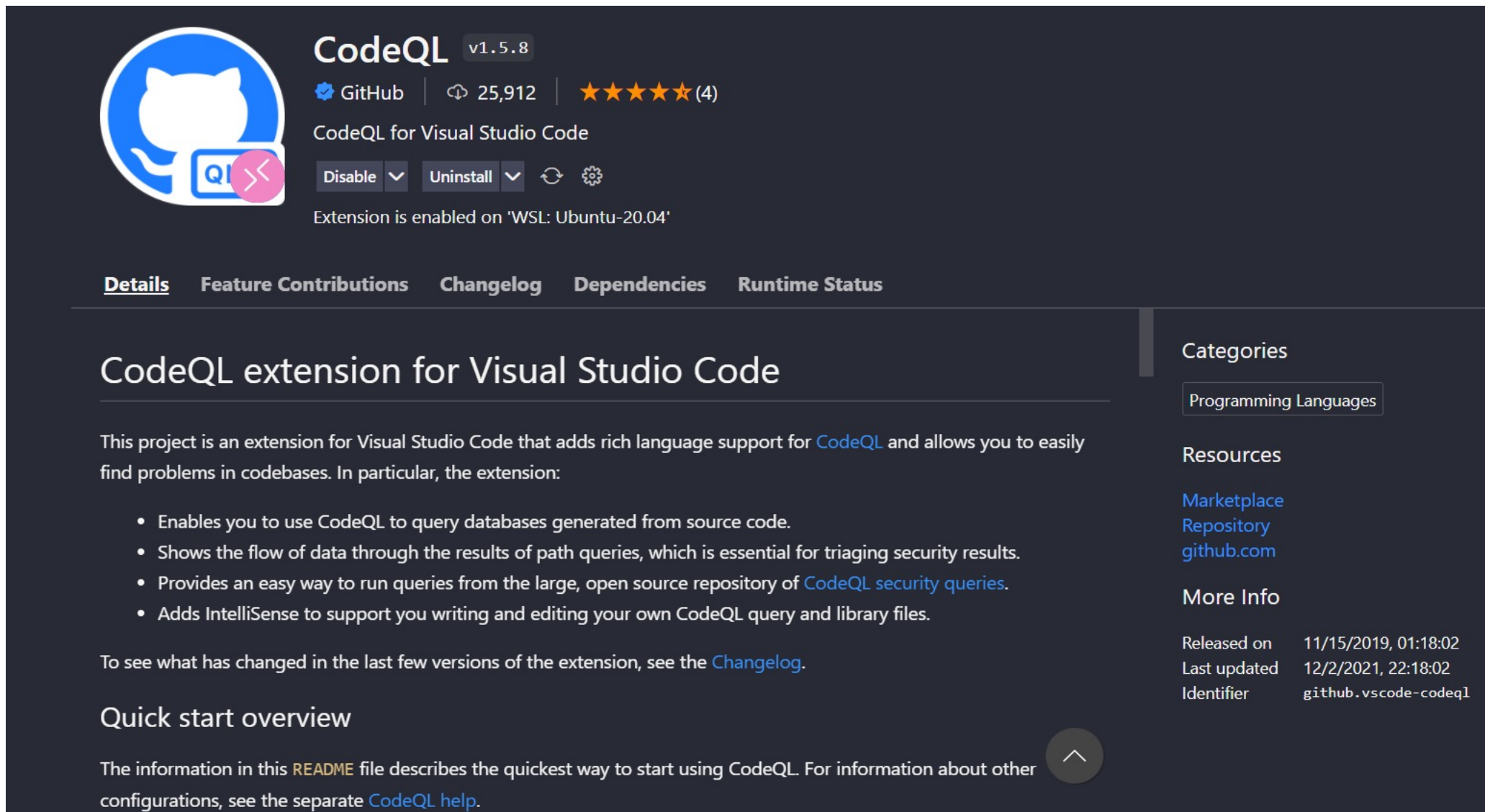
CodeQL 是什么？

- 一种代码审计工具
- 主要思想是将代码转换成数据库，利用类似 SQL 语言的查询语句分析代码是否存在漏洞

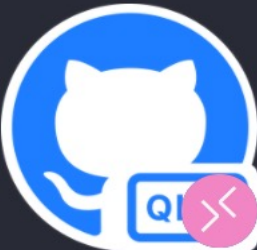


CodeQL 关键组件

- QL : CodeQL 使用的查询语言
- CLI : 运行查询的命令行工具
- Libraries : CodeQL 的库
- Databases: 由代码生成的数据库

CodeQL VSCode 插件



The image shows the CodeQL extension page in the Visual Studio Code marketplace. At the top, there's a header with the extension name 'CodeQL' and version 'v1.5.8'. Below this, it shows the GitHub logo, a star rating of 4 stars, and the number of installations (25,912). The description states 'CodeQL for Visual Studio Code'. There are buttons for 'Disable', 'Uninstall', and a settings gear. A status message says 'Extension is enabled on 'WSL: Ubuntu-20.04''. Below the header, there are tabs for 'Details', 'Feature Contributions', 'Changelog', 'Dependencies', and 'Runtime Status'. The 'Details' tab is selected, showing the title 'CodeQL extension for Visual Studio Code'. The description explains that the project is an extension for Visual Studio Code that adds rich language support for CodeQL and allows users to easily find problems in codebases. It lists four features: enabling CodeQL for querying databases, showing data flow through path queries, providing an easy way to run queries from the CodeQL security queries repository, and adding IntelliSense for writing and editing CodeQL queries and library files. There's a link to the 'Changelog' to see what has changed in the last few versions. A 'Quick start overview' section follows, mentioning the 'README' file for the quickest way to start using CodeQL and a link to 'CodeQL help' for other configurations. On the right side, there are sections for 'Categories' (Programming Languages), 'Resources' (Marketplace, Repository, github.com), and 'More Info' (Released on, Last updated, Identifier).

**CodeQL** v1.5.8
GitHub | 25,912 | ★★★★★ (4)
CodeQL for Visual Studio Code
Disable | Uninstall |  
Extension is enabled on 'WSL: Ubuntu-20.04'

Details | Feature Contributions | Changelog | Dependencies | Runtime Status

CodeQL extension for Visual Studio Code

This project is an extension for Visual Studio Code that adds rich language support for [CodeQL](#) and allows you to easily find problems in codebases. In particular, the extension:

- Enables you to use CodeQL to query databases generated from source code.
- Shows the flow of data through the results of path queries, which is essential for triaging security results.
- Provides an easy way to run queries from the large, open source repository of [CodeQL security queries](#).
- Adds IntelliSense to support you writing and editing your own CodeQL query and library files.

To see what has changed in the last few versions of the extension, see the [Changelog](#).

Quick start overview

The information in this [README](#) file describes the quickest way to start using CodeQL. For information about other configurations, see the separate [CodeQL help](#).

Categories

Programming Languages

Resources

[Marketplace](#)
[Repository](#)
[github.com](#)

More Info

Released on	11/15/2019, 01:18:02
Last updated	12/2/2021, 22:18:02
Identifier	github.vscode-codeql

QL 基础语法

- QL 是类似 SQL 的一种查询语言

```
from /* ... variable declarations ... */  
where /* ... logical formulas ... */  
select /* ... expressions ... */
```

- 例子

```
from int x, int y  
where x = 6 and y = 7  
select x * y
```

Tips : QL 中的变量其实代表的是满足对应类型的集合

QL 基础语法

QL 是一种强类型的语言，每个变量都必须指定类型。

QL 中的 Type 代表满足某种性质的值的集合

Types

- Primitive types: int、float、boolean、string、date
- Classes

QL 基础语法

- 例子 —— 输出10以内的勾股数

```
from int x, int y, int z
where x in [1..10] and y in [1..10] and z in [1..10] and
|      | x*x + y*y = z*z
select x, y, z
```

- 例子 —— 输出 python 代码中参数个数大于 7 的函数

```
import python
from Function f
where count(f.getAnArg()) > 7
select f
```

QL 基础语法

- 谓词 (predicate) 用来表达逻辑关系

```
predicate isSouthern(Person p) {  
    p.getLocation() = "south"  
}  
  
from Person p  
where isSouthern(p)  
select p
```

QL 基础语法

- class 在 QL 中代表了一个逻辑性质，若一个值满足该性质，则为该 class 的一个成员

```
class SmallInt extends int {  
    SmallInt() { this in [1..10] }  
    int square() { result = this*this }  
}  
  
from SmallInt x, SmallInt y, SmallInt z  
where x.square() + y.square() = z.square()  
select x, y, z
```

来点实战(Golang)

目标

MinIO 是亚马逊开发的对象存储库，安全研究人员在2020年发现 MinIO 的管理 API 中的身份验证绕过漏洞。

validateAdminSignature 函数中，由于没有正确处理 isReqAuthenticated 返回值导致身份验证绕过。（鉴权失败后缺少 return 语句）

```
137      137          s3Err = isReqAuthenticated(ctx, r, region, serviceS3)
138      138      }
139      139      if s3Err != ErrNone {
140      140          reqInfo := (&logger.ReqInfo{}).AppendTags("requestHeaders", dumpRequest(r))
141      141          ctx := logger.SetReqInfo(ctx, reqInfo)
142      142          logger.LogIf(ctx, errors.New(getAPIError(s3Err).Description), logger.Application)
143      143      +      return cred, nil, owner, s3Err
144      144      }
145      145      claims, s3Err := checkClaimsFromToken(r, cred)
```

[commit](#)

Step1 查找所有 ErrNone 的引用

ErrNode 是一种标识符

```
from Ident x
where x.getName() = "ErrNone"
select x
```

Ident 是 Golang 分析包中的类，该类表示所有标识符的集合。

The screenshot shows a Golang code editor on the left and a table of results on the right. A red arrow points from the 'ErrNone' value in the code to the first row of the table.

Code snippet:

```
guessIsBrowserReq(r))
```



```
; s3Error != ErrNone {
q(r))
```

Table of results:

#	
1	ErrNone
2	ErrNone
3	ErrNone
4	ErrNone
5	ErrNone
6	ErrNone
7	ErrNone
8	ErrNone
9	ErrNone
10	ErrNone
11	ErrNone
12	ErrNone
13	ErrNone
14	ErrNone
15	ErrNone

Step2 查找条件判断 != ErrNone

```
from EqualityTestExpr e, Ident idt
where e.getAChild*() = idt
      and idt.getName() = "ErrNone"
select e
```

条件测试表达式中的子表达式存在标识符，且该标识符的名称是 ErrNone

```
., guessIsBrowserReq(r))
```

```
"; s3Error != ErrNone {
Req(r))
```

#	
1	...
2	...
3	...
4	...
5	...
6	...
7	...
8	...
9	...
10	...
11	...
12	...
13	...
14	...
15	...
16	...
17	...

Step3 查找包含上一步的测试条件的 if 语句

```
from IfStmt ifstatm, EqualityTestExpr e, Ident idt
where ifstatm.getAChildExpr() = e
      and e.getAChild *() = idt
      and idt.getName() = "ErrNone"
select ifstatm
```

```
43         return cred, nil, owner, s3Err
44     }
45
46     claims, s3Err := checkClaimsFromToken(r, cred)
47     if s3Err != ErrNone {
48         return cred, nil, owner, s3Err
49     }
50
51     return cred, claims, owner, ErrNone
52 }
53
54 // checkAdminRequestAuthType checks whether the request is a vali
```

Step4

查找 return 语句（漏洞是由于缺少 return 语句造成）

```
from ReturnStmt rs
select rs
```

查找不含 return 语句的 if-block

```
from IfStmt is
where not (is.getThen().getASmt() instanceof ReturnStmt)
select is

206     for _, nerr := range globalNotificationSys.SignalService(serv
207     if nerr.Err != nil {
208         logger.GetReqInfo(ctx).SetTags("peerAddress", nerr.Ho
209         logger.LogIf(ctx, nerr.Err)
210     }
211 }
```

Step5

结合前面的步骤，查找包含 ErrNone 的 if 且 if-then 没有 return 语句

```
from IfStmt is, Ident idt, EqualityTestExpr et
where not(is.getThen().getASmt() instanceof ReturnStmt)
    and is.getAChildExpr() = et
    and et.getAChild() = idt
    and idt.getName() = "ErrNone"
select is, et.getParent()
```

存在很多误报

```
respBytes, errCode := globalAllHealState.PopHealStatusJSON(
    healPath, hip.clientToken)
if errCode != ErrNone {
    writeErrorResponseJSON(ctx, w, errorCodes.ToAPIErr(errCode), r.URL)
} else {
    writeSuccessResponseJSON(w, respBytes)
}
return
```

查找对 isReqAuthenticated 函数返回值的条件测试

#	is	[1]
	if statement	if statement
	if statement	if statement
	if statement	if statement
	if statement	if statement
	if statement	if statement
	if statement	if statement
7	if statement	if statement

数据流分析、污点分析

数据流分析

1. 计算变量在不同程序点可能的值
2. 分析变量的值在程序中如何传播
3. 分析变量的值在什么地方被使用

Source：污点源、数据源

Sink：敏感点

利用数据流分析可以判断 Source 数据是否影响（传播）到 Sink

CodeQL 做污点分析或数据流分析需要继承 Configuration 类，并实现 isSource 和 isSink 谓词

Step6 数据流分析

查找对 `isReqAuthenticated` 函数返回值的条件测试

```
class AuthTestConfig extends DataFlow::Configuration {  
  AuthTestConfig() {  
    this = "auth-test-config"  
  }  
  
  // 定义 Source 为 isReqAuthenticated 的返回值  
  override predicate isSource(DataFlow::Node source) {  
    source = any(DataFlow::CallNode cn |  
      cn.getTarget().hasQualifiedName("github.com/minio/minio/cmd", "isReqAuthenticated")  
    ).getResult()  
  }  
  
  // 定义 Sink 为条件测试中的子表达式  
  override predicate isSink(DataFlow::Node sink) {  
    sink = any(  
      DataFlow::EqualityTestNode n).getAnOperand()  
    )  
  }  
}
```


Step6 数据流分析

查询结果例子

```
l17      case userTypeSigned:
l18          s3Err := isReqAuthenticated(ctx, r, globalServerRegion, serviceSTS)
l19          if STSErrorCode(s3Err) != ErrSTSNone {
l20              return user, STSErrorCode(s3Err)
l21          }
l22          var owner bool
```

Step7 结合前面所有步骤

结合前面的步骤，我们可以做出准确的查询：

1. 用数据流分析找出 isReqAuthenticated 影响的条件测试
2. 判断条件测试子表达式里是否存在 ErrNone 标识符
3. 判断条件测试的父表达式是否为 if-stmt
4. 判断 if-stmt-then 代码块是否存在 return

同时满足上述所有条件的代码片段存在身份绕过漏洞

```
from AuthTestConfig config, DataFlow::Node sink, DataFlow::EqualityTestNode comparsion
where config.hasFlow(_, sink) and comparsion.getAnOperand() = sink
and comparsion.asExpr().getAChildExpr().(Ident).getName() = "ErrNone"
and comparsion.asExpr().getParent() instanceof IfStmt
and not comparsion.asExpr().getParent().(IfStmt).getThen().getAChildStmt() instanceof ReturnStmt
select comparsion
```

Step7 结合前面所有步骤

最后我们写的 QL 代码只查询出一个漏洞

#select ▼	1 result
#	comparsion
1	...!=...

```
// we only support V4 (no presign) with auth body
s3Err = isReqAuthenticated(ctx, r, region, serviceS3)
}
if s3Err != ErrNone {
    reqInfo := (&logger.ReqInfo{}).AppendTags("requestHeaders", dumpRequest(r))
    ctx := logger.SetReqInfo(ctx, reqInfo)
    logger.LogIf(ctx, errors.New(getAPIError(s3Err).Description), logger.Applic
}
```

参考来源

- [go-and-dont-return](#)
- [about-data-flow-analysis](#)
- [codeql-overview](#)