



京麒
网络安全大会



以安全研究论开放式程序分析

母浩文@meizjm3i



自我介绍

母浩文

- 寻臻科技创始人&CEO
- Nu1L半退役选手
- BlackHat USA、BlackHat Europe 演讲者
- 多年软件供应链安全研究、漏洞攻防及挖掘经验



寻找小伙伴：安全研究、静态分析、编译器

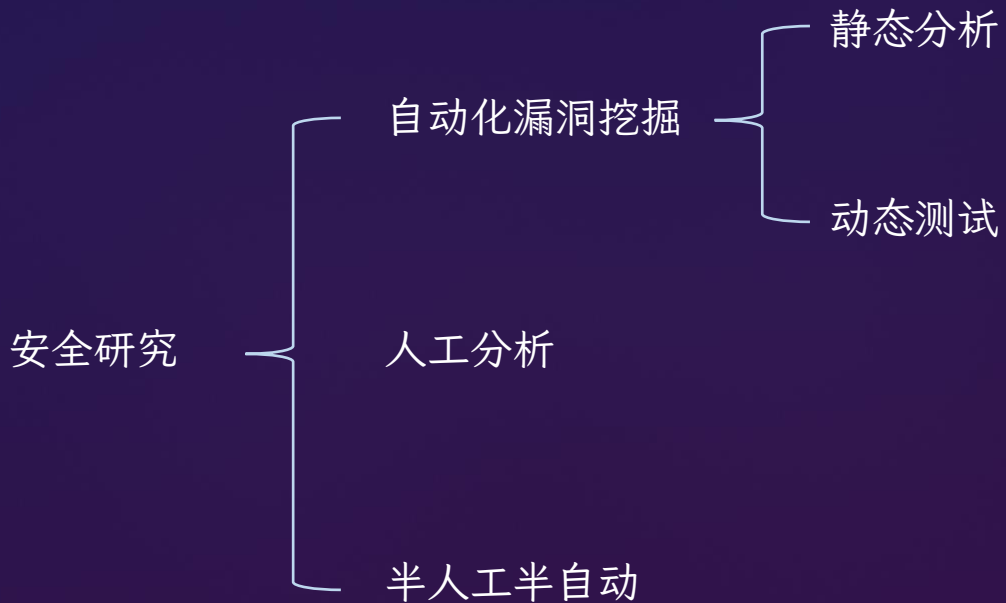


目录

- 开放式程序分析与Datalog
- 安全研究与Datalog
- Change It !



开放式程序分析与Datalog





开放式程序分析与Datalog

静态程序分析的经典流程样例图

程序源代码 → 抽象语法树 → 中间表示 → 静态分析 → 输出结果

静态程序分析的经典流程样例图

程序源代码 → 抽象语法树 → 中间表示 → 静态分析 → 输出结果

Call Graph
Control Flow Graph
IFDS/IDE
Inter Control Flow Graph
Program Dependency Graph
Pointer Analysis
Taint Tracking

.....

对于使用者：纯黑盒



开放式程序分析与Datalog

开放式.....程序分析？



开放式程序分析与Datalog

开放式 程序分析



核心分析能力开放



开放式程序分析与Datalog

开放式 程序分析

核心分析能力开放

与规则定义SDK的区别：

能直接修改分析算法的核心逻辑，不只是调用接口



开放式程序分析与Datalog

开放式 程序分析

是新东西吗？



开放式程序分析与Datalog

开放式 程序分析

是新东西吗？

NO!



开放式程序分析与Datalog

开放式 程序分析

CodeQL

Doop

Joern

Vandal

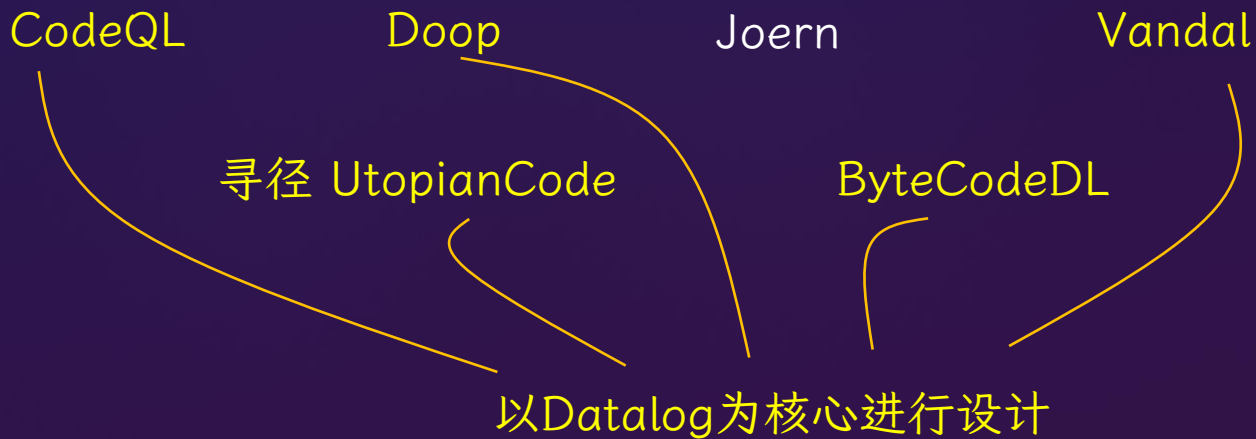
寻径 UtopianCode

ByteCodeDL



开放式程序分析与Datalog

开放式 程序分析





开放式程序分析与Datalog

什么是Datalog: 一种声明式的逻辑编程语言[规范]

数据库内核 ➡ Prolog(Programming in Logic) ➡ Datalog



开放式程序分析与Datalog

什么是Datalog: 一种声明式的逻辑编程语言[规范]

EDB: 外部预置数据



推理引擎: 通过规则对EDB数据做推理

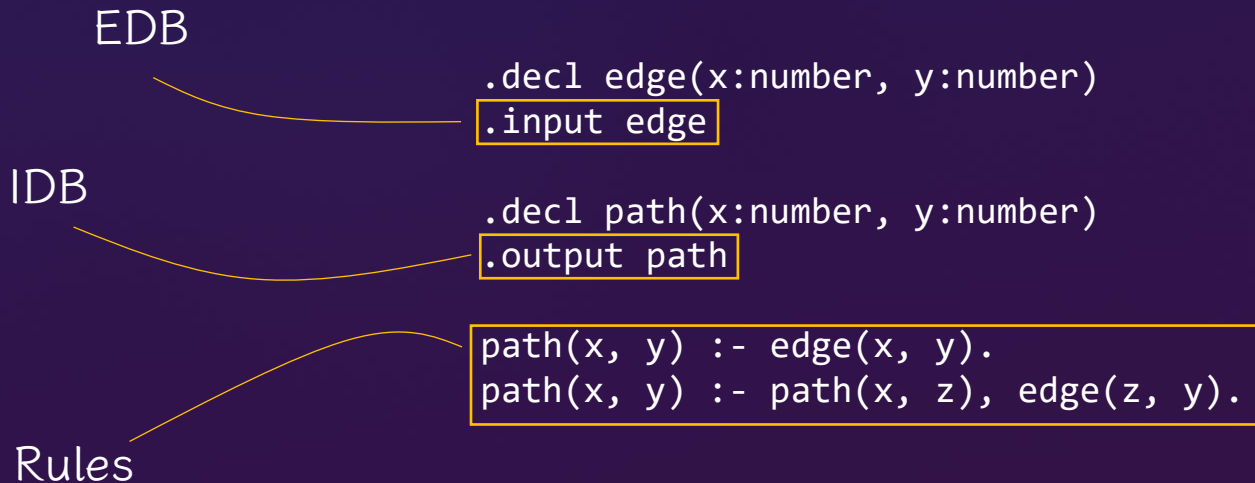


IDB: 内部推导所得数据



Rules: datalog推理规则

什么是Datalog: 一种声明式的逻辑编程语言[规范]





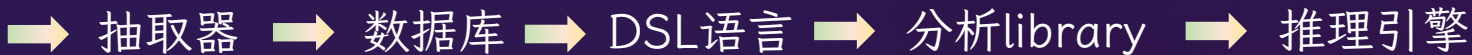
开放式程序分析与Datalog

Datalog推理核心流程

源代码

字节码

.....





开放式程序分析与Datalog

Datalog推理核心流程



CodeQL: 需要通过编译过程生成数据库

Doop: 无编译生成数据库



开放式程序分析与Datalog

Datalog推理核心流程



CodeQL: 需要通过编译过程生成数据库



Datalog可以与编译脱钩

Doop: 无编译生成数据库



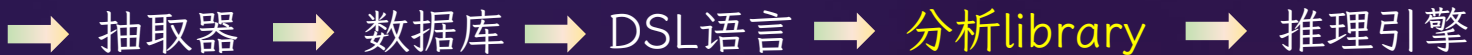
开放式程序分析与Datalog

Datalog推理核心流程

源代码

字节码

.....



核心静态分析算法的构建



开放式程序分析与Datalog

Datalog推理核心流程

源代码

字节码

.....



核心静态分析算法的构建：一切皆rules



开放式程序分析与Datalog

Datalog推理核心流程

源代码

字节码

.....

➡ 抽取器 ➡ 数据库 ➡ DSL语言 ➡ 分析library ➡ 推理引擎

核心静态分析算法的构建：一切皆rules

```
import semmle.code.java.dataflow.TaintTracking2
```

CodeQL: 调用污点分析模块

```
#include "souffle-logic/analyses/context-insensitive/analysis.dl"
```

Doop: 调用上下文不敏感分析模块



安全研究与Datalog

一个核心问题



安全研究与Datalog

一个核心问题

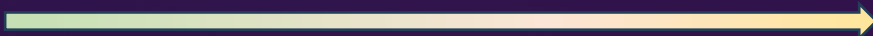
从工具的角度看，什么对安全研究是重要的？



一个核心问题

从工具的角度看，什么对安全研究是重要的？

扩展性 自由度 准确率 速度



重要性依次递减



安全研究与Datalog

一个基本事实



安全研究与Datalog

一个基本事实

现有的工具，默认配置对于安全对抗都是不足的



京麒
网络安全大会



安全研究与Datalog

一个基本事实

现有的工具，默认配置对于安全对抗都是不足的

BurpSuite

IDA

CodeQL

AFL++

Joern

Z3



一个基本事实

现有的工具，默认配置对于安全对抗都是不足的

BurpSuite

IDA

CodeQL

AFL++

Joern

Z3

均需要通过大量的定制化设计以满足真实场景



安全研究与Datalog

以Spring Web应用程序的自动化漏洞挖掘为例

静态分析 → 识别可达方法 → 分析数据流向 → 得到分析结果

以Spring Web应用程序的自动化漏洞挖掘为例

静态分析 → 识别可达方法 → 分析数据流向 → 得到分析结果

```
@Controller
public class HelloController {
    @RequestMapping("/index")
    public String index(User user) {
        return user.toString();
    }
}
```

POJO

```
User {
    name: String -> getter setter
    address: Address -> getter setter
}

Address {
    country: String -> getter setter
    city: String -> getter setter
}
```



安全研究与Datalog

以Spring Web应用程序的自动化漏洞挖掘为例

静态分析 → 识别可达方法 → 分析数据流向 → 得到分析结果

```
@Controller
public class HelloController {
    @RequestMapping("/index")
    public String index(User user) {
        return user.toString();
    }
}
```

POST:


name=liu&address.country=china

以Spring Web应用程序的自动化漏洞挖掘为例

静态分析 → 识别可达方法 → 分析数据流向 → 得到分析结果

```
@Controller
public class HelloController {
    @RequestMapping("/index")
    public String index(User user) {
        return user.toString();
    }
}
```

方法调用栈:



User.getAddress()
Address.getCountry()
Address.setCountry()

以Spring Web应用程序的自动化漏洞挖掘为例

静态分析 → 识别可达方法 → 分析数据流向 → 得到分析结果

```
@Controller
public class HelloController {
    @RequestMapping("/index")
    public String index(User user) {
        return user.toString();
    }
}
```

方法调用栈:

User.getAddress()

Address.getCountry()

Address.setCountry(): 隐式可达

在绝大多数静态分析工具都会忽略该调用

以Spring Web应用程序的自动化漏洞挖掘为例

静态分析 → 识别可达方法 → 分析数据流向 → 得到分析结果

开放式程序分析 + 特定场景快速建模 = 隐式方法调用



反射调用
数据容器
动态注入
动态注入



安全研究与Datalog

以Spring Web应用程序的自动化漏洞挖掘为例

静态分析 → 识别可达方法 → 分析数据流向 → 得到分析结果



以Spring Web应用程序的自动化漏洞挖掘为例

静态分析 → 识别可达方法 → 分析数据流向 → 得到分析结果



以Spring Web应用程序的自动化漏洞挖掘为例

静态分析 → 识别可达方法 → 分析数据流向 → 得到分析结果



以Spring Web应用程序的自动化漏洞挖掘为例

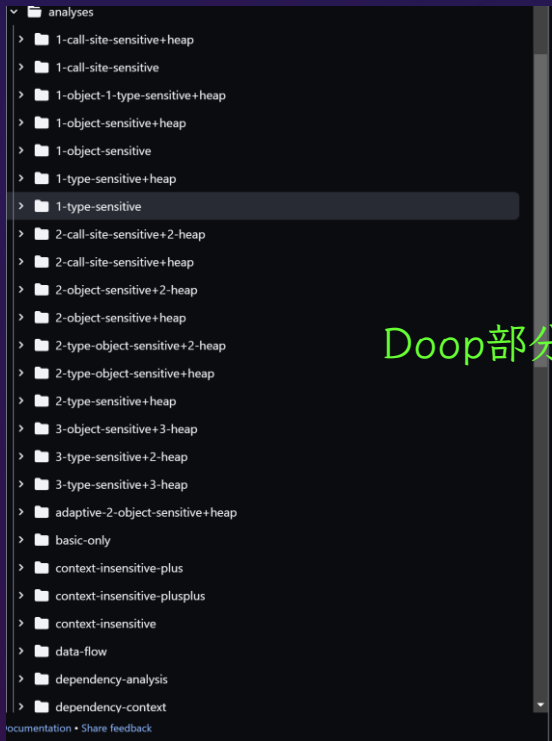
静态分析 → 识别可达方法 → 分析数据流向 → 得到分析结果

静态分析算法种类多、实现复杂、修改难度大

流敏感
上下文敏感
路径敏感
调用点敏感
对象敏感
类型敏感
K-limit 敏感度
1-call-site
2-call-site
1-object
.....



安全研究与Datalog



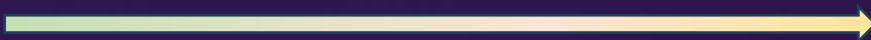
Doop部分规则



CodeQL部分规则

从工具的角度看，什么对安全研究是重要的？

扩展性 自由度 准确率 速度



重要性依次递减



以Datalog为核心构建开放式程序分析系统



京麒
网络安全大会



Change It !

开放式程序分析系统

如何算“好的”？



Change It !

开放式程序分析系统



开发者友好的DSL

具有良好抽象的规则library

准确的分析算法

高效的推理引擎

完善的配套生态



京麒
网络安全大会



Change It !

开放式程序分析系统



安全研究



Change It !

使用静态分析工具难点:

1. 了解静态分析算法
2. 对漏洞模式进行抽象
3. 熟悉框架模块组织
4. 对原有静态分析工具架构的改动



京麒
网络安全大会



Change It !

复杂的编码流程 + 高学习成本



京麒
网络安全大会



Change It !

~~复杂的编码流程 + 高学习成本~~

如何降低难度？



京麒
网络安全大会



Change It !

~~复杂的编码流程 + 高学习成本~~

如何降低难度？

开放式程序分析：一切皆rules



Change It !

~~复杂的编码流程 + 高学习成本~~

如何降低难度？

开放式程序分析：一切皆rules



AIGC + 开放式程序分析 = 自动化rules编写



Change It !

复杂的编码流程 + 高学习成本

如何降低难度？

新的问题：这对所有的开放式程序分析都能适用吗？



AIGC + 开放式程序分析 = 自动化rules编写



Change It !

一个基本要求：阅读友好的DSL

Soufflé : 一门datalog变体的逻辑编程语言 + Soufflé datalog推理引擎

CodeQL : 一门datalog变体的逻辑编程语言 + CodeQL datalog推理引擎



Change It !

使用Doop(基于Soufflé的指针分析框架)进行漏洞挖掘:

```
TaintedValueIntroduced(?declaringType, ?to, ?ctype, "default", ValueIdMacro(?to, ?ctype,
DEFAULT_BREADCRUMB)),
mainAnalysis.VarPointsTo(?hctx, ValueIdMacro(?to, ?ctype, DEFAULT_BREADCRUMB), ?ctx, ?to) :-
EntryPointMethod(?meth), Method_DeclaringType(?meth, ?declaringType), FormalParam(_, ?meth, ?to),
mainAnalysis.ReachableContext(?ctx, ?meth), Var_Type(?to, ?type),
mainAnalysis.TypeToConcreteImplementation(?type, ?ctype),
mainAnalysis.configuration.RecordContextResponse(?ctx, ValueIdMacro(?to, ?ctype,
DEFAULT_BREADCRUMB), ?to, ?hctx), Method_DeclaringType(?dtoMeth, ?type).
```

复杂繁多的rule

Token排序颠倒

引用关系难以理解

极其复杂的逻辑关系

生态缺失样本稀少

AIGC难以被学习并进行代码生成



Change It !

CodeQL:

```
import java
import semmle.code.java.dataflow.FlowSources
import DataFlow::PathGraphpredicate
class Sourcex extends RefType{
  Sourcex(){
    this.hasQualifiedName(
      "org.springframework.kafka.listener","KafkaMessageListenerContainer$ListenerConsumer"
    )
  }
}

from
  TKConfiguration config ,DataFlow::PathNode source, DataFlow::PathNode sink
where
  config.hasFlowPath(source, sink)
select
  source.getNode(), source, sink, "Find Vul"
```



Change It !

CodeQL:

```
import java
import semmle.code.java.dataflow.FlowSources
import DataFlow::PathGraphpredicate
class Sourcex extends RefType{
  Sourcex(){
    this.hasQualifiedName(
      "org.springframework.kafka.listener", "KafkaMessageListenerContainer$ListenerConsumer"
    )
  }
}

from
  TKConfiguration config ,DataFlow::PathNode source, DataFlow::PathNode sink
where
  config.hasFlowPath(source, sink)
select
  source.getNode(), source, sink, "Find Vul"
```

易于理解的语法

更清晰的引用关系

有规律的推理顺序

AIGC能快速学习并进行代码生成



京麒
网络安全大会



Change It !

一个简陋的例子 (CodeQL):

You
你知道CodeQL吗

ChatGPT
当然知道！CodeQL是一种强大的静态分析工具，主要用于发现和修复软件中的安全漏洞。它由GitHub开发，可以帮助开发者在其代码中找到潜在的安全问题。你对CodeQL有什么具体的问题或者想了解的方面吗？

You
CodeQL可以进行数据流分析和污点分析，你对CodeQL的Java语言规则库中的数据流分析了解吗

ChatGPT
当然了解！CodeQL的数据流分析和污点分析功能非常强大，特别是在Java语言规则库中。数据流分析主要用于理解程序中数据的流动方式，帮助检测潜在的安全漏洞和错误。而污点分析则关注数据在程序中的传递，特别是涉及到用户输入的地方，以帮助发现潜在的安全风险。

在Java规则库中，你可以使用CodeQL来编写查询，以识别程序中的敏感数据如何被处理，是否存在潜在的安全隐患。例如，你可以检查用户输入是否经过适当的验证和清理，以防止潜在的注入攻击。

如果你有具体的问题或者需要更详细的信息，我可以帮你查找或解释相关的内容。

You
能否用CodeQL编写一段寻找Java代码中存在SQL注入漏洞的例子

ChatGPT

当然可以！以下是一个简单的例子，用CodeQL查询来寻找Java代码中的潜在SQL注入漏洞：

```
ql

import java
import semmle.code.java.dataflow.DataFlow
import semmle.code.java.dataflow.TaintTracking

class SQLInjection extends TaintTracking::Configuration {
    SQLInjection() { this = "SQLInjection" }

    override predicate isSource(DataFlow::Node source) {
        exists(MethodAccess ma |
            ma.getMethod().getQualifiedName().matches(".*execute(Query)?")
            and source.asExpr() = ma
        )
    }

    override predicate isSink(DataFlow::Node sink) {
        exists(MethodAccess ma |
            ma.getMethod().getDeclaringType().hasQualifiedName("java.sql", "Statement")
            and sink.asExpr() = ma
        )
    }
}

from SQLInjection cfg, DataFlow::PathNode source, DataFlow::PathNode sink
where cfg.hasFlow(source, sink)
select source, sink, source.getNode().getLocation().getFile().toString()
```

这个例子使用了CodeQL的数据流分析功能，通过定义数据流的源（用户输入）和污点（传递到执行SQL语句的地方），来查找潜在的SQL注入漏洞。

请注意，这只是一个简单的例子，实际使用中可能需要更复杂的规则和查询来覆盖更多情况。此外，确保在实际应用中测试和验证这样的查询，以确保其准确性和有效性。

一个简陋的例子 (Doop):

第一次: 认为Doop做不到

You

你能用Doop写出查找Java代码中存在的SQL注入漏洞的例子吗

ChatGPT

Doop主要用于指针分析和静态程序分析, 而SQL注入漏洞通常涉及到动态的数据流和用户输入的处理。Doop可能不是最适合直接用于SQL注入漏洞检测的工具, 因为它更专注于指针分析而不是动态数据流。

通常, SQL注入漏洞检测更适合使用静态分析工具, 例如CodeQL, 以及运行时安全性工具。这些工具可以更容易地模拟和分析程序中的数据流, 以检测潜在的注入点。

以下是一个简单的例子, 使用CodeQL来查找Java代码中的潜在SQL注入漏洞:

```
ql

import java
import semmle.code.java.dataflow.DataFlow
import semmle.code.java.dataflow.TaintTracking

class SQLInjection extends TaintTracking::Configuration {
    SQLInjection() { this = "SQLInjection" }

    override predicate isSource(DataFlow::Node source) {
        exists(MethodAccess ma |
            ma.getMethod().getQualifiedName().matches(".*execute(Query)?")
            and source.asExpr() = ma
        )
    }
}
```

第二次: 给出了错误的用法

You

Doop也是一个静态分析工具, 它通过指针分析的做法对来Java应用进行静态分析, 从而发现安全漏洞

ChatGPT

我明白了, 抱歉之前的误解。虽然Doop主要是用于指针分析, 但它确实可以应用于静态分析, 包括安全漏洞的检测。在Doop中, 用户可以定义自己的分析规约, 以识别程序中的潜在问题, 包括安全漏洞。

以下是一个简化的示例, 演示如何在Doop中使用Datalog查询语言来查找Java代码中的潜在SQL注入漏洞。请注意, 这只是一个概念性的示例, 实际使用时可能需要更复杂的规则和配置。

```
datalog

class SourceFlow(Source, Flow) :- {
    SourceFlow(Source, Flow), SourceFlow(Flow, Source).
class SinkFlow(Sink, Flow) :- {
    SinkFlow(Sink, Flow), SinkFlow(Flow, Sink).
class SQLInjectionFlow(Flow) :- ().

hasFlow(Flow) :- SourceFlow(Source, Flow), SinkFlow(Sink, Flow), not hasSanitizer(Sink, Sanitizer) :- SinkFlow(Sink, Flow), SourceFlow(Sanitizer, Sink).

SQLInjectionFlow(Flow) :- hasFlow(Flow), not hasSanitizer(., Flow).
```

这个简化的Datalog规约尝试找到从源到汇点的数据流, 而没有经过清理 (即没有安全措施)。你可以根据你的需求扩展这个规约以更全面地涵盖SQL注入漏洞的情况。



Change It !

第n次: 仍然无法编写出正确的DooP规则样例

1. Soufflé语法不正确
2. DooP的模块无法正确使用, 基本不知道DooP具备哪些模块
3. 规则繁多, 即使做出引导, 也无法正确使用需要的relation

第n次: 仍然无法编写出正确的Dooop规则样例

1. Soufflé语法不正确
2. Dooop的模块无法正确使用, 基本不知道Dooop具备哪些模块
3. 规则繁多, 即使做出引导, 也无法正确使用需要的relation

样本极其缺失

Soufflé的推导方式复杂

模块依赖层级混乱



垂直领域专有模型

理解友好的DSL

抽象良好的library



Change It !

And More: 动静态结合、框架建模、IaC安全



寻找小伙伴：安全研究、静态分析、编译器



京麒
网络安全大会



THANKS