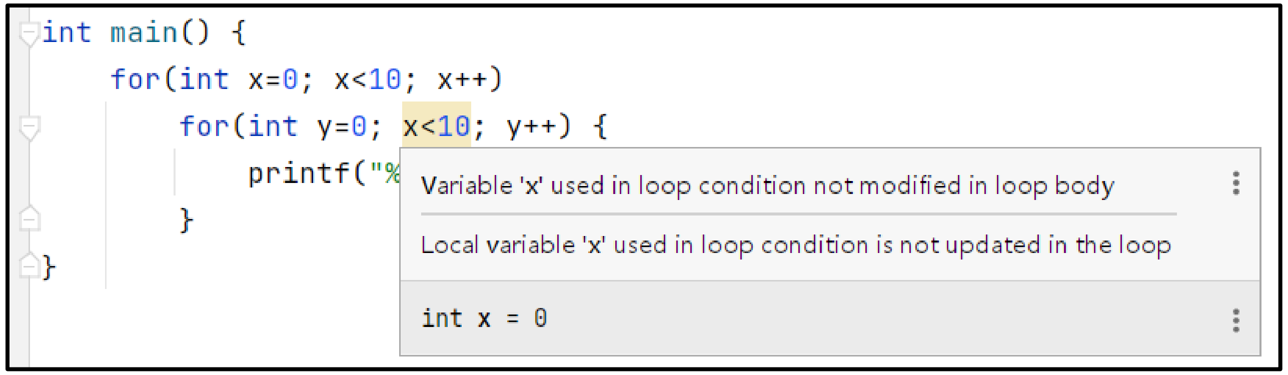
PredicateFix：利用桥接谓词修复静态分析警报

熊英飞 北京大学

背景

- 静态分析工具在实践中被广泛用于安全漏洞扫描
  - 如：CodeQL
- 修复静态分析工具的漏洞警报耗费程序员大量精力
  - 大模型出现后，修复任务可以交给大模型完成
  - 真的吗？



```
int main() {
    for(int x=0; x<10; x++)
        for(int y=0; x<10; y++) {
            printf("%
        }
}
```

Variable 'x' used in loop condition not modified in loop body

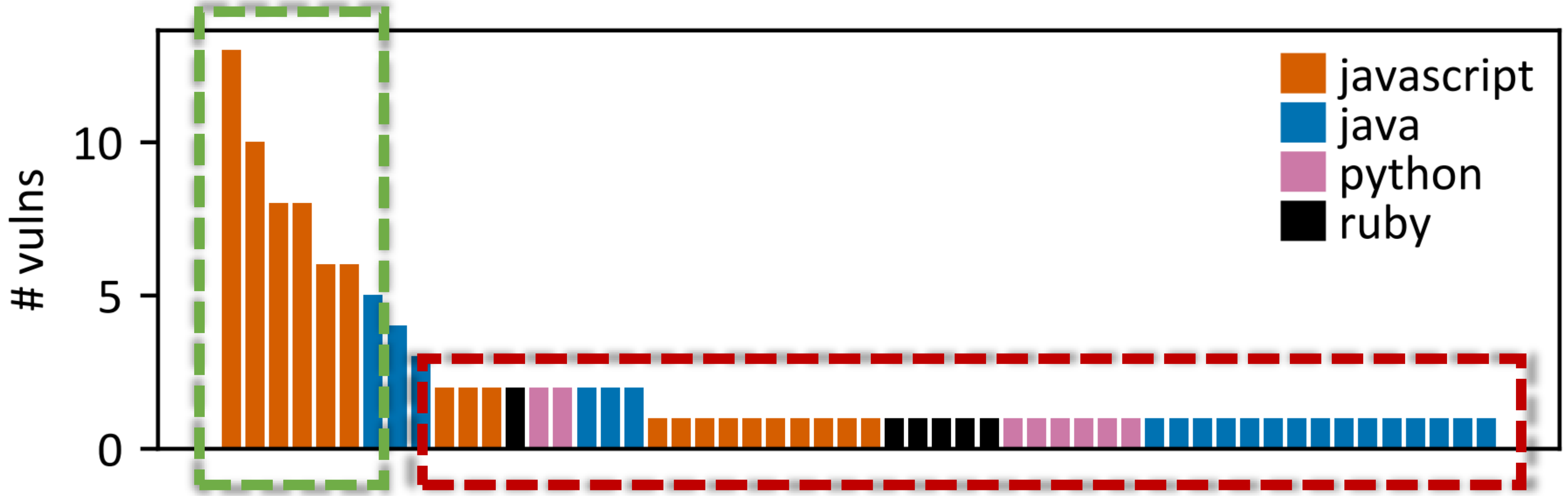
Local variable 'x' used in loop condition is not updated in the loop

```
int x = 0
```

静态分析警报分类

"长尾分布"

↓ 根据CVE分类的CodeQL警报数

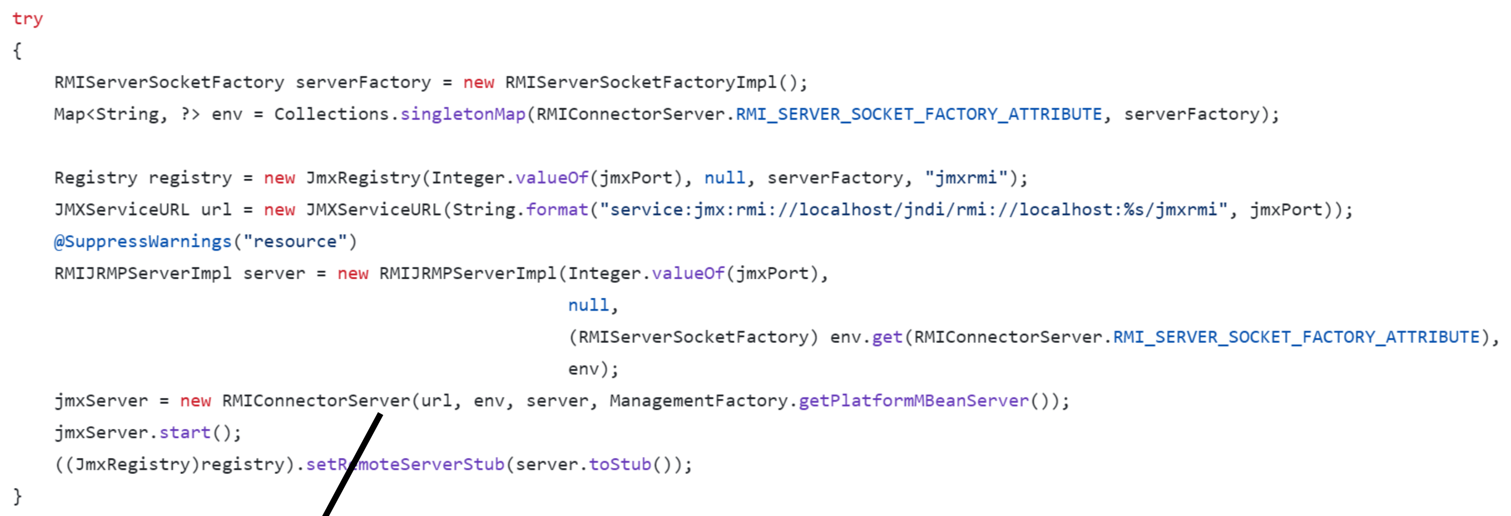


常见类型：
大模型可较好修复

不常见类型（但总数多）：
大模型常常产生幻觉

修复示例

CVE-2020-13946 for Apache Cassandra



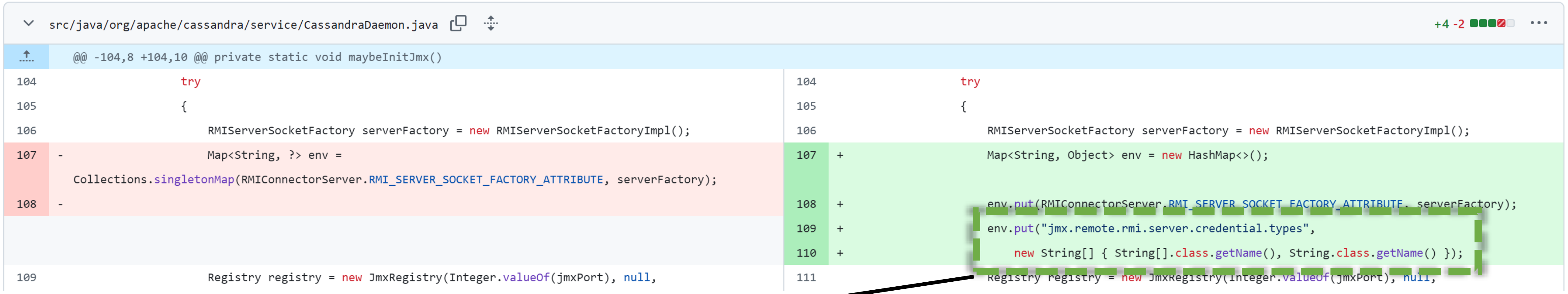
```
try
{
    RMIServerSocketFactory serverFactory = new RMIServerSocketFactoryImpl();
    Map<String, ?> env = Collections.singletonMap(RMIConnectorServer.RMI_SERVER_SOCKET_FACTORY_ATTRIBUTE, serverFactory);

    Registry registry = new JmxRegistry(Integer.valueOf(jmxPort), null, serverFactory, "jmxrmi");
    JMXServiceURL url = new JMXServiceURL(String.format("service:jmx:rmi://localhost/jndi/rmi://localhost:%s/jmxrmi", jmxPort));
    @SuppressWarnings("resource")
    RMIJRMPServerImpl server = new RMIJRMPServerImpl(Integer.valueOf(jmxPort),
                                                     null,
                                                     (RMIServerSocketFactory) env.get(RMIConnectorServer.RMI_SERVER_SOCKET_FACTORY_ATTRIBUTE),
                                                     env);
    jmxServer = new RMIConnectorServer(url, env, server, ManagementFactory.getPlatformMBeanServer());
    jmxServer.start();
    ((JmxRegistry)registry).setRemoteServerStub(server.toStub());
}
```

CodeQL alert: InsecureRmiJmxAuthenticationEnvironment

"This query detects if a JMX/RMI server is created with a potentially dangerous environment, which could lead to code execution through insecure deserialization."

修复示例

CVE-2020-13946 for Apache Cassandra



```
src/java/org/apache/cassandra/service/CassandraDaemon.java                +4 -2
@@ -104,8 +104,10 @@ private static void maybeInitJmx()
```

```
            try
            {
                RMIServerSocketFactory serverFactory = new RMIServerSocketFactoryImpl();
-               Map<String, ?> env =
Collections.singletonMap(RMIConnectorServer.RMI_SERVER_SOCKET_FACTORY_ATTRIBUTE, serverFactory);
-
                Registry registry = new JmxRegistry(Integer.valueOf(jmxPort), null,
```

```
            try
            {
                RMIServerSocketFactory serverFactory = new RMIServerSocketFactoryImpl();
+               Map<String, Object> env = new HashMap<>();

+               env.put(RMIConnectorServer.RMI_SERVER_SOCKET_FACTORY_ATTRIBUTE, serverFactory);
+               env.put("jmx.remote.rmi.server.credential.types",
+                   new String[] { String[].class.getName(), String.class.getName() });
                Registry registry = new JmxRegistry(Integer.valueOf(jmxPort), null,
```

开发人员修复 (commit **63f4da9**)：

+ env.put(“jmx.remote.rmi.server.credential.types”,
    new String[] { String[].class.getName(), String.class.getName() });

通过设置配置项，保证只有字符串能用于JMX验证

尝试1：直接提示

[System]
You are a helpful programming assistant to fix security vulnerabilities in Java code. You are given a code snippet and a description of the vulnerability. You should briefly describe how to fix the vulnerability, and finally output the modification.

[User]

Vulnerability des
Code snippet: *(1*

大模型修复（删除原有配置项，完全不正确）：

```
- Map<String, ?> env = Collections.singletonMap(
-     RMIConnectorServer.RMI_SERVER_SOCKET_FACTORY_ATTRIBUTE, serverFactory
- );
+ Map<String, Object> env = new HashMap<>();
```

[Assistant] (GPT-4o)

To fix the vulnerability, we need to ensure that the environment map used to create the RMIConnectorServer does not include any potentially dangerous attributes that could lead to insecure deserialization. Specifically, we **should avoid using RMI_SERVER_SOCKET_FACTORY_ATTRIBUTE directly** and instead use a more secure configuration.

……

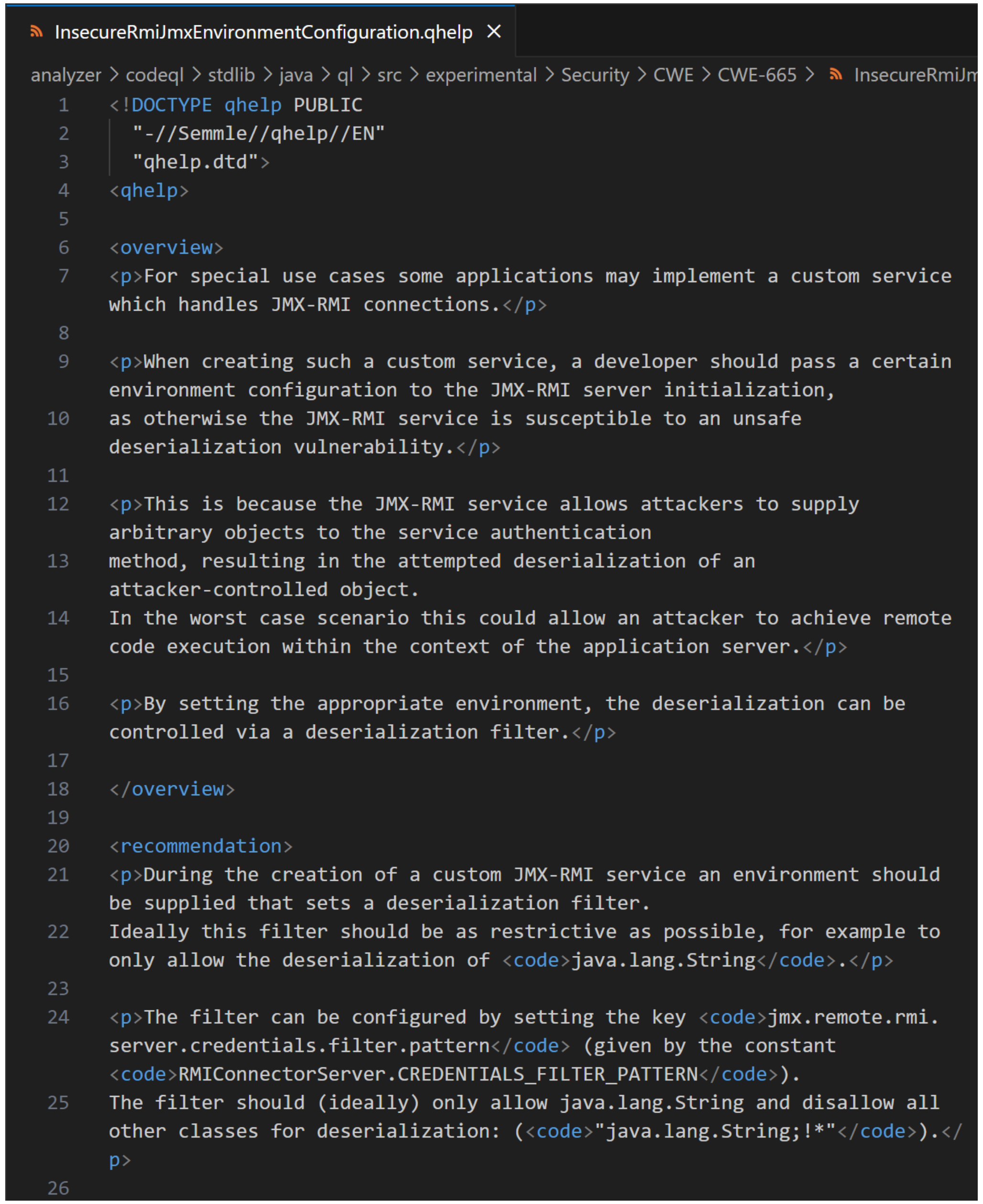
尝试2：提示CodeQL分析规则

- 分析规则过于复杂，大模型难以正确理解
- 在本示例上，GPT 4o直接摆烂：
  - 删除原有验证机制
  - 添加todo，让程序员修复

```
module SafeFlowConfig implements DataFlow::ConfigSig {
  predicate isSource(DataFlow::Node source) {
    putsCredentialtypesKey(source.asExpr()) }
  ......
  private predicate putsCredentialtypesKey(Expr qualifier) {
    exists(MapPutCall put |
      put.getKey().(CompileTimeConstantExpr).getStringValue() = [
        "jmx.remote.rmi.server.credential.types",
         "jmx.remote.rmi.server.credentials.filter.pattern"
      ] or ......
     | put.getQualifier() = qualifier and
      put.getMethod().(MapMethod).getReceiverKeyType()
        instanceof TypeString and
      put.getMethod().(MapMethod).getReceiverValueType()
        instanceof TypeObject ) } }
module SafeFlow = DataFlow::Global<SafeFlowConfig>;
......
from Call c, Expr envArg
where (isRmiOrJmxServerCreateConstructor(c.getCallee())
    or isRmiOrJmxServerCreateMethod(c.getCallee()))
  and envArg = c.getArgument(1)
  and not SafeFlow::flowToExpr(envArg)
select c, getRmiResult(envArg), envArg, envArg.toString()
```

尝试3：提示CodeQL文档

- 大模型确实有较好的文档理解能力
- 但
  - 文档常常不全，不包含修复方法
  - 文档的修复方法不适用于当前项目
    - 当前项目有特殊的编译环境
    - 当前项目有项目级别的安全规范
- 在本示例上，
  - 大模型产生一个Java 10支持的修复，
  - 但项目采用Java 8



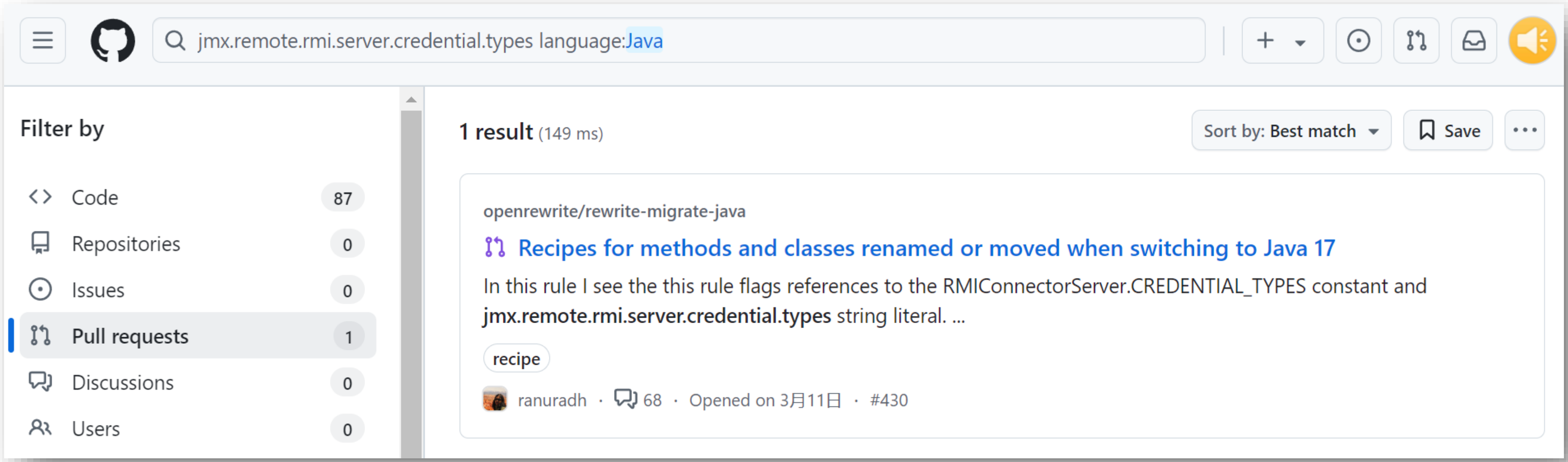
InsecureRmiJmxEnvironmentConfiguration.qhelp ×

analyzer > codeql > stdlib > java > ql > src > experimental > Security > CWE > CWE-665 > InsecureRmiJm

```
<!DOCTYPE qhelp PUBLIC
  "-//Semmle//qhelp//EN"
  "qhelp.dtd">
<qhelp>

<overview>
<p>For special use cases some applications may implement a custom service
which handles JMX-RMI connections.</p>

<p>When creating such a custom service, a developer should pass a certain
environment configuration to the JMX-RMI server initialization,
as otherwise the JMX-RMI service is susceptible to an unsafe
deserialization vulnerability.</p>

<p>This is because the JMX-RMI service allows attackers to supply
arbitrary objects to the service authentication
method, resulting in the attempted deserialization of an
attacker-controlled object.
In the worst case scenario this could allow an attacker to achieve remote
code execution within the context of the application server.</p>

<p>By setting the appropriate environment, the deserialization can be
controlled via a deserialization filter.</p>

</overview>

<recommendation>
<p>During the creation of a custom JMX-RMI service an environment should
be supplied that sets a deserialization filter.
Ideally this filter should be as restrictive as possible, for example to
only allow the deserialization of <code>java.lang.String</code>.</p>

<p>The filter can be configured by setting the key <code>jmx.remote.rmi.
server.credentials.filter.pattern</code> (given by the constant
<code>RMIConnectorServer.CREDENTIALS_FILTER_PATTERN</code>).
The filter should (ideally) only allow java.lang.String and disallow all
other classes for deserialization: (<code>"java.lang.String;!*"</code>).</
p>

```

尝试4: 检索增强的生成 (RAG)-1

- 检索相关的历史补丁，提示给大模型
- 对于少见的漏洞，基本找不到历史补丁



1 result (149 ms)

openrewrite/rewrite-migrate-java

Recipes for methods and classes renamed or moved when switching to Java 17

In this rule I see the this rule flags references to the RMIConnectorServer.CREDENTIAL_TYPES constant and **jmx.remote.rmi.server.credential.types** string literal. ...

recipe

ranuradh · 68 · Opened on 3月11日 · #430

尝试5: 检索增强的生成 (RAG)-2

- 检索相关的相似安全代码，提示给大模型
- 因为合适代码数量少，目前方法很难精准匹配对应代码

出错代码:

```
RMIServerSocketFactory serverFactory = new RMIServerSocketFactoryImpl();
Map<String, ?> env = Collections.singletonMap(
    RMIConnectorServer.RMI_SERVER_SOCKET_FACTORY_ATTRIBUTE, serverFactory);
......
jmxServer = new RMIConnectorServer(url, env, server,
    ManagementFactory.getPlatformMBeanServer());
```

BM25匹配:

```
final MBeanServer jmxServer = ManagementFactory.getPlatformMBeanServer();
try {
    jmxServer.unregisterMBean(new ObjectName(this.getMBeanName()));
}
……
```

观察

```
module SafeFlowConfig implements DataFlow::ConfigSig {
  predicate isSource(DataFlow::Node source) { putsCredentialtypesKey(source.asExpr()) }
  ......
  private predicate putsCredentialtypesKey(Expr qualifier) {
    exists(MapPutCall put |
      put.getKey().(CompileTimeConstantExpr).getStringValue() = [
         "jmx.remote.rmi.server.credential.types",
         "jmx.remote.rmi.server.credentials.filter.pattern"
      ] or ......
    | put.getQualifier() = qualifier and
      put.getMethod().(MapMethod).getReceiverKeyType() instanceof TypeString and
      put.getMethod().(MapMethod).getReceiverValueType() instanceof TypeObject )
  }
}
module SafeFlow = DataFlow::Global<SafeFlowConfig>;
......
from Call c, Expr envArg
where
  (isRmiOrJmxServerCreateConstructor(c.getCallee()) or isRmiOrJmxServerCreateMethod(c.getCallee()))
  and envArg = c.getArgument(1)
```

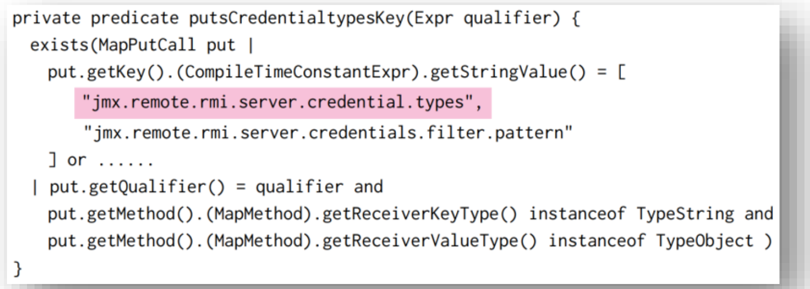
该谓词包含修复关键元素，可用来寻找样例
修复规则是否总是存在这样的谓词？

分析：如何撰写规则漏洞检查代码漏洞？

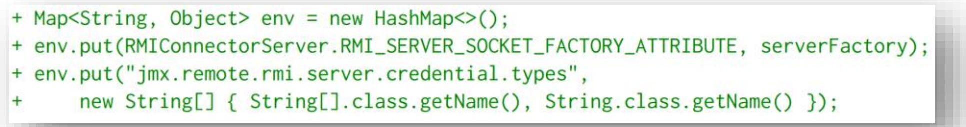
- 要么检查某种危险代码存在
  - 修复方法1：删除危险代码
    - 大模型通常不需要额外信息就能完成
  - 修复方法2：换成安全代码，如将不安全的函数调用换成安全调用
    - 大模型通常能从名字找到安全调用：XXX->XXX_Safe
- 要么检查某种安全机制不存在
  - 修复方法：添加安全机制
    - 需要添加全新代码，对大模型比较困难
  - 添加的关键代码一定包含在静态分析规则中！

我们的方法

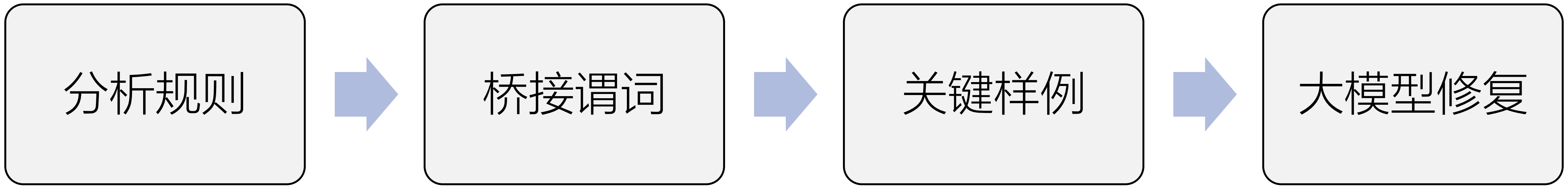
- 找到规则中用来匹配关键元素的部分
  - 称为“**桥接谓词**”
- 采用桥接谓词找到对应的样例
  - 称为“**关键样例**”



```
private predicate putsCredentialtypesKey(Expr qualifier) {
  exists(MapPutCall put |
    put.getKey().(CompileTimeConstantExpr).getStringValue() = [
      "jmx.remote.rmi.server.credential.types",
       "jmx.remote.rmi.server.credentials.filter.pattern"
    ] or ......
  | put.getQualifier() = qualifier and
    put.getMethod().(MapMethod).getReceiverKeyType() instanceof TypeString and
    put.getMethod().(MapMethod).getReceiverValueType() instanceof TypeObject )
}
```



```
+ Map<String, Object> env = new HashMap<>();
+ env.put(RMIConnectorServer.RMI_SERVER_SOCKET_FACTORY_ATTRIBUTE, serverFactory);
+ env.put("jmx.remote.rmi.server.credential.types",
+     new String[] { String[].class.getName(), String.class.getName() });
```



桥接谓词和关键样例

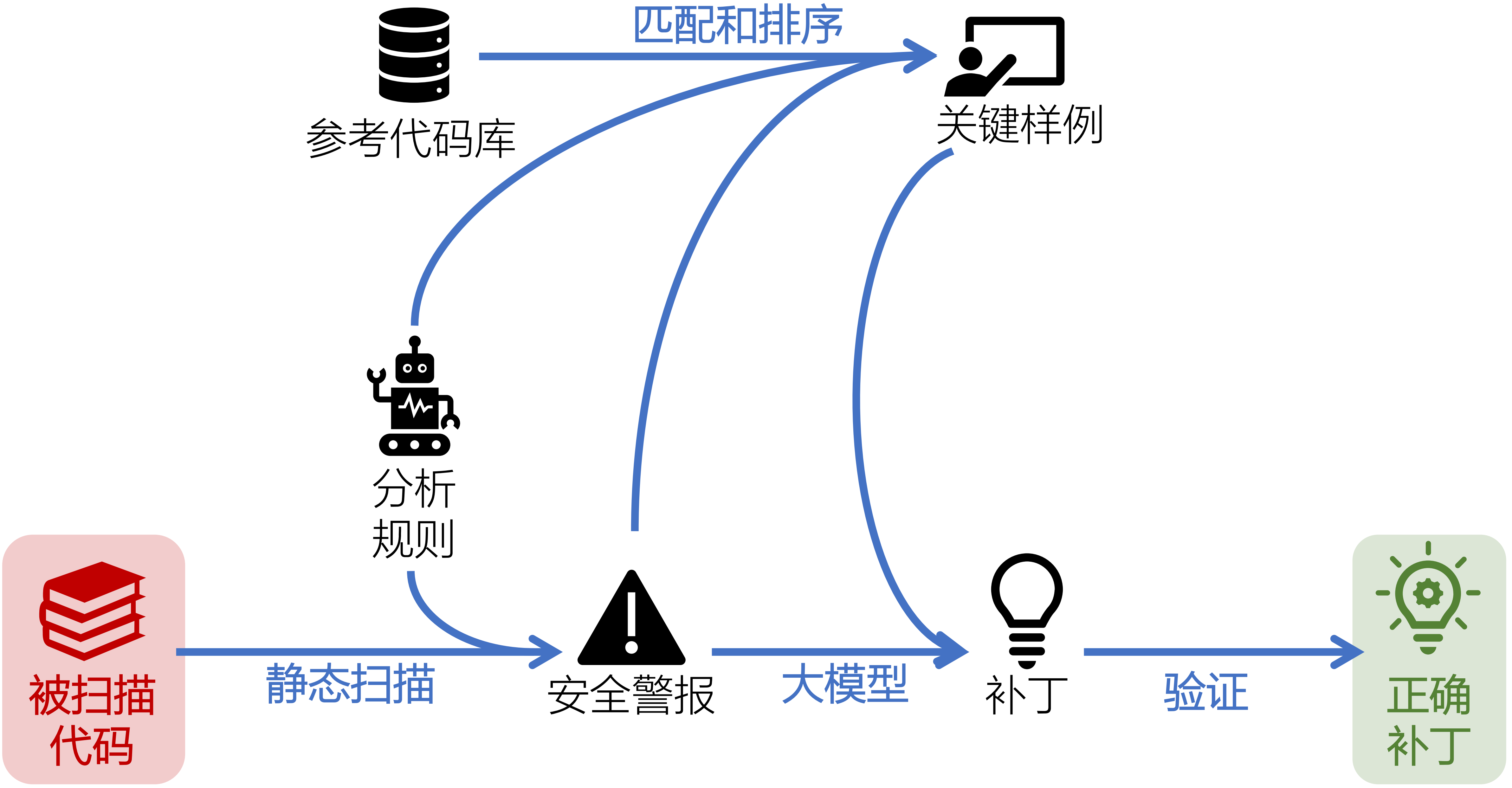
- 给定谓词*p*和样例*s*， *p*是桥接谓词且*s*是关键样例， 当且仅当

- **条件1:** *p*匹配*s*
- **条件2:** 在出错代码上翻转*p* → 漏洞警报消失
- **条件3:** 在*s*上翻转*p* → 新漏洞警报出现

💡 *p* 是漏洞代码和安全代码的关键区分

首先用条件2找到桥接谓词候选， 然后用条件1和3找到关键样例

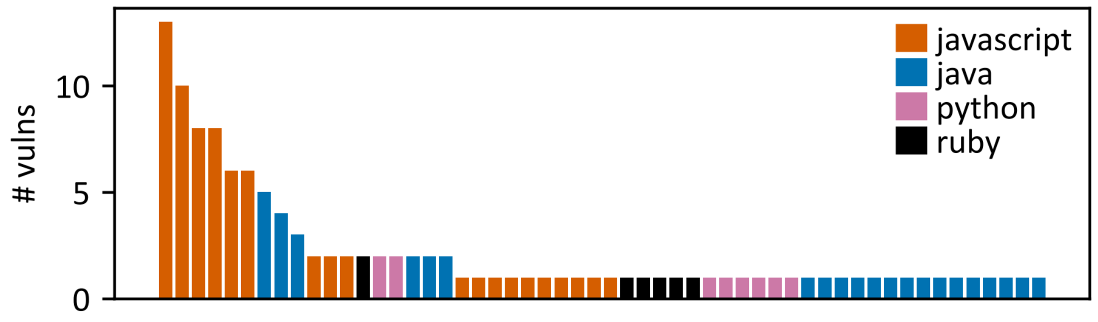
整体流程



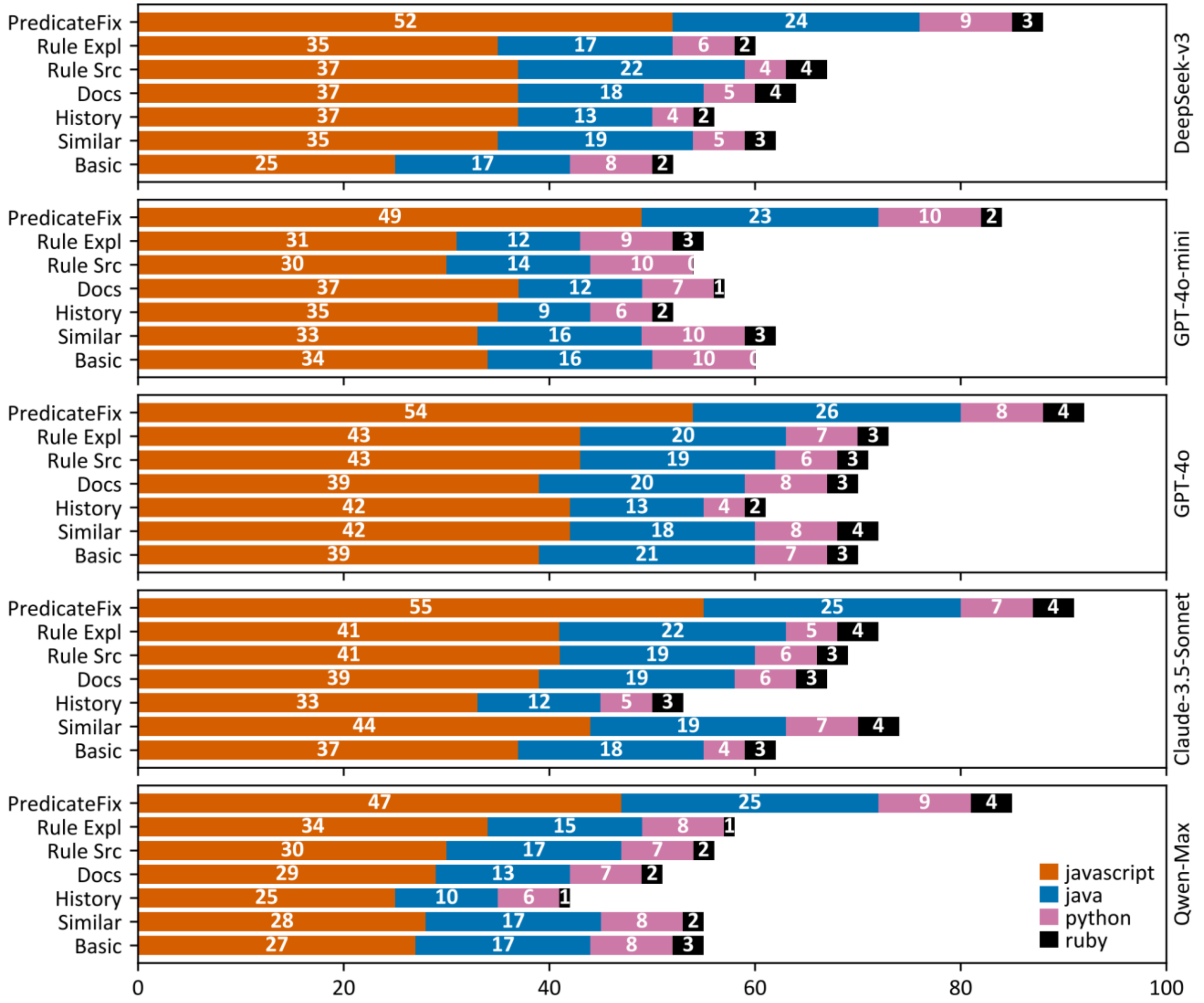
验证

数据集:

- 收集6027带修复的CVE漏洞
- 运行CodeQL检查修复前后代码
  - 修复前代码有漏洞警报
  - 修复之后代码没有
- 获得117个验证数据



验证结果



- **Basic:** 直接提示
- **Similar:** bm25查找相似代码
- **History:** bm25查找历史补丁
- **Docs:** 提示CodeQL文档

- **Rule Src:** 提示CodeQL规则源码
- **Rule Expl:** 大模型解释后的规则源码

- **PredicateFix:** 我们的方法

中兴的部署

- GoInsight： 中兴内部的Go语言静态分析工具
  - 基于命令式语言编写静态分析规则
- 中兴工程师将本项目方法实现在GoInsight上
  - Predicate： If语句中的条件
- 工具的修复比例从40.9%提升到69.2%
  - 部署一个月支持了16个项目，修复了280个缺陷

中兴操作系统工具平台
程圣宇经理

大模型自己会写静态分析规则吗？

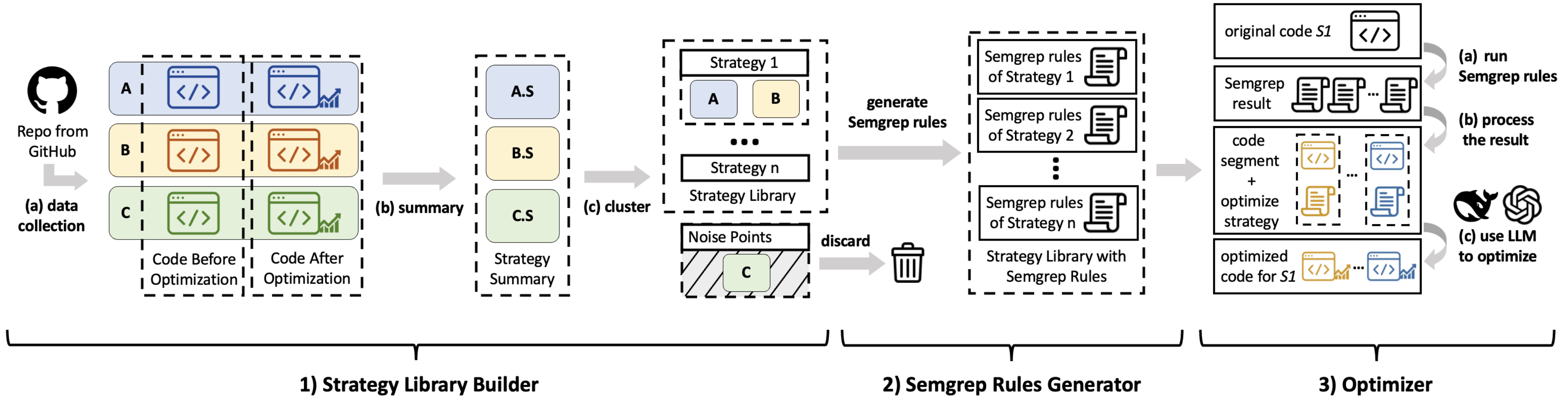
大模型做优化

- 大模型打算法竞赛已经超越人类
- AlphaEvolve已经设计出新算法
- 为什么大模型还没有大规模自动帮我们优化软件性能?
- 问题：大模型找不到优化方向
  - 已有解决方法：通过例子提示优化策略
  - 收集历史优化补丁，用BM25寻找最合适的优化补丁
  - 但BM25常常不能准确匹配
- 思路：遍历所有的优化示例
  - 软件中有n段代码，补丁库中有m个示例，就要至少提示n*m次
  - n和m都巨大，考虑LLM开销和时间，几乎不可能完成

我们的方法SemOpt

- 静态分析规则可以低层本扫描大量代码
- 让大模型生成静态分析规则，捕获优化策略的应用条件
- 对于匹配上的代码，再让大模型进行优化

我们的方法SemOpt



实验结果

- 复现程序员的历史优化

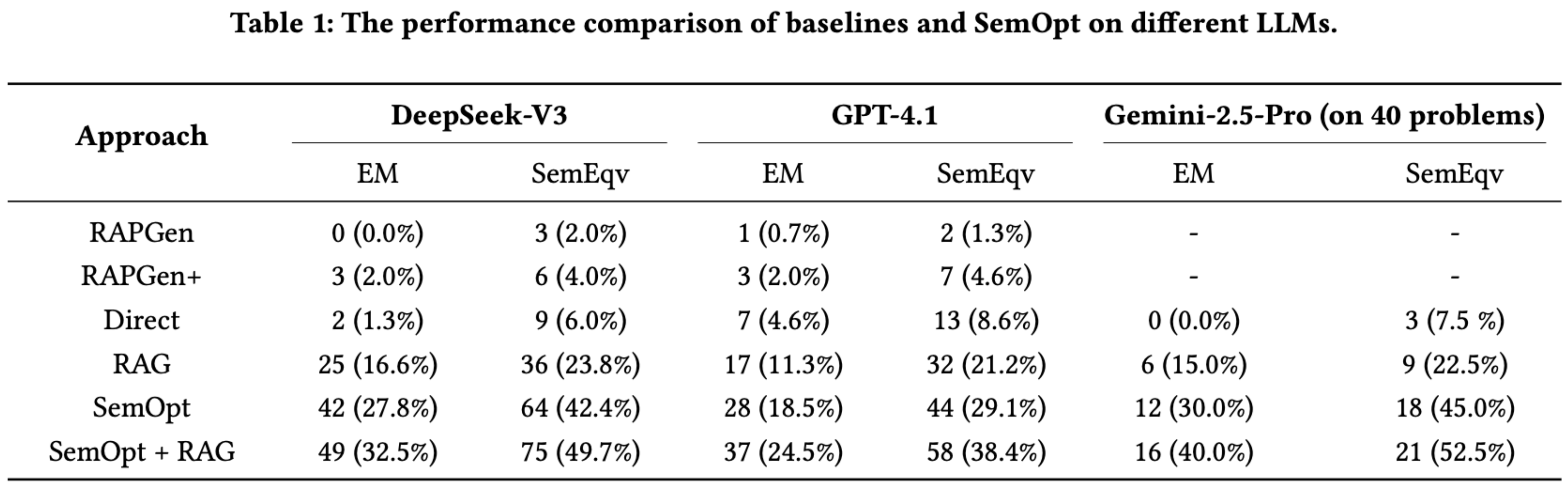


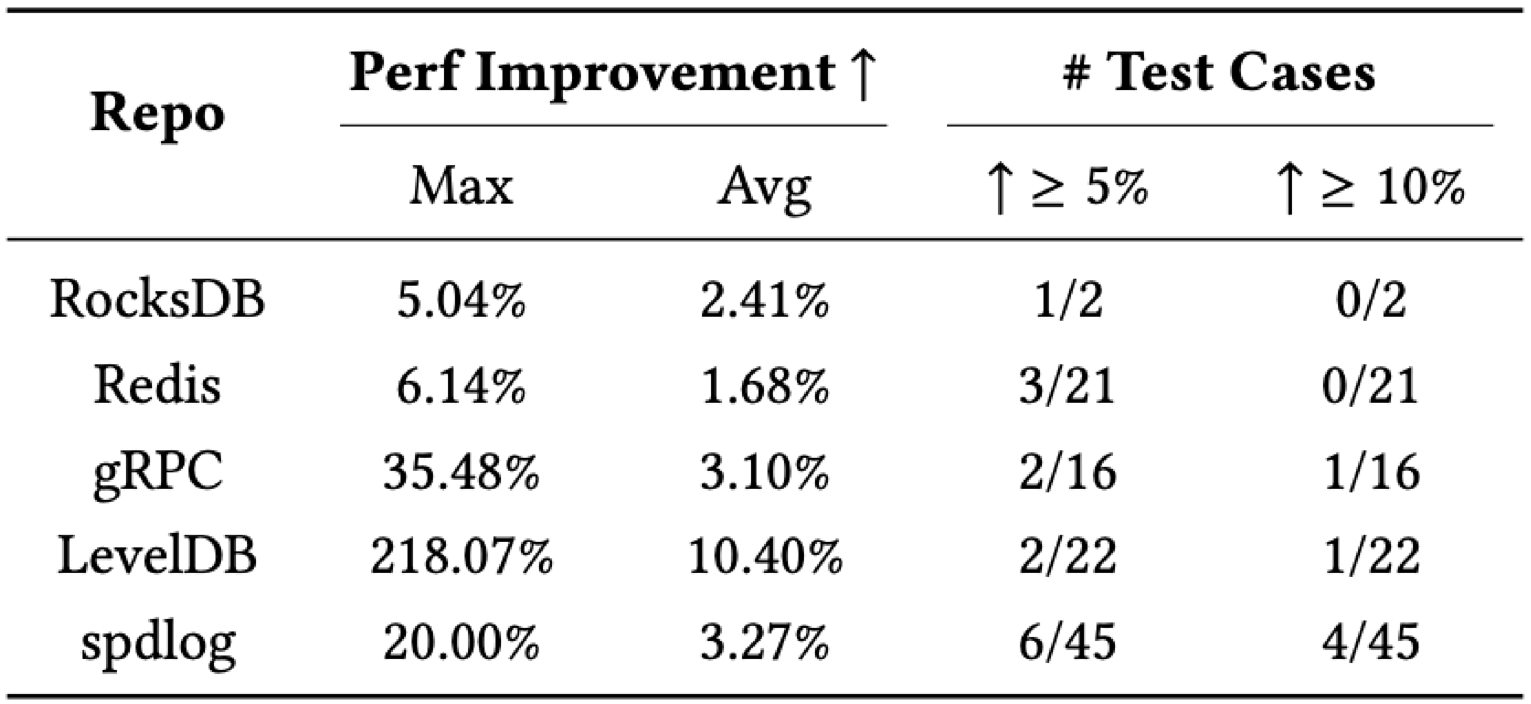
Table 1: The performance comparison of baselines and SemOpt on different LLMs.

Approach	DeepSeek-V3		GPT-4.1		Gemini-2.5-Pro (on 40 problems)	
	EM	SemEqv	EM	SemEqv	EM	SemEqv
RAPGen	0 (0.0%)	3 (2.0%)	1 (0.7%)	2 (1.3%)	-	-
RAPGen+	3 (2.0%)	6 (4.0%)	3 (2.0%)	7 (4.6%)	-	-
Direct	2 (1.3%)	9 (6.0%)	7 (4.6%)	13 (8.6%)	0 (0.0%)	3 (7.5 %)
RAG	25 (16.6%)	36 (23.8%)	17 (11.3%)	32 (21.2%)	6 (15.0%)	9 (22.5%)
SemOpt	42 (27.8%)	64 (42.4%)	28 (18.5%)	44 (29.1%)	12 (30.0%)	18 (45.0%)
SemOpt + RAG	49 (32.5%)	75 (49.7%)	37 (24.5%)	58 (38.4%)	16 (40.0%)	21 (52.5%)

实验结果

- 直接优化大型项目的结果

Table 4: In-the-wild evaluation result

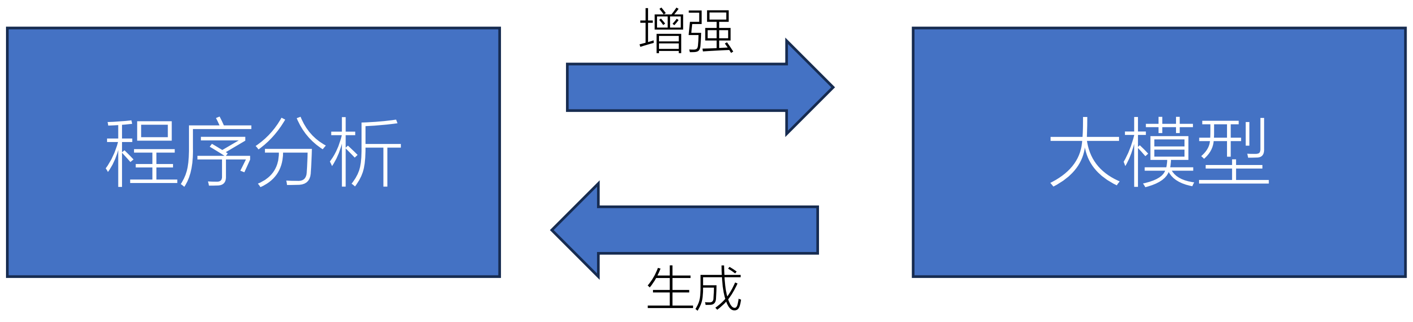


Repo	**Perf Improvement ↑**		**# Test Cases**	
	Max	Avg	$\uparrow \geq 5\%$	$\uparrow \geq 10\%$
RocksDB	5.04%	2.41%	1/2	0/2
Redis	6.14%	1.68%	3/21	0/21
gRPC	35.48%	3.10%	2/16	1/16
LevelDB	218.07%	10.40%	2/22	1/22
spdlog	20.00%	3.27%	6/45	4/45

小结

程序分析和大模型能力互补

整合二者可以达到更好效果


增强
程序分析
大模型
生成