black hat ARSENAL

DECEMBER 9-12, 2024
EXCEL LONDON / UNITED KINGDOM

Tabby: Simplifying the Art of Java Vulnerability Hunting

Speaker: BaiZhu Wang
AntGroup MYBank
Other Contributors: XingChen Chen

Who Are We

Baizhu Wang (@wh1t3p1g)

Senior Security Engineer at AntGroup MYBank

Focused on program analysis and red team

Xingchen Chen (@cxc)

PhD student at Institute of Information Engineering, Chinese Academy of Sciences

Focused on program analysis and vulnerability detection

Agenda

- Introduction
- Implementation
- Application Scenarios & Demos
- Next in Tabby

Introduction

Background About Tabby

Background

The Reality About Java Code Auditing

- **Increasing Code Complexity:** Modern Java programs are becoming increasingly complex with larger codebases.
- **High Technical Requirements for Analysts:** The technical expertise required to analyze these large-scale applications is continually growing.
- **Incomplete Automatic Tool Analysis:** Many automatic tools do not provide comprehensive analysis, leading to high false-negative rates
- **Time-consuming Intermediate Data Generation:** Generating intermediate data during the program analysis can be time-intensive and cannot be easily reused.
- **Lack of Access to Source Code:** In many cases, analysts do not have access to source code, making it difficult to perform effective audits.

Tabby is designed to address these challenges.

What is Tabby?

Tabby is an **automatic code auditing suite** specifically designed for the Java language. It transforms code into graph data stored in Neo4j, enabling users to identify various Java vulnerabilities using simple Cypher queries.

Tabby shares a similar concept with CodeQL but is more effective for Java language analysis.
- **Ease of Use**: Provides fixed Cypher templates or YAML templates for straightforward adoption.
- **Superior Effectiveness**: Supports multiple scenarios, accepts various types of inputs, and detects numerous vulnerabilities.
- **Comprehensive Analysis**: Includes third-party libraries in the audit process for a broader scope.
- **Greater Customization**: Allows custom logic to be incorporated into the analysis.

Tabby Findings:
CVE-2021-21346、 CVE-2021-21351、 CVE-2021-39147、 CVE-2021-39148、 CVE-2021-39152、 CVE-2021-43297、 CVE-2021-43297、 CVE-2022-39198、 CVE- 2022-39198、 CVE-2024-52012、 CVE-2024-48988 and 100+ commercial application vulnerabilities.

Implementation

Technical Details of Tabby

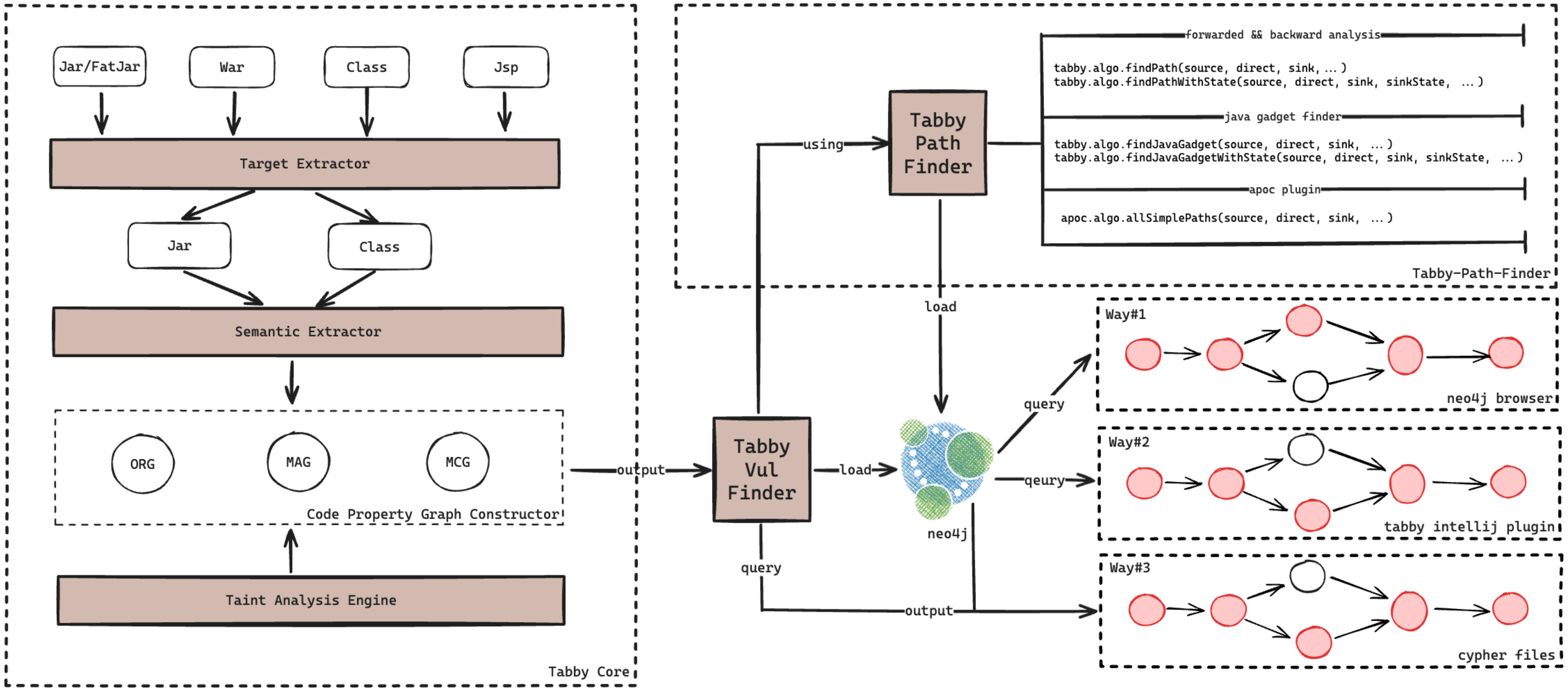
Architecture

Tabby Core

Covert code to a specialized **Code Property Graph (CPG)**, including Target Extractor, Semantic Extractor and Taint Analysis Engine.

Tabby Vul Finder

A tool can load CPG csv files to Neo4j, and also can find vulnerable function chains on database.



Referernces:
https://github.com/wh1t3p1g/tabby
https://github.com/wh1t3p1g/tabby-vul-finder
https://github.com/wh1t3p1g/tabby-path-finder
https://github.com/wh1t3p1g/tabby-intellij-plugin

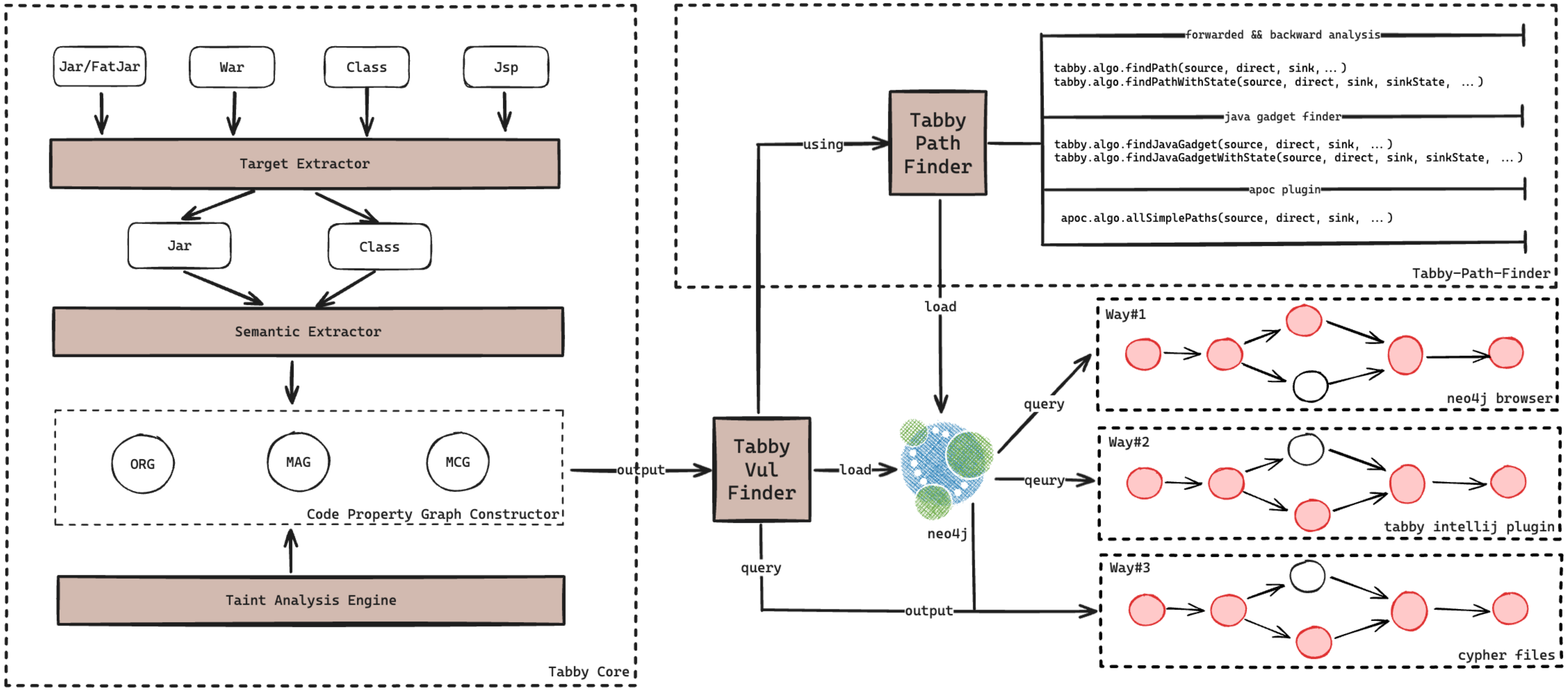
Architecture

Tabby Path Finder

A path traversal plugin on Neo4j implements specialized logic for inter-procedural analysis, enabling the detection of vulnerabilities across multiple methods and function calls.

Tabby Intellij Plugin

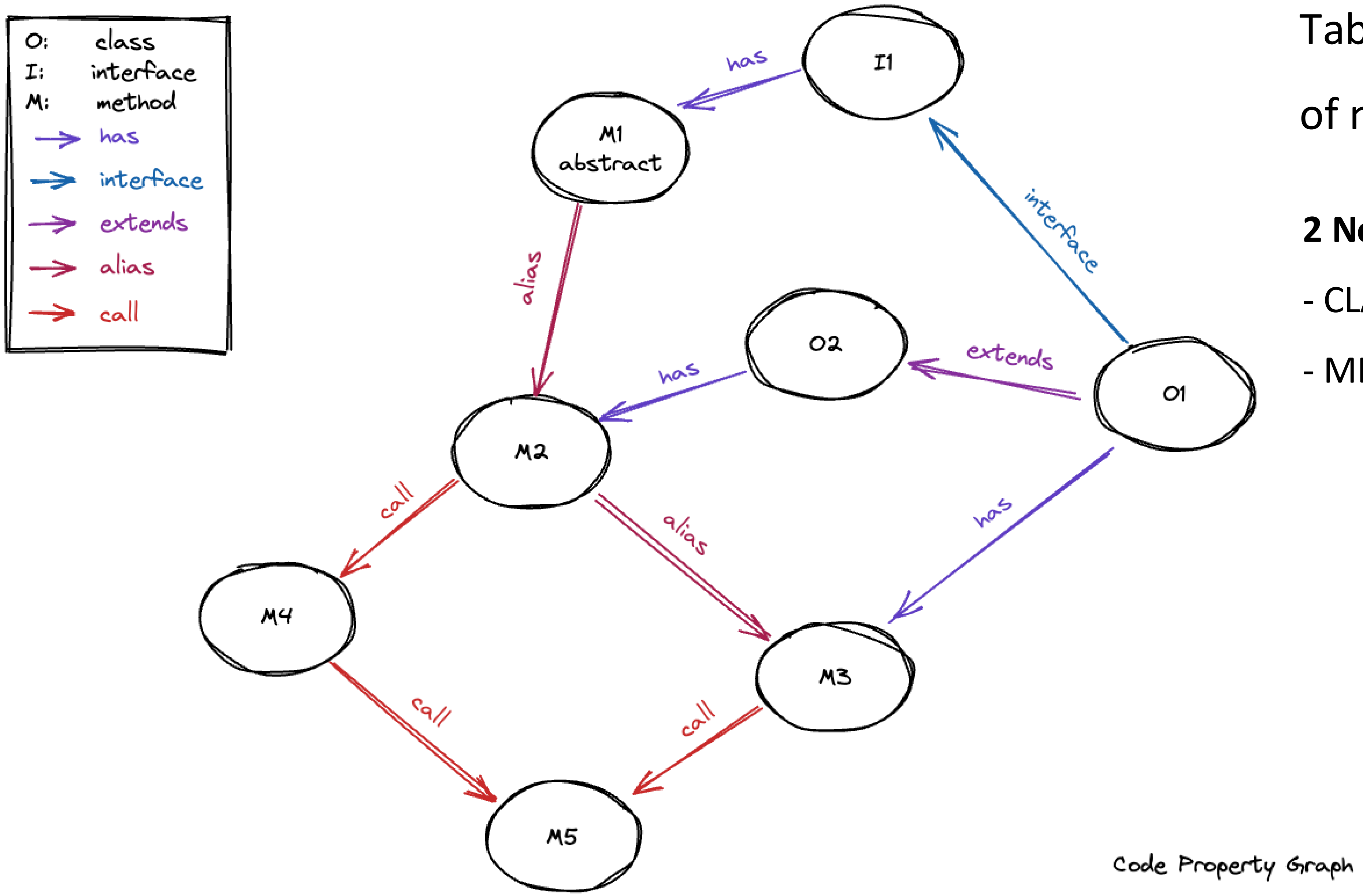
A plugin on Jetbrain IDEA can query customize cyphers and jump from nodes or edges to source code.



Referernces:
https://github.com/wh1t3p1g/tabby
https://github.com/wh1t3p1g/tabby-vul-finder
https://github.com/wh1t3p1g/tabby-path-finder
https://github.com/wh1t3p1g/tabby-intellij-plugin

black hat ARSENAL

Tabby CPG on Neo4j



Tabby Code Property Graph contains 2 types of nodes and 5 types of edges.

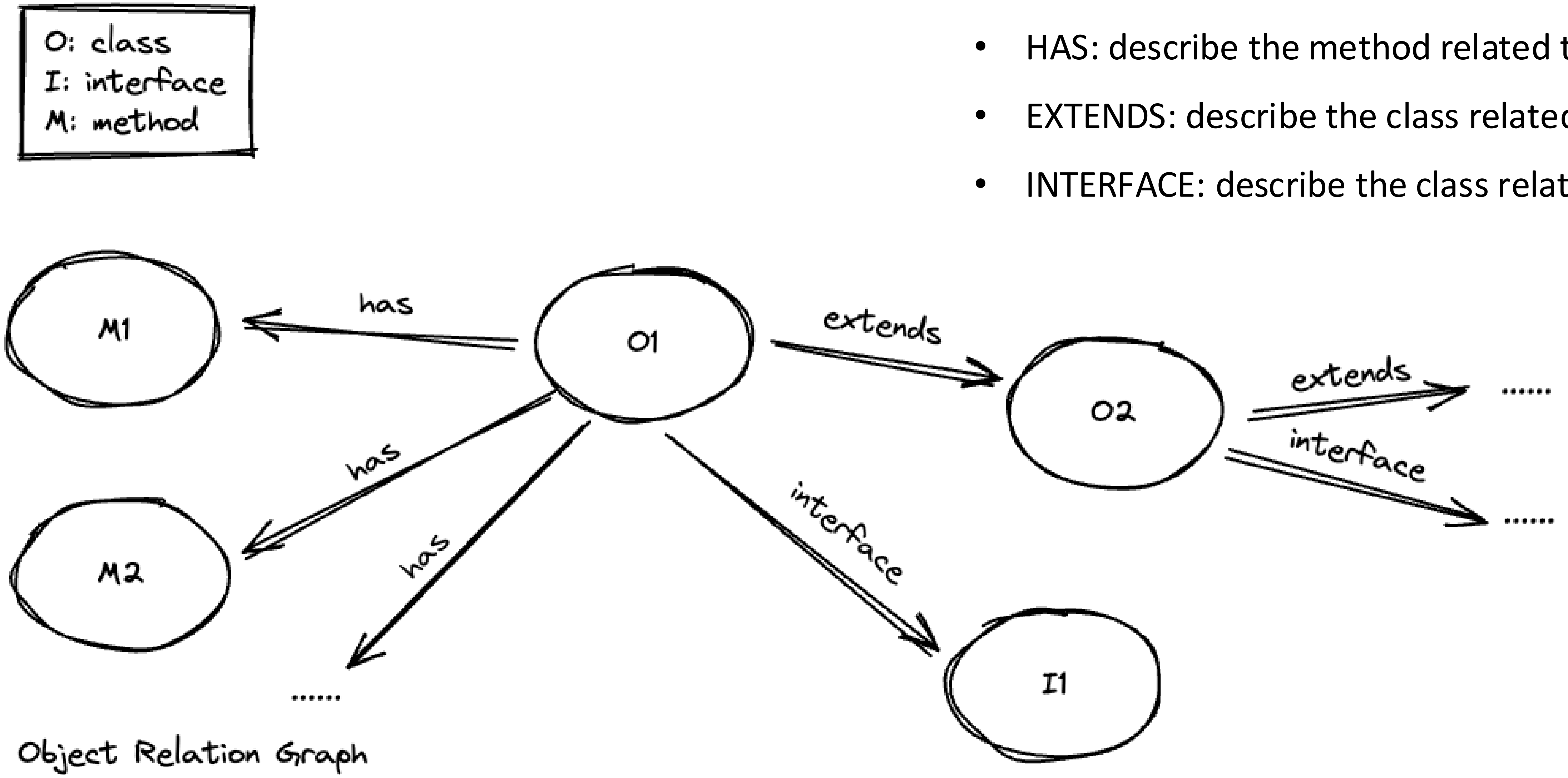
2 Nodes:

- CLASS node
- METHOD node

5 Edges:

- HAS edge
- CALL edge
- ALIAS edge
- EXTENDS edge
- INTERFACE edge

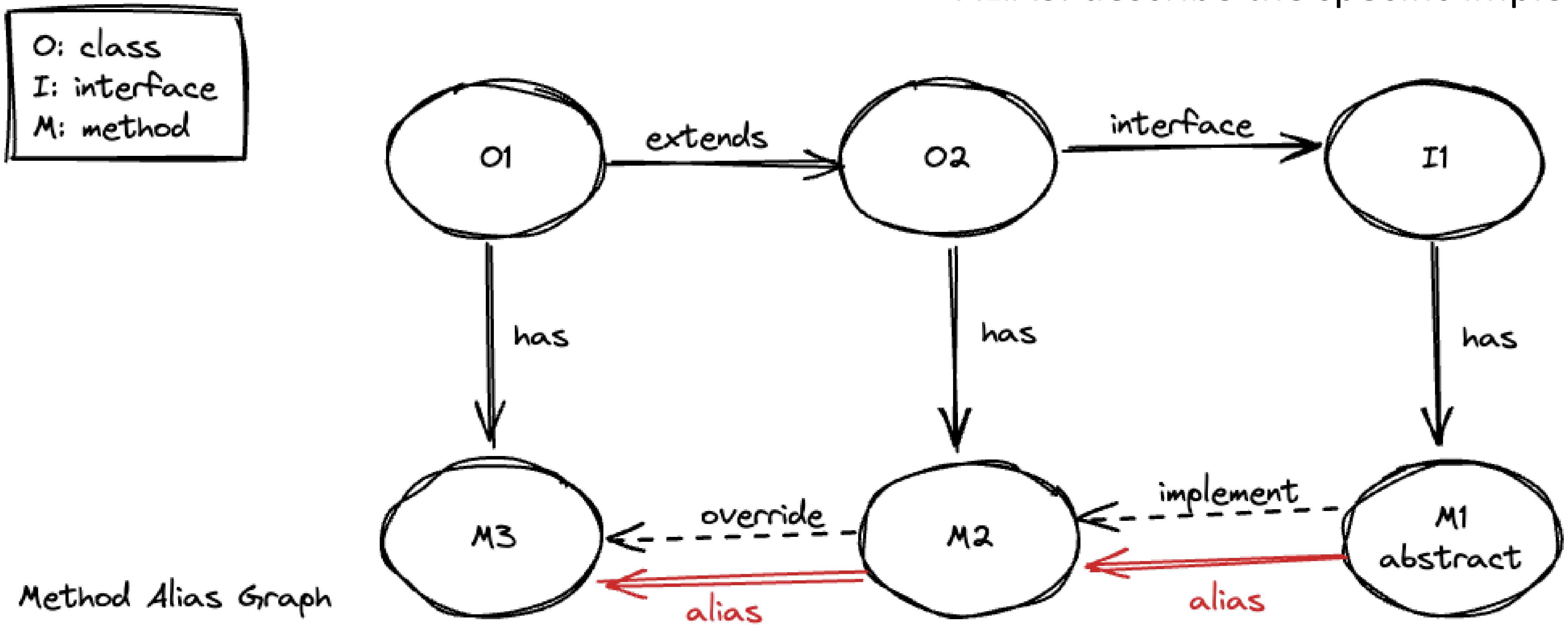
Object Relation Graph



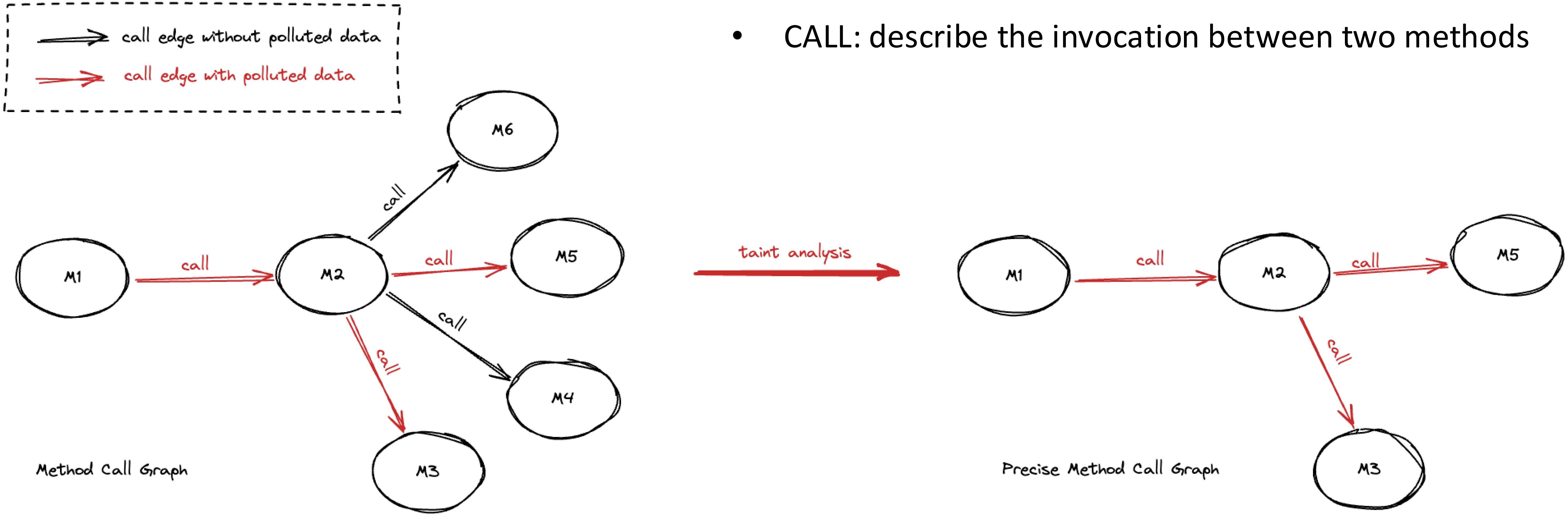
- HAS: describe the method related to class node.
- EXTENDS: describe the class related to its parent class node.
- INTERFACE: describe the class related to its interface class node.

Method Alias Graph

- ALIAS: describe the specific implementations of a method



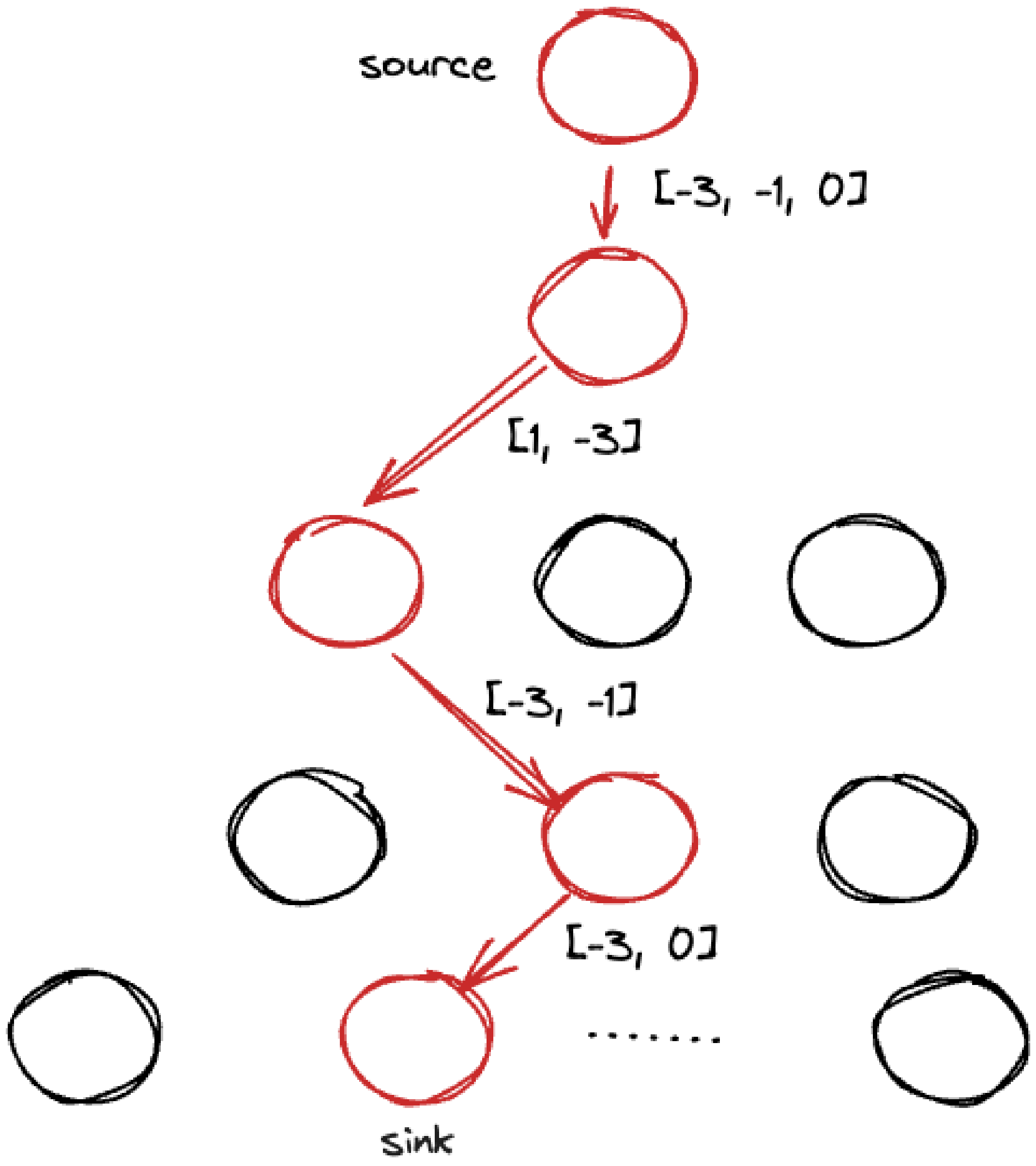
Method Call Graph



- CALL: describe the invocation between two methods

Taint analysis help to reduce the unnecessary call edges

Taint Analysis Engine

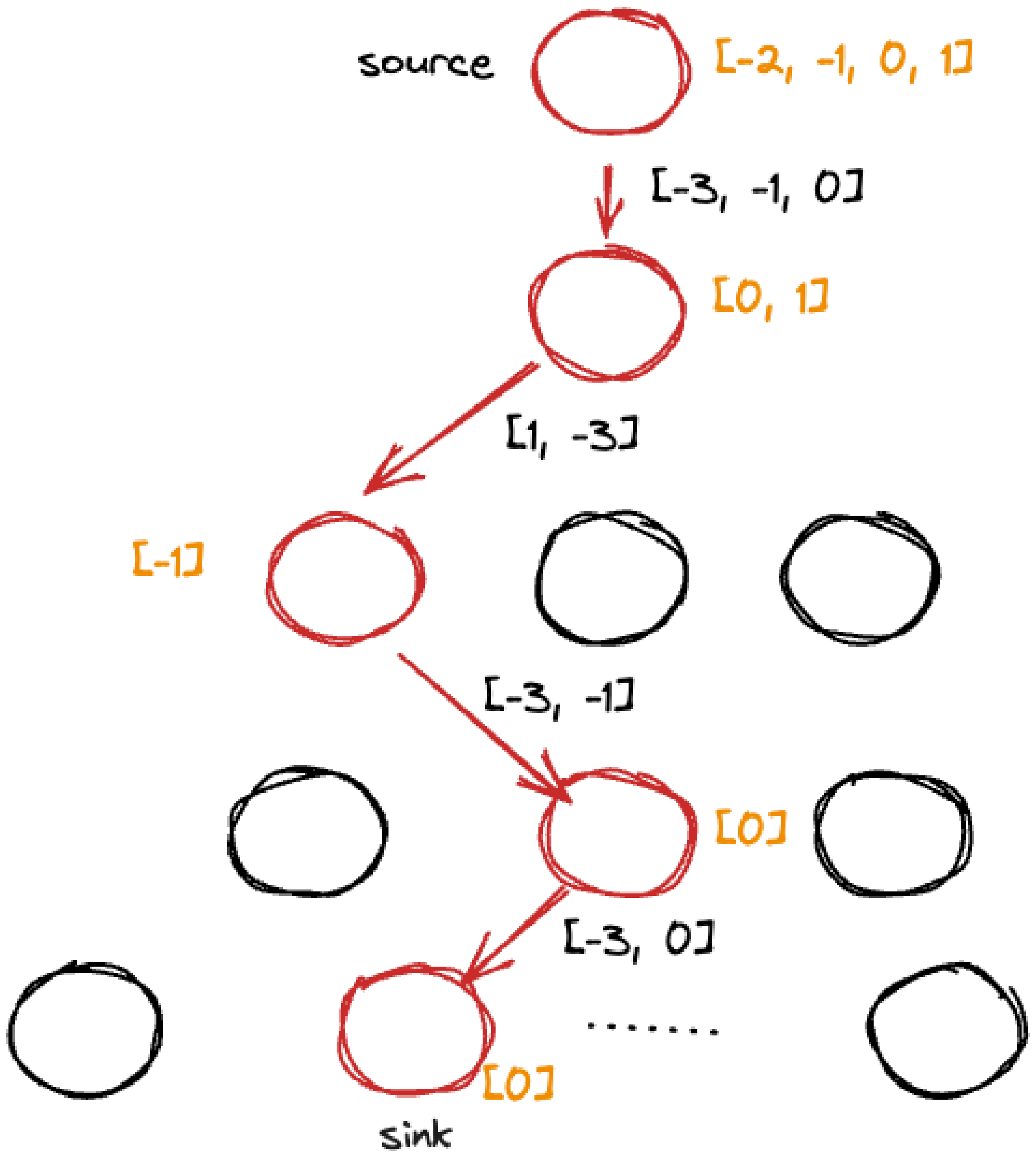


The Stage of Producing Sematic Cache on Memory

1. Apply the taint analysis algorithm to analyze each method body individually, recording the tainted data flows.
2. Store the tainted semantic cache in call edges, as illustrated by [-3, -1, 0] in the diagram.

This functionality is implemented in Tabby-Core.

Taint Analysis Engine



The Stage of Producing vulnerable call path on Neo4j

1. Start from the initial state and use the tainted semantic cache to calculate the temporary tainted state along the call path.

2. Remove function nodes that fail to properly propagate the tainted state during the calculation process.

3. Stop traversing further nodes when the taint analysis state reaches a sink function.

This functionality is implemented in Tabby-Path-Finder.

Application Scenarios & Demos

Finding vulnerabilities in real-world

Basic Usages

Finding Classic Web Application Vulnerabilities

- The cypher structure with three main parts include Source Node Identification, Sink Node Identification and Node Path Traversal.

```json
{
  "name": "servlet",
  "type": "class&method",
  "value": "web",
  "annotations": [],
  "classes": [
    "javax.servlet.Servlet",
    "javax.servlet.http.HttpServlet",
    "javax.servlet.GenericServlet"
  ],
  "methods": [
    "doGet", "doPost", "doPut", "doDelete",
    "doHead", "doOptions", "doTrace", "service"
  ],
  "whitelist": []
},
```

Servlet type

```json
{
  "name": "jsp",
  "type": "method",
  "value": "web",
  "annotations": [],
  "classes": [],
  "methods": [
    "_jspService"
  ],
  "whitelist": []
},
```

JSP compiled type

```json
{
  "name": "web-tags",
  "type": "annotation",
  "value": "web",
  "annotations": [
    "%Mapping",
    "javax.ws.rs.%",
    "javax.jws.%"
  ],
  "classes": [],
  "methods": [],
  "whitelist": []
},
```

Common Annotation type

```json
{
  "name": "struts actions",
  "type": "class",
  "value": "web",
  "annotations": [],
  "classes": [
    "com.opensymphony.xwork2.ActionSupport",
    "com.opensymphony.xwork2.Action",
    "org.apache.struts.actions.DispatchAction"
  ],
  "methods": [],
  "whitelist": [
    "com.opensymphony.xwork2.ActionSupport",
    "com.opensymphony.xwork2.Action",
    "org.apache.struts.actions.DispatchAction"
  ]
},
```

Struts type

```json
{
  "name": "netty-handler",
  "type": "class&method",
  "value": "netty",
  "annotations": [],
  "classes": [
    "io.netty.channel.ChannelInboundHandler",
    "org.jboss.netty.channel.SimpleChannelUpstreamHandler"
  ],
  "methods": [
    "channelRead",
    "channelRead0",
    "messageReceived"
  ],
  "whitelist": [
    "io.netty.channel.ChannelInboundHandler",
    "org.jboss.netty.channel.SimpleChannelUpstreamHandler"
  ]
},
```

Netty Handler type

```json
{
  "name": "netty-decoder",
  "type": "class&method",
  "value": "netty",
  "annotations": [],
  "classes": [
    "io.netty.handler.codec.ByteToMessageDecoder",
    "io.netty.handler.codec.MessageToMessageDecoder",
    "org.jboss.netty.handler.codec.frame.FrameDecoder"
  ],
  "methods": ["decode"],
  "whitelist": [
    "io.netty.handler.codec.ByteToMessageDecoder",
    "io.netty.handler.codec.MessageToMessageDecoder",
    "org.jboss.netty.handler.codec.frame.FrameDecoder"
  ]
},
```

Netty Decoder type

```
// normal web endpoint
MATCH (source:Method {IS_ENDPOINT: true})
// netty-style endpoint
MATCH (source:Method {IS_NETTY_ENDPOINT: true})
```

Custom Source Identification (tags.json)

Basic Usages

Finding Classic Web Application Vulnerabilities

- The cypher structure with three main parts include Source Node Identification, Sink Node Identification and Node Path Traversal.

```json
{"name": "java.lang.Runtime","rules": [
  {"function": "exec","type": "sink","vul": "EXEC","actions": {"return<s>": ["param-0"]},"polluted": [[0]],"max": 0},
  {"function": "load","type": "sink","vul": "CODE","actions": {},"polluted": [[0]],"max": 0},
  {"function": "loadLibrary","type": "sink","vul": "CODE","actions": {},"polluted": [[0]],"max": 0}
]},

{"name": "java.lang.ProcessBuilder","rules": [
  {"function": "<init>","type": "sink","vul": "EXEC","actions": {"this<s>": ["param-0"]},"polluted": [[0]],"max": 0}
]},
```

VUL:EXEC

```json
{"name": "org.apache.velocity.app.Velocity","rules": [
  {"function": "evaluate","type": "sink","vul": "CODE","actions": {},"polluted": [[3]],"max": 3}
]},

{"name": "org.springframework.expression.common.TemplateAwareExpressionParser","rules": [
  {"function": "parseExpression","type": "sink","vul": "CODE","actions": {"return<s>": ["param-0"]},"polluted": [[0]],"max": 1}
]},

{"name": "org.springframework.expression.Expression","rules": [
  {"function": "getValue","type": "sink","vul": "CODE","actions": {"return<s>": ["param-0"]},"polluted": [[-1]],"max": -1},
  {"function": "setValue","type": "sink","vul": "CODE","actions": {"return<s>": ["param-0"]},"polluted": [[0]],"max": 0}
]},
```

VUL:CODE

```json
{"name": "org.springframework.jndi.JndiTemplate","rules": [
  {"function": "lookup","type": "sink","vul": "JNDI","actions": {},"polluted": [[0]],"max": 0}
]},
```

VUL:JNDI

```json
{"name": "org.xml.sax.XMLReader","rules": [
  {"function": "parse","type": "sink","vul": "XXE","actions": {},"polluted": [[0]],"max": 0}
]},
```

VUL:XXE

- Tabby already has 125 built-in sinks.
- Tabby currently supports 10 VUL types, like SQLI、 CODE、 EXEC、 FILE、 FILE_WRITE、 JNDI、 SSRF、 REFLECTION、 XXE、 SERIALIZE.
- Tabby also supports complex sink identification by allowing users to extend the **Plugin** class. This enables the implementation of custom logic to identify new sinks or sources, providing greater flexibility in the analysis process.

```
// built-in sink endpoint
MATCH (sink:Method {IS_SINK: true})
// built-in sink endpoint with vul-type
MATCH (sink:Method {IS_SINK: true, VUL:"EXEC"})
```

Custom Sink Identification (sinks.json)

Basic Usages

Finding Classic Web Application Vulnerabilities

- The cypher structure with three main parts include Source Node Identification, Sink Node Identification and Node Path Traversal.

```
// tabby.beta.findPath
CALL tabby.beta.findPath(source, "-", sink, 8, false) YIELD path
// tabby.beta.findPathWithState
CALL tabby.beta.findPathWithState(source, "-", sink, "[[0]]", 8, false) YIELD path
// tabby.algo.findJavaGadget
CALL tabby.beta.findJavaGadget(source, "-", sink, 8, false) YIELD path
// tabby.algo.findJavaGadgetWithState
CALL tabby.beta.findJavaGadgetWithState(source, "-", sink, "[[0]]", 8, false) YIELD path
```

According to specific inference rules, **Tabby-Path-Finder** implements taint propagation and dynamic pruning on Neo4j by following the aforementioned procedures.

Built-in Path Traversal Procedure

Basic Usages

Finding Classic Web Application Vulnerabilities

- Example of finding command execution vulnerabilities

```
MATCH (source:Method {IS_ENDPOINT: true})
MATCH (sink:Method {IS_SINK: true, VUL:"EXEC"})
CALL tabby.beta.findPath(source, "-", sink, 8, false) YIELD path
RETURN path LIMIT 10
```

- Example of finding specific vulnerability by using tabby.beta.findPathWithState

```
MATCH (source:Method {IS_ENDPOINT: true})
MATCH (sink:Method {some_special_settings})
CALL tabby.beta.findPathWithState(source, "-", sink, "special_state", 8, false) YIELD path
RETURN path LIMIT 10
```

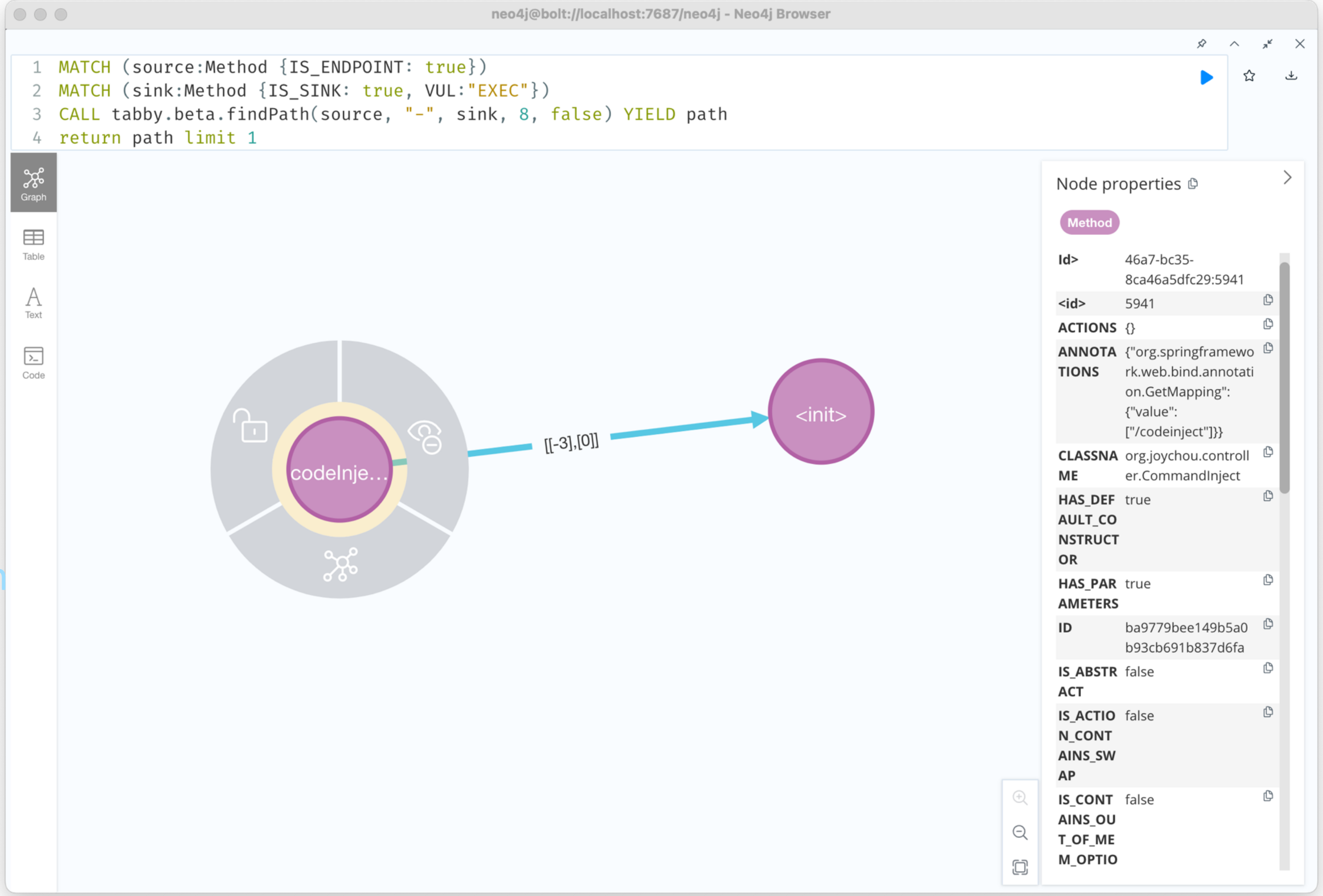
- Also using tabby.algo.findJavaGadget to find java deserialize gadget chains

Real-World Demos

#1 Finding Classic Web Application Vulnerabilities

- **Target**: Choose `java-sec-code` project which contains multiple java vulnerabilities as a demo.
- **Cypher**: As demo, set up the VUL as `EXEC` to find command execution vulnerability.

```
MATCH (source:Method {IS_ENDPOINT: true})
MATCH (sink:Method {IS_SINK: true, VUL:"EXEC"})
CALL tabby.beta.findPath(source, "-", sink, 8, false) YIELD path
return path limit
```



Simply use the above cypher template to complete vulnerability mining of common web vulnerabilities.

Reference:

- https://github.com/JoyChou93/java-sec-code

Real-World Demos

#2 Finding Deserialization Gadget Chains | XStream Gadget

- **Target:** Find some gadget in JDK8 to bypass XStream 1.4.16 blacklist.

Some prior knowledges about CVE-2021-29505 which successfully exploited the version 1.4.15.

```
javax.naming.ldap.Rdn$RdnEntry#compareTo
com.sun.org.apache.xpath.internal.objects.XString#equal
com.sun.xml.internal.ws.api.message.Packet#toString
com.sun.xml.internal.ws.message.saaj.SAAJMessage#copy
com.sun.xml.internal.ws.message.saaj.SAAJMessage#getAttachments
com.sun.xml.internal.ws.message.saaj.SAAJMessage$SAAJAttachmentSet#<init>
com.sun.xml.internal.messaging.saaj.soap.ver1_1.Message1_1Impl#getAttachments
com.sun.xml.internal.messaging.saaj.soap.ver1_1.Message1_1Impl#initializeAllAttachments
com.sun.xml.internal.messaging.saaj.packaging.mime.internet.MimePullMultipart#getCount
com.sun.xml.internal.messaging.saaj.packaging.mime.internet.MimePullMultipart#parse
com.sun.xml.internal.messaging.saaj.packaging.mime.internet.MimePullMultipart#parseAll
com.sun.xml.internal.org.jvnet.mimepull.MIMEMessage#getAttachments
com.sun.xml.internal.org.jvnet.mimepull.MIMEMessage#parseAll
com.sun.xml.internal.org.jvnet.mimepull.MIMEMessage#makeProgress
com.sun.org.apache.xml.internal.security.keys.storage.implementations.KeyStoreResolver$KeyStoreIterator#hasNext
com.sun.org.apache.xml.internal.security.keys.storage.implementations.KeyStoreResolver$KeyStoreIterator#findNextCert
com.sun.jndi.toolkit.dir.LazySearchEnumerationImpl#nextElement
com.sun.jndi.toolkit.dir.LazySearchEnumerationImpl#findNextMatch
com.sun.jndi.rmi.registry.BindingEnumeration#next
sun.rmi.registry.RegistryImpl_Stub#lookup
```

Hit
```
Pattern.compile(
        ".*\\.Lazy(?:Search)?Enumeration.*");
```

Hit
```
Pattern.compile("(?:java|sun)\\.rmi\\..*");
```

CVE-2021-29505

Not in xstream 1.4.16 blacklist

Hit by blacklist

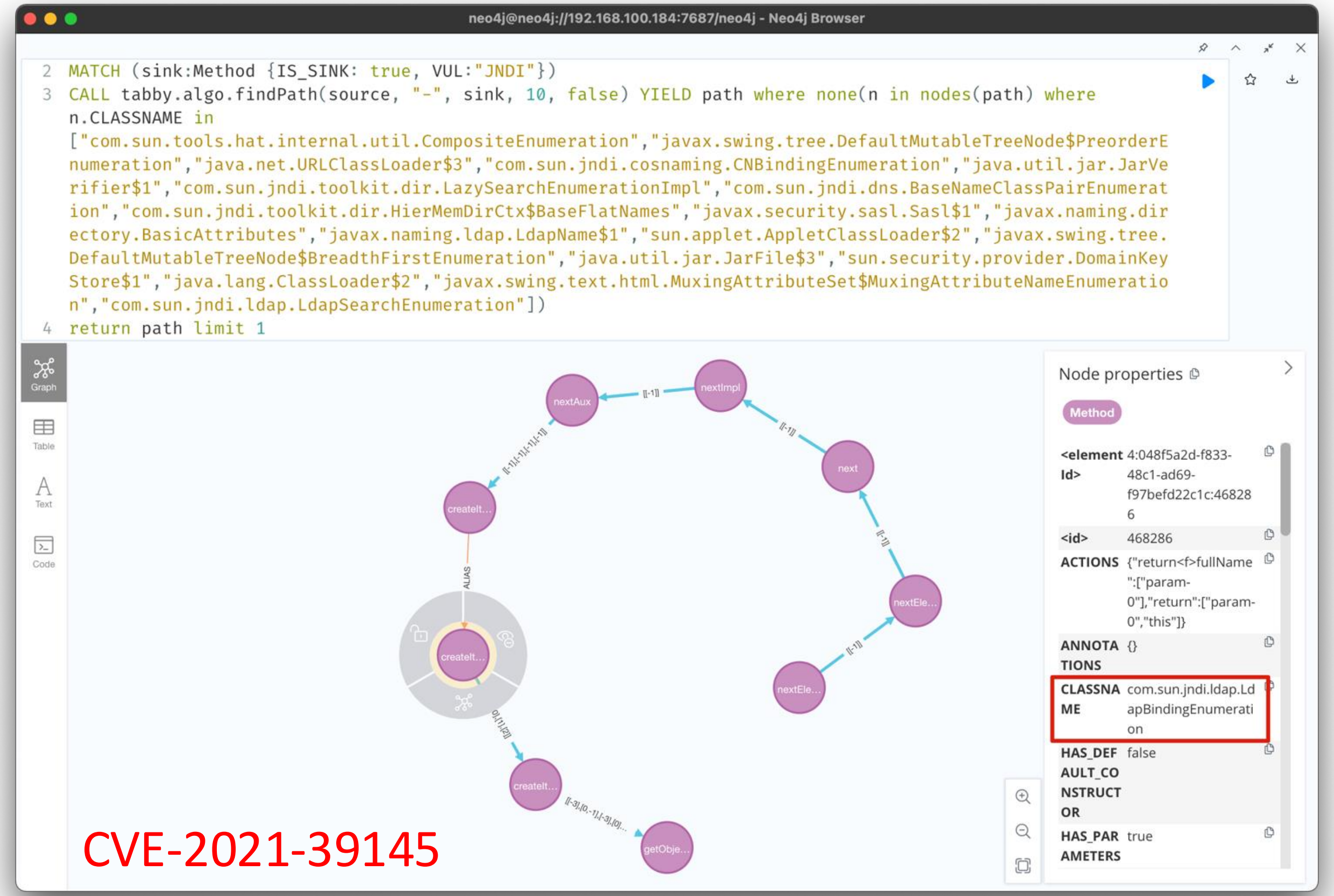
Just find another `nextElement` function to a sink function

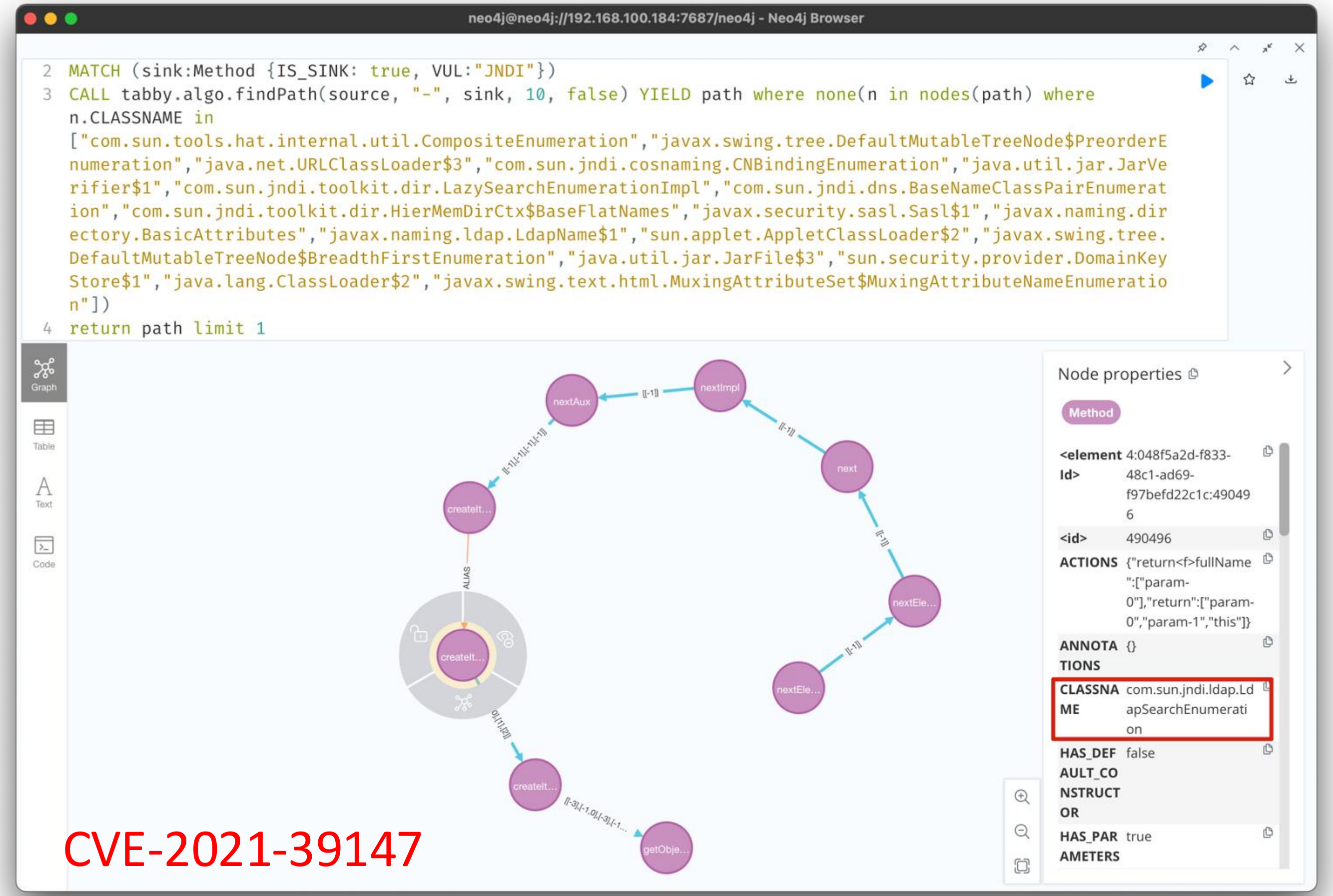
Real-World Demos

#2 Finding Deserialization Gadget Chains | XStream Gadget

- **Target:** Find some gadget in JDK8 to bypass XStream 1.4.16 blacklist.

Find a new gadget to replace the one affected by the XStream 1.4.16 blacklist.



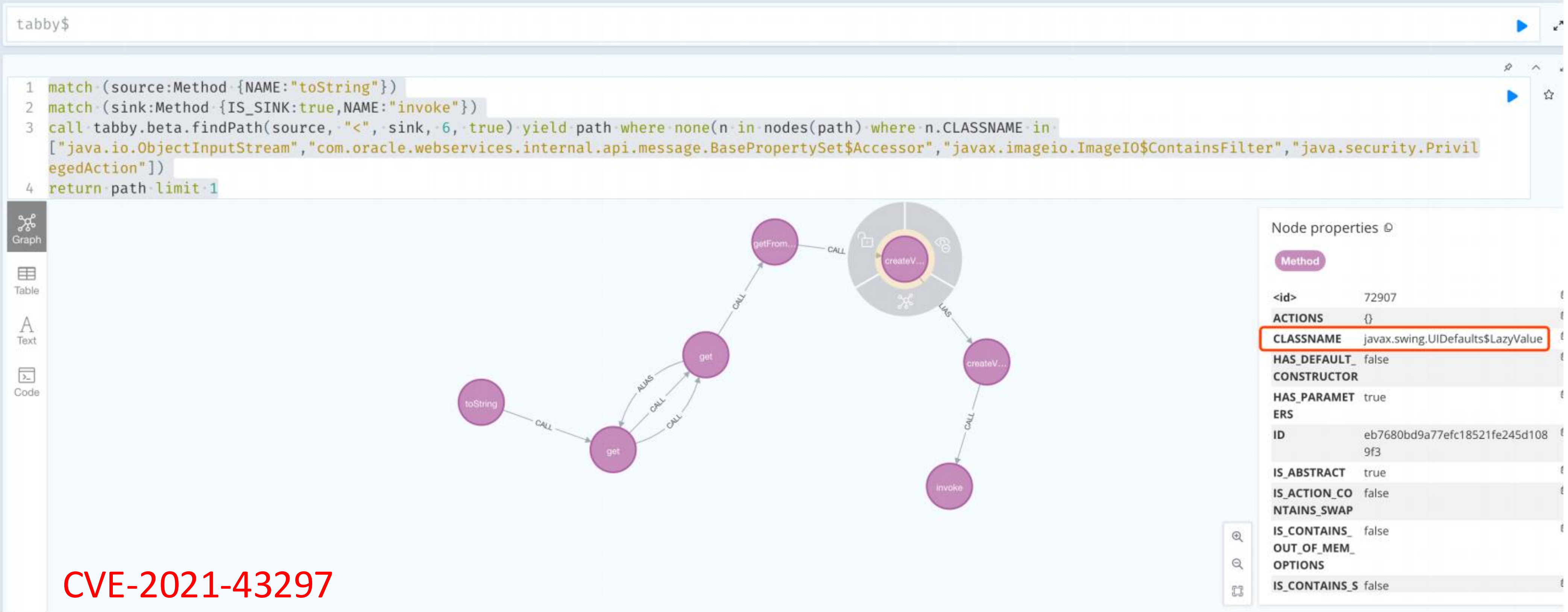


Real-World Demos

#3 Finding Deserialization Gadget Chains | Hessian Gadget

- Target:

1. Dubbo will trigger some magic functions like `toString` func.
2. We needs to find a gadget which not in dubbo’s blacklist



Real-World Demos

#4 Finding Specific Setter Functions | CobaltStrike CVE-2022-39197

- Target:

1. Package of Swing will trigger any component's setters.
2. Setter's parameters needs to contain `java.lang.String`.

Use the property `IS_SETTER` and specific limitation like the pic showed

```
match (source:Method {IS_SETTER:true})<-[:HAS]-(c:Class)-[:INTERFACE|EXTENDS*]->(c1:Class {NAME:"java.awt.Component"}) where
source.SUB_SIGNATURE =~ ".*java.lang.String.*"
return count(source)
```

	count(source)
1	53

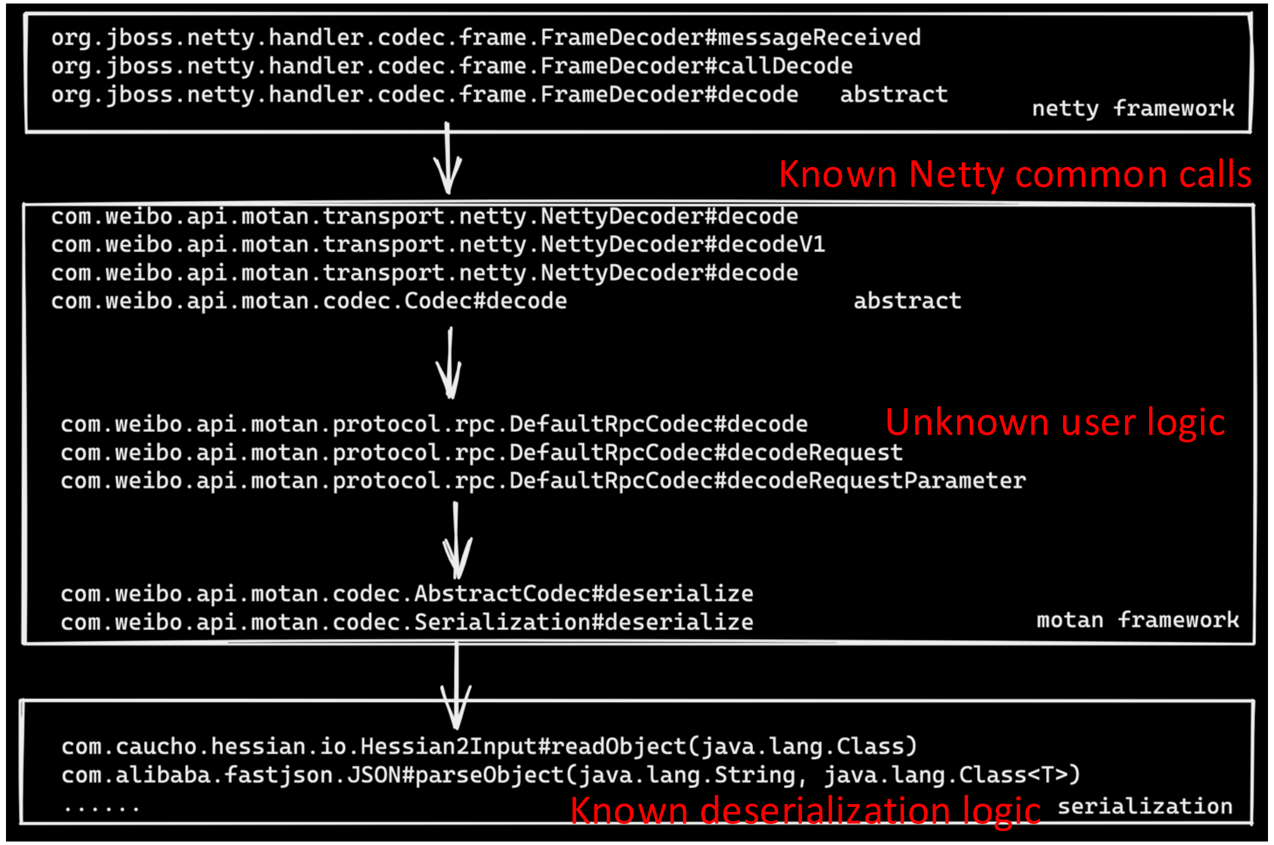
Started streaming 1 records after 2 ms and completed after 52 ms.

Real-World Demos

#5 Finding Vulnerabilities in Netty-Style RPC Framework

- **Target:** Finding the deserialization vulnerabilities in Motan RPC Framework

Netty-style RPC framework always need developer to implement decoder or handler. In general, that implements will contains deserialization. So just finding the path between decoder or handler to deserialization functions.

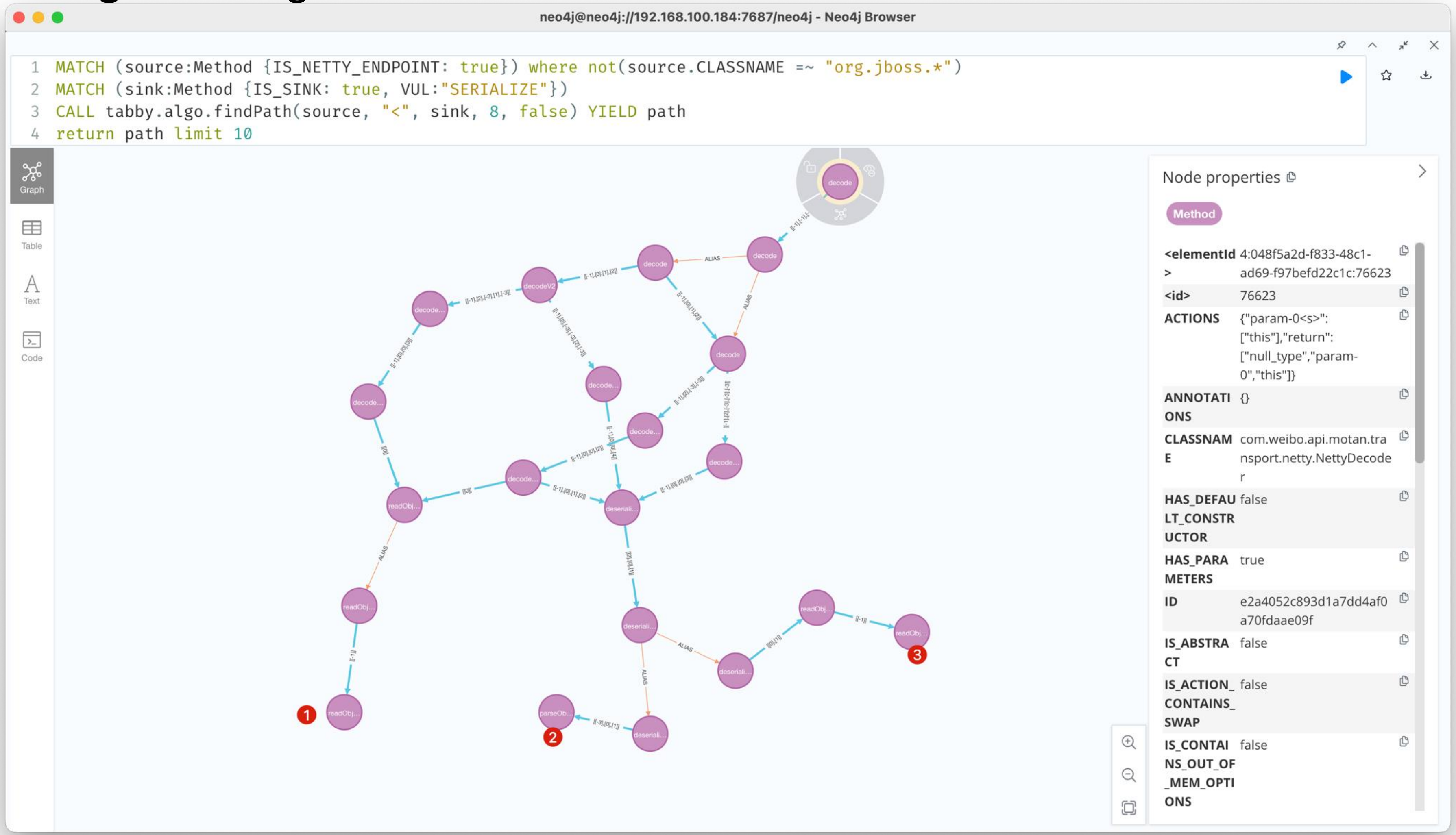



```
MATCH(source:Method {IS_NETTY_ENDPOINT:true})
MATCH(sink:Method {IS_SINK:true, VUL:"SERIALIZABLE"})
CALL tabby.beta.findPath(source, "-", sink, 8, false) YIELD path
return path limit 1
```

Real-World Demos

#5 Finding Vulnerabilities in Netty-Style RPC Framework

- **Target:** Finding the deserialization vulnerabilities in Motan RPC Framework



Using the Cypher queries above, we can identify three vulnerable function invocation paths:

1. **Java-Origin Deserialization:** java.io.ObjectInputStream#readObject
2. **FastJSON Deserialization:** com.alibaba.fastjson.JSON#parseObject
3. **Hessian Deserialization**: com.caucho.hessian.io.Hessian2Input#readObject

Automated Workflow

```yaml
name: "web_to_code"
enable: false
type: "web"
source:
  type: "source"
  endpoint: true
  name: "test"
  name0: "test"
  ...
sink:
  type: "sink"
  sink: true
  vul: "CODE"
  ...
depth: 8
limit: 10
depthFirst: false
direct: "-"
procedure: "tabby.beta.findPath"
sourceBlacklists: []
pathBlacklists: []
```

```yaml
name: "web_to_specific_sink"
enable: false
type: "web"
source:
  type: "source"
  endpoint: true
  name: "test"
  name0: "test"
  ...
sink:
  type: "sink"
  cypher: "..."
  ...
depth: 8
limit: 10
depthFirst: false
direct: "-"
procedure: "tabby.beta.findPathWithState"
sinkState: "[[0]]"
sourceBlacklists: []
pathBlacklists: []
```

Automated Workflow by using YAML Conf

1. **Set Up Configuration:** Use predefined templates to create the required configuration.

2. **Load CSV Files and Query**: Import CSV files and execute queries on the project based on the configuration.

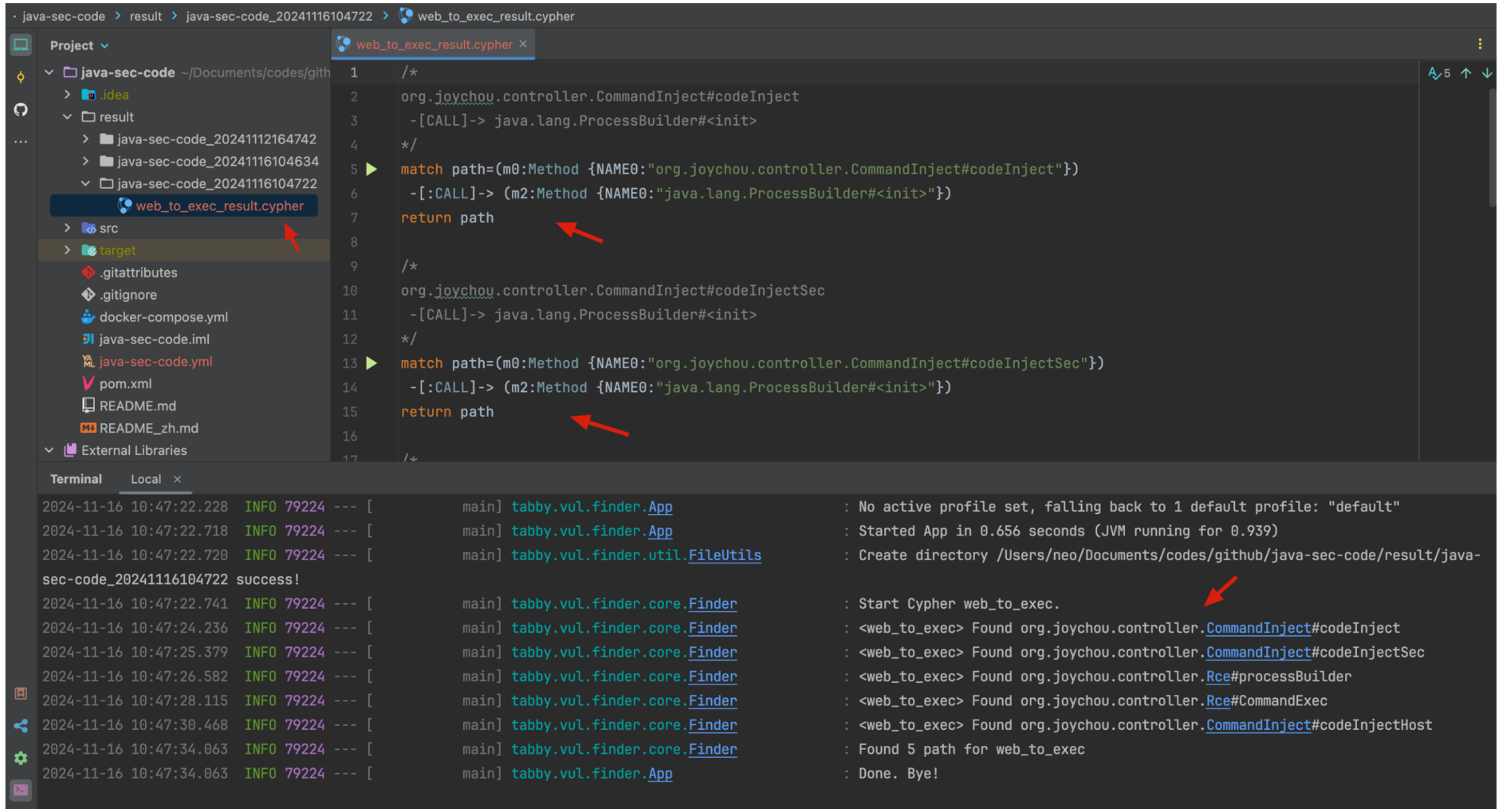
3. **Review Results**: After the query execution is complete, examine the function invocation paths outlined in the .cypher files.

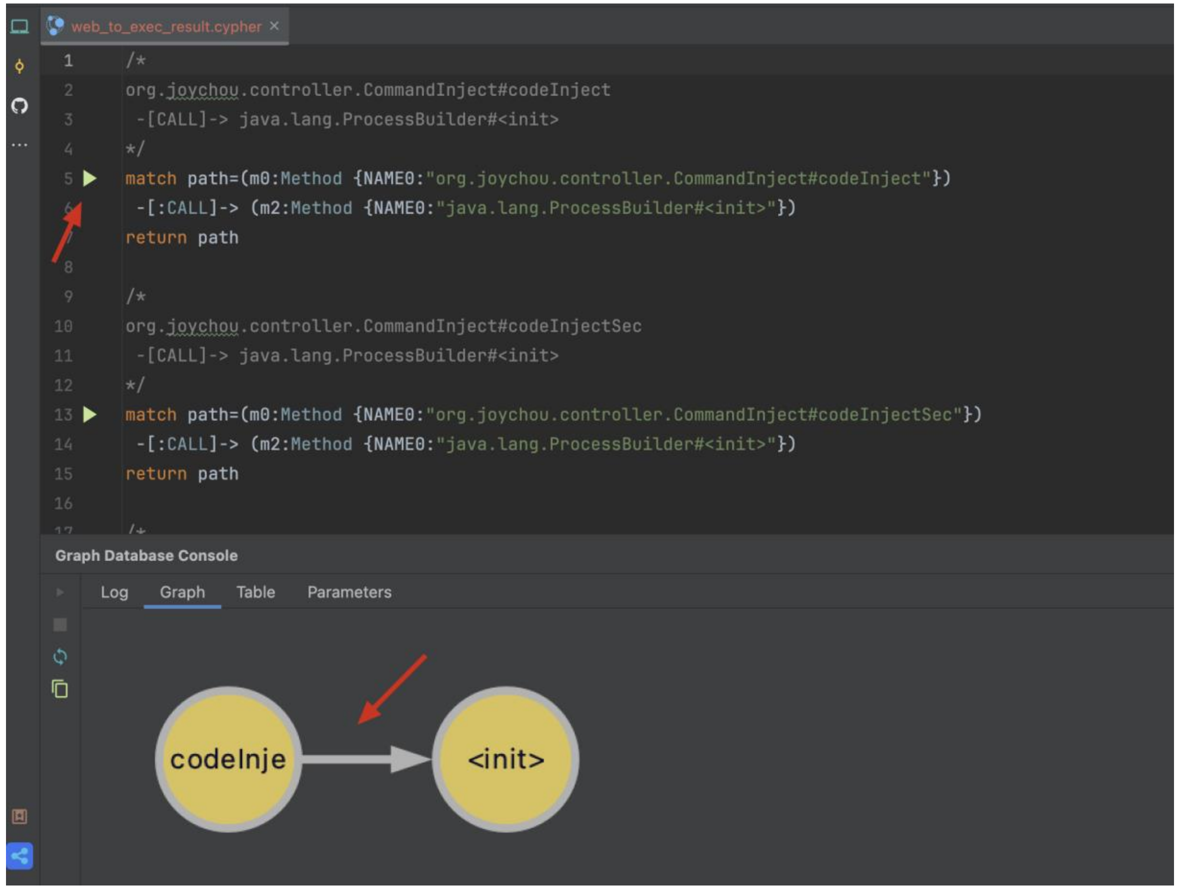
References:

- Examples of configuration, https://github.com/wh1t3p1g/tabby-vul-finder/tree/main/rules

Automated Workflow

Example of `java-sec-code` project by using yaml configuration





1. **Tabby-Vul-Finder** generates several .cypher files, each containing multiple Cypher statements.
2. Use the **Tabby IntelliJ Plugin** to execute these Cypher queries and quickly locate the source code by double-clicking on nodes or edges.

What's Next

- **Enhance Taint Analysis Engine**:
  - Support analysis of dynamic reflection mechanisms.
  - Improve threading code analysis.
  - ...

- **Expand Built-in Rules**:
  - Introduce additional rules to enhance the effectiveness of the analysis.

- **Address Cross-Language and Cross-Application Scenarios**:
  - Enable seamless analysis across languages and applications using the CPG.

- **Optimize Performance**:
  - Under limited memory conditions: Enable analysis of large-scale applications.
  - Under unlimited memory conditions: Deliver fast and precise analysis.

TABBY

Feel free to share your ideas or report issues on GitHub.
https://github.com/wh1t3p1g/tabby

Thanks