# 程序预分析&存储（binary\_analysis\_frontend.py）

本部分用于构建 SPDG图（semantics-based program dependency graph），结合控制流分析、数据依赖分析和符号执行技术，支持结果同步到 Neo4j 图数据库。

初始化参数说明

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **参数** | **类型** | **必填** | **默认值** | **说明** |
| binary\_path | str | 是 | 无 | 待分析二进制文件路径 |
| cfg\_type | str | 否 | "emulated" | CFG类型 ("fast"/"emulated") |
| cfg\_options | dict | 否 | 见下方 | CFG构建配置 |
| data\_dep\_enabled | bool | 否 | True | 是否启用数据依赖分析 |
| data\_dep\_options | dict | 否 | 见下方 | 数据依赖分析配置 |
| symbolic\_enabled | bool | 否 | True | 是否启用符号执行 |
| symbolic\_options | dict | 否 | 见下方 | 符号执行配置 |
| neo4j\_enabled | bool | 否 | True | 是否启用Neo4j同步 |
| neo4j\_options | dict | 否 | 见下方 | Neo4j连接配置 |
| log\_level | int | 否 | logging.INFO | 日志级别 |

**CFG 配置 (cfg\_options)**

{

'context\_sensitivity\_level': 2, # 上下文敏感级别 (范围1-3)

'resolve\_indirect\_jumps': True, # 解析间接跳转

'cross\_references': True, # 跨函数引用分析

'collect\_data\_refs': True, # 收集数据引用

'normalize': True # 标准化处理

}

**数据依赖配置 (data\_dep\_options)**

{

'track\_tmps': True, # 跟踪临时变量

'track\_memory': True, # 跟踪内存访问

'def\_use\_threshold': None, # 定义-使用阈值

'cross\_function': True # 跨函数分析

}

**符号执行配置 (symbolic\_options)**

{

'max\_steps': 100, # 最大执行步数

'timeout': None, # 超时设置（秒）

'avoid\_loops': True, # 避免循环执行

'concretize\_memory': False, # 具体化内存访问

'symbolic\_memory': True # 符号化内存

}

**Neo4j 配置 (neo4j\_options)**

{

'uri': "bolt://localhost:7687", # 数据库URI

'auth': ("neo4j", "password"), # 认证信息

'batch\_size': 1000, # 批量操作大小

'sync\_nodes': True, # 同步节点

'sync\_edges': True, # 同步边

'sync\_constraints': True # 同步约束条件

}

# 查询引擎(spdg\_query\_engine.py)

本模块提供针对 SPDG的多维度查询能力，支持从基础图查询到符号执行验证的多层次分析。

#### 初始化参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **必填** | **说明** |
| neo4j\_graph | Graph | 是 | Neo4j 图数据库连接 |
| angr\_project | Project | 否 | Angr 项目实例（符号执行需要） |

支持的查询类型

|  |  |
| --- | --- |
| **查询类型** | **说明** |
| CYPHER | 原始Cypher查询 |
| SEMANTIC | 语义模式匹配查询 |
| DATAFLOW | 数据流追踪查询 |
| CONSTRAINED\_PATH | 带符号执行约束的路径验证 |
| VISUALIZATION | 可视化图表生成 |

**动态参数说明**（根据查询类型变化）

|  |  |  |  |
| --- | --- | --- | --- |
| **查询类型** | **参数** | **类型** | **验证规则** |
| CYPHER | cypher\_pattern | str | 合法Cypher语句 |
|  | cypher\_params | dict | 参数键值对 |
| SEMANTIC | instructions\_regex | str | 合法正则表达式 |
|  | constraints\_regex | str | 合法正则表达式 |
| DATAFLOW | var\_name | str | 非空字符串 |
|  | max\_depth | int | >0 的整数 |
| CONSTRAINED\_PATH | start\_addr | int/str | 有效地址格式 |
|  | end\_addr | int/str | 有效地址格式 |
|  | constraints | list[str] | 符号表达式列表 |
| VISUALIZATION | path\_addrs | list[int/str] | 有效地址列表 |

查询使用示例

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **查询类型** | **应用场景** | **问题类型** | **典型漏洞** | **示例场景** | **示例查询语句** |
| 基础约束查询 | 安全输入验证分析 | 检测程序是否对用户输入进行有效性检查 | 缓冲区溢出、注入攻击 | 识别所有处理网络数据包输入（stdin\_81\_480）的代码块，验证是否包含非法字符过滤逻辑 | params = QueryParams(  query\_type=QueryType.CYPHER,  cypher\_pattern="""  MATCH (b:Block)  WHERE ANY(constr IN b.constraints WHERE constr CONTAINS $pattern)  RETURN b.address AS addr, b.instructions AS insns  """,  cypher\_params={'pattern': 'stdin\_81\_480'}  )  stdin\_checks = engine.execute\_query(params) |
| 指令+约束组合查询 | 危险函数调用分析 | 识别未经验证的敏感操作 | 命令注入、内存越界访问 | 发现未经验证直接将用户输入（rax寄存器）传递给系统调用（call 0x401080）的代码模式 | params = QueryParams(  query\_type=QueryType.SEMANTIC,  instructions\_regex=r"mov\s+rdi,.\*rax",  constraints\_regex=r"stdin\_81\_480.\*!="  )  pointer\_checks = engine.execute\_query(params) |
| 精确位检查查询 | 协议规范合规性验证 | 检测协议解析是否符合RFC标准 | 协议解析错误 | 验证HTTP头解析是否正确过滤控制字符（如\n/\r） | params = QueryParams(  query\_type=QueryType.CYPHER,  cypher\_pattern=r"""  MATCH (b:Block)  WHERE ANY(constr IN b.constraints  WHERE constr CONTAINS '[479:472]'  AND (constr CONTAINS '!= 10' OR constr CONTAINS '!= 0xA'))  RETURN  b.address AS address,  [c IN b.constraints WHERE c CONTAINS '[479:472]'] AS constraints,  b.instructions AS instructions  LIMIT 20  """  ) |
| 多条件约束查询 | 输入过滤完整性检查 | 检测白名单/黑名单过滤的完备性 | 过滤绕过漏洞 | 检查空白字符过滤是否覆盖TAB/CR/LF等所有控制字符 | params = QueryParams(  query\_type=QueryType.CYPHER,  cypher\_pattern="""  MATCH (b:Block)  WHERE SIZE([c IN b.constraints WHERE  c CONTAINS $c1 OR  c CONTAINS $c2 OR  c CONTAINS $c3 OR  c CONTAINS $c4]) >= $min\_count  RETURN b.address AS addr  """,  cypher\_params={  'c1': '!= 9>',  'c2': '!= 10>',  'c3': '!= 11>',  'c4': '!= 13>',  'min\_count': 3  }  )  whitespace\_checks = engine.execute\_query(params) |
| 数据流追踪增强查询 | 污点传播分析 | 追踪敏感数据流向 | 信息泄露、未初始化内存使用 | 追踪网络数据包（stdin\_81\_480）是否未经消毒直接用于内存分配参数 | params = QueryParams(  query\_type=QueryType.DATAFLOW,  var\_name="stdin\_81\_480",  max\_depth=5  )  taint\_paths = engine.execute\_query(params) |
| 带符号约束路径验证 | 漏洞利用可行性验证 | 确认攻击路径可达性 | 逻辑漏洞 | 验证攻击者是否可通过构造特定输入（如字节值65/66）触发危险函数调用 | params = QueryParams(  query\_type=QueryType.CONSTRAINED\_PATH,  start\_addr=0x401000,  end\_addr="0x401200",  constraints=[  "state.solver.BVS('stdin\_81\_480', 8) == 65",  "state.solver.BVS('stdin\_81\_480', 8) == 66"  ]  )  valid\_paths = engine.execute\_query(params) |
| 复杂指令模式匹配 | 函数调用模式识别 | 逆向工程中的功能识别 | - | 定位加密算法初始化函数（如识别lea rax加载密钥表的模式） | params = QueryParams(  query\_type=QueryType.CYPHER,  cypher\_pattern="""  MATCH (b:Block)  WHERE b.instructions =~ $insn\_regex  RETURN b.address AS call\_site  """,  cypher\_params={'insn\_regex': r'.\*lea rax,.\*; mov rdi, rax; call.\*'}  )  call\_preps = engine.execute\_query(params) |
| 约束有效性验证查询 | 代码逻辑矛盾检测 | 发现程序中的矛盾约束 | 死代码、逻辑错误 | 检测同一代码块中同时存在var != 9和var == 9的矛盾约束 | params = QueryParams(  query\_type=QueryType.CYPHER,  cypher\_pattern="""  MATCH (b:Block)  WHERE ANY(c1 IN b.constraints WHERE c1 CONTAINS $c1)  AND ANY(c2 IN b.constraints WHERE c2 CONTAINS $c2)  RETURN b.address AS conflict\_block  """,  cypher\_params={  'c1': '!= 9>',  'c2': '== 9>'  }  )  constraint\_conflicts = engine.execute\_query(params) |
| 路径可视化 | 漏洞原理分析 | 理解漏洞触发路径 | - | 生成从用户输入点到危险操作的完整路径图 | - |