How It Works

\_\_main\_\_.py 모든 큰 흐름이 이루어지는 파일.

Step 1

커맨드 라인 인자들을 해석한다.

usage.py에 있는 parse\_args 함수에서 이루어짐

Step 2

추상 구문 트리(AST)를 실행한다.

결정적으로 ast 모듈의 이 코드 라인에서 이루어진다.

ast\_helper.py의 generate\_ast 함수에서 이루어짐

Step 3

AST를 거쳐 제어 흐름 그래프(CFG)를 만든다.

\_\_main\_\_.py에서 make\_cfg 함수를 호출하는데, 이는 AST를 통해 ExprVisitor를 생성하고 CFG를 반환한다.

stmt\_visitor.py(statements)와 expr\_visitor.py(expressions)는 Python의 추상 문법을 비춘다. statements는 expressions(표현식)을 포함할 수 있지만 그 반대의 식은 포함할 수 없습니다. 이것이 ExprVisitor가 표준 라이브러리인 ast.NodeVisitor의 자식인 StmtVisitor로부터 상속받는 이유이다. 이하는 ast.NodeVisitor의 작동원리이다.

def visit(self, node):

"""Visit a node."""

method = 'visit\_' + node.\_\_class\_\_.\_\_name\_\_

visitor = getattr(self, method, self.generic\_visit)

return visitor(node)

보시다시피 거의 모든 AST 노드 타입에는 visit 함수가 있다. 우리는 방문하는 동안 모든 노드를 self.node에 추가하고, ingoing과 outgoing 노드 속성을 통해 모든 노드를 연결하여 계속 추적합니다.

stmt\_star\_handler와 expr\_star\_handler가 2개의 가장 대표적인 함수라 할 수 있다.

expr\_star\_handler는 아직 master branch에 통합되지 않았기 때문에 stmt\_star\_handler에 대해 이야기해보면 좋을 것 같다.

if:문 다루기

Step 4

CFG를 거쳐 web\_frameworks의 framework\_adaptor로 이동한다. 여기서 특정 함수들의 인자들을 감염된 소스로 마크할 것이다.

framework\_helper.py 파일에는 현재 4가지 framework\_route\_criteria가 있다.

코드 작동 원리

FrameworkAdaptor는 \_\_main\_\_.py가 만드는데, 이것은 —adaptor cli 인자를 통해 선택되어진다. framework\_route\_criteria는 ast.FunctionDef를 가지고 있고 선택한 웹 프레임워크의 경로인지 여부를 반환하는 함수이다.

노드들을 돌아다니면서 더럽혀진 인자들을 마크하고 그들을 TaintedNode 타입으로 만든다. 그리고 그들을 소스의 리스트에 추가한다.

Step 5

reaching definitions analysis(도달정의분석)를 실행한다. 정의가 어디에 도착하는지 알기 위해.

Step 6.

소스가 어디에, 어떻게 도달하는지를 관찰하여 취약점을 발견한다.

Step 7.

baseline(이전 PyT를 실행했을 때의 JSON파일)이 제공되었다면 이미 알려진 취약점들을 제거한다.

Step 8

결과를 text(표준출력)나 JSON(출력파일) 형태로 출력한다.