Análise de `constraints` no módulo de detecção de vulnerabilidades `External_calls`

Definição de Constraints

```python
constraints = Constraints(
    [UGT(gas, symbol_factory.BitVecVal(2300, 256)), to ==
ACTORS.attacker]
)
```

O objetivo dessa configuração é verificar duas condições principais:

1. **Se o gás disponível (`gas`) é maior que 2300.**
2. **Se o endereço de destino (`to`) corresponde ao atacante (`ACTORS.attacker`).**

Processo de Análise

Método `_analyze_state`

A análise começa no método `_analyze_state`, que recebe como entrada um `GlobalState` contendo informações sobre a execução atual do contrato.

1. **Checagem Inicial**:
   - Verifica se não estamos no construtor (momento da implantação do contrato).
2. **Obtenção de Valores da Pilha**:
   - `gas`: Representa o gás disponível (nas informações de debug aparece como `2_gas`).
   - `to`: O endereço de destino, que é uma composição complexa obtida a partir do `calldata`:

```python
Concat(0,
       If(1_calldatasize <= 12, 0, 1_calldata[12]),
       If(1_calldatasize <= 13, 0, 1_calldata[13]),
       If(1_calldatasize <= 14, 0, 1_calldata[14]),
       If(1_calldatasize <= 15, 0, 1_calldata[15]),
       If(1_calldatasize <= 16, 0, 1_calldata[16]),
       If(1_calldatasize <= 17, 0, 1_calldata[17]),
```

```
    If(1_calldatasize <= 18, 0, 1_calldata[18]),
    If(1_calldatasize <= 19, 0, 1_calldata[19]),
    If(1_calldatasize <= 20, 0, 1_calldata[20]),
    If(1_calldatasize <= 21, 0, 1_calldata[21]),
    If(1_calldatasize <= 22, 0, 1_calldata[22]),
    If(1_calldatasize <= 23, 0, 1_calldata[23]),
    If(1_calldatasize <= 24, 0, 1_calldata[24]),
    If(1_calldatasize <= 25, 0, 1_calldata[25]),
    If(1_calldatasize <= 26, 0, 1_calldata[26]),
    If(1_calldatasize <= 27, 0, 1_calldata[27]),
    If(1_calldatasize <= 28, 0, 1_calldata[28]),
    If(1_calldatasize <= 29, 0, 1_calldata[29]),
    If(1_calldatasize <= 30, 0, 1_calldata[30]),
    If(1_calldatasize <= 31, 0, 1_calldata[31]))
```

- Essa composição utiliza instruções `Concat` e `If` para processar cada byte do `calldata` e construir o endereço.

Criação das Restrições

1. **Restrição 1: `UGT(gas, 2300)`**
    - Verifica se o gás disponível é maior que 2300.
      Processo:
    - `BitVecVal(2300, 256)`: Cria um valor inteiro de 256 bits para representar 2300.
    - A comparação é feita usando `UGT` (Unsigned Greater Than):
        - Internamente, `UGT` utiliza `_comparison_helper` para criar a expressão.
        - `_comparison_helper` combina as anotações dos operandos e gera um objeto `Bool` representando `Z3.UGT(gas, 2300)`.
2. **Restrição 2: `to == ACTORS.attacker`**
    - Verifica se o endereço de destino é igual ao do atacante (`1271270613000041655817448348132275889066893754095`).
      Processo:
    - `ACTORS.attacker`: Representado como um valor fixo de 256 bits.
    - O valor de `to` é a composição detalhada do `calldata`.

- A comparação utiliza o operador `==`, implementado pelo método `__eq__` da classe `BitVec`.
- Antes da comparação, a função `_padded_operation` é usada para garantir que ambos os operandos (`to` e `ACTORS.attacker`) tenham o mesmo tamanho:
  - Se necessário, um dos valores é preenchido com zeros (`padding`) à esquerda.

Objeto `Constraints`

- A classe `Constraints` encapsula as restrições em uma estrutura unificada:
  - Internamente, herda de `list` para armazenar os objetos `Bool` que representam as condições.
  - O método `_get_smt_bool_list` verifica que todas as expressões na lista são do tipo `Bool`.

Exemplo Final de `constraints`:

```python
constraints = [UGT(2_gas, 2300),
127127061300004165581744834813227588906689375409 5 ==
Concat(0,
       If(1_calldatasize <= 12, 0, 1_calldata[12]),
       If(1_calldatasize <= 13, 0, 1_calldata[13]),
       If(1_calldatasize <= 14, 0, 1_calldata[14]),
       If(1_calldatasize <= 15, 0, 1_calldata[15]),
       If(1_calldatasize <= 16, 0, 1_calldata[16]),
       If(1_calldatasize <= 17, 0, 1_calldata[17]),
       If(1_calldatasize <= 18, 0, 1_calldata[18]),
       If(1_calldatasize <= 19, 0, 1_calldata[19]),
       If(1_calldatasize <= 20, 0, 1_calldata[20]),
       If(1_calldatasize <= 21, 0, 1_calldata[21]),
       If(1_calldatasize <= 22, 0, 1_calldata[22]),
       If(1_calldatasize <= 23, 0, 1_calldata[23]),
       If(1_calldatasize <= 24, 0, 1_calldata[24]),
       If(1_calldatasize <= 25, 0, 1_calldata[25]),
       If(1_calldatasize <= 26, 0, 1_calldata[26]),
       If(1_calldatasize <= 27, 0, 1_calldata[27]),
       If(1_calldatasize <= 28, 0, 1_calldata[28]),
       If(1_calldatasize <= 29, 0, 1_calldata[29]),
```

```
        If(1_calldatasize <= 30, 0, 1_calldata[30]),
        If(1_calldatasize <= 31, 0, 1_calldata[31]))]
```

Informações de Depuração Adicionais

1. **Valores Raw da Pilha**:
    - `gas`: Identificado como `2_gas`.
    - `to`: Representado como uma expressão `Concat` complexa com múltiplos `If`.
2. **Resultado de Operações Internas**:
    - `UGT`: Produz `Bool(2_gas > 2300)` com base no `BitVecVal` de 2300.
    - `to == ACTORS.attacker`: Gera uma expressão `Bool` comparando o `Concat` de `to` com o valor fixo de `ACTORS.attacker`.

Resumo do Processo

1. **Valores de Entrada:**
    - `gas`: Gás disponível na execução.
    - `to`: Endereço de destino, derivado do `calldata`.
2. **Criação de Restrições:**
    - `UGT(gas, 2300)`: Verifica a condição de gás suficiente.
    - `to == ACTORS.attacker`: Confirma se o destino é o atacante.
3. **Encapsulamento:**
    - Ambas as condições são combinadas em um objeto `Constraints`.
4. **Propósito Final:**
    - Essas restrições são enviadas ao solver (Z3) para verificar vulnerabilidades potenciais, como chamadas externas com gás suficiente direcionadas a um endereço malicioso.

Essa análise detalha como o Mythril transforma os valores do estado global em condições lógicas (`constraints`) que são avaliadas para encontrar vulnerabilidades no contrato.