Passo 1: Preparação do Contrato

Consideremos o contrato SpyErcBridge. Este contrato permite a transferência de tokens e possui funções para transferir a propriedade do contrato e dos tokens.

Passo 2: Inicialização do Mythril

Ao executar o Mythril para analisar este contrato, o processo começa com a criação de uma instância da classe LaserEVM, que é o coração do mecanismo de execução simbólica.
laser_evm = LaserEVM(...)

- LaserEVM: Esta classe coordena a execução simbólica, gerenciando estados globais, transações e o espaço de estados.

Passo 3: Início da Execução Simbólica

O método sym_exec é chamado para iniciar a execução simbólica.
laser_evm.sym_exec(creation_code=contract_code, contract_name='SpyErcBridge')

- sym_exec: Determina se o contrato será analisado a partir de um estado pré-configurado ou se será criado simbolicamente. Neste caso, estamos fornecendo o código de criação do contrato.

Dentro de sym_exec, se o código de criação é fornecido, o método execute_contract_creation é chamado.
new_account = execute_contract_creation(laser_evm, creation_code, contract_name)

- execute_contract_creation: Simula a criação do contrato, gerando um novo estado de mundo com o contrato implantado.

Passo 4: Execução das Transações

Após a criação do contrato, o Mythril executa transações simbólicas para explorar diferentes caminhos de execução.
laser_evm.execute_transactions(new_account.address)

- execute_transactions: Gerencia a execução de múltiplas transações simbólicas. Por padrão, o Mythril executa duas transações para explorar variáveis de estado após interações do usuário.

Dentro de execute_transactions, chama-se _execute_transactions_incremental.
self._execute_transactions_incremental(address)

- _execute_transactions_incremental: Executa transações de forma incremental, iterando sobre os estados abertos e chamando execute_message_call.

execute_message_call(self, address)

- execute_message_call: Prepara e inicia a execução simbólica de uma chamada de mensagem para o contrato.

Passo 5: Configuração do Estado Global

Dentro de execute_message_call, o método _setup_global_state_for_execution é chamado para configurar o estado global inicial para a transação.

_setup_global_state_for_execution(laser_evm, transaction)

- _setup_global_state_for_execution: Cria um novo estado global, configura a pilha de transações e adiciona restrições iniciais, como a definição de remetentes possíveis.

Passo 6: Execução das Instruções

Após a configuração do estado global, o Mythril começa a executar as instruções EVM simbolicamente usando o método exec.

laser_evm.exec()

- exec: Este método é responsável por executar as instruções em cada estado global. Ele itera sobre os estados na work_list e chama execute_state para cada um.

Dentro de exec, ocorre o seguinte:

1. Verificação de Tempo: Checa se o timeout de execução foi atingido.
2. Chamada de execute_state: Para cada estado global, executa a próxima instrução.

new_states, op_code = self.execute_state(global_state)

- execute_state: Executa uma única instrução no estado global. Determina o opcode a ser executado e lida com exceções.

Dentro de execute_state, ocorre o seguinte:

- Obtém o Opcode Atual: Baseado no contador do programa (pc).
- Verifica Pré-Requisitos da Pilha: Verifica se há elementos suficientes na pilha para executar a instrução.
- Chamada de Hooks Pré-Instrução: Executa quaisquer funções registradas para serem chamadas antes da instrução.
- Executa a Instrução: Usa a classe Instruction para avaliar o opcode.

new_global_states = Instruction(op_code, ...).evaluate(global_state)

- Instruction.evaluate: Avalia a instrução atual e retorna novos estados globais resultantes.

Passo 7: Verificação de Restrições (Verificação Formal)

Após a execução da instrução, o Mythril realiza a verificação formal das restrições acumuladas.

```
if self.strategy.run_check() and (
   len(new_states) > 1 and random.uniform(0, 1) < args.pruning_factor
):
   new_states = [
      state
      for state in new_states
      if state.world_state.constraints.is_possible()
   ]
```

- state.world_state.constraints.is_possible(): Verifica se as restrições (constraints) no estado atual são satisfatíveis. Isso é feito usando um solver SMT, como o Z3.

Se as restrições não forem satisfatíveis, o estado é descartado (pruning), reduzindo o espaço de busca e focando em caminhos possíveis.

Passo 8: Gerenciamento do Grafo de Fluxo de Controle (CFG)

O método manage_cfg atualiza o grafo de fluxo de controle com os novos estados.
self.manage_cfg(op_code, new_states)

- manage_cfg: Gerencia os nós e arestas do CFG, criando novos nós para estados resultantes de saltos, chamadas e retornos.

Passo 9: Iteração e Conclusão

O processo continua iterativamente, executando instruções, atualizando estados e verificando restrições, até que não haja mais estados a serem explorados ou o limite de profundidade seja atingido.

Simulando o Resultado

O Mythril gera um relatório com potenciais vulnerabilidades encontradas, como reentrância, overflow, etc. Essas vulnerabilidades são detectadas pelos módulos de detecção que analisam os estados finais ou intermediários em busca de padrões inseguros.

Alterando o Contrato e Observando as Diferenças

Se fizermos alterações no contrato, por exemplo, removendo uma verificação em um require, o Mythril poderá detectar novas vulnerabilidades.

Exemplo:

Suponha que removamos a linha require(msg.sender == admin); da função transferOwnership.

- Efeito: Agora, qualquer endereço pode chamar transferOwnership e mudar o administrador.
- Resultado da Análise: O Mythril detectará uma vulnerabilidade de Controle de Acesso. Durante a execução simbólica, ele encontrará um caminho onde um atacante (representado simbolicamente) pode chamar essa função e assumir o controle.

Conclusão

Simular a execução do contrato com o Mythril nos permite ver como as diferentes partes do mecanismo de execução simbólica trabalham juntas para realizar a verificação formal. Entender a ordem das chamadas de funções e métodos é crucial para compreender como o Mythril constrói o espaço de estados e detecta vulnerabilidades potenciais.