Dear Mythril Developers, I hope this message finds you well. My name is M. Silva, and I'm a Computer Science student currently working on a project that involves understanding how Mythril transforms smart contract code for analysis. I've been diving deep into the documentation and even debugging the codebase, but I'm struggling to fully grasp the intricacies, particularly when it comes to the symbolic execution process. I understand this is a foundational aspect of Mythril, and I truly want to comprehend it, not just for my project but also to deepen my understanding of these critical security concepts. This project is very important to me, and I'm reaching out with a heartfelt request: could you kindly provide some guidance or resources that might help clarify how Mythril processes and transforms code during analysis? I'd be immensely grateful for any insight you could share. Thank you so much for your time and for the incredible work you've done with Mythril. It's an inspiring tool, and I hope to understand it better with your help. Warm regards, M. Silva

@norhh
Collaborator norhh commented last week
Hi, Mythril operates on bytecode. So, each instruction has to be symbolically modeled as in this file:
https://github.com/Consensys/mythril/blob/develop/mythril/laser/ethereum/instructions.py.
Consider JUMPI instruction as an example. The condition and the jump destination are popped from the stack (EVM is a stack based machine). Usually, if the condition is true, program counter jumps to the jump destination, otherwise, the program counter is just incremented. In symbolic execution, when the condition is a symbol and can resolve to both true and false, both the paths get added to the list of states. Hence, this instruction returns two states. Whenever an instruction is being evaluated by symbolic vm, this code is wrapped by analysis modules. One such example is arbitrary jump destination module. This module wraps around JUMP and JUMPI instructions as prehooks, so, this analysis module is executed before symbolically executing the instructions JUMP and JUMPI. This analysis module returns an issue if the jump destination popped from the stack (the jump dest still exists in stack during pre_hooks) is symbolic rather than a concrete value. Feel free to ask me any questions.

Em português:

Prezados Desenvolvedores do Mythril,

Espero que esta mensagem os encontre bem. Meu nome é M. Silva e sou estudante de Ciência da Computação, atualmente trabalhando em um projeto que envolve compreender como o Mythril transforma o código de contratos inteligentes para análise.

Tenho me aprofundado na documentação e até depurado a base de código, mas estou encontrando dificuldades para entender completamente os detalhes, especialmente no que diz respeito ao processo de execução simbólica. Entendo que este é um aspecto fundamental do Mythril e realmente quero compreendê-lo, não apenas para meu projeto, mas também para aprofundar meu entendimento sobre esses conceitos críticos de segurança.

Este projeto é muito importante para mim, e estou entrando em contato com um pedido sincero: poderiam, por gentileza, fornecer alguma orientação ou recursos que possam

ajudar a esclarecer como o Mythril processa e transforma o código durante a análise? Ficaria imensamente grato por qualquer insight que possam compartilhar.

Muito obrigado pelo tempo de vocês e pelo trabalho incrível que realizaram com o Mythril. É uma ferramenta inspiradora, e espero compreendê-la melhor com a ajuda de vocês.

Atenciosamente,
M. Silva

@norhh
Colaborador

norhh comentou na semana passada:
Olá, o Mythril opera no bytecode. Assim, cada instrução precisa ser modelada simbolicamente, como neste arquivo: mythril/laser/ethereum/instructions.py.

Considere a instrução JUMPI como exemplo.
A condição e o destino do salto são retirados da pilha (a EVM é uma máquina baseada em pilha). Geralmente, se a condição for verdadeira, o contador de programa salta para o destino do salto; caso contrário, o contador de programa é apenas incrementado. Na execução simbólica, quando a condição é um símbolo que pode ser resolvido tanto como verdadeira quanto como falsa, ambos os caminhos são adicionados à lista de estados. Assim, essa instrução retorna dois estados.

Sempre que uma instrução é avaliada pela VM simbólica, este código é envolvido por módulos de análise. Um exemplo é o módulo "arbitrary jump destination". Este módulo envolve as instruções JUMP e JUMPI como pre-hooks, ou seja, este módulo de análise é executado antes da execução simbólica dessas instruções. Este módulo retorna um problema se o destino do salto retirado da pilha (o destino do salto ainda existe na pilha durante os pre-hooks) for simbólico, em vez de um valor concreto.

Fique à vontade para fazer perguntas.