NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

School of Computer Science and Engineering
College of Engineering

Smart Contract Analysis and Verification (SCSE22-0507)

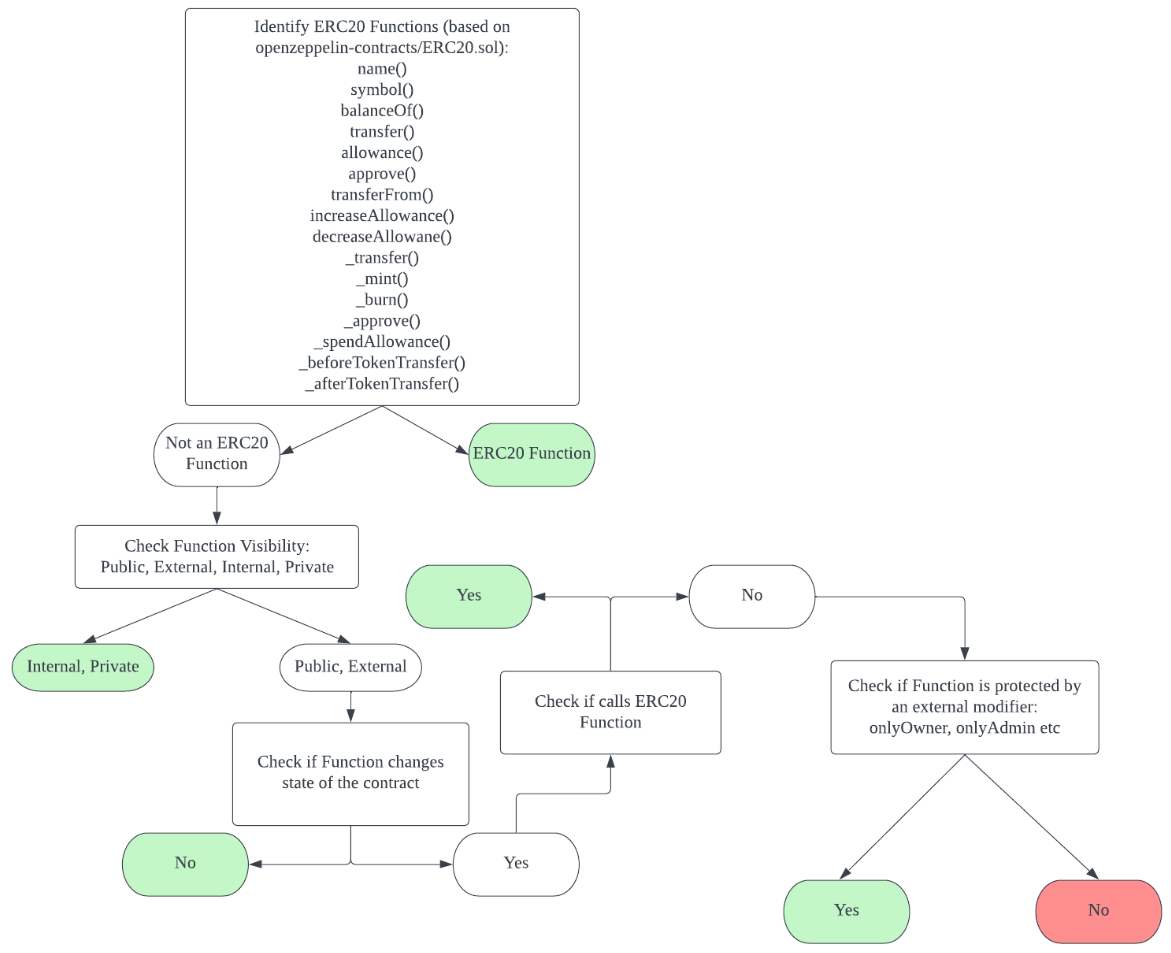
Student: Tan Keng Kai Luke   Supervisor: Asst Prof Lin Shang-Wei

Project Objectives:

We explore how we could eliminate the need for developers to include the selfdestruct function in their Solidity smart contracts. While the selfdestruct function does have its benefits like mitigating a malicious attack by destroying the vulnerable contract, the selfdestruct function creates a new attack vector for attackers to take advantage of and reduces the trust that users would have in the contract, as the owner could transfer out the Ethers from the contract at any point of time. Based on the reasons identified by smart contract developers for including the selfdestruct function, we fixate on the reason of Limits of Permission. We create a tool – Limites, which detects issues with respect to a Solidity function's permission checks and concludes if the function is safe or not.

Introducing Limites:

Limites is a Limits of Permission issues detection tool for Solidity smart contracts. In general, Limites does a static analysis on the Solidity contract parsed through the tool and determines if each function is safe based on its limits of permission as well as other conditions identified throughout the course of this project.



Identify ERC20 Functions (based on openzeppelin-contracts/ERC20.sol):
name()
symbol()
balanceOf()
transfer()
allowance()
approve()
transferFrom()
increaseAllowance()
decreaseAllowane()
_transfer()
_mint()
_burn()
_approve()
_spendAllowance()
_beforeTokenTransfer()
_afterTokenTransfer()

Not an ERC20 Function

ERC20 Function

Check Function Visibility: Public, External, Internal, Private

Internal, Private

Public, External

Check if Function changes state of the contract

No

Yes

Check if calls ERC20 Function

Yes

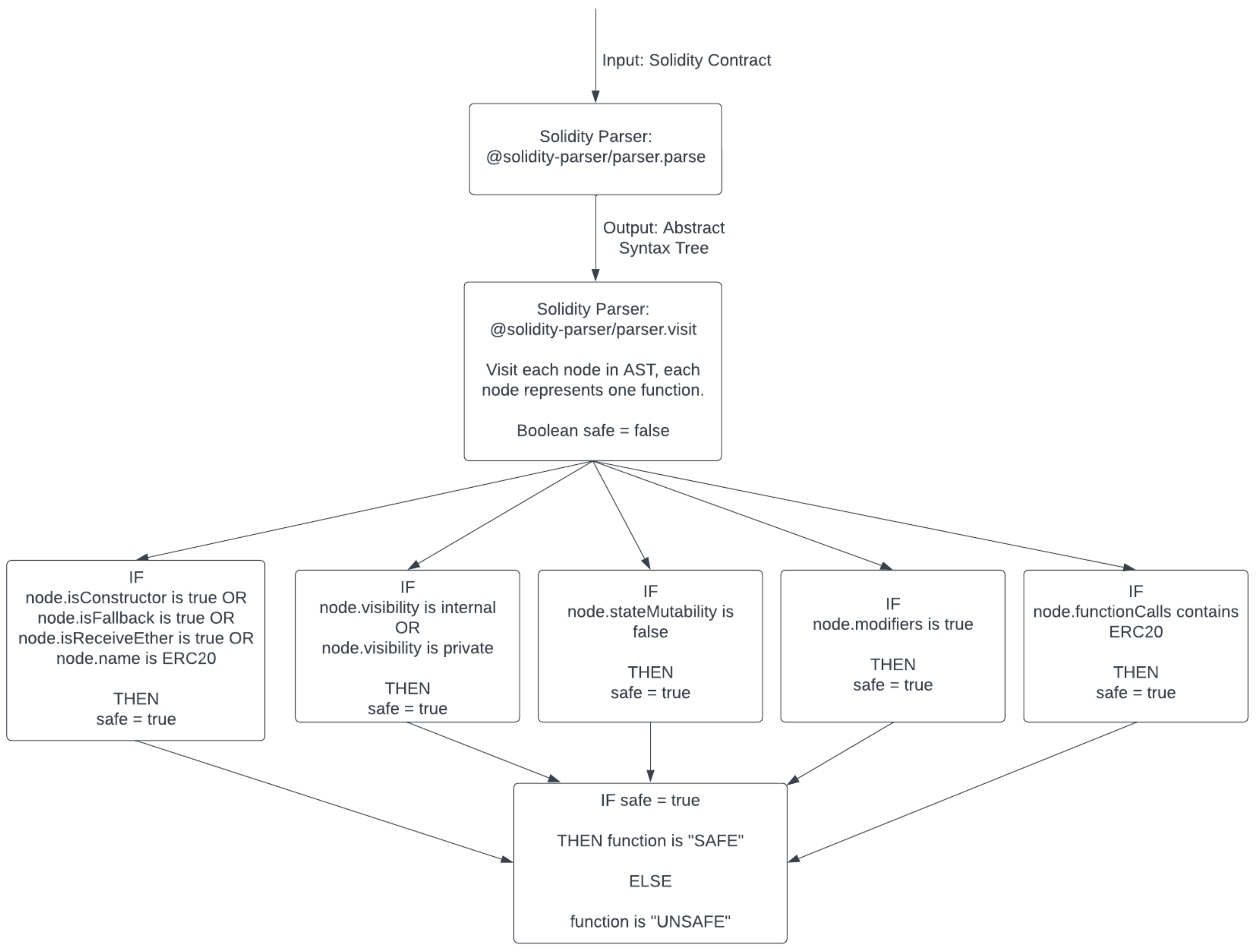
No

Check if Function is protected by an external modifier: onlyOwner, onlyAdmin etc

Yes

No

Figure 2: Flowchart of Conditions in Limites



Input: Solidity Contract

Solidity Parser: @solidity-parser/parser.parse

Output: Abstract Syntax Tree

Solidity Parser: @solidity-parser/parser.visit

Visit each node in AST, each node represents one function.

Boolean safe = false

IF node.isConstructor is true OR node.isFallback is true OR node.isReceiveEther is true OR node.name is ERC20 THEN safe = true

IF node.visibility is internal OR node.visibility is private THEN safe = true

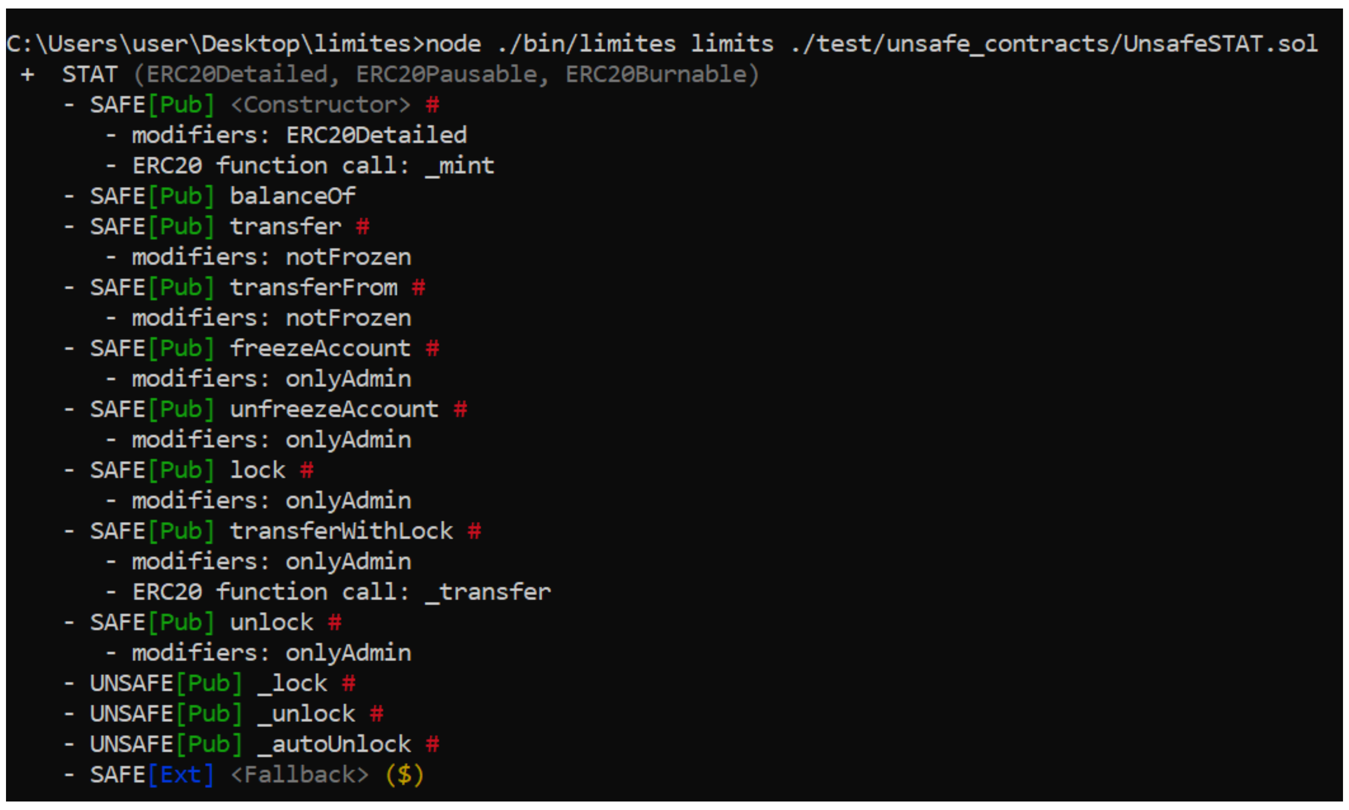
IF node.stateMutability is false THEN safe = true

IF node.modifiers is true THEN safe = true

IF node.functionCalls contains ERC20 THEN safe = true

IF safe = true THEN function is "SAFE" ELSE function is "UNSAFE"

Figure 1: Software Architecture of Limites



```
C:\Users\user\Desktop\limites>node ./bin/limites limits ./test/unsafe_contracts/UnsafeSTAT.sol
 +  STAT (ERC20Detailed, ERC20Pausable, ERC20Burnable)
    - SAFE[Pub] <Constructor> #
       - modifiers: ERC20Detailed
       - ERC20 function call: _mint
    - SAFE[Pub] balanceOf
    - SAFE[Pub] transfer #
       - modifiers: notFrozen
    - SAFE[Pub] transferFrom #
       - modifiers: notFrozen
    - SAFE[Pub] freezeAccount #
       - modifiers: onlyAdmin
    - SAFE[Pub] unfreezeAccount #
       - modifiers: onlyAdmin
    - SAFE[Pub] lock #
       - modifiers: onlyAdmin
    - SAFE[Pub] transferWithLock #
       - modifiers: onlyAdmin
       - ERC20 function call: _transfer
    - SAFE[Pub] unlock #
       - modifiers: onlyAdmin
    - UNSAFE[Pub] _lock #
    - UNSAFE[Pub] _unlock #
    - UNSAFE[Pub] _autoUnlock #
    - SAFE[Ext] <Fallback> ($)
```

Figure 3: Limites in Action

https://www.ntu.edu.sg/scse