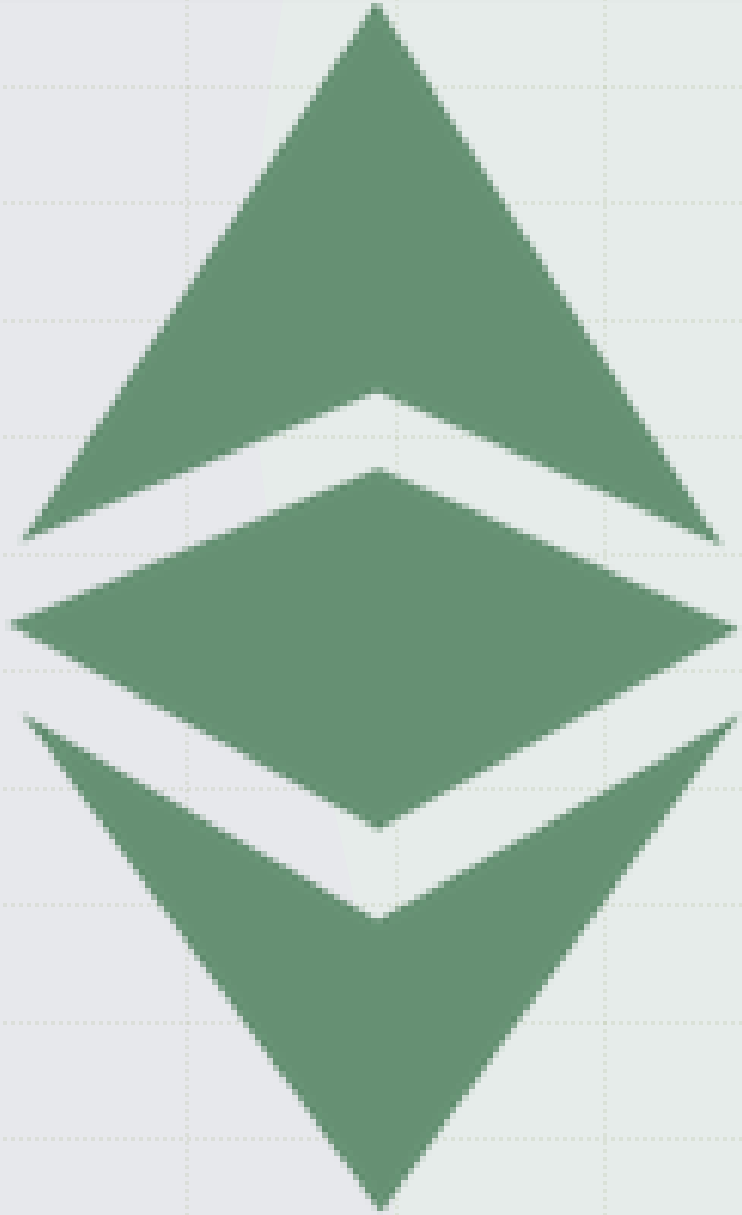
Vulnerability analysis using Conkas tool



By – Team 2

Hema Sri Sesham(1001955263)

Moulin Chandrakant Jariwala (1001858654)

Parshva Urmish Shah(1001838879)

Vineeth Kumar Ananthula(1001953922)

OUTLINE

- Project Plan
- Risks and plan to mitigate risks
- Specification and Design
- Use Case Diagram
- What will be Input & Output
- References

Project Plan

Iterations	Goals	Achieved Goals
1	• Install Conkas tool and run any sample code and see whether tool works good or not • Decide how and which feature will be implemented first • Start Implement of features	• Successfully installed the tool and fixed the versions of the library, solve some compatibility and dependency issues of the code and run successfully the sample code • Planned how and which feature will be implemented • Start implementing of one of the feature
2	• Implement and test Tx.origin feature • Add this feature into existing conkas tool • Plan implementation technique and algorithm of second feature	
3	• Implement and test Costly loop feature • Add this feature into existing conkas tool	

Risks and plan to mitigate risks

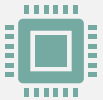
Not Achieving iteration goals:

meeting after each class or on weekends where we will work together by dividing the work equally.

Final program is not portable enough: **ensuring** to maintain clear documentation and compiling as well as testing it on different platforms.

Not having enough knowledge of the tool: all team members should study the tool deeply and discuss and share their knowledge with one another.

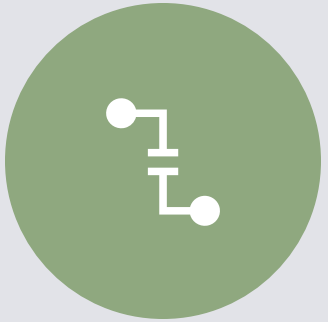
Risk of using a tool which has been recently updated, forcing us to start working from the beginning: creating the feature code robust and independent of the tool.



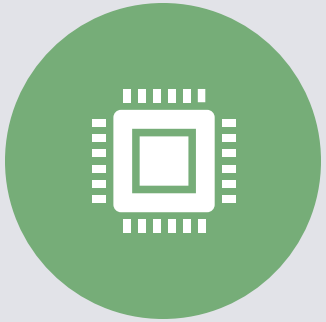
Logic Failure: The risk can be handled by testing and debugging the code for issues.

Specification and Design

We are working on security analysis of the solidity smart contracts using the conkas tool.



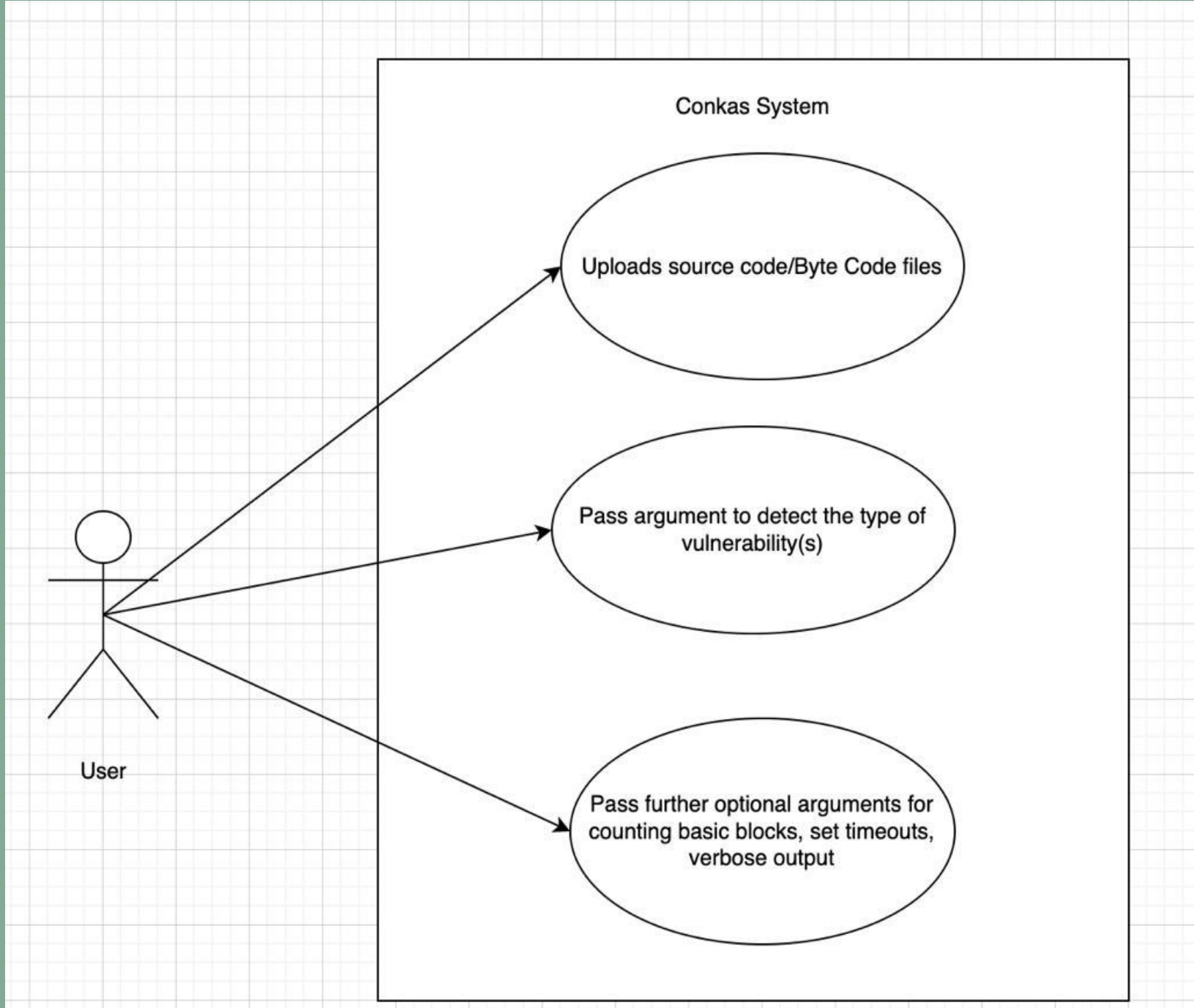
We plan to add a new feature that detects the vulnerabilities regarding the costly loops and Tx.origin.



The vulnerability will be detected by taking the source code as bytecode or solidity code from CLI to compile the code which detects the function, line, type of the vulnerability as well as will detect the function that uses Tx.origin.

Costly loops will be detected by looking out for the unbounded 'for' loop in the code.

Use Case Diagram



Expected Input and Output

Users can enter the following if they have a Solidity-written Smart Contract and want to look for any specific kind of vulnerabilities. Below is one of the example:

$ python3 conkas.py -vt reentrancy -s some_file.sol

User can enter following to search for all of Conkas' vulnerabilities if user have the compiled runtime bytecode hex string that Conkas provide user can type:

$ python3 conkas.py some_file.bin

User will need the “requirements.txt” file from the project github link to install all the dependencies required or by setting up a virtual environment with the following steps:

python3 -m venv venv

source venv/bin/activate

pip install -r requirements.txt

Expected Input

```
(venv) ➜  conkas-master python3 conkas.py --help
usage: conkas.py [-h] [--solidity-file] [--verbosity VERBOSITY] [--vuln-type VULN_TYPE] [--max-depth MAX_DEPTH] [--find-all-vulnerabilities] [--timeout TIMEOUT] file

Symbolic execution tool for EVM

positional arguments:
  file                  File with EVM bytecode hex string to analyse

options:
  -h, --help            show this help message and exit
  --solidity-file, -s   Use this option when file is a solidity file instead of EVM bytecode hex string. By default it is unset
  --verbosity VERBOSITY, -v VERBOSITY
                        Log output verbosity (NotSet, Debug, Info, Warning, Error, Critical). Default = Error
  --vuln-type VULN_TYPE, -vt VULN_TYPE
                        VULN_TYPE can be ['arithmetic', 'reentrancy', 'time_manipulation', 'transaction_ordering_dependence', 'unchecked_ll_calls']. Default =
                        ['arithmetic', 'reentrancy', 'time_manipulation', 'transaction_ordering_dependence', 'unchecked_ll_calls']
  --max-depth MAX_DEPTH, -md MAX_DEPTH
                        Max recursion depth. The counting is how many basic blocks should be analysed. Default = 25
  --find-all-vulnerabilities, -fav
                        When set it will try to find all possible vulnerabilities. It will take some time. By default it is unset
  --timeout TIMEOUT, -t TIMEOUT
                        Timeout to Z3 Solver. Default = 100
```

Expected Output

```
(venv) ➜  conkas-master python3 conkas.py --find-all-vulnerabilities -s /Users/parshvashah/Downloads/not-so-smart-contracts-master/unchecked_external_call/KotET_source_code
/KingOfTheEtherThrone.sol
Analysing /Users/parshvashah/Downloads/not-so-smart-contracts-master/unchecked_external_call/KotET_source_code/KingOfTheEtherThrone.sol:KingOfTheEtherThrone...
Vulnerability: Reentrancy. Maybe in function: _fallthrough. PC: 0x446. Line number: 108.
Vulnerability: Unchecked Low Level Call. Maybe in function: sweepCommission(uint256). PC: 0xa2d. Line number: 162.
Vulnerability: Unchecked Low Level Call. Maybe in function: _fallthrough. PC: 0x3fa. Line number: 101.
Vulnerability: Integer Overflow. Maybe in function: claimThrone(string). PC: 0x23b. Line number: 95.
If a = 31
and b = 115792089237316195423570985008687907853269984665640564039457584007913129639905
Vulnerability: Integer Overflow. Maybe in function: _fallthrough. PC: 0xa4. Line number: 91.
If a = 32
and b = 115792089237316195423570985008687907853269984665640564039457584007913129639904
Vulnerability: Integer Overflow. Maybe in function: _fallthrough. PC: 0x569. Line number: 127.
If a = 4
and b = 2894802230932904885589274625217197696331749616641014100986439600197828240998 4
Vulnerability: Transaction Ordering Dependence. Maybe in function: _fallthrough. PC: 0x539. Line number: 121.
Vulnerability: Integer Underflow. Maybe in function: _fallthrough. PC: 0x55c. Line number: 127.
If a = 115792089237316195423570985008687907853269984665640564039457584007913129639935
and b = 1
Vulnerability: Reentrancy. Maybe in function: _fallthrough. PC: 0x539. Line number: 121.
Vulnerability: Reentrancy. Maybe in function: sweepCommission(uint256). PC: 0xa2d. Line number: 162.
Vulnerability: Reentrancy. Maybe in function: _fallthrough. PC: 0x3fa. Line number: 101.
Vulnerability: Transaction Ordering Dependence. Maybe in function: _fallthrough. PC: 0x446. Line number: 108.
Vulnerability: Time Manipulation. Maybe in function: _fallthrough. PC: 0x6cc. Line number: 128.
```

Project GitHub Link

- https://github.com/shahparshva72/CSE6324-Team2-Project

References

- [1] Veloso, Nuno. Conkas: A Modular and Static Analysis Tool for Ethereum Bytecode - ULisboa. Jan.2021,https://fenix.tecnico.ulisboa.pt/downloadFile/1689244997262417/94080-Nuno-Veloso_resumo.pdf.
- [2] Polak, Kamil. “Hacking Solidity: Contracts Using Tx.origin for Authorization Are Vulnerable to Phishing.” *HackerNoon*, 16 Jan.2022, https://hackernoon.com/hacking-solidity-contracts-using-txorigin-for-authorization-are-vulnerable-to-phishing
- [3] Sebastian Sinclair. “Blockchain Protocol Thorchain Suffers $8M Hack” Coindesk, 14 Sep. 2021, https://www.coindesk.com/markets/2021/07/23/blockchain-protocol-thorchain-suffers-8m-hack/
- [4] Veloso, Nuno. (n.d.). *Nveloso/conkas: Ethereum Virtual Machine (EVM) bytecode or solidity smart contract static analysis tool based on symbolic execution*. GitHub. Retrieved from https://github.com/nveloso/conkas

THANK YOU!!