CSE 6324:Advanced Topics in Software Engineering

Vulnerability analysis using Conkas tool

Team 2:

Hema Sri Sesham(1001955263)

Moulin Chandrakant Jariwala(1001858654)

Parshva Urmish Shah(1001838879)

Vineeth Kumar Ananthula(1001953922)

TABLE OF CONTENT

Sr No.	Title	Page No
1.	Abstract	3
2.	Architecture	3
3.	Project Plan	4
4.	List of Features	4
5.	Key Data Structure	5
6.	List of Biggest Risk	5
7.	Plans to deal with the Risk	6
8.	Specification & Design	6
9.	Use Case	8
10.	Comparison	9
11.	Target Users/Customers	10
12.	References	10

ABSTRACT

Modern blockchains like Ethereum, enable any user to create a program, called Smart Contract, and run-in distributed network. Ethereum also has a cryptocurrency called Ether and those Smart Contracts can send and receive Ether between each other via transactions. Once a Smart Contract is deployed, the owner of that contract can no longer make updates to fix it if someone finds a vulnerability. To try to reduce the probability of new future attacks happening Conkas was introduced, a modular static analysis tool that uses symbolic execution to find traces that lead to vulnerabilities and uses an intermediate representation. The users can interact with Conkas via Command-Line Interface (CLI) and the output will be the result of the analysis. Conkas supports Ethereum bytecode or contracts written in Solidity and is compatible with all versions of Solidity, but the analysis is done at the bytecode level. [1]

Conkas supports 5 modules that detect vulnerabilities: Arithmetic, Front-Running, Reentrancy, Time Manipulation and Unchecked Low-Level Calls. Conkas is easy to extend, meaning that you can add your custom modules to detect other types of vulnerabilities. We plan to improve conkas by adding Tx.Origin and costly loops feature in conkas.

ARCHITECTURE

Conkas uses a modular architecture. The users can provide byte-code and this byte code is passed to the rattle module, in this the symbolic execution engine is responsible to iterate and generate traces. These are provided to the Detectors module, in this one of the sub modules is responsible to detect a category of vulnerability.

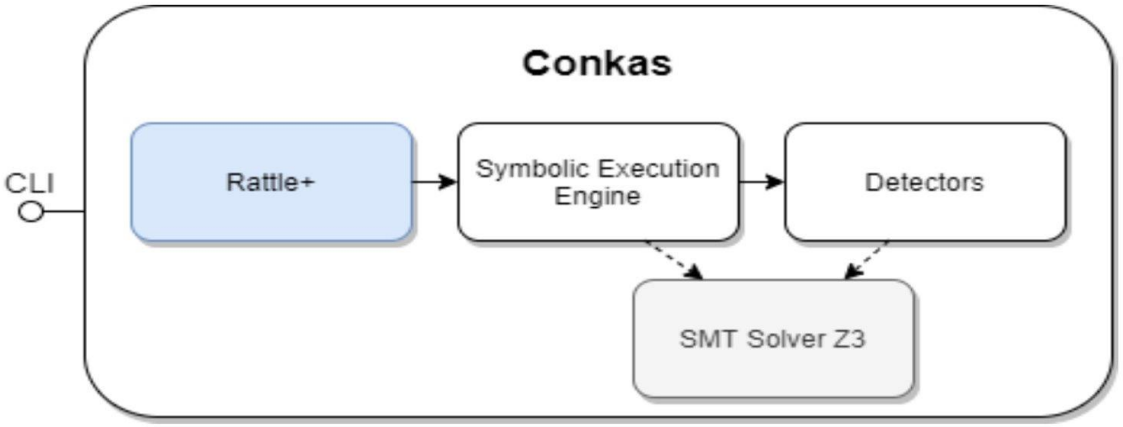


Figure 1: Architecture of Conkas tool.[1]

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	

LIST OF FEATURES

We will be implementing the following two features phase wise to improve the conkas tool.

1. Tx.Origin: Tx.Origin is a global variable in Solidity which returns the address of the account that sent the transaction. Contracts that use the tx.origin to authorize users are vulnerable to phishing attacks. It is critical to never use tx.origin for authorization since another contract can contact a contract using a fallback function and get authorization because the permitted address is recorded in tx.origin. [2]. In June,2021 THORChain Org suffered this attack and lost around 8 million dollars. [3] Detecting the use of the tx.origin variable will eliminate the vulnerability of getting phished.

Qualitative value to Customers -Medium

2. Costly Loops: An EVM is constrained by the block gas limit, and the cost of gas connected with contracts is one of the factors that encourage users to interact with your contracts. In Solidity, loops may be used for any function. However, if the loop involves updating certain contract state variables, it should be constrained; otherwise, the contract may get stuck if the loop iteration exceeds the block's gas limit. If a loop consumes more gas than the block's gas limit, the transaction will not be added to the blockchain and will revert. As a result, the transaction fails. As a result, it is important to avoid employing loops that alter contract state variables in the first place. Costly loops and gas limit

refers to an attacker including infinite loops within an array to exhaust the gas limit mimicking DoS attack and freeing the transaction.

Qualitative value to Customers -Low

KEY DATA STRUCTURES

- List – used to keep track of Unbound loops whether it is visited or keep track of functions for checking vulnerability
- Set – used to check and store analyse the vulnerability and blocks

LIST OF BIGGEST RISKS

1. Not Achieving iteration goals

- P=30% and E=15, so extra 4.5 hours

2. Final program is not portable enough

- P=10% and E=5, so extra 0.5 hours

3. Not having enough knowledge of the tool

- P=30% and E=5, so extra 1.5 hours

4. Risk of using a tool which has been recently updated, forcing us to start working from the beginning.

- P=10% and E=40, so extra 4 hours

5. Unexpected personal or health issue of teammate

- P=5% and E=30, so extra 1.5 hours

NEW RISKS

1. Logic Failure

- P=30% and E=20, so extra 6 hours

2. New feature merge failure

- P=40% and E=10, so extra 4 hours

PLANS TO DEAL WITH RISK

1. Dealing with Risk 1: One of the possible risks is not achieving the iteration goal. The plan to handle this risk is to have a meeting after each class or on weekends where we will work together by dividing the work equally.

2. Dealing with Risk 2: The risk of our code not being portable enough can be handled by ensuring to maintain clear documentation and compiling as well as testing it on different platforms.

3. Dealing with Risk 3: To handle this risk all team members should study the tool deeply and discuss and share their knowledge with one another.

4. Dealing with Risk 4: The risk can be handled by creating the feature code robust and independent of the tool.

5. Dealing with Risk 5: In such extreme conditions the team member's work can be divided with the remaining teammates, such that it does not affect the progress of the project.

6. Dealing with New Risk 1: The risk can be handled by testing and debugging the code for issues.

7. Dealing with New Risk 2: The risk can be handled by following conkas new feature addition guidelines.

SPECIFICATION & DESIGN

In this project we will be working on security analysis of the solidity smart contracts using conkas tool. We plan to add a new feature that detects the vulnerabilities regarding the costly loops and Tx.origin. We will detect the vulnerability by taking the source code as bytecode or solidity code from the command line interface then compile the code and the code will detect 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 for the unbounded for loop in the code.

CODE AND TEST:

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:

1. python3 -m venv venv
2. source venv/bin/activate
3. pip install -r requirements.txt

Expected Input for Application:

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

Below is an complete information for users what to pass and how to give an input

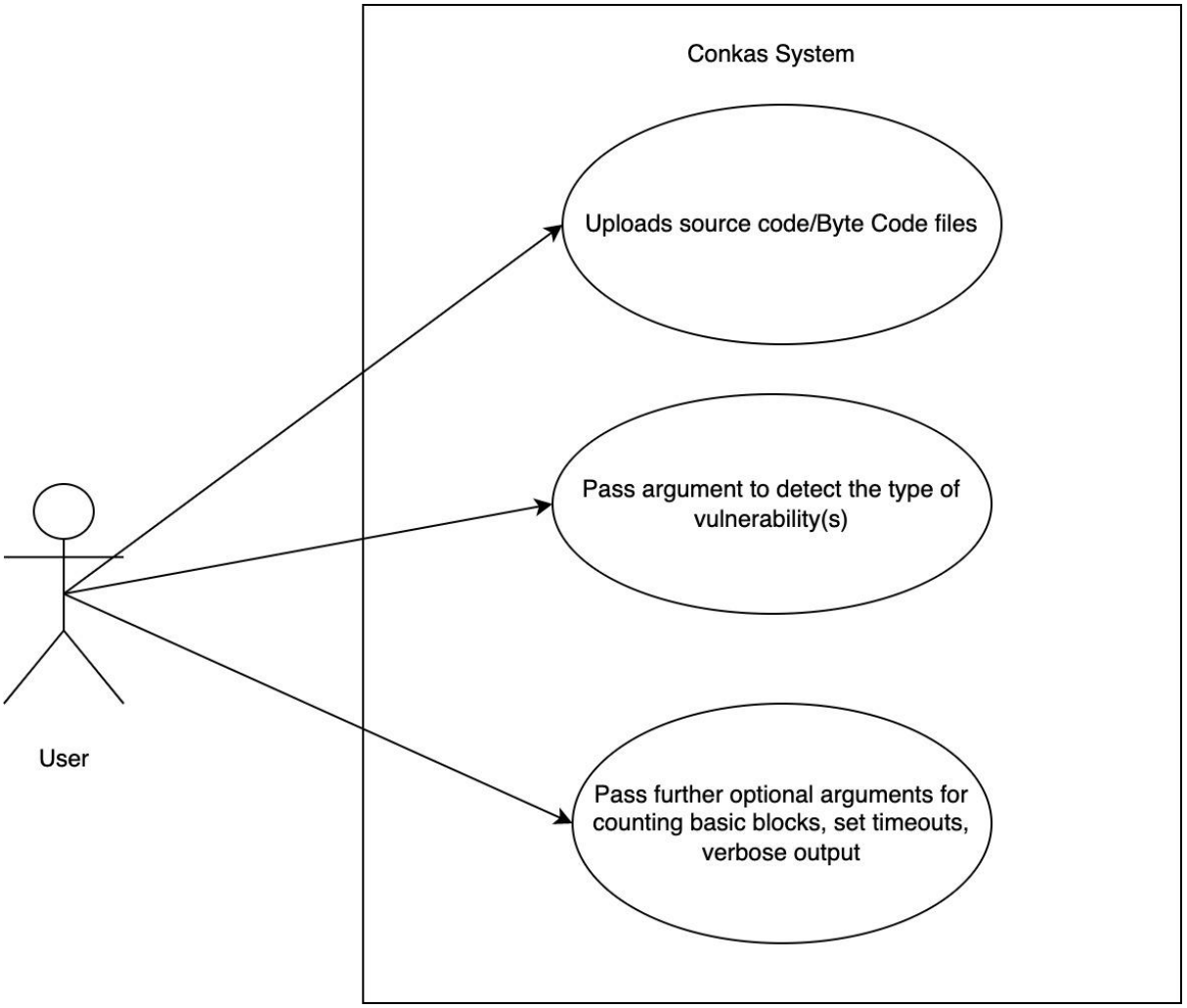
```
(venv) ➜  conkas-master python3 conkas.py --help
usage: conkas.py [-h] [--solidity-file] [--verbosity VERBOSITY] [--vuln-type VULN_TYPE] [--max-depth MAX_DEPTH] [--find-all-vulnerabilities] [--timeout TIMEOUT] file

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 for Application: Output will be displayed as per input the user passes the vulnerability name on command line argument. Below mention is an example

- Sample output of checking the all the vulnerability

```
(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 = 28948022309329048855892746252171976963317496166410141009864396001978282409984
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 = 115792089237316195423570985008687907853269846656405640394575840079131296399335
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.
```

USE CASE DIAGRAM



COMPARISON TO OTHER SMART CONTRACT VULNERABILITY DETECTION TOOLS

Several tools for smart contracts have been developed to address vulnerabilities detection. Other prominent tools that compete with this Conkas include Slither, Oyente, Osiris, Manticore, Mythril, and Securify, Smartcheck. Using the SmartBugs framework, Conkas' performance was examined compared to that of other tools. When the true positive rate of all tools was calculated, Conkas had the highest percentage.

Conkas false positives are decreasing in practically every area, but the arithmetic category is the most penalised. We have less false positives than Mythril, and we have the fewest false positives in Unchecked Low Calls.

Category	**Conkas**	**Honeybadger**	**Maian**	**Manticore**	**Mythril**	**Osiris**	**Oyente**	**Securify**	**Slither**	**Smartcheck**	**Total**
Access Control	0/24 0%	0/24 0%	0/24 0%	5/24 21%	4/24 17%	0/24 0%	0/24 0%	1/24 4%	6/24 25%	2/24 8%	8/24 33%
Arithmetic	19/23 83%	0/23 0%	0/23 0%	13/23 57%	16/23 70%	13/23 57%	18/23 78%	0/23 0%	0/23 0%	1/23 4%	22/23 96%
Denial Service	0/14 0%	0/14 0%	0/14 0%	0/14 0%	0/14 0%	0/14 0%	0/14 0%	0/14 0%	0/14 0%	1/14 7%	1/14 7%
Front Running	2/7 29%	0/7 0%	0/7 0%	0/7 0%	2/7 29%	0/7 0%	2/7 29%	2/7 29%	0/7 0%	0/7 0%	2/7 29%
Reentrancy	30/34 88%	19/34 56%	0/34 0%	15/34 44%	25/34 74%	21/34 62%	28/34 82%	14/34 41%	33/34 97%	30/34 88%	33/34 97%
Time Manipulation	7/7 100%	0/7 0%	0/7 0%	4/7 57%	0/7 0%	2/7 29%	0/7 0%	0/7 0%	3/7 43%	2/7 29%	7/ 7 100%
Unchecked Low Calls	62/75 83%	0/75 0%	0/75 0%	9/75 12%	60/75 80%	0/75 0%	0/75 0%	50/75 67%	51/75 68%	61/75 81%	70/75 93%
Other	0/5 0%	0/5 0%	0/5 0%	0/5 0%	0/5 0%	0/5 0%	0/5 0%	0/5 0%	0/5 0%	0/5 0%	0/ 5 0%
Total	120/224 54%	19/224 8%	0/224 0%	46/224 21%	107/224 48%	36/224 16%	48/224 21%	67/224 30%	93/224 42%	97/224 43%	143/224 64%

Table 1: Results of Conkas compared to other tools. [4]

SmartBugs was used to evaluate Conkas compared to other tools. In the above table, different vulnerabilities are checked for different kinds of analysis tools. SmartBugs only count the true positive rate as the accuracy for evaluation. The data in the cell represents the number of actual vulnerabilities that were predicted correctly along with the line numbers at which vulnerability might be present in the source code, by the tools compared to Conkas.[4]

TARGET USERS/CUSTOMERS OF THE TOOL

- The main users of Conkas are the web3 developers and smart contract auditors. Auditing a Smart Contract is about extensively looking through the code to avoid any data breaches or bugs. This is required because once this code is launched, it cannot be modified, which might result in massive security concerns, data loss, monetary loss and so on. Hence, it is essential to use an analysis tool to detect and eliminate all the vulnerabilities in a contract before it is deployed.
- As it will be available on GitHub so anyone can access the tool.
- Will take feedback from Shovon Niverd Pereira as well as from Arfin Mohammad Rifat(TA).

Project Github Repo: 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