CSE –6324 Advance Topics In Software Engineering

Iteration 1 Written Deliverables

Conkas tool for vulnerability analysis

TEAM -3
Sarang Mehta (1002002420)
Aknith Reddy Mekala (1001946768)
Shresth Pal (1001965426)
Aarush Verma (1001965431)
Sai Likhith Palasala (1001980407)

TABLE OF CONTENT

ABSTRACT

- Conkas, a modular and static analysis tool based on symbolic execution that looks for a trace that points to a weakness.

- It is also simple to develop unique modules to find new varieties of vulnerabilities. Users can communicate with Conkas via the tool's Command-Line Interface (CLI).[1]

ARCHITECTURE

- The architecture of Conkas is modular.
- Users have the option of providing bytecode, which is then given to the rattle module, where the symbolic execution engine oversees iterating and generating traces.
- They are sent to the Detectors module, one of whose submodules oversees identifying a certain type of vulnerability.

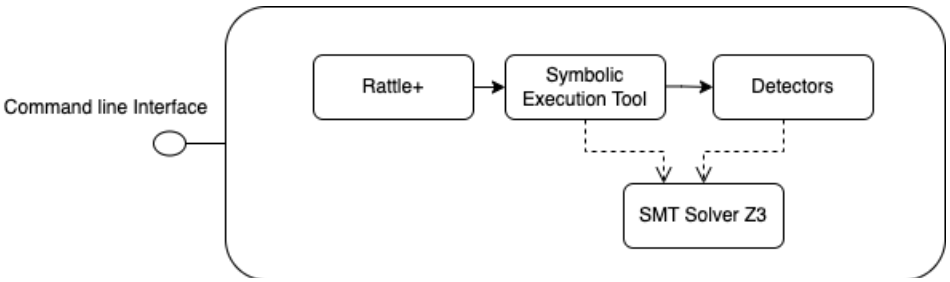


Fig: Conkas Architecture [1]

Project Plan

Iterations	*Goals*	*Achieved Goals*
1	• Install the Conkas tool and test it out with some example code to see how well it performs. • Choose which feature will be implemented first and how. • Begin implementing the features	• The tool was installed successfully, the library versions were updated, some code compatibility and dependency issues were resolved, and the test code was successfully ran. • Planned the implementation of the feature and how it would be done • Begin implementing the features
2	• TX. Origin feature implementation and testing • Including TX. Origin function in	

	the current Conkas tool • Implementation strategy and algorithm for the second feature	
3	• Construct and evaluate expensive loop feature • Including this function in the current Conkas tool • Denial of Service feature implementation and testing	

LIST OF FEATURES

1. Tx. Origin: In order to obtain authorization, the "TX. Origin" is used. The attacker can still use this for a phishing attempt. For purposes of authentication, "Ms. Sender" ought to be used rather than "TX. Origin".

2. Gas Related Issues: Gas-related problems can include sending a transaction with inadequate gas, having worthless code in the smart contract, or having gas-expensive loops in the contract. To carry out the instructions of the smart contract, gas is used as a transnational fee; each type of operation calls for a particular gas, which is charged in ether.

3. Denial of Service: It happens because the user intentionally tries to stop another user from carrying out their caller contract by repeatedly reversing the call.

KEY DATA STRUCTURES

- Control flow graph (CFG): A CFG is a directed graph that depicts a smart contract's control flow. The execution of conditional and unconditional leaps demonstrates how control moves from one instruction to another.

- Abstract syntax tree (AST): An AST is a tree-like data structure that depicts the Solidity code's structure. The mapping between the bytecode and the relevant source code is done by Conkas, which creates an AST from the Solidity source code.

- Dominator tree: A dominator tree is a data structure that resembles a tree and displays the dominance connections between the fundamental building components in a CFG. A simple block is a set of instructions with only one point of entrance and one point of exit.

- Data dependency graph: A directed graph called a data dependency graph shows the dependencies between instructions that use the same data. It is used to find possible weaknesses like reentrancy attacks.

- Symbolic execution tree: The execution pathways of a smart contract are represented by a tree-like data structure called a symbolic execution tree. Conkas employs symbolic execution to examine every conceivable smart contract execution path and find any potential weaknesses.

LIST OF FIVE BIGGEST RISKS

1. Conkas is written in python, we have two members who primarily use python and three members who are new to python language we are adding the functionality which might enables conkas to detect denial of service attacks which requires extremely proficient coding practices this factor is risk in our project as we have couple of team members who might be vulnerable to poor coding practice:

   Risk Exposure = Probability of Risk * Effect of Risk = 1 * 5 =5 hours

2. Attacker can instigate a fallback code to steal all the money while hiding a withdraw function within the TX. Origin variable to construct a phishable contract:

   Risk Exposure = Probability of Risk * Effect of Risk = 0.3 * 5 =1.5 hours

3. Infinite loops within an array are used by an attacker to exhaust the gas limit, simulating a denial-of-service attack, and liberate the transaction. This is referred to as using costly loops and gas limits:

   Risk Exposure = Probability of Risk * Effect of Risk = 0.4 * 20 =8 hours

4. Unforeseen personal or medical crisis with a teammate:

   Risk Exposure = Probability of Risk * Effect of Risk = 0.1 * 20 =2 hours

5. Failure of Implementation Logic:

   Risk Exposure = Probability of Risk * Effect of Risk = 0.2 * 30 =6 hours

PLANS TO DEAL WITH RISK

1. Dealing with Risk 1: To mitigate this category of risk we will be carrying out cross training sessions for new members mentored by experienced members and we will pre-schedule key personnel

2. Dealing with Risk 2: Put access controls in place to prevent unwanted access to the fallback function or the TX. Origin variable.

3. Dealing with Risk 3: Gas restrictions should be put in place to stop expensive loops from using too much gas. Consider using the `GasLeft` variable to track the quantity of gas left over during the contract's execution and use `require` statements to enforce caps on how much gas may be spent by a loop.

4. Dealing with Risk 4: In such difficult circumstances, the team member's job might be split across the other teammates so that it does not impede the project's development.

5. Dealing with Risk 5: Testing and debugging the code for errors helps reduce the risk.

SPECIFICATION & DESIGN

In this project, we will use the Conkas tool to analyze the security of Solidity smart contracts. We intend to introduce a new feature that finds flaws in costly loops, Denial of Service and TX. Origin. We will identify the vulnerability by extracting the source code in the form of bytecode or solidity code from the command line interface, compiling it, and then identifying the function that uses TX. Origin as well as the function, line, and type of the vulnerability. By scanning the code for the unbounded for loop, costly loops can be found.

Code Execution & Tests:

The following are the dependencies of the conkas tool:

- cbor2
- py-solc-x
- pycryptodome
- pyevmasm
- python3
- solidity_parser
- z3-solver

For the successful execution of conkas tool following dependencies must be installed by the user.

To install all these dependencies the user can utilize requirements.txt file provided in the official GitHub repository of the conkas tool by using the command:

pip install -r requirements.txt

To check for specific vulnerabilities for e.g., Reentrancy following command can be used:

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

Conkas also detects vulnerabilities using compiled runtime bytecode hex string by the following command:

python3 conkas.py some_file.bin

INPUT:

The input types accepted by conkas can be seen by using the following command:

python3 conkas.py –help

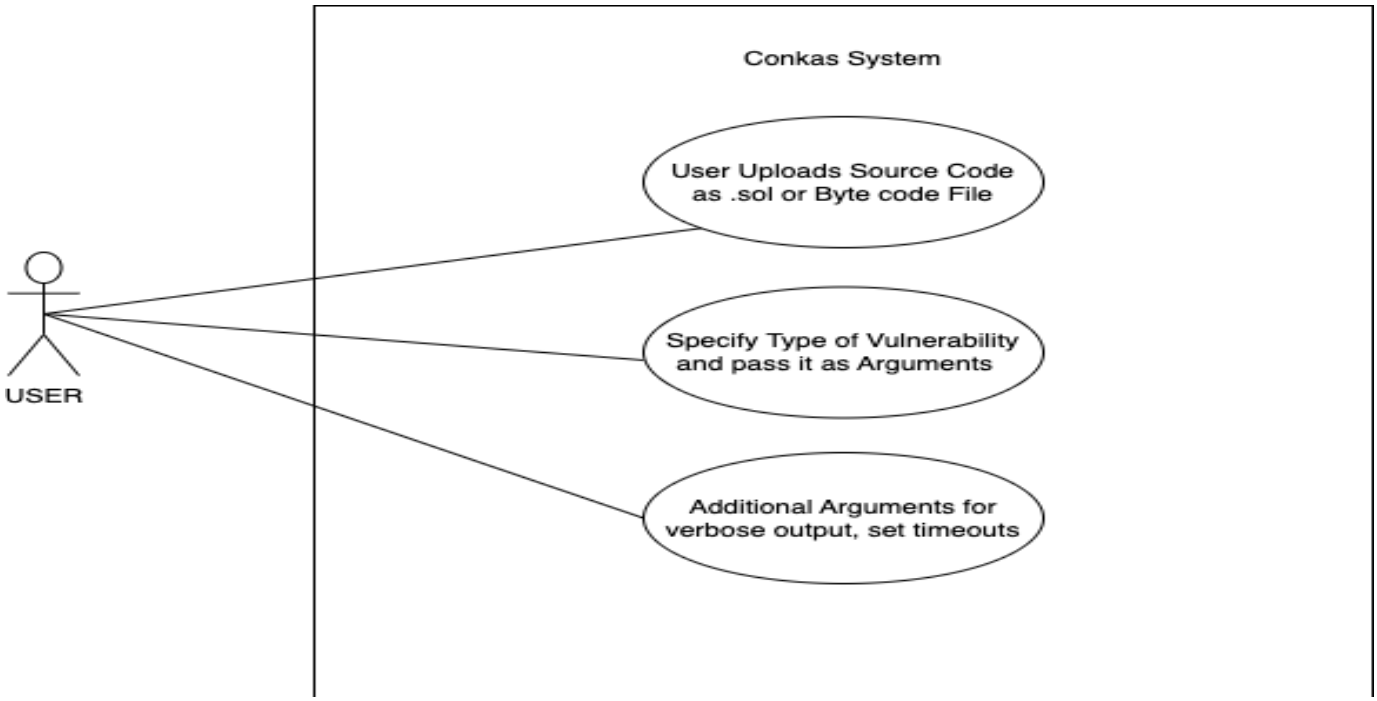
```
(base) ➜  CONKAS_OFFICIAL python3 conkas.py --help
usage: conkas.py [-h] [--solidity-file] [--verbosity VERBOSITY] [--vuln-type VULN_TYPE] [--max-depth MAX_DEPTH]
                 [--find-all-vulnerabilities] [--timeout TIMEOUT]
                 file

Symbolic execution tool for EVM

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

optional arguments:
  -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
(base) ➜  CONKAS_OFFICIAL
```

OUTPUT:

The below figure shows the output of a sample smart contract notice in the command line arguments we have mentioned to find all vulnerabilities hence it would return all the vulnerabilities.

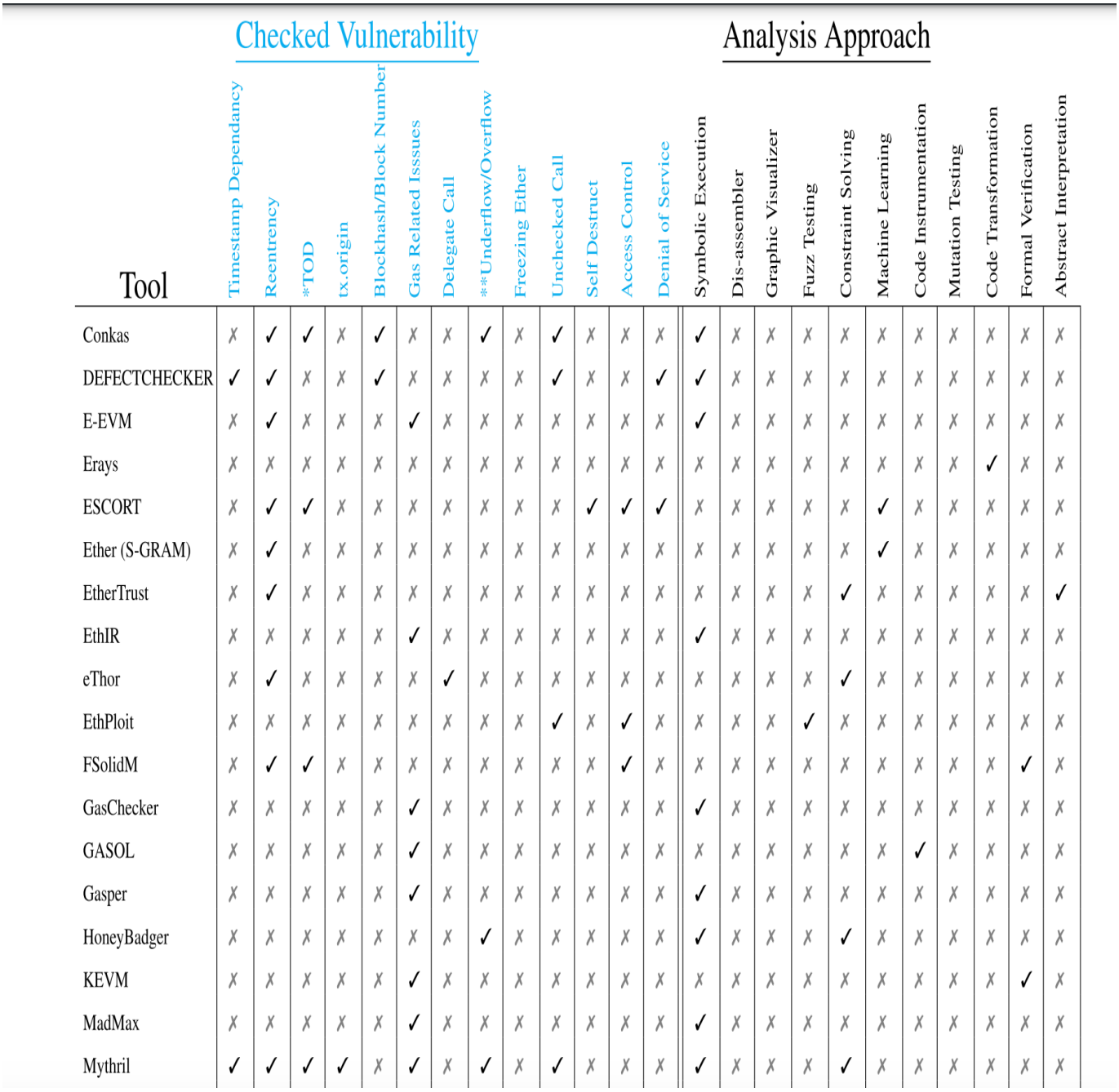
```
(base) ➜  CONKAS_OFFICIAL python3 conkas.py --find-all-vulnerabilities -s /Users/aarushverma/Desktop/ADV_SE/CONKAS_OFFICIAL/sol_exa
mples/forced_ether_reception/coin.sol
Analysing /Users/aarushverma/Desktop/ADV_SE/CONKAS_OFFICIAL/sol_examples/forced_ether_reception/coin.sol:MyAdvancedToken...
Vulnerability: Integer Overflow. Maybe in function: approveAndCall(address,uint256,bytes). PC: 0x53c. Line number: 129.
If a = 128
and b = 115792089237316195423570985008687907853269984665640564039457584007913129639752
Vulnerability: Reentrancy. Maybe in function: approveAndCall(address,uint256,bytes). PC: 0xca8. Line number: 134.
Vulnerability: Integer Overflow. Maybe in function: buy(). PC: 0xaff. Line number: 174.
If a = 796600874683013103561001735008643384869432533028102351060306747615016339693 64
and b = 702126042901620883320092516248003272324799019893886719851935142348691413747 09
Vulnerability: Integer Overflow. Maybe in function: approveAndCall(address,uint256,bytes). PC: 0x7b5. Line number: 116.
If a = 32
and b = 115792089237316195423570985008687907853269984665640564039457584007913129639744
Vulnerability: Integer Overflow. Maybe in function: approveAndCall(address,uint256,bytes). PC: 0xbe8. Line number: 134.
If a = 32
and b = 115792089237316195423570985008687907853269984665640564039457584007913129639708
Analysing /Users/aarushverma/Desktop/ADV_SE/CONKAS_OFFICIAL/sol_examples/forced_ether_reception/coin.sol:TokenERC20...
Vulnerability: Reentrancy. Maybe in function: approve(address,uint256). PC: 0x9ec. Line number: 134.
Vulnerability: Integer Overflow. Maybe in function: approveAndCall(address,uint256,bytes). PC: 0x423. Line number: 129.
If a = 31
and b = 115792089237316195423570985008687907853269984665640564039457584007913129639912
Vulnerability: Integer Overflow. Maybe in function: approve(address,uint256). PC: 0x668. Line number: 116.
If a = 32
and b = 115792089237316195423570985008687907853269984665640564039457584007913129639713
Vulnerability: Integer Overflow. Maybe in function: approve(address,uint256). PC: 0x95e. Line number: 134.
If a = 32
and b = 115792089237316195423570985008687907853269984665640564039457584007913129639672
Analysing /Users/aarushverma/Desktop/ADV_SE/CONKAS_OFFICIAL/sol_examples/forced_ether_reception/coin.sol:owned...
Analysing /Users/aarushverma/Desktop/ADV_SE/CONKAS_OFFICIAL/sol_examples/forced_ether_reception/coin.sol:tokenRecipient...
Nothing to analyse
(base) ➜  CONKAS_OFFICIAL
```

Use Case Diagram:



Comparison To Other Smart Contract Vulnerability Detection Tools

The Following figure below shows a comprehensive comparison according to features available in various tools, it compares conkas features mainly in two criteria checked vulnerability and analysis approach.



Tool	Checked Vulnerability													Analysis Approach										
	Timestamp Dependancy	Reentrency	*TOD	tx.origin	Blockhash/Block Number	Gas Related Isssues	Delegate Call	**Underflow/Overflow	Freezing Ether	Unchecked Call	Self Destruct	Access Control	Denial of Service	Symbolic Execution	Dis-assembler	Graphic Visualizer	Fuzz Testing	Constraint Solving	Machine Learning	Code Instrumentation	Mutation Testing	Code Transformation	Formal Verification	Abstract Interpretation
Conkas	✗	✓	✓	✗	✓	✗	✗	✓	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
DEFECTCHECKER	✓	✓	✗	✗	✓	✗	✗	✗	✗	✓	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
E-EVM	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Erays	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗
ESCORT	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗
Ether (S-GRAM)	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗
EtherTrust	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✓
EthIR	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
eThor	✗	✓	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
EthPloit	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗
FSolidM	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
GasChecker	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
GASOL	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗
Gasper	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
HoneyBadger	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
KEVM	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
MadMax	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Mythril	✓	✓	✓	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗

Fig: Comparison of various tools [2]

Target Users/Customers

- Decentralized application (dApp) developers: Conkas is a tool that Ethereum blockchain dApp developers may use to assess their smart contracts and make sure they are trustworthy and safe.

- Cryptocurrency exchanges: The Conkas tool can be used by cryptocurrency exchanges to evaluate the security of the smart contracts underlying the ERC-20 tokens and other Ethereum-based assets that are listed on their exchanges.

- Auditing firms: Conkas technology is used by auditing companies to conduct security audits of smart contracts on behalf of their clients, confirming the security and dependability of the smart contracts.

Feedback From Customers:

We were able to identify some feedbacks from customers via official GitHub Issues portal, some of the developers were particularly worried about compatibility of Conkas in analyzing vulnerabilities using compiled runtime bytecode hex string and various reported issues like maximum recursion depth, bit vector size and many more.

Fig: User Feedback [3]

GitHub Link: https://github.com/grudge17/CSE_6324_Conkas2.0.git

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] Ethereum Smart Contract Analysis Tools: A Systematic Review https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9762279

[3] https://github.com/nveloso/conkas/issues/1