Iteration 1

CSE – 6324
TEAM -3

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

PROF.- Christoph Csallner
UNIVERSITY OF TEXAS AT ARLINGTON

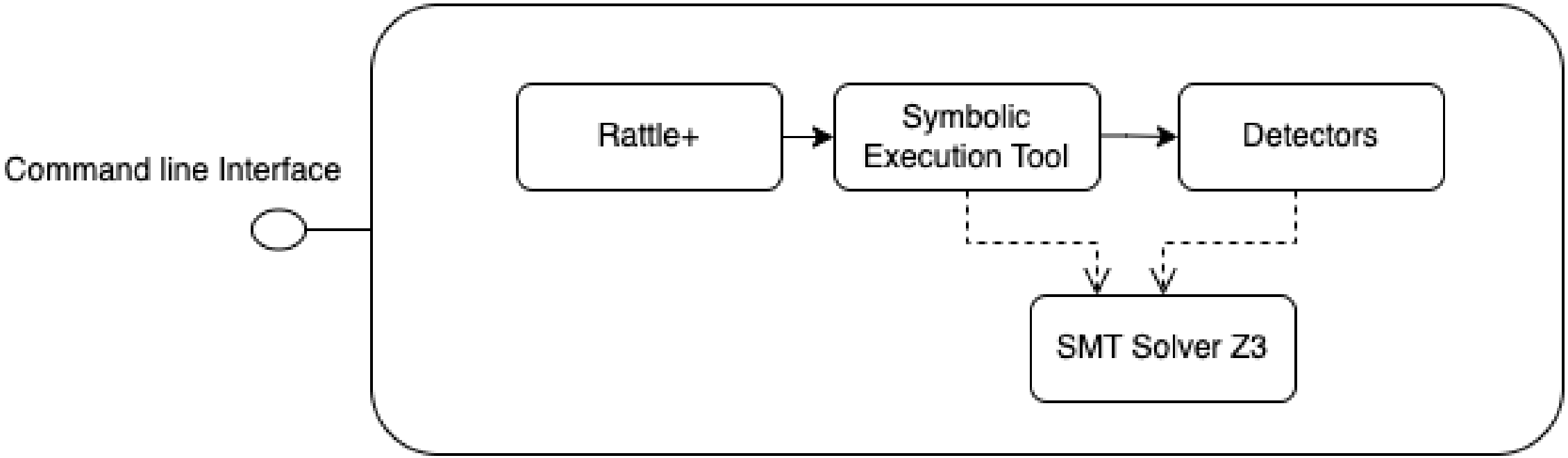
TABLE OF CONTENT

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.



Project Plan

Iterations	Goals	Achieved Goals
Iteration 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

Iteration 2	☐ TX. Origin feature implementation and testing ☐ Including TX. Origin function in the current Conkas tool ☐ Implementation strategy and algorithm for the second feature	
Iteration 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:
- 2. Gas Related Issues:
- 3. Denial of Service

KEY DATA STRUCTURES

- Control flow graph (CFG)
- Abstract syntax tree (AST)
- Dominator tree
- Data dependency graph
- Symbolic execution tree

LIST OF FIVE BIGGEST RISKS

- Lack of Programming Knowledge: We plan to mitigate this risk by carrying cross training sessions.
- Tx.origin issues: By placing access controls to prevent unwanted access.
- Costly-loop issues: By using Gas left and require variable to keep track of gas and loops.
- Unforeseen personal or medical crisis with a teammate: We plan to split the team members tasks among the rest of us.
- Failure of Logic: By testing and debugging the code for errors regularly we can mitigate this 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.

Code Execution & Tests:

Following are the dependencies of the conkas tool:

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

$ pip install -r requirements.txt

INPUT:

python3 conkas.py --help

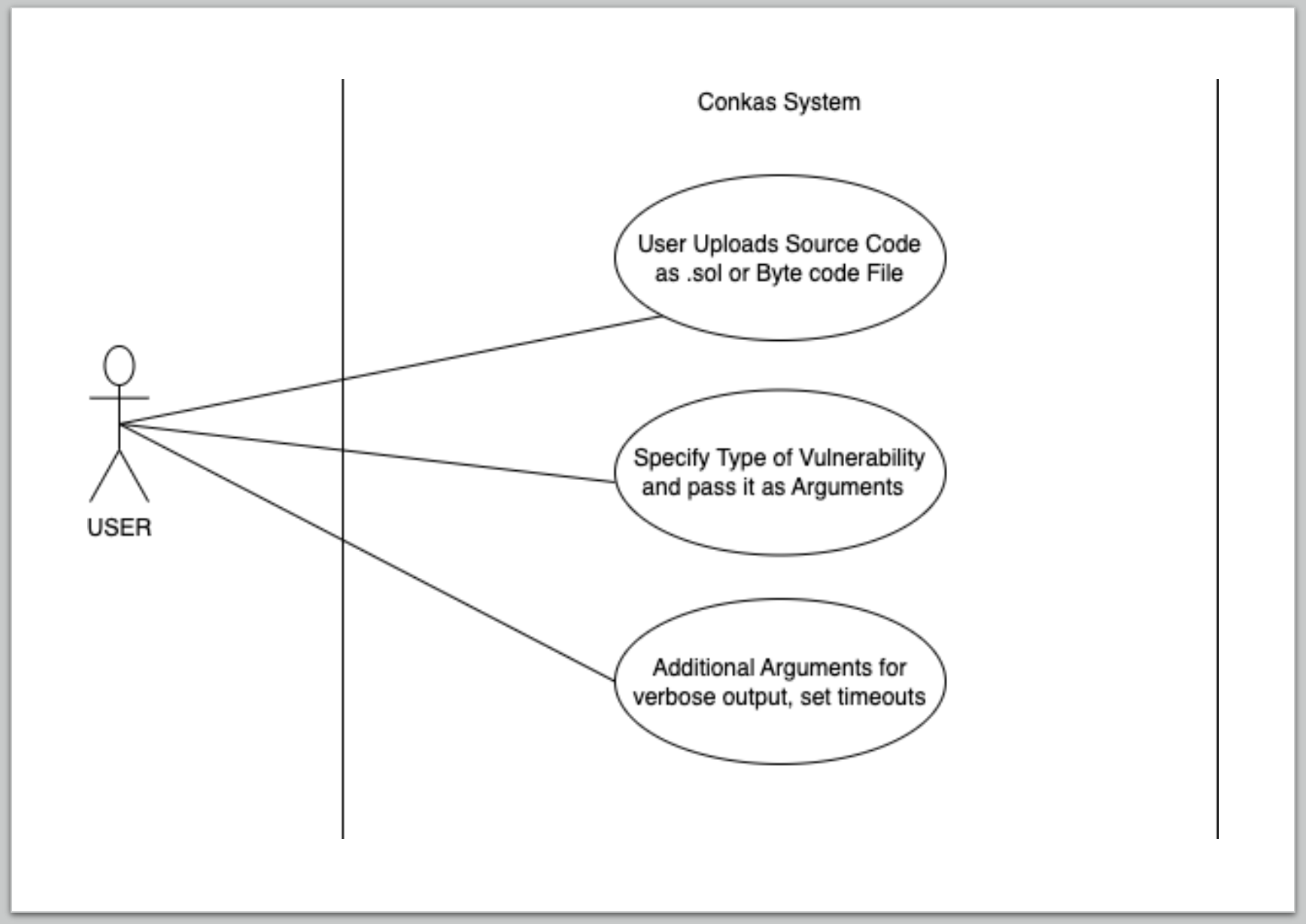
```
Symbolic execution tool for EVM

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:

$ python3 conkas.py --find-all-vulnerabilities (PATH)

```
(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

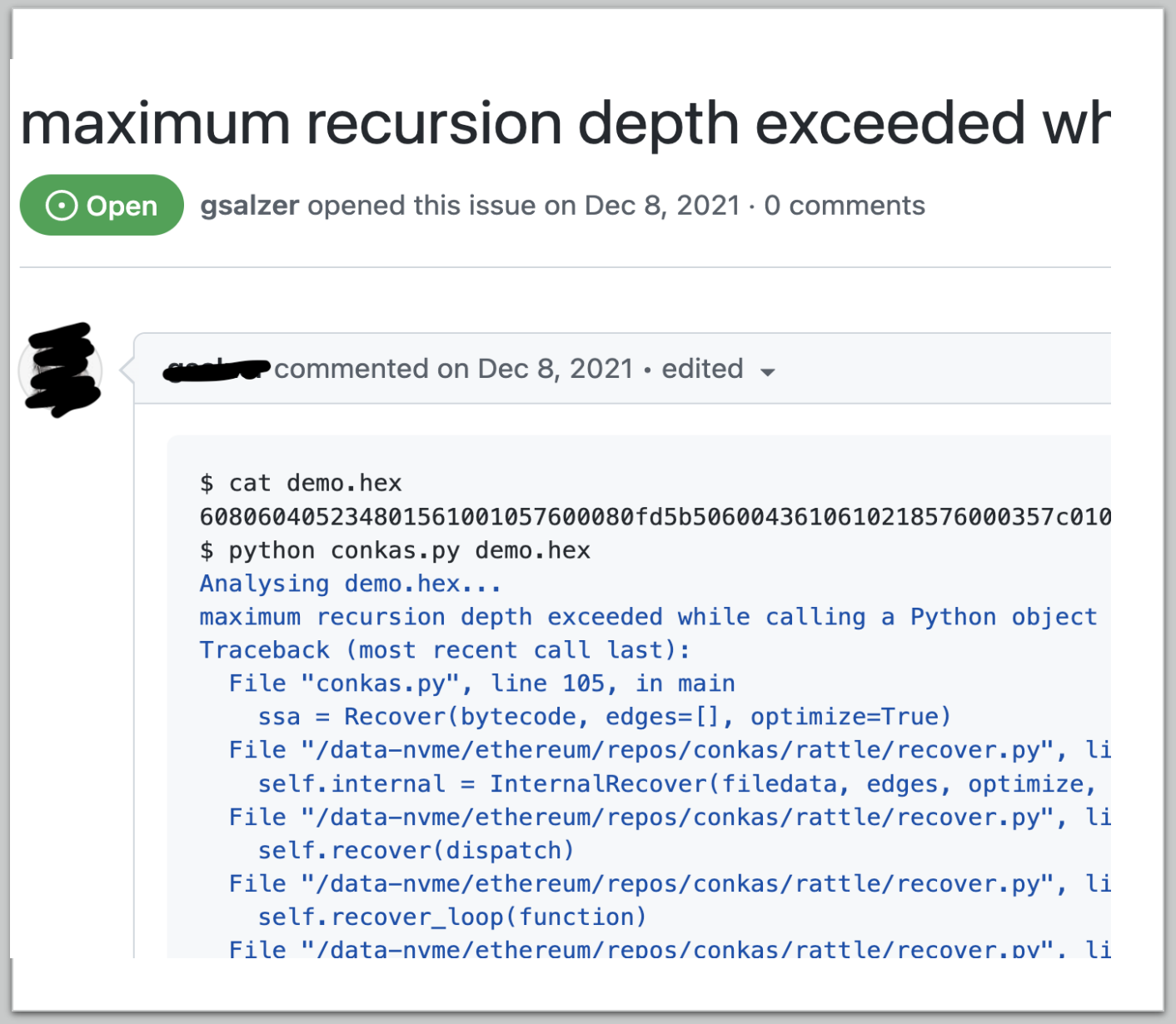
	Checked Vulnerability														Analysis Approach										
Tool	Timestamp Dependancy	Reentrency	*TOD	tx.origin	Blockhash/Block Number	Gas Related Issues	Delegate Call	**Underflow/Overflow	Freezing Ether	Unchecked Call	Self Destruct	Access Control	Denial of Service	Symbolic Execution	Dis-assembler	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	✓	✓	✓	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗

Target Users/Customers

- Decentralized application (dApp) developers:
- Cryptocurrency exchanges:
- Auditing firms:

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.



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

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


THANK
YOU