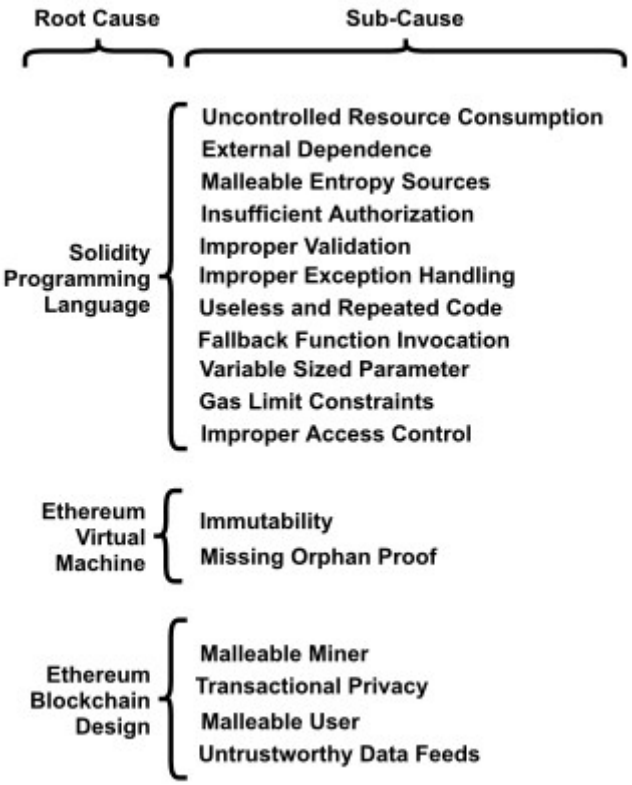
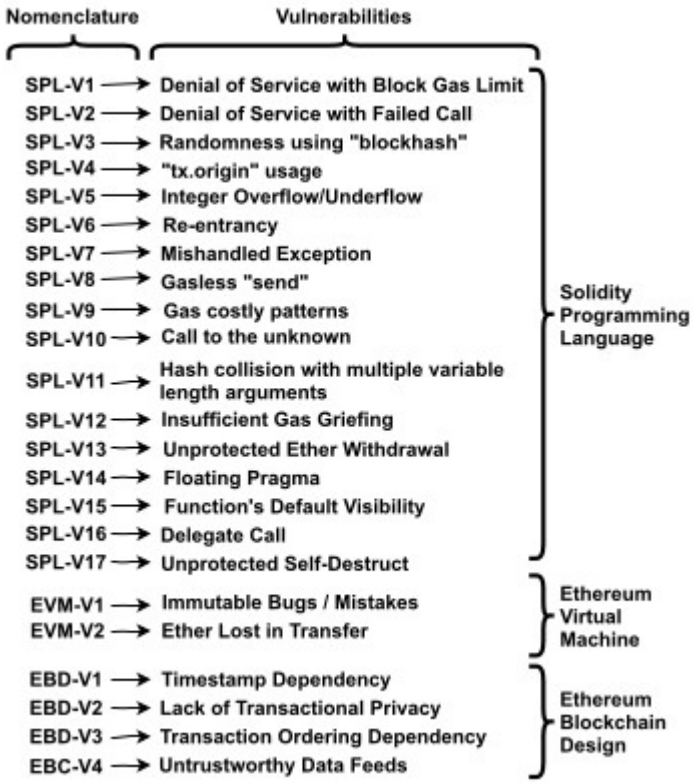
Systematic Review of Security Vulnerabilities in Ethereum Blockchain Smart Contract

Abstract
This paper presents a systematic review of security vulnerabilities in Ethereum blockchain-based smart contracts. It highlights the risks associated with these vulnerabilities, which can result in significant financial losses due to the large amounts of cryptocurrency involved. The paper discusses detection tools, real-life attacks, and preventive measures related to Ethereum smart contract security. It also compares different Ethereum smart contract analysis tools based on various features. The review sheds light on the issues associated with Ethereum smart contracts and concludes with suggestions for future research directions in this field.

Ethereum smart contract vulnerabilities



Root cause and sub-cause categorization of Ethereum smart contract vulnerabilities



Classification of Ethereum smart contract vulnerabilities and their root causes

Summary of Ethereum Smart Contract Security Vulnerabilities

1. SPL-V1 (Denial of Service with Block Gas Limit): Exceeding the gas limit of a block can lead to denial of service. Breaking transactions into multiple blocks can mitigate this vulnerability.

2. SPL-V2 (Denial of Service with Failed Call): Intentionally or accidentally failed calls can prevent other smart contract nodes from interacting. Proper contract logic handling failed calls is necessary.

3. SPL-V3 (Randomness Using 'Block Hash'): Using 'block hash' for randomness can be manipulated. External sources of randomness, such as oracles or RNGs, should be used instead.

4. SPL-V4 ('tx.origin' Usage): Using 'tx.origin' to check ownership authorization can be exploited. 'msg.sender' should be used instead to validate ownership.

5. SPL-V5 (Integer Overflow/Underflow): Calculation exceeding variable size limits can lead to undesired effects. Careful implementation with assert(), require(), and 'SafeMath.sol' library is recommended.

6. SPL-V6 (Re-Entrancy): Recursive call attacks can allow malicious users to repeatedly withdraw without affecting their balance. Using the "Check-Effect-Interaction" pattern and mutex locks can mitigate re-entrancy attacks.

7. SPL-V7 (Mishandled Exception): Mishandling exceptions can attract attackers to execute malicious code. Proper exception handling and checks on return values are crucial.

8. SPL-V8 (Gasless 'Send'): Mishandling gas limits in ether transfers can lead to out-of-gas exceptions. Thorough exception handling and efficient fallback functions are recommended.

9. SPL-V9 (Gas Costly Pattern): Inefficient code patterns can result in high gas costs. Identifying and replacing gas costly patterns with efficient alternatives is crucial.

10. SPL-V10 (Call to the Unknown): Unknown contract calls can be risky. External calls to unknown addresses should be performed as the last operation with checks and safeguards.

11. SPL-V11 (Hash Collision with Multiple Variable-Length Arguments): Improper use of 'abi.encodePacked()' can lead to hash collisions. Single value parameters and careful use of 'abi.encode()' should be considered.

12. SPL-V12 (Insufficient Gas Griefing): Insufficient gas provided in sub-calls can cause transaction failure. Implementing checks for sufficient gas and rewarding relayers can prevent this vulnerability.

13. SPL-V13 (Unprotected Ether Withdrawal): Missing or inadequate access control can allow attackers to withdraw ether. Proper access control and function naming should be implemented.

14. SPL-V14 (Floating Pragma): Using outdated compiler versions can introduce bugs. Smart contracts should be compiled using the same version used for testing.

15. SPL-V15 (Function's Default Visibility): Incorrect visibility specifiers can make contracts vulnerable. Consistently determining the visibility of all functions is essential.

16. SPL-V16 (Delegate Call): Improper use of 'DELEGATECALL' can lead to unintended execution. Stateless libraries and careful use of 'DELEGATECALL' are recommended.

17. SPL-V17 (Unprotected 'Self-Destruct'): Improper use of 'selfdestruct()' can result in ether loss. Proper access control and limiting the use of 'selfdestruct()' are preventive measures.

18. EVM-V1 (Immutable Bugs or Mistakes): Immutable smart contracts can be problematic if they contain bugs. Applying legal contract standards and redefining them for Ethereum smart contracts can allow for modifications when necessary.

19. EVM-V2 (Ether Lost in Transfer): Transferring ether to orphan contracts can result in permanent loss. Manually verifying recipient addresses can prevent ether loss.

20. EBD-V1 (Timestamp Dependency): Relying on block timestamp can be exploited. Using block number instead of timestamp is recommended.

21. EBD-V2 (Lack of Transactional Privacy): Public visibility of transaction details can limit user privacy. Encryption and privacy-preserving contract frameworks can address this issue.

22. EBD-V3 (Transaction Ordering Dependency): Vulnerabilities arise from the execution order of dependent transactions. Enforcing transaction order and guard conditions can prevent such attacks.

23. EBD-V4 (Untrustworthy Data Feeds): Dependence on external data sources can introduce vulnerabilities. Tools like Town Crier can act as trustworthy intermediaries for data feeds.

Ethereum smart contract analysis tools

Tool	Byte Code	Solidity Code	Static Analysis	Dynamic Analysis	Type of Tool	Implementation Language	Publicly Available
ContractLarva [103]	✗	✗	✗	✗	Academic	Haskell	✓
E-EVM [25]	✗	✗	✗	✗	Academic	Python	✓
EtherTrust [104]	✓	✗	✓	✗	Academic	java	✓
EthIR [105]	✓	✗	✓	✗	Academic	Python	✓
FSolidM [106]	✗	✗	✓	✗	Academic	Java Script	✓
Gasper [42]	✓	✗	✓	✗	*SDSLabs	Go	✓
KEVM [107]	✓	✗	✓	✗	Academic	Python	✓
MAIAN [59]	✓	✗	✓	✗	Academic	Python	✓
Manticore [108]	✓	✗	✓	✓	*Trail of Bits	Python	✓
Mythril [72]	✓	✗	✓	✗	*ConsenSys	Python	✓
Osiris [61]	✓	✗	✓	✗	Academic	Python	✓
Oyente [10]	✓	✗	✓	✗	Community	Python	✓
Porosity [109]	✓	✗	✓	✗	*Comae Technologies	C++	✓
Rattle [110]	✓	✗	✓	✗	*Trail of Bits	Python	✓
ReGaurd [76]	✓	✓	✗	✗	*Chieftin Lab	C++	✗
Remix-IDE [111]	✗	✓	✓	✓	Community	Java Script	✓
SASC [112]	✗	✓	✓	✗	*Fujitsu	Python	✗
sCompile [113]	✗	✓	✓	✗	Academic	C++	✗
Securify [71]	✓	✗	✓	✗	Academic	Java	✓
SmartCheck [69]	✗	✓	✓	✗	Academic	Java	✓
Solgraph [114]	✗	✓	✓	✗	*Raine Revere	Java Script	✓
SolMet [93]	✗	✓	✓	✗	Academic	Java	✓
teEther [26]	✓	✗	✓	✗	Academic	Python	✗
Vandal [115]	✓	✗	✓	✓	Academic	Python	✓
Zeus [116]	✗	✓	✓	✗	*IBM Research India	C++	✗

* Tool development company name