JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

Formalistic Occurrence of Vulnerabilities in Transactions of Blockchain Technology

Dr Challa Narasimham
IOCL- Chair Professor
Andhra University

Patnana Meghana
Department of CSE,
GMRIT

Siddamurthi Nitin Reddy
Department of CSE,
GMRIT

I.ABSTRACT

In this research, a robust approach to establishing secure smart contracts devoid of vulnerabilities such as timestamp manipulation, re-entrancy exploits, and overflow vulnerabilities. Leveraging the versatility of Python, this method meticulously addresses these critical issues that often plague smart contracts, compromising their integrity and security. This methodology involves thorough validation mechanisms, specifically designed to detect and prevent timestamp discrepancies, re-entrancy attacks, and eliminate potential overflow risks within the smart contract codebase. By integrating these security measures, smart contract implementation ensures a resilient and tamper-resistant foundation, bolstering the trustworthiness of decentralized applications. This work contributes to the advancement of secure blockchain technologies, offering a reliable framework for the development of smart contracts resistant to common vulnerabilities, thereby enhancing the overall security landscape of decentralized system.

Keywords: *Smart contract, Timestamp, Re-entrancy, Overflow, Decentralized*

II.INTRODUCTION

The integration of blockchain technology has revolutionized the landscape of smart contracts, enhancing their standards and robustness. This project explores the pivotal role of blockchain in fortifying smart contract standards, offering improved security, transparency, and trust in decentralized systems. One of the primary strengths lies in Ethereum's pioneering use of blockchain to establish contracts without reliance on traditional third-party banking sectors. Ethereum, as a decentralized platform, facilitates peer-to-peer transactions and agreements, eliminating the need for intermediaries and fostering a more efficient and inclusive financial ecosystem. This section delves into the transformative impact of Ethereum in reshaping contract establishment by leveraging the power of blockchain. Vulnerabilities such as timestamp manipulation, re-entrancy exploits, and overflow risks pose significant threats to the integrity of smart contracts. This project critically examines the adverse effects of these vulnerabilities on the reliability and security of smart contracts, emphasizing the urgency to address these issues. In response to these challenges, our work proposes a novel method for establishing smart contracts that mitigates vulnerabilities such as timestamp manipulation, re-entrancy exploits, and overflow risks. By meticulously addressing these issues, our approach aims to create a more secure foundation for smart contracts, contributing to the ongoing efforts to fortify the reliability of blockchain-based systems. Through this research,

we seek to advance the understanding of secure smart contract development and foster the continued evolution of decentralized technologies.

Vulnerabilities

a) Timestamp:

Timestamp manipulation poses a significant threat to the reliability and security of smart contracts, as malicious actors may exploit vulnerabilities in the timekeeping mechanism to gain unfair advantages or disrupt the intended functionality. In the context of smart contracts, timestamps are often utilized to record the exact moment of contract initiation or execution. However, these timestamps are susceptible to manipulation, allowing bad actors to tamper with the chronological order of transactions and compromise the integrity of the contract's logic. To address this concern, a critical aspect of our project involves the implementation of a robust timestamp module in Python. This module is specifically designed to accurately record and verify timestamps within the smart contract codebase. By incorporating secure timestamp handling in Python, ensuring the authenticity of temporal data, reducing the risk of manipulation and providing a more dependable foundation for smart contract execution. In the coding aspect, our Python module includes measures to prevent timestamp manipulation. For instance, we enforce a time restriction mechanism, preventing transactions from occurring more than 60 seconds after the initiation of the smart contract. This restriction acts as a safeguard against potential timestamp manipulations, enhancing the temporal integrity of the smart contract and reinforcing the overall security of blockchain-based systems.

b) Re-entrancy:

Re-entrancy exploits represent a critical security concern in smart contracts, where a malicious contract repeatedly calls back into the same or other contracts before the initial call completes. This type of attack can lead to unexpected outcomes, including unauthorized fund withdrawals and the manipulation of contract states. The vulnerability arises when a contract allows external calls before completing its own state changes, enabling attackers to execute additional functions during the execution of the initial call. To mitigate the risks associated with re-entrancy exploits, our project incorporates robust prevention measures in the Solidity programming language. Solidity, a language designed for writing smart contracts on blockchain platforms like Ethereum, offers features and best practices to prevent re-entrancy vulnerabilities. By strategically placing checks and controls within the contract code, we ensure that external calls cannot be re-entrant, safeguarding against unauthorized access and manipulation of contract states. Through the implementation of secure coding practices in Solidity, our project contributes to the development of smart contracts resilient to re-entrancy exploits, bolstering the overall security of decentralized applications.

c) Overflow:

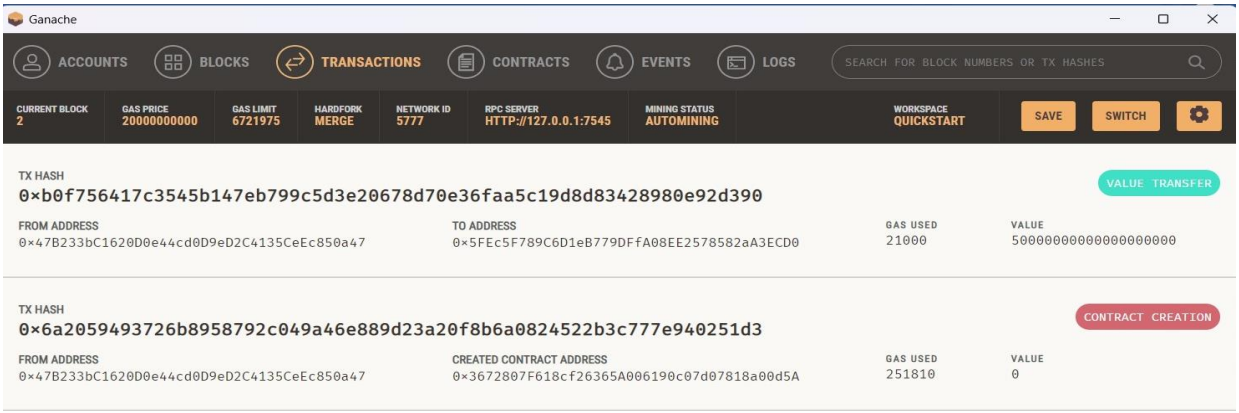
In smart contracts, overflow conditions can pose serious security risks. An overflow occurs when a numerical variable exceeds its maximum representable value, leading to unexpected behaviour and potential vulnerabilities. To mitigate this risk, it's essential to implement measures such as validating transaction parameters. The provided Python code snippet addresses overflow concerns by limiting the length of the transaction hash or receiver address to 32 characters.

III.METHODOLOGY

A smart contract is a self-executing contract where the terms and conditions of an agreement are directly written into code. These contracts are designed to automatically enforce, execute, or verify the terms of a contract without the need of trusted third parties.

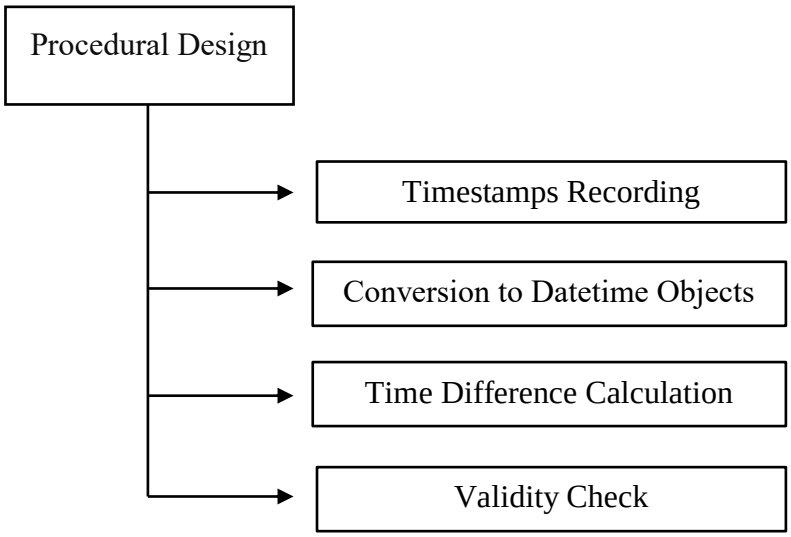
Ganache:

Ganache is open-source platform in Block chain development tool. It is a user-friendly interface which contains some accounts that generates address and private key where we can access and use them for to test our own smart contract. We can connect the Ganache platform to localhost by using RPC server.



1) Procedural Design of Time Stamp:

This mechanism helps in avoiding contracts that take an unexpectedly long time, potentially indicating an issue or manipulation in the contract execution process.



Implementation:

#Time stamp function

```
def times():
    a = time.time()
```

return a,time.ctime(a)

Findings:

```
#Contract creation
signed_tx = w3.eth.account.sign_transaction(transaction_hash, private_key=private_key)
tx_hash = w3.eth.send_raw_transaction(signed_tx.rawTransaction)
tx_receipt = w3.eth.wait_for_transaction_receipt(tx_hash)
print(f"Contract deployed to {tx_receipt.contractAddress}")
ts1,t1=times()
print("Timestamp created at :",ts1 )
print("Time is (in GMT) : ",t1)
```

```
Contract deployed to 0xd7EF539e1f5cB1b2E7F06040F8AC02a9983937c4
Timestamp created at : 1705731806.7598867
Time is (in GMT) :  Sat Jan 20 11:53:26 2024
```

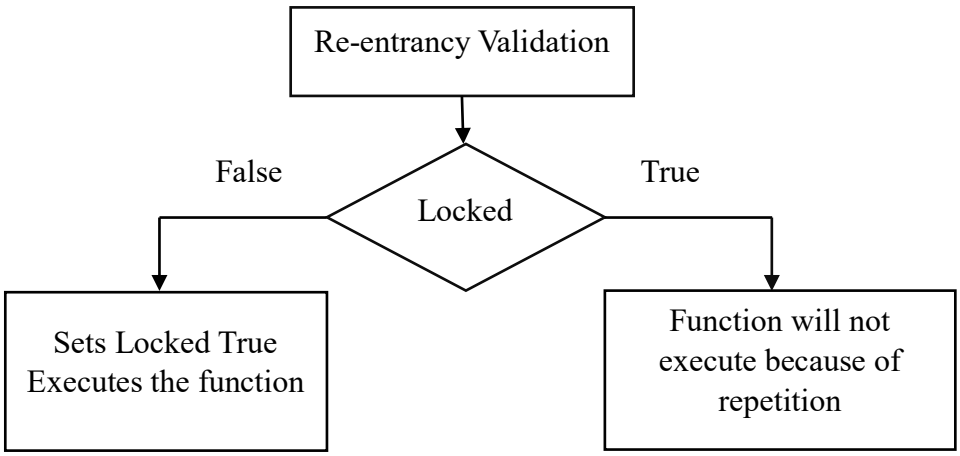
```
print(f"Transaction completed.......... \nTransaction hash: {transaction_hash.hex()}")
ts2,t2=times()
print("Timestamp created at :",ts2 )
print("Time is (in GMT) : ",t2)
```

```
Transaction completed..........
Transaction hash: 0xeb56b22b0e2edefcc8c5e1ad0f286b01c3af8146593c996c0ba264e745f2ce99
Timestamp created at : 1705731865.0876222
Time is (in GMT) :  Sat Jan 20 11:54:25 2024
```

```
from datetime import datetime
# Convert string timestamps to datetime objects
timestamp1 = datetime.utcfromtimestamp(ts1)
timestamp2 = datetime.utcfromtimestamp(ts2)

# Calculate the time difference
time_difference = timestamp2 - timestamp1
time_difference_Sec=time_difference.total_seconds()
print(f"Timestamp 1: {timestamp1}")
print(f"Timestamp 2: {timestamp2}")
print(f"Time difference: {time_difference}")
print("Time difference in seconds : ",time_difference_Sec)
```

```
Timestamp 1: 2024-01-20 06:23:26.759887
Timestamp 2: 2024-01-20 06:24:25.087622
Time difference: 0:00:58.327735
Time difference in seconds :  58.327735
```

2) Procedural Design of Re-entrancy:

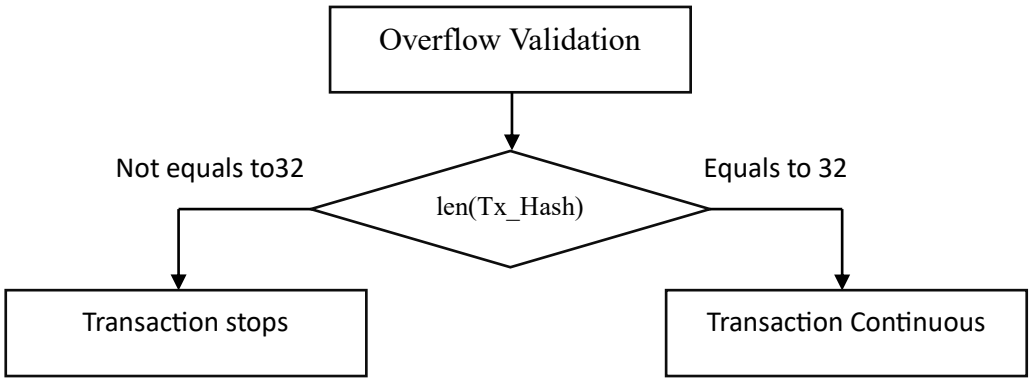


`

Findings:

```
contract_source_code = """
pragma solidity ^0.6.0;

contract SimpleStorage {
    uint256 public storedData;
    bool private locked;

    modifier nonReentrant() {
        require(!locked, "ReentrancyGuard: reentrant call");
        locked = true;
        _;
        locked = false;
    }

    function set(uint256 x) public nonReentrant {
        storedData = x;
    }

    function get() public view returns (uint256) {
        return storedData;
    }

    function store(uint256 _value) public nonReentrant {
        storedData = _value;
    }

    function retrieve() public nonReentrant returns (uint256) {
        return storedData;
    }
}
"""
```

3) Procedural Design of Overflow:



a) Findings:

```
if len(transaction_hash)==32:
    confirmations+=1
    print("No overflow in the Transation")
else:
    print("Transation overflow")
```

```
No overflow in the Transation
```

IV.CONCLUSION:

In conclusion, this research has introduced a robust approach to the establishment of secure smart contracts, effectively mitigating vulnerabilities such as timestamp manipulation, re-entrancy exploits, and overflow risks. Leveraging the flexibility and power of Python, our methodology incorporates meticulous validation mechanisms tailored to detect and prevent discrepancies that commonly compromise the integrity and security of smart contracts. By integrating these security measures, our approach ensures a resilient and tamper-resistant foundation for decentralized applications, significantly enhancing the trustworthiness of blockchain technologies. Through thorough examination and strategic addressing of timestamp-related challenges, re-entrancy risks, and potential overflow vulnerabilities, our work contributes to the evolution of secure blockchain technologies.

V.REFERENCES

[1] Sun, Y., & Gu, L. (2021, March). Attention-based machine learning model for smart contract vulnerability detection. In *Journal of physics: conference series* (Vol. 1820, No. 1, p. 012004). IOP Publishing.

[2] Singh, A., Parizi, R. M., Zhang, Q., Choo, K. K. R., & Dehghantanha, A. (2020). Blockchain smart contracts formalization: Approaches and challenges to address vulnerabilities. Computers & Security, 88, 101654.

[3] Sayeed, S., Marco-Gisbert, H., & Caira, T. (2020). Smart contract: Attacks and protections. IEEE Access, 8, 24416-24427.

[4] Perez, D., & Livshits, B. (2019). Smart contract vulnerabilities: Does anyone care?. arXiv preprint arXiv:1902.06710, 1-15.

[5] Kushwaha, S. S., Joshi, S., Singh, D., Kaur, M., & Lee, H. N. (2022). Systematic review of security vulnerabilities in ethereum blockchain smart contract. IEEE Access, 10, 6605-6621.

[6] Chen, T., Li, X., Luo, X., & Zhang, X. (2017, February). Under-optimized smart contracts devour your money. In *2017 IEEE 24th international conference on software analysis, evolution and reengineering (SANER)* (pp. 442-446). IEEE.

[7] Dhawan, M. (2017, February). Analyzing safety of smart contracts. In *Proceedings of the Conference: Network and Distributed System Security Symposium, San Diego, CA, USA* (pp. 16-17).

[8] Alharby, M., & Van Moorsel, A. (2017). Blockchain-based smart contracts: A systematic mapping study. *arXiv preprint arXiv:1710.06372*.

[9] Kushwaha, S. S., Joshi, S., Singh, D., Kaur, M., & Lee, H. N. (2022). Ethereum smart contract analysis tools: A systematic review. IEEE Access, 10, 57037-57062.

[10] Wang, S., Ouyang, L., Yuan, Y., Ni, X., Han, X., & Wang, F. Y. (2019). Blockchain-enabled smart contracts: architecture, applications, and future trends. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *49*(11), 2266-2277.