Ethereum Smart Contracts

Devansh Patel
Information System Security
Concordia University
Montreal, Canada
pa_devan@live.concordia.ca

Darshit Gajjar
Information System Security
Concordia University
Montreal, Canada
d_gajj@live.concordia.ca

Rushi Chandalia
Information System Security
Concordia University
Montreal, Canada
ru_chan@live.concordia.ca

Kamal Sharma
Information System Security
Concordia University
Montreal, Canada
kamal.sharma@mail.concordia.ca

Palak Kaur Sodhi
Information System Security
Concordia University
Montreal, Canada
p_sod@live.concordia.ca

Sachin Verma
Information System Security
Concordia University
Montreal, Canada
v_sachi@live.concordia.ca

Gulshan Joshi
Information System Security
Concordia University
Montreal, Canada
gu_joshi@live.concordia.ca

Varsha Vivek
Information System Security
Concordia University
Montreal, Canada
va_vivek@live.concordia.ca

Abstract*— To provide flash loan to users, the Unstoppable and Naive Receiver contract was created. In this paper, we thoroughly examined both contracts and found a number of weaknesses that attackers could take advantage of. We discovered that the contract is particularly susceptible to denial-of-service and reentrancy attacks. We also suggested a number of mitigation techniques, such as building fail-safe methods and providing input validation, to address these issues. Overall, our findings emphasize the significance of carrying out exhaustive security evaluations and putting in place suitable security controls to guarantee the security of DeFi applications.

Keywords—smart contracts, ethereum, blockchain, vulnerability, code analysis, exploitation.

I. INTRODUCTION

A smart contract is a self-executing commitment with integrated lines of code that define the parameters of the agreement between both the contract's counterparties. A smart agreement serves as a digital replica of the traditional paper contract that actively maintains and carries out its conditions. The smart contract is performed over a blockchain system, and the network's various computers, each contain a copy of the contract's script. This guarantees a safer and more open implementation and execution of the contract terms. Furthermore, since the code of a smart contract is checked by every member of the blockchain network, smart contracts do not need an intermediary to be validated. The cost to counterparties is diminished significantly by eliminating the intermediary from the deal. Blockchain technology serves as the foundation for the notion of smart contracts.

A blockchain is a distributed network made up of a continuously expanding list of records (blocks) connected by encryption. Unlike a traditional database, a blockchain technology does not have a single central location. Each machine on the network has permission to view the information that is recorded in the blockchain. The network is consequently less susceptible to mistakes or attacks. A data on one device cannot be changed in a blockchain without also updating the identical information on other computers in the network. With a blockchain, transactions are organised into blocks that are connected by a chain. Only once the preceding block is finished is a new block formed. Each block comprises a cryptographic hash of the preceding block and is presented in a linear chronological sequence.

There are multiple steps involved in working of smart contracts. The contract's parameters should initially be decided by the counterparties. The completed contractual requirements are then converted into a computer program. In essence, the program contains a variety of conditional statements that outline several circumstances for a potential future transaction. As a piece of code is written, it is copied across the blockchain's users and saved in the network. The code is then compiled and executed by all machines on the network. The appropriate transaction is carried out if a condition of the contract is met and has been confirmed by every user of the blockchain network.

Code analysis plays a major role in this process. Basically, code analysis investigates programming code to discover potential pitfalls or mistakes, assure compliance with coding standards, and guarantee quality. Code analysis can be done statically (viewing the program without running it) or dynamically (executing it and evaluating its performance). There are several tools and methodologies for doing code review.

II. CONTEXT

A. Common Types of Vulnerabilities in Ethereum Smart Contracts:

Smart contracts, compared to numerous other contracts, are mainly focused on monetary assets. As a result of the Blockchain's immutability, failures in smart contracts cannot be corrected after they've been implemented. Smart contract flaws might be harmful to security, as well as a tempting aim for suspicious computer hackers. In fact, regardless of whether there are outer manipulators, there is a risk of investment breakdown and economic difficulties in certain instances.

Here are some common vulnerabilities:

Reentrancy:

A risk in which an intruder can approach a contract feature recurrently before the preceding call has finished, likely to result in the intruder trying to execute their own script within the contract.

Integer overflow and underflow:

A form of security vulnerability in which a computation generates an outcome that is greater or less than the highest or lowest value that can be kept in a variable, resulting in malicious activity.

Unused variables:

Variables that are declared but not utilized can imply a coding error and may result in possible errors.

Uninitialized storage pointers:

A threat in which a memory pointer is not initialised prior to application, resulting in errors.

Function visibility:

Proclaimed public functions can be called by anybody, which includes hackers, which could also ultimately lead to security flaws.

Replay signatures attacks:

Signatures allow one account to transfer payments from another account to the blockchain. The actual account will sign a message, and the delivery account will transmit it to a smart contract so that the money transfer fees are paid by the shipping account rather than the main account.

Block gas limit vulnerability:

The block gas limit assists in preventing blocks from becoming too large. If a transfer of funds absorbs excessive gas, it's unlikely to fit in the block and will thus be ignored. Therefore, if information is maintained in arrays and then retrieved via loops over such arrays, the exchange may exhaust its gas and receive a refund. It might result in a denial-of-service (DoS) attack.

Using the Block Hash Function:

By using the block hash function, comparable to the timestamp reliance, is a strategy of attempts to hack smart contracts. It is not suggested that it be utilized for vital parts for the identical purpose as timestamp dependency: miners can modify such features and alter the funds that are withdrawn to their own benefit. This would be particularly apparent whenever the block cipher is employed as a generator of arbitrary numbers.

Incorrect Calculation of the Output Token Amount:

A large variety of actions in the contract argument are linked to token transactions to and from the contract. It opens a wide range of possibilities for errors related to calculating appropriate proportions, service charges, and revenues. That is why it is critical to work with a trustworthy token advancement business to guarantee the achievement of your endeavour.

The following are the most common errors: Incorrect digits processing, especially when interacting with a token like USDT; inaccurate command of procedure during service charge estimations, leading to substantial precision failure; and the precision incessant that was genuinely neglected in the maths processes.

B. Overview of Code Analysis

Code analysis can be done using a variety of tools and methods. They can be roughly divided into two types: static code analysis and dynamic code analysis.

Before releasing a code, static code analysis is performed to discover numerous common coding issues. It implies manually inspecting the code or using tools to automate the procedure. Analysis of static code software can examine a program without running it. They could detect syntax errors, ensure coding guidelines are adhered to, and seek out possible data breaches. It is frequently conducted before the code is deployed, at an early stage of development. It can aid in the identification of difficulties that may be hard to miss during diagnostics and save both money and time by detecting issues prior to become quite severe.

Dynamic code analysis comprises deploying the code and observing its performance. They can detect runtime issues, test the program's efficiency, and ensure that it operates correctly. It is usually done after the program has been published. It can detect issues that were not obvious throughout assessment, such as performance problems or security flaws that arise only under certain circumstances.