Smart Contracts Exercise 07: Out of Gas

1 Introduction

Denial of service attacks in smart contracts aim to render a contract temporarily or permanently unusable by manipulating its execution flow or exploiting resource limitations. Unlike traditional web applications where DoS attacks typically involve overwhelming a system with traffic, blockchain DoS attacks might be more sophisticated and can have permanent consequences. In this exercise, you will learn about several types of DoS attacks on blockchain and try to implement them in practical exercises. You will also become familiar with the concept of decentralized autonomous organization.

Project Setup

You have two options for working with this exercise: using a Docker container or local installation. Choose the one that best fits your preferences.

1.1 Using Docker with VS Code

This option uses Docker to create a development environment with all the necessary tools and dependencies preinstalled.

Prerequisites

- **Docker** - A platform for developing, shipping, and running applications in containers.
- **Visual Studio Code** - A lightweight but powerful source code editor.
- **Dev Containers** - An extension to VS Code that lets you use a Docker container as a full-featured development environment.

Setting Up the Project:

1. Visit the following GitLab repository and clone it to your local machine.
2. Open the repository folder in VS Code.
3. When prompted, click "Reopen in Container" or use the command palette (F1) and run `Dev Containers: Reopen in Container`.

1.2 Local Setup

If you prefer working directly on your machine without Docker, you can set up the development environment locally.

Prerequisites

- **Node.js**: https://nodejs.org/en/ - An open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser.
- **NPM**: Node Package Manager, which comes bundled with Node.js.

Open your terminal and run the following commands to verify the installations:

```
$ node -v
$ npm -v
```

Both commands should return the installed version numbers of Node.js and NPM respectively. Node.js provides the runtime environment required to execute JavaScript-based tools like Hardhat, while NPM is used to manage the packages and dependencies needed for development.

Setting Up the Project

1. Visit the following GitLab repository and clone it to your local machine.
2. Open a terminal and navigate to the project directory.
3. Install the project dependencies by running `npm install`.

2 Denial of Service Attacks

Denial of Service (DoS) attacks in Ethereum smart contracts occur when a malicious actor prevents legitimate users from using a contract's functionality. There are several types of DoS attacks that can affect smart contracts.

2.1 DoS with Block Gas Limit

Ethereum blocks have a maximum gas limit. If a function requires more gas than this limit to execute, it becomes impossible to call that function. This commonly happens with functions that loop through unbounded arrays, make multiple external calls, or perform complex computations. Read more about gas: https://ethereum.org/gas/.

The following code shows a contract that iterates through an unbounded array to check if someone is a member of the organization. With each additional member in the organization, the join function requires more and more gas to execute, making it increasingly expensive to run. Eventually, when the member count becomes too high, the join() function will become impossible to call due to exceeding the block gas limit. You can try this yourself in the prepared file in REMIX IDE.

```
contract Organisation {
    mapping(uint => address) members;
    uint public membersCount;
    uint public gasNeeded;

    function isMember(address addr) public view returns (bool) {
        for(uint i = 0; i < membersCount; i++) {
            if(members[i] == addr) return true;
        }
        return false;
    }

    function join() public {
        uint gas = gasleft();
        if(!isMember(msg.sender)) {
            members[membersCount++] = msg.sender;
        }
        gasNeeded = gas - gasleft();
    }
}
```

Listing 1: DoS with Block Gas Limit Example

2.2 DoS with Unexpected Revert

When a contract requires all operations in a batch to succeed, a single rejected operation can block the entire process. See this example in REMIX IDE. In this case, the successful execution of the `claimThrone()` function depends on the transfer operation completing successfully. If a smart contract becomes the current king and doesn't implement a receive function, it will remain the king forever.

```
contract KingOfEther {
    address public king;
    uint public balance;

    function claimThrone() external payable {
        require(msg.value > balance, "Need to pay more to become the king");
        payable(king).transfer(balance);
        balance = msg.value;
        king = msg.sender;
    }
}

contract MaliciousKing {
    function attack(KingOfEther kingOfEther) external payable {
        kingOfEther.claimThrone{value: msg.value}();
    }

    // No receive function - will cause DoS when transfer is attempted
}
```

Listing 2: DoS with Revert Example

2.3 Owner Operations DoS

If a contract relies on an owner for critical operations and the owner's private key is lost or the owner becomes unresponsive, the contract can become permanently unusable.

```
function withdrawFunds() public onlyOwner {
    // What if the owner loses their private key?
    payable(owner).transfer(address(this).balance);
}
```

Listing 3: Owner Operations DoS Example

In this example, if the owner loses their private key or becomes unresponsive, the funds are permanently locked.

2.4 DoS Mitigation Strategies

Pull Over Push Payment Pattern

Instead of pushing payments to users, let users pull their own funds. This pattern shifts the responsibility of handling failed transactions to individual users rather than affecting everyone.

```
mapping(address => uint256) public pendingWithdrawals;

// Allow users to claim their funds individually
function withdraw() public {
    uint amount = pendingWithdrawals[msg.sender];
    // Remember to set pending withdrawals to 0 before transfer
    pendingWithdrawals[msg.sender] = 0;
    payable(msg.sender).transfer(amount);
}
```

Listing 4: Pull Over Push Example

Avoid Unbounded Operations

Avoid iterating through unbounded arrays or performing unbounded operations that could exceed the block gas limit.

```
function processBatch(uint start, uint batchSize) public {
    // Process only a limited number of items per transaction
    uint end = start + batchSize;
    require(end <= array.length, "Invalid batch");

    for(uint i = start; i < end; i++) {
        // Process items in batches
        processItem(array[i]);
    }
}
```

Listing 5: Bounded Operations Example

External Calls Handling

When making external calls, handle failures gracefully without causing the entire function to revert.

```
function processPayments() public {
    for(uint i = 0; i < recipients.length; i++) {
        // Note missing require statement - we continue even if one transfer fails
        (bool success, ) = recipients[i].call{value: amounts[i]}("");
        if(success) {
            emit PaymentSuccessful(recipients[i], amounts[i]);
        } else {
            emit PaymentFailed(recipients[i], amounts[i]);
            // Store failed payments for later retry
            failedPayments[recipients[i]] = amounts[i];
        }
    }
}
```

Listing 6: Safe External Call Example

3 Decentralized Autonomous Organizations (DAOs)

DAOs are blockchain-based organizations governed by smart contracts and controlled by their members rather than by a central authority. They rely on on-chain voting mechanisms to make decisions and allocate resources. The foundation of a DAO is a smart contract on the blockchain that precisely establishes the rules by which the organization will be governed. Examples of DAOs include charities (where members decide where to contribute funds), collective ownership structures, and ventures and grants - where venture funds pool investment capital and members vote on which ventures to support. Key components of DAOs include:

1. **Membership Management**: Rules for joining the DAO, often by holding governance tokens or paying membership fees.
2. **Proposal Mechanism**: Allowing members to submit proposals for changes or funds allocation.
3. **Voting System**: A system for members to vote on proposals.
4. **Execution Mechanism**: Logic for implementing approved proposals.

One of the most infamous DAO organizations was the notorious The DAO, which you learned about in Exercise 05 - Re-entrancy. You will see a simple example of a DAO in Task 2.

DAO	Traditional Organization
Usually flat, and fully democratized.	Usually hierarchical.
Voting required by members for any changes to be implemented.	Depending on structure, changes can be demanded from a sole party, or voting may be offered.
Votes tallied, and outcome implemented automatically without trusted intermediary.	If voting allowed, votes are tallied internally, and outcome of voting must be handled manually.
Services offered are handled automatically in a decentralized manner (for example distribution of philanthropic funds).	Requires human handling, or centrally controlled automation, prone to manipulation.
All activity is transparent and fully public.	Activity is typically private, and limited to the public.

Table 1: DAOs and Traditional Organizations (from: Ethereum.org)

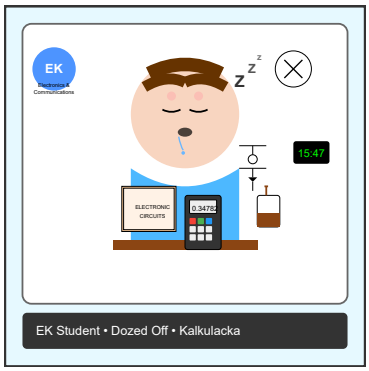
More about DAOs:

- **Read:** What is a DAO?
- **Video:** What is a DAO in Crypto?
- **TED Talk:** Could a DAO Build the Next Great City?

4 Task

Task 1: NFT Auction Sabotage

One of the most valuable students from the FEL NFT Students collection is now for sale. It depicts an Electronics and Communication student who dozed off with a calculator. The owner (seller) has decided not to sell this rare piece on an ordinary marketplace but through an auction. The starting price is 0.5 ETH, but it quickly rises to 2.8 ETH. You have 3 ETH available and want to acquire this piece at all costs! Suddenly, a well-known speculator and reseller appears and announces on the auction forum that they'll pay 50 ETH for the NFT student. You realize that you no longer have any chance of winning the auction. But you decide: if you can't have the NFT, no one will!



Your Mission:

- Prevent the reseller from acquiring the NFT Student - your code will run before the reseller (bidder 3) places their 50 ETH bid

- Ensure that none of the other auction participants can acquire the NFT Student either

Code your solution in the `test/NFTAuction.js` file. Use only the player account or contracts that you deploy using the player account. Verify your solution by running the following command:

```
$ npm run auction
```

Files that are relevant for this challenge:

- test/**NFTAuction.js**: The test file where you should code your solution.
- contracts/**NFTAuction.sol**: The auction contract where the auction is taking place.
- contracts/**FELStudentNFT.sol**: The NFT collection contract.

Task 2: Save the DAO Funds From the Cats

You decided to join a DAO with other people to collectively manage your finances. The founder of the DAO contributed 10 ETH, and anyone who wants to join the DAO must pay a membership fee of 1 ETH. Everything looks promising until one member proposes to donate all the funds to a cat charity organization. This proposal is very popular and all members (except you) vote in favor of it. By coincidence, it's the same cat charity that you exploited in Exercise 4, but this time it's sufficiently protected against reentrancy attacks. You hate cats and don't want the funds to reach them under any circumstances.

Your Mission:

- Prevent the execution of the proposal to donate funds to the cat charity
- Ensure the treasury funds remain in the DAO
- You have 1.5 ETH available to achieve this goal

Code your solution in the `test/DAO.js` file. Use only the player account, which is already a DAO member. Verify your solution by running the following command:

```
$ npm run dao
```

Files that are relevant for this challenge:

- test/**DAO.js**: The test file where you should code your solution.
- contracts/**DAO.sol**: The contract forming your Decentralized Autonomous Organization.
- contracts/**CatCharity.sol**: The cat charity contract. You do not need to pay any special attention to this contract.