Introduction to Decentralized Finance

Homework 2

1 Announcement

- The assignment contains 6 programming problem.
- If you encounter any issues with the homework, please first visit the discussion forum on NTU COOL.
- If you need further assistance, attend the TA hour so the teaching assistant can help you with your problem.
- If you do not receive a response in the forum, or if posting your question would reveal your answer, leave a comment in your pull request (PR), and the TA will reach out to you.
- If you encounter any issues with Homework 2 that cannot be resolved using the methods above, please reach out to `2b@csie.ntu.edu.tw` following these guidelines:
  1. Title your email `[DeFi-HW2][Summary-Of-Your-Issue]`. Please note that we will **NOT** receive emails in other formats as we have applied filters to our email system.
  2. Provide detailed information about your computer, including the operating system.
  3. Outline the methods you attempted previously, the resources you consulted, the steps you followed, and the results of your efforts.
  4. Note that ambiguous requests, such as attaching screenshots without proper descriptions, will not be answered.
- There is **NO** late submission allowed. After the deadline, you will automatically lose write access to the repository. Please ensure you push your code changes to GitHub before the deadline.
- You are **NOT** allowed to modify any protected files, such as `.github/**/*`, and we will provide these paths for you. You will receive zero points if any protected file is modified.

2 Preliminaries

- Create a homework repository by clicking this link.
- **Important**: Select your name and link it to your GitHub account. Failure to complete this step may result in a penalty.
- Clone the repository, enter the project directory, and install dependencies using: `cd hw && forge install`.
- Add necessary packages, run `forge install OpenZeppelin/openzeppelin-contracts --no-commit`
- **Optional**: If you encounter any issues with package linking, run `forge remappings`.

3 Problem 1: Merkle (35 pt)

The Merkle tree is an essential data structure in smart contract development. Why can't we simply distribute tokens using a for-loop? This approach can lead to potential DoS vulnerabilities and incur high gas costs. After students developed the `NFinTech` NFT for Professor Liao, he decided to distribute these NFTs using a Merkle distributor. He wrote two versions, but both contain bugs in their implementation. Please help him identify the issues.

3.1 Description

- `BadLeaf`: Professor Liao has prepared 4 NFTs for students who performed exceptionally well during the course. The recipients are `user0`, `user1`, `user2`, and `user3`. `user0` generates a leaf node by providing the account and the lucky number given by the professor, and then claims the NFT by submitting the leaf node and the corresponding Merkle proof. After verification, `user0` successfully receives the NFT! Although you are not eligible for the NFT reward, you are able to mint 2 NFTs from the contract. How can you accomplish this?
- `BadProof`: Less code means fewer bugs. To minimize potential vulnerabilities, the Merkle distributor's codebase has been simplified. This time, you're eligible for the reward! (See the comments in `BadProof.t.sol` for claim instructions.) However, you can claim 7 NFTs as a reward, even though the teacher intended to give you only 1 NFT. Identify the vulnerability in the contract that allows you to claim more rewards.

3.2 Judge

The tests are in `hw/test/Merkle/BadLeaf/BadLeaf.t.sol` and `hw/test/Merkle/BadProof/BadProof.t.sol`.

- BadLeaf (20 pt): `cd hw && forge test --mc BadLeafTest --mt testExploit`
- BadProof (15 pt): `cd hw && forge test --mc BadProofTest --mt testExploit`

3.3 Protected Files

- `hw/src/Merkle/BadLeaf.sol`
- `hw/src/Merkle/BadProof.sol`
- `hw/test/Merkle/BadLeaf/BadLeafBase.t.sol`
- `hw/test/Merkle/BadProof/BadProofBase.t.sol`

4 Problem 2: Upgradeable (30 pt)

The upgradeable proxy pattern is crucial in the DeFi space, with three distinct variations: UUPS, Transparent Proxy, and Beacon Proxy. Complete the following exercise to understand the structure of these proxies and their underlying vulnerabilities.

4.1 Description

- `UUPS`: There is a potential vulnerability in the UUPS contract where a hacker could re-initialize the contract, maliciously upgrade it, and ultimately destroy it. Please identify the issue in the proxies and try to eliminate that the logic contract byte code. (Note: Here we assume the contract is deployed on an EVM-compatible chain that `selfdestruct` is not deactivated.)
- `Transparent`: There is a potential issue called 'function clashing' in the Transparent Proxy Pattern, which occurs when function selectors are duplicated in both the proxy and implementation contracts. Please complete the proxy pattern in `Transparent.t.sol` to ensure this issue is avoided.

4.2 Judge

Tests are in `hw/test/Upgradeable/UUPS/UUPS.t.sol` and `hw/test/Upgradeable/Transparent/Transparent.t.sol`.

- UUPS (15 pt): `cd hw && forge test --mc UUPSTest --mt testExploit`
- Transparent (15 pt): `cd hw && forge test --mc Transparent --mt testExploit`

4.3 Protected Files

- `hw/src/Upgradeable/UUPS.sol`
- `hw/test/Upgradeable/UUPS/UUPSBase.t.sol`
- `hw/test/Upgradeable/Transparent/TransparentBase.t.sol`
- `hw/test/Upgradeable/Transparent/Transparent.t.sol`

5 Problem 3: Logic (35 pt)

The most common attack vector for smart contracts is logic errors. And there are two cases for you: `GPU` and `Multisig`.

In the problem of the 'Multisig' wallet, it is a 3-out-of-5 multisig, meaning that a transaction (or proposal) will only be executed if at least 3 signers approve it. Anyone in this 'Multisig' implementation can submit a transaction (or proposal). Although it is designed as a 3-out-of-5 multisig wallet, there are only 4 participants—`user0`, `user1`, `user2`, and `user3`—so they decided to leave the fifth signer empty.

They initially deployed their multisig wallet on an EVM-compatible chain, `LiaoChain`, due to its high speed. `user0` attempted to transfer `LiaoToken` from the multisig wallet to support charity activities. `user0` first signed a message with an ID of 3, where the target address was `LiaoToken`'s address, and obtained the following signature.

0xddbfaf1f1237db98f2e8517d9c2111c6184e927a9489
c3ec30a95903fd16de59110b30fee6f7fb2f440dc22d58a7a4ec2ee596 6d81e3e1bf66c9473bda6a911e1b

`user0` collected the remaining 2 signature from `user1` and `user3`, and used these signatures for verification, each signature was split into three fields: `(bytes32 r, bytes32 s, uint8 v)`.The `ecrecover` function was then used to recover the signer's address. The multisig wallet **should** ensure that the number of signatures exceeds the required threshold for the transaction to be approved.

They used `LiaoChain` for some time but decided to migrate their project to the `Ethereum` Mainnet, so they deployed a new multisig wallet with the same signer settings on the mainnet. During the migration, the deployer accidentally transferred all of their `LiaoToken` on Ethereum into the multisig wallet. While they can transfer the tokens out of the wallet, hackers have the opportunity to act first and claim all the tokens before they do. As an ethical white hat, how would you help transfer the tokens out of the multisig wallet before bad actors exploit this vulnerability? (Note: The address of `LiaoToken` is the same on both `Ethereum` and `LiaoChain`. This can be achieved by `create2` opcode.)

5.1 Description

- `GPU`: This is a real-world case study that you might find online. A poorly designed ERC-20 token can be easily exploited. Initially, you have `10**18` GPU tokens, but your goal is to increase that to `10**19` GPU tokens.
- `Multisig`: A multisig wallet is a type of cryptocurrency wallet that requires multiple private keys to authorize a transaction.

5.2 Judge

The tests are in `hw/test/Logic/GPU/GPU.t.sol` and `hw/test/Logic/Multisig/Multisig.t.sol`

- `GPU` (15 pt): `cd hw && forge test --mc GPUTest --mt testExploit`
- `Multisig` (20 pt): `cd hw && forge test --mc MultisigTest --mt testExploit`

5.3 Protected Files

- `hw/src/Logic/GPU.sol`
- `hw/src/Logic/Multisig.sol`
- `hw/test/Logic/GPU/GPUBase.t.sol`
- `hw/test/Logic/Multisig/MultisigBase.t.sol`