Lecture 3: Understanding and Breaking Smart Contracts

Total de pontos 2/56

Email *

lalanda.alberto@gmail.com

✗ What is a smart contract? * ···/1

A smart contract is a publicly verifiable, immutable and programmatic third party. ✗

Feedback

A program that runs on the blockchain. You can call it trusted third party with public state.

✗ Name and explain two type of accounts on Ethereum * ···/4

There are two types of accounts in Ethereum.
First, externally owned accounts, controlled by people with private keys and second contract accounts, controlled by smart contract code and storage.

Feedback

Externally owned account.
This is a private-public key pair. Owned by a user.

Contract account.
Code and storage for the program.

✗ Name and explain three types of transactions in Ethereum * ···/6

The three types of transactions in Ethereum are:
Value Transaction: Transfer coins from an externally owned account to another Ethereum account
Invocation Transaction: Calls a function in a smart contract.
Deployment Transaction: The bytecode of a smart contract is used to instantiate it in the database of the EVM.

Feedback

Value transaction.
Send coins to another account. No program execution/deployment.

Deployment Transaction.
Deploys a smart contract to the network.

Invocation Transaction.
User can call a function in a smart contract.

✓ What is gas? * 1/1

- ○ Gas is a standard metric for measuring only the storage of a transaction.
- ◉ Gas is a standard metric to associate the approximate cost of an operation cost in the EVM in terms of storage and execution. ✓
- ○ Gas is a standard metric for measuring only the computation of a transaction.

✗ What is a fee market? * ···/1

Users compete by attaching a tip to their transaction that is paid to the block producer for including their transaction. ✗

Feedback

Users attempt to entice a block producer to include their transaction in a block. It is competitive as all users are competing to entice the block producer.

✗ At a high level, what is the compilation process for Solidity before it is deployed to the network? *···/1

The compilation process for solidity before deployment is, first the solidity code is transformed into assembly code, then the assembly code is transformed into hexadecimal format. ✗

Feedback

Source code is compiled to bytecode. The bytecode is a representation of the opcodes.

Breaking smart contracts

✗ What is a front-running attack for a smart contract and how can a user perform it without collaborating with a miner? Please explain it with an example. *···/6

A front-running attack is when a transaction is broadcast to be added to a new block and an attacker front runs with his own transaction for his benefit with the detriment of the victims transaction. This can be made by a miner, by adding his own transaction before the victim, or it can be made by paying a higher fee than the victim, to be added first.
For example the victim can try to register a ENS domain name and the attacker can register it first, and profit by trying to sell the domain to the victim afterwards.

Feedback

A front-running attack requires the attacker to publish a transaction in ad- vance that changes the execution of an honest party's transaction. For example, if the user thinks they will withdraw some coins from a smart contract, the adversary can broadcast a transaction that changes the contract's state such that the coins are no longer available. Typically, miners include the highest fee paying transactions first, so an adversary just needs to pay a higher fee than an honest user.

✗ What is the checks-effects-interaction paradigm? Please provide an example. *···/3

The checks-effects-interaction paradigm is a best practice guideline for safe smart contracts. If there is a function that sends a transaction to another account the steps should be, first checks(all the input validation and checks), effects (update the local state) and interactions (interact with other contracts). This is to prevent re-entrancy attacks, the most important part being, updating the state before interacting with other contracts. The attacker contract can use the fallback funtion to call the same function of the victim contract on a loop, so if the local state is not updated, the function can send funds to the attacker contract until the victim contract is empty or there is no more gas left for the transaction execution.

Feedback

When coding a function, we should always check the pre-conditions are satisfied (i.e. user has sufficient balance to withdraw), then we should update the local state (i.e. remove the user's balance), and then interact with other accounts/contracts (i.e. send the balance).

✗ Why is a contract-in-the-middle attack possible if tx.origin is used instead of msg.sender? *···/4

For example a victim can interact with a malicious contract, which can call a vulnerable vault contract. The malicious contract can call the withdraw functions, and if the vulnerable contract uses tx.origin (victim account) to check the address funds in the vault, he can validate the withdraw to the attacker contract. The withdraw would not be validated if the vulnerable vault contract used msg.sender, since that would return the attacker contract address.

Feedback

tx.origin identifies the transaction signer and msg.sender identifies the immediate caller. If contract B relies on tx.origin then it cannot distinguish if the immediate caller is the transaction signer or a malice contract. (i.e. malice contract can trick a user to call it, and then it calls contract B to steal masquerade as the user). If contract B relies on msg.sender, then it will always distinguish the immediate caller's address.

✗ What is an out-of-gas exception and how can it be used to lock up coins? *···/4

An out-of-gas exception is thrown when a function is not able to continue execution because there is no gas left provided by the caller. It can lock up coins if the point on a function when coins are withdrawn can not be reached because there needs to be commited too much gas.

Feedback

Whenever a transaction runs out of gas (i.e. it has used up all purchased gas), then all computation will be reverted as if the transaction never happened. If there is an expensive computation in the contract before it sends coins to a user, then the expensive computation may prevent the transaction ever reaching the code that sends the coins.

✗ Figure 1 outlines a contract that is vulnerable. How can a malice contract use selfdestruct to attack it ···/4

```
contract NotAPonzi {

  function playGame() public payable {
    require(msg.value== 1 ether); // 1 coin deposit

    // Contract has 10 eth?
    if(address(this).balance == 10 ether) {
      msg.sender.transfer(address(this).balance);
    }
  }
}
```

Fig. 1: A broken contract

A malice contract can break this contract by using selfdestruct to send the ethereum balance of the malice contract to the provided address. This will force the victim contract to receive ethereum, therefore making the requirement of the balance being precisely 10 ether impossible to achieve, and locking up the remaining ether inside the contract.

Feedback

The command selfdestruct will always send coins to the appointed con- tract, even if that contract has no payable function. Thus an adversary can set up a malice contract with 1 wei, and then selfdestruct it with the inten- tion to send 1 wei to the victim contract. Since its balance has 1 wei, it may never satisfy the condition (i..e balance == 10 eth) for coins to be sent.

✗ At a high level, how does a re-entrancy attack work? And how can we prevent it? *···/6

Re-entrancy works when there is a victim contract, with an withdraw function, that can send funds from his balance to an account. When the attacker contract calls the withdraw function the first time, and funds are sent to the attacker address, the fallback function is triggered. An attacker contract can be made, that, using the fallback function can call the withdraw function in a loop, emptying the funds of the victim contract, by withdrawing more than he his allowed. We can prevent re-entrancy by using the checks-effects-interaction paradigm.

Feedback

A re-entrancy attack requires the victim contract to send coins before up- dating its internal state. This lets an adversary contract re-call the victim contract's withdrawal contract several times via its fallback function. After the attack, several withdrawals will have happened, but the balance is potentially only deducted once. To prevent the attack, we can either use the checks-effects-interaction paradigm or reduce the gas allocation for the receiving contract (i.e. transfer/send function).

Advanced Contracts

✗ What is an oracle smart contract? And why do we need it? * ···/4

An oracle smart contract is a contract used to verify, store and provide access to information from the external world. That information can then be accessed by smart contracts on the EVM. Oracles are essencial, for smart contracts to have access to information brought from outside the EVM, e.g. weather, price of oil etc.

Feedback

An oracle smart contract verifies information from the external world and stores it for later use. It is required as a smart contract cannot interact with the external world (i.e. pull information) as this would prevent users deterministically validating the blockchain at some point in the future.

✗ What is a smart contract factory? And why is it useful to let another smart contract verify the code deployed? *···/4

A smart contract factory is a contract to deploy multiple instances of smart contracts of the same type. It is usefull to be sure of the code that was actually deployed, a child contract produced by this factory will always have equal code, with different addresses.

Feedback

A factory lets us re-deploy different instances of a smart contract using the same code. It is useful for a smart contract to use the factory as it is confident that the instance matches the expected code. Otherwise, it can only provided an address of a pre-deployed contract and there is no guarantee what code it is actually executing.

✗ Name and explain three permission systems that can be built into smart contracts. *···/6

Three example of permission systems are, multi-sig agreement, that requires a certain number of signers to aprove a transaction, time-delay, that requires that a certain period of time passes before the function call can be triggered, and owner authorisation, that requires that only the contract owner can call the function.

Feedback

Onlyowner.
The contract can have a single owner who can call special functions. Typically relies on the onlyOwner modifier.

Multi-signature.
A function cannot be called unless it has received a sig- nature from k out of n parties.

Pre-conditions.
A function may only be ready to call after a list of pre- conditions are satisfied. i.e. a vote function only works during the voting phase of a voting contract.

✓ What does SafeTransfer in ERC271 do? * 1/1

- ○ It checks additional information about the sender before transferring the NFT.
- ◉ It checks the receiver is a smart contract that is capable of transferring the NFT. ✓
- ○ It checks the receiver is NOT a smart contract account. It is an externally owned account.