History

- 2009 - Bitcoin
- 2014/2015 - Ethereum
Ethereum is a computer
— Gavin Wood
<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ETH =</td>
<td>$563.97</td>
</tr>
<tr>
<td>24 Hour Change</td>
<td>$11.63</td>
</tr>
<tr>
<td>24 Hour High</td>
<td>$566.92</td>
</tr>
<tr>
<td>24 Hour Low</td>
<td>$517.19</td>
</tr>
<tr>
<td>24 Hour Volume</td>
<td>$306.75M</td>
</tr>
<tr>
<td>Market Cap</td>
<td>$55.45B</td>
</tr>
</tbody>
</table>
What on earth is Vitalik Buterin wearing now?  

**Something awesome, that’s what!**  

Scroll down to sauce these badboys...

---

**NOTE:** Anyone could have ripped off these designs & sold them using print on demand. Instead, I have done my best to find and support the original artists.  

**TIPJAR:** 0x953d20Ee7d304B3DcB5D9e9C3bb261644C2F14A
A Next-Generation Smart Contract and Decentralized Application Platform

Satoshi Nakamoto's development of Bitcoin in 2009 has often been hailed as a radical development in money and currency, being the first example of a digital asset which simultaneously has no backing or "intrinsic value" and no centralized issuer or controller. However, another - arguably more important - part of the Bitcoin experiment is the underlying blockchain technology as a tool of distributed consensus, and attention is rapidly starting to shift to this other aspect of Bitcoin. Commonly cited alternative applications of blockchain technology include using on-blockchain digital assets to represent custom currencies and financial instruments ("colored coins"), the ownership of an underlying physical device ("smart property"), non-fungible assets such as domain names ("Namecoin"), as well as more complex applications involving having digital assets being directly controlled by a piece of code implementing arbitrary rules ("smart contracts") or even blockchain-based "decentralized autonomous organizations" (DAOs). What Ethereum
ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER
EIP-150 REVISION

DR. GAVIN WOOD
FOUNDER, ETHEREUM & ETHCORE
GAVIN@ETHCORE.IO

ABSTRACT. The blockchain paradigm when coupled with cryptographically-secured transactions has demonstrated its utility through a number of projects, not least Bitcoin. Each such project can be seen as a simple application on a decentralised, but singleton, compute resource. We can call this paradigm a transactional singleton machine with shared-state.

Ethereum implements this paradigm in a generalised manner. Furthermore it provides a plurality of such resources, each with a distinct state and operating code but able to interact through a message-passing framework with others. We discuss its design, implementation issues, the opportunities it provides and the future hurdles we envisage.

1. INTRODUCTION

With this in mind, we first present an overview of the evolutionary process from the initial Ethereum design and its implementation to the present state, before exploring the blockchain paradigm in detail.
Beigepaper:
An Ethereum Technical Specification

Micah Dameron

Abstract

The Ethereum Protocol is a deterministic but practically unbounded state-machine with two basic functions; the first being a globally accessible singleton state, and the second being a virtual machine that applies changes to that state. This paper explains the individual parts that make up these two factors.

1. Imagining Bitcoin as a Computer

...
Ethereum 2.0 Mauve Paper

Over the past decade, projects such as Bitcoin, Namecoin and Ethereum have shown the power of cryptoeconomic consensus networks to bring about the next stage in evolution of decentralized systems, potentially expanding their scope from simply providing for the storage of data and messaging services to managing the “backend” of arbitrary stateful applications. Proposed and implemented applications for such systems range from globally accessible cheap payment systems to financial contracts, prediction markets, registration of identity and real world property ownership, building more secure certificate authority systems and even keeping track of the movement of manufactured goods through the supply chain.

However, there remain serious efficiency concerns with the technological underpinnings of such systems. Because every full node in the network must maintain the entire state of the system and process every transaction, the network can never be more powerful than a single computer. The consensus mechanism most often used in existing systems, proof of work, consumes a very large amount of electricity in order to operate; the largest working blockchain using this mechanism, Bitcoin, has been shown to consume as much electricity as the entire country of Ireland.

This document proposes a solution to these problems based on the combination of proof of stake and sharding. Proof of stake itself is not a novel idea, having existed since 2011, but this new algorithm presents substantial benefits, both solving flaws in previous systems and even introducing new properties that are not present in proof of work. Proof of stake can be thought of as a kind of “virtual mining”: whereas in proof of work, users can spend real-world dollars to buy real computers which expend electricity and stochastically produce blocks at a rate roughly proportional to the cost expended, in proof of stake, users spend real-world dollars to buy virtual coins inside the system, and then use an in-protocol mechanism to convert the virtual coins into virtual computers, which are simulated by the protocol to be “mining” blocks. These virtual computers can be sold to other users, and the user that bought them can then spend the virtual coins to do so.
MELON PROTOCOL: A BLOCKCHAIN PROTOCOL FOR DIGITAL ASSET MANAGEMENT
DRAFT

RETO TRINKLER AND MONA EL ISA

ABSTRACT. The Melon protocol is a blockchain protocol for digital asset management on the Ethereum platform. It enables participants to set up, manage and invest in digital asset management strategies in an open, competitive and decentralised manner.

1. INTRODUCTION

The value and importance of a wide range of digital assets\(^1\) has risen dramatically over the last few years. Hence the question naturally arises how to manage this new and fast-growing asset class in the most advantageous way.

This could be done by investing in a hedge fund which

Digital assets which do not gain their value from collateralisation, called *un-collateralised assets*. Finally, digital assets which are derived from existing digital assets called *derivatives*.

2.1. Collateralised Assets. Collateralised Assets, are assets which gain their value from the collateralisation of

\(^1\)
POLKADOT: VISION FOR A HETEROGENEOUS MULTI-CHAIN FRAMEWORK

DRAFT 1

DR. GAVIN WOOD
FOUNDER, ETHEREUM & PARITY
GAVIN@PARITY.IO

ABSTRACT. Present-day blockchain architectures all suffer from a number of issues not least practical means of extensibility and scalability. We believe this stems from tying two very important parts of the consensus architecture, namely canonicality and validity, too closely together. This paper introduces an architecture, the heterogeneous multi-chain, which fundamentally sets the two apart.

In compartmentalising these two parts, and by keeping the overall functionality provided to an absolute minimum of security and transport, we introduce practical means of core extensibility in situ. Scalability is addressed through a divide-and-conquer approach to these two functions, scaling out of its bonded core through the incentivisation of untrusted public nodes.

The heterogeneous nature of this architecture enables many highly divergent types of consensus systems interoperating in a trustless, fully decentralised “federation”, allowing open and closed networks to have trust-free access to each other.

We put forward a means of providing backwards compatibility with one or more pre-existing networks such as Ethereum [6, 22]. We believe that such a system provides a useful base-level component in the overall search for a practically implementable system capable of achieving global-commerce levels of scalability and privacy.
what are smart contracts?
Solidity

Solidity is a contract-oriented, high-level language for implementing smart contracts. It was influenced by C++, Python and JavaScript and is designed to target the Ethereum Virtual Machine (EVM).

Solidity is statically typed, supports inheritance, libraries and complex user-defined types among other features.

As you will see, it is possible to create contracts for voting, crowdfunding, blind auctions, multisignature wallets and more.

**Note**

The best way to try out Solidity right now is using Remix (it can take a while to load, please be patient).
pragma solidity 0.4.19;

contract BlackHat {
  string public conferanceName = "Black Hat";
  string public country = "Asia";
  uint256 public numberOfAttendees;
  uint256 public ticketPrice

  mapping(address => uint256) signedUp;

  function BlackHat(uint256 _ticketPrice) public {
    ticketPrice = _ticketPrice;
  }

  function signUpToBlackHat() public payable {
    require(msg.value > ticketPrice);
    signedUp[msg.sender] = true;
  }

  event AttendingTalk(address who, uint16 talkId);

  function attendBlackHat(uint16 _talkId) public {
    AttendingTalk(msg.sender, _talkId);
  }
}
EVM assembly:

```assembly
/* "SafeMath.sol":26:1060 library SafeMath {... */
mstore(0x40, 0x60)
jumpi(tag_1, iszero(callvalue))
0x0
dup1
revert
tag_1:
dataSize(sub_0)
dup1
dataOffset(sub_0)
0x0
codecopy
0x0
return
stop

sub_0: assembly {
    /* "SafeMath.sol":26:1060 library SafeMath {... */
mstore(0x40, 0x60)
0x0
dup1
revert

auxdata: 0xa165627a7723058206c9bd66e0efe163a41f5663b686835819321d24be410768f14bb77ef7c2f8e2d0029
}

======= TokenCity.sol:TokenCity =======

EVM assembly:

```assembly
/* "TokenCity.sol":52:19141 contract TokenCity {... */
mstore(0x40, 0x60)
/* "TokenCity.sol":128:159 string public name = "CityCoin" */
0x40
what are smart contracts used for?
IPO
IPO
ICO
Preamble

EIP: 20
Title: ERC-20 Token Standard
Author: Fabian Vogelsteller <fabian@ethereum.org>, Vitalik Buterin <vitalik.buterin@ethereum.org>
Type: Standard
Category: ERC
Status: Accepted
Created: 2015-11-19

Simple Summary

A standard interface for tokens.

Abstract

The following standard allows for the implementation of a standard API for tokens within smart contracts. This standard provides basic functionality to transfer tokens, as well as allow tokens to be approved so they can be spent by another on-chain third party.

Motivation
Box solution for conducting ICOs for 40 BTC

2 WEEKS FOR PREPARATION

+ 1 MONTH FOR PR CAMPAIGN

ICOBox is the first and the biggest new generation Blockchain Growth Promoter and Business Facilitator for companies seeking to sell their products via ICO crowdsales
Get The Best Ico Whitepaper Services

Find the best Ico whitepaper services you need to help you successfully meet your project planning goals and deadline

Join Fiverr
The ethereum network is getting jammed up because people are rushing to buy cartoon cats on its blockchain.
watch the statistics

The DAO has been created

968.42 M
DAO tokens created

9.68 M
Total ETH

97.81 M
USD equivalent

1.00
Last exchange rate ETH / 100 DAO tokens

22 hours
Next price phase

13 days
Since creation period ended
Created 28 May 09:00 GMT

Thank you all for your contribution
The DAO Raises More Than $117 Million in World's Largest Crowdfunding to Date

THE DAO IS AUTONOMOUS.

1071.36 M DAO TOKENS CREATED
10.73 M TOTAL ETH
116.81 M USD EQUIVALENT
1.10 CURRENT RATE ETH / 1 DAO TOKEN
15 hours NEXT PRICE PHASE
11 days LEFT ENDS 23 MAY 2016 CEST

Trending Now
- South Korea Allows Cryptocurrency Trading for Real-Name Registered Accounts
- Edge’s Paul Puey: “Digital Security Will Take Place on the Edges”
- Physical Bitcoins: Our Hands-On, End-to-End Review of Opendime
- Driving Blockchain Adoption in the Developing World With Spire
- Study Suggests 25 Percent of Bitcoin Users Are Associated With Illegal Activity
A $50 Million Hack Just Showed That the DAO Was All Too Human
Look how much I have.

Can I hold it?

cRYPTO-COMICS.COM
pragma solidity ^0.4.20;

contract Reentrance {
    mapping (address => uint256) public balances;
}
pragma solidity ^0.4.20;

contract Reentrance {
    mapping (address => uint256) public balances;

    function donate() public payable {
        balances[msg.sender] += msg.value;
    }
}
pragma solidity ^0.4.20;

contract Reentrance {

    mapping (address => uint256) public balances;

    function donate() public payable {
        balances[msg.sender] += msg.value;
    }

    function withdraw(uint256 amount) public {
        require(balances[msg.sender] >= amount);
        msg.sender.call.value(amount)();
        balances[msg.sender] -= amount;
    }

}
pragma solidity ^0.4.20;

contract Reentrance {

    mapping (address => uint256) public balances;

    function donate() public payable {
        balances[msg.sender] += msg.value;
    }

    function withdraw(uint256 amount) public {
        require(balances[msg.sender] >= amount);
        msg.sender.call.value(amount)();
        balances[msg.sender] -= amount;
    }
}
pragma solidity ^0.4.20;

contract Reentrance {

    mapping (address => uint256) public balances;

    function donate() public payable {
        balances[msg.sender] += msg.value;
    }

    function withdraw(uint256 amount) public {
        require(balances[msg.sender] >= amount);
        msg.sender.call.value(amount)();
        balances[msg.sender] -= amount;
    }
}

pragma solidity ^0.4.20;

contract Reentrance {

    mapping (address => uint256) public balances;

    function donate() public payable {
        balances[msg.sender] += msg.value;
    }

    function withdraw(uint256 amount) public {
        require(balances[msg.sender] >= amount);
        msg.sender.call.value(amount)();
        balances[msg.sender] -= amount;
    }
}
pragma solidity ^0.4.20;

contract Reentrance {
    mapping (address => uint256) public balances;

    function donate() public payable {
        balances[msg.sender] += msg.value;
    }

    function withdraw(uint256 amount) public {
        require(balances[msg.sender] >= amount);
        msg.sender.call.value(amount)();
        balances[msg.sender] -= amount;
    }
}
msg.sender

withdraw()
msg.sender is a smart contract

DAO

withdraw()
msg.sender is a smart contract

withdraw()
msg.sender is a smart contract

fallback()

withdraw()
msg.sender is a smart contract

DAO

fallback()
withdraw()
pragma solidity ^0.4.20;

class Reentrance {
    mapping (address => uint256) public balances;

    function donate() public payable {
        balances[msg.sender] += msg.value;
    }

    function withdraw(uint256 amount) public {
        require(balances[msg.sender] >= amount);
        msg.sender.call.value(amount)();
        balances[msg.sender] -= amount;
    }
}
pragma solidity ^0.4.20;

contract Reentrance {
    mapping (address => uint256) public balances;

    function donate() public payable {
        balances[msg.sender] += msg.value;
    }

    function withdraw(uint256 amount) public {
        require(balances[msg.sender] >= amount);
        msg.sender.call.value(amount)();
        balances[msg.sender] -= amount;
    }
}
pragma solidity ^0.4.20;

contract Reentrance {
    mapping (address => uint256) public balances;

    function donate() public payable {
        balances[msg.sender] += msg.value;
    }

    function withdraw(uint256 amount) public {
        require(balances[msg.sender] >= amount);
        msg.sender.call.value(amount)();
        balances[msg.sender] -= amount;
    }
}
pragma solidity ^0.4.20;

contract Reentrance {

    mapping (address => uint256) public balances;

    function donate() public payable {
        balances[msg.sender] += msg.value;
    }

    function withdraw(uint256 amount) public {
        require(balances[msg.sender] >= amount);
        msg.sender.call.value(amount)();
        balances[msg.sender] -= amount;
    }
}

msg.sender is a smart contract

withdraw()
msg.sender is a smart contract

fallback()

withdraw()
Reentrancy

Race-to-empty

• Sending ether to an address might lead to code execution
• Changing the state after calling another contract
The fastest and most secure way of interacting with the Ethereum blockchain. Our client powers much of the infrastructure of the public Ethereum network and is used by companies and users alike.
• function **initWallet**(address[] _owners, …)
• function confirm(bytes32 _h)
• function kill(address _to)
• …
Wallet

- function **initWallet**(address[] _owners, …)
- function confirm(bytes32 _h)
- function kill(address _to)
- …

owners: David, Mason and Thomas
required: 2
Wallet

- function() { ... }

WalletLibrary

- function initWallet(address[] _owners, ...)
- function confirm(bytes32 _h)
- function kill(address _to)
- ...
• function() { … }

Wallet

• function initWallet(address[] _owners, …)
• function confirm(bytes32 _h)
• function kill(address _to)
• …

WalletLibrary

• function() { … }

Wallet
Wallet

- function() { ... }

**call**: initWallet()

**owners**: David, Mason and Thomas

**required**: 2

WalletLibrary

- function `initWallet`(address[] _owners, ...)
- function `confirm`(bytes32 _h)
- function `kill`(address _to)
- ...
Wallet

- function() { … }

owners: David, Mason and Thomas
required: 2

call: initWallet()

decleteCall

WalletLibrary

- function initWallet(address[] _owners, …)
- function confirm(bytes32 _h)
- function kill(address _to)
- …
Wallet

- owners: David, Mason, Thomas
- function() { … }

WalletLibrary

- function `initWallet`([address] _owners, …)
- function confirm(bytes32 _h)
- function kill(address _to)
- …
function clearPending() internal {
  uint length = m_pendingIndex.length;
  for (uint i = 0; i < length; ++i) {
    delete m_txs[m_pendingIndex[i]];
    if (m_pendingIndex[i] != 0)
      delete m_pending[m_pendingIndex[i]];
  }
  delete m_pendingIndex;
}

// FIELDS
address constant _walletLibrary = 0xa657491c1e7f16ad39b9b60e87bbb8d93988bc3;

// the number of owners that must confirm the same operation before it is run.
uint public m_required;
// pointer used to find a free slot in m_owners
uint public m_numOwners;

uint public m_dailyLimit;
uint public m_spentToday;
uint public m_lastDay;

// list of owners
uint[256] m_owners;

uint constant c_maxOwners = 250;
// index on the list of owners to allow reverse lookup
mapping(uint => uint) m_ownerIndex;
// the ongoing operations.
mapping(uint32 => PendingState) m_pending;
Wallet

- owners: David, Mason, Thomas
- function() { … }

delegateCall

call: initWallet()
owners: Eve
required: 1

WalletLibrary

- function initWallet(address[] _owners, …)
- function confirm(bytes32 _h)
- function kill(address _to)
- …
Wallet

- owners: **Eve**
- function() { … }

WalletLibrary

- function `initWallet` (address[] _owners, …)
- function `confirm` (bytes32 _h)
- function `kill` (address _to)
- …
$30 Million: Ether Reported Stolen Due to Parity Wallet Breach

Smart contract coding company Parity has issued a security alert, warning of a vulnerability in version 1.5 or later of its wallet software.

So far, 150,000 ethers, worth $30 million, have been reported by the company as stolen, data confirmed by Etherscan.io. As reported by the startup, the issue is the result of a bug in a specific multi-signature contract known as wallet.sol. Data suggests the issue was mitigated, however, as 377,000 ethers that were potentially vulnerable to the issue were recovered by white hat hackers.
Part 2: Electric Boogaloo
Wallet

- function() { … }

WalletLibrary

- function initWallet(address[] _owners, …)
- function confirm(bytes32 _h)
- function kill(address _to)
- …

owners: devops199
required: 1
Wallet

- function() { … }

WalletLibrary

- owner: devops199
- function initWallet(address[] _owners, …)
- function confirm(bytes32 _h)
- function kill(address _to)
anyone can kill your contract #6995

ghost commented on 6 Nov 2017 • edited by ghost

I accidentally killed it.

https://etherscan.io/address/0x863df6bfa44baf3ead0be8f9f2aae51c91a907b4

jtakalai commented on 7 Nov 2017

Hmmm, clearly the kill came from registered owner, and required signatures was 0, see initWallet transaction arguments https://etherscan.io/tx/0x05f7f1e1b2ca3b3547739db15d080fd30c989eda04d37ce6264c5686e0722c9

ghost commented on 7 Nov 2017

Will it effect the dependent multisig wallets? When i query "isowner(<any_addr>)" the multisig wallets returns TRUE.
More ETH Troubles: Someone Triggered a Bug That Has Frozen Over 280 Million in Ethereum

By Rafia Shaikh
Nov 7, 2017
Other vulnerabilities
Randomness

Nothing is secret
pragma solidity ^0.4.18;

contract GameRandom {
    uint256 private seed;
    uint256 private iteration = 0;

    function play() public payable {
        require(msg.value >= 1 ether);

        iteration++;

        uint randomNumber = uint(keccak256(seed + iteration));

        bool won = (randomNumber % 2) == 0;

        if (won) {
            msg.sender.transfer(2 ether);
        }
    }
}
pragma solidity ^0.4.18;

contract GameRandom {
    uint256 private seed;
    uint256 private iteration = 0;

    function play() public payable {
        require(msg.value >= 1 ether);
        iteration++;
        uint randomNumber = uint(keccak256(seed + iteration));
        bool won = (randomNumber % 2) == 0;
        if (won) {
            msg.sender.transfer(2 ether);
        }
    }
}
pragma solidity ^0.4.18;

contract GameRandom {
    uint256 private seed;
    uint256 private iteration = 0;

    function play() public payable {
        require(msg.value >= 1 ether);
        iteration++;

        uint randomNumber = uint(keccak256(seed + iteration));
        bool won = (randomNumber % 2) == 0;

        if (won) {
            msg.sender.transfer(2 ether);
        }
    }
}
pragma solidity ^0.4.18;

contract GameRandom {

    uint256 private seed;
    uint256 private iteration = 0;

    function play() public payable {
        require(msg.value >= 1 ether);

        iteration++;

        uint randomNumber = uint(keccak256(seed + iteration));

        bool won = (randomNumber % 2) == 0;

        if (won) {
            msg.sender.transfer(2 ether);
        }
    }
}
NOTHING IS SECRET
pragma solidity ^0.4.18;

contract GameRandom {

    uint256 private seed;
    uint256 private iteration = 0;

    function play() public payable {
        require(msg.value >= 1 ether);

        iteration++;

        uint randomNumber = uint(keccak256(seed + iteration));
        bool won = (randomNumber % 2) == 0;

        if (won) {
            msg.sender.transfer(2 ether);
        }
    }
}
Block and Transaction Properties

- `block.blockhash(uint blockNumber) returns (bytes32)`: hash of the given block - only works for 256 most recent blocks excluding current
- `block.coinbase(address)`: current block miner's address
- `block.difficulty(uint)`: current block difficulty
- `block.gaslimit(uint)`: current block gaslimit
- `block.number(uint)`: current block number
- `block.timestamp(uint)`: current block timestamp as seconds since unix epoch
- `gasleft() returns (uint256)`: remaining gas
- `msg.data(bytes)`: complete calldata
- `msg.gas(uint)`: remaining gas - deprecated in version 0.4.21 and to be replaced by `gasleft()`
- `msg.sender(address)`: sender of the message (current call)
- `msg.sig(bytes4)`: first four bytes of the calldata (i.e. function identifier)
- `msg.value(uint)`: number of wei sent with the message
- `now(uint)`: current block timestamp (alias for `block.timestamp`
- `tx.gasprice(uint)`: gas price of the transaction
- `tx.origin(address)`: sender of the transaction (full call chain)
12.2. **Random Numbers.** Providing random numbers within a deterministic system is, naturally, an impossible task. However, we can approximate with pseudo-random numbers by utilising data which is generally unknowable at the time of transacting. Such data might include the block’s hash, the block’s timestamp and the block’s beneficiary address. In order to make it hard for malicious miner to control those values, one should use the BLOUCKHASH operation in order to use hashes of the previous 256 blocks as pseudo-random numbers. For a series of such numbers,
SmartBillions lottery contract just got hacked!  self.ethereum
Submitted 5 months ago * by supr3m

Someone made it in the “hackathon” (lol). The hacker could withdraw 400 ETH before the owners, who wrote "the successful hacker keeps ALL of the 1500 ETH reward", withdrew quickly the remaining 1100 ETH, that happened 5min before the next transaction (from the "hacker") would have emptied the whole contract. So that's already a lie from their side. The other point is that the owners were able to withdraw ALL contract funds; which in theory they could have done after ICO and run with all the investor money. They always remained anon, which also shows there weren't good intentions in first place.

How did it happen? Their lottery functions were flawed, if you place a bet (systemPlay() function) with betting on number value "0" and then call the won() function after 256+ blocks (after you placed the bet) the returning value will be "0" so you would have bet on "000000" and result would be "000000" and baam you have the jackpot. The lucky guys first bet was "1" so "000001" and result after 256+ blocks calling won() would be "000000" so he matched 5 correctly which is 20000x and with 0.01ETH bet amount a win of 200ETH. He managed to pull that 2 time and corrected to "0" and for that transaction he had to wait for 256+ blocks, but 5 min before he could call won() the owners withdraw all funds.

Moral of the story, that ICO was a scam seeing the owners remains anon all the time AND were able to withdraw all contract funds (doing that after ICO would have been fatal for investors).

They thought they are clever, building a honeypot for investors but at the end their poor coded contract caused them damage of 400ETH and no damage to potential investors.
pragma solidity ^0.4.18;

contract GameRandom {
    function play() public payable {
        require(msg.value >= 1 ether);
        bool won = (block.blockhash(block.number) % 2) == 0;
        if (won) {
            msg.sender.transfer(2 ether);
        }
    }
}
is blockhash of block.number even?

attack()

play()

Game
is blockhash of block.number even?

Game

attack() → play()
is blockhash of block.number even?

Game

attack()

play()
pragma solidity ^0.4.18;

contract GameRandom {
    function play() public payable {
        require(msg.value >= 1 ether);

        bool won = (block.blockhash(block.number) % 2) == 0;

        if (won) {
            msg.sender.transfer(2 ether);
        }
    }
}
Randomness

Nothing is secret

• Be wary of "randomness"
• Strong reliance on time is bad
Front-running
also known as race condition, TOCTOU, TOD
pragma solidity ^0.4.18;

contract GameRandom {
    uint rsaNumber = 2194012421903219301151; // N = p * q
    
    function play(uint p, uint q) public payable {
        if (p * q == rsaNumber) {
            msg.sender.transfer(1000 ether);
        }
    }
}
Alice

Bob

Game

solution

play()
Alice

Bob

Game

play()
Alice

Bob

solution

Game

play()
transferFrom

Transfers \_value amount of tokens from address \_from to address \_to, and MUST fire the Transfer event.

The transferFrom method is used for a withdraw workflow, allowing contracts to transfer tokens on your behalf. This can be used for example to allow a contract to transfer tokens on your behalf and/or to charge fees in sub-currencies. The function SHOULD throw unless the \_from account has deliberately authorized the sender of the message via some mechanism.

Note Transfers of 0 values MUST be treated as normal transfers and fire the Transfer event.

```solidity
function transferFrom(address \_from, address \_to, uint256 \_value) returns (bool success)
```

approve

Allows \_spender to withdraw from your account multiple times, up to the \_value amount. If this function is called again it overwrites the current allowance with \_value.

NOTE: To prevent attack vectors like the one described here and discussed here, clients SHOULD make sure to create user interfaces in such a way that they set the allowance first to 0 before setting it to another value for the same spender. THOUGH The contract itself shouldn't enforce it, to allow backwards compatibility with contracts deployed before

```solidity
function approve(address \_spender, uint256 \_value) returns (bool success)
```

allowance

Returns the amount which \_spender is still allowed to withdraw from \_owner.

```solidity
function allowance(address \_owner, address \_spender) constant returns (uint256 remaining)
```
Front-running
also known as race condition, TOCTOU, TOD

• Can transactions re-ordering strongly affect the outcome?
• Do some transactions contain secret information?
Arithmetic Errors

Integer overflows and underflows
\textbf{uint8}

\begin{tabular}{ccccccccccc}
\textbf{1} & \textbf{1} & \textbf{1} & \textbf{1} & \textbf{1} & \textbf{1} & \textbf{1} & \textbf{1} & \textbf{1} & \textbf{1} & \textbf{1} \\
\end{tabular}

\[+1\]

\begin{tabular}{ccccccccccc}
\textbf{0} & \textbf{0} & \textbf{0} & \textbf{0} & \textbf{0} & \textbf{0} & \textbf{0} & \textbf{0} & \textbf{0} & \textbf{0} & \textbf{0} \\
\end{tabular}
pragma solidity ^0.4.15;

contract Overflow {
    uint private sellerBalance=0;

    function add(uint value) returns (bool){
        sellerBalance += value; // possible overflow
        // possible auditor assert
        // assert(sellerBalance >= value);
    }

    function safe_add(uint value) returns (bool){
        require(value + sellerBalance >= sellerBalance);
        sellerBalance += value;
    }
}

https://github.com/trailofbits/not-so-smart-contracts/blob/master/integer_overflow/integer_overflow_1.sol
library SafeMath {

    /**
     * @dev Multiplies two numbers, throws on overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }
        uint256 c = a * b;
        assert(c / a == b);
        return c;
    }

    /**
     * @dev Integer division of two numbers, truncating the quotient.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        // assert(b > 0); // Solidity automatically throws when dividing by 0
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold
        return c;
    }

    /**
     * @dev Subtracts two numbers, throws on overflow (i.e. if subtrahend is greater than minuend).
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        assert(b <= a);
        return a - b;
    }

    /**
     * @dev Adds two numbers, throws on overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        assert(c >= a);
        return c;
    }
}
The main reason I opened separate issues is that each requires discussion if we want it or not, but if we are set to go over everything, here is a list:

- exception on overflow in unsigned-\rightarrow signed conversion
- exception on overflow in signed-\rightarrow unsigned conversion
- exception on overflow in size-decreasing implicit conversion
- exception on overflow in addition of two signed numbers
- exception on overflow in addition of two unsigned numbers
- exception on underflow in subtraction of two signed numbers
- exception on underflow in subtraction of two unsigned numbers
- exception on overflow in multiplication of two signed numbers
- exception on overflow in multiplication of two unsigned numbers
- exception on overflow in shifts
- exception on overflow in ++ on a signed number
- exception on overflow in ++ on an unsigned number
- exception on underflow in -- on a signed number
- exception on underflow in -- on an unsigned number
- exception on overflow in +=
- exception on overflow in -=
- exception on overflow in *=
- exception on overflow in /=
- make sure no optimizations are relying on \( (a + b - b == a) \); lest they remove overflow exceptions
- compiler error on an out-of-range constant expression when the constant expression is interpreted into a type \( \text{uint} \ x = -1 \)
- exception on overflow in INT_MIN/-1 #1091
What to Look For

• Unchecked arithmetic operation
• Can be increased/decreased by large enough leaps (incrementing upward generally too expensive)
• Accessible from lower levels of privilege
Denial of Service
### Blocks

Showing Block (#5216258 to #5216234) out of 5216259 total blocks

<table>
<thead>
<tr>
<th>Height</th>
<th>Age</th>
<th>txn</th>
<th>Uncles</th>
<th>Miner</th>
<th>GasUsed</th>
<th>(%)</th>
<th>GasLimit</th>
<th>Avg.GasPrice</th>
<th>Reward</th>
</tr>
</thead>
<tbody>
<tr>
<td>5216258</td>
<td>40 secs</td>
<td>146</td>
<td>0</td>
<td>DwarfPool1</td>
<td>7985670</td>
<td>99.87%</td>
<td>7996085</td>
<td>25.95 Gwei</td>
<td>3.207222 Ether</td>
</tr>
<tr>
<td>5216257</td>
<td>1 min</td>
<td>124</td>
<td>0</td>
<td>SparkPool</td>
<td>7989541</td>
<td>99.97%</td>
<td>7992185</td>
<td>12.52 Gwei</td>
<td>3.1 Ether</td>
</tr>
<tr>
<td>5216256</td>
<td>1 min</td>
<td>6</td>
<td>0</td>
<td>waterhole</td>
<td>1260000</td>
<td>1.58%</td>
<td>7999959</td>
<td>62.17 Gwei</td>
<td>3.00783 Ether</td>
</tr>
</tbody>
</table>
play() {
    add_user()
}

payToAllUsers() {
    for user in users
    }

many contributions

pay out pls
```javascript
play() {
  add_user()
}

too many contributions

payToAllUsers() {
  for user in users
}

pay out pls
```
But Wait—There’s More!
function() {
    revert();
}

payBackBids()
Get size of an account's code.
contract BidThing {

    function bid(uint256 object) public payable {
        // place ethers
    }

    function cancelBid(uint256 object) public {
        // refund ethers
    }

    function acceptBid(uint256 object) public {
        // transfer ethers
    }
}

contract BidThing {

    function bid(uint256 object) public payable {
        // place ethers
    }

    function cancelBid(uint256 object) public {
        // refund ethers
    }

    function acceptBid(uint256 object) public {
        // transfer ethers
    }

    function withdraw(uint256 _amount) public {
        address owner = 0x32523459909435...;
        owner.transfer(_amount);
    }
}
What to Look For

• For loops dependent on storage values
• Functions dependent on transactions
• Public functions affecting contract balance
Methodology Takeaways

• Prerequisite: knowledge of common Ethereum gotchas
• **Understand** the contract
• Look for
  • Calls to external contracts and low level function calls
  • Denial of Services
  • Reliance on public or secret values
  • Logic relying on user input
  • Re-inventing the wheel
  • Excessive use of inline assembly
• Analyze any **publicly-accessible** functions in depth
Tools

• Oyente, Mythril, Manticore, …
  • Run them in the beginning, but don’t expect much unless there is an evident issue

• **Remix** and Linters
  • really useful to catch quick mistakes and bad behaviors that might lead to issues. Awesome to **test** and **debug** quickly.

• **Truffle**:
  • is OK

• Disassembler and decompilers
Future

- Formal verification
- Frameworks and re-usable code
- Ethereum rapidly evolves
- Solidity, Mist, Ethereum Wallet still have experimental versions.
- Audit process is immature
- Lots of money put in lightly-audited smart contracts
- Unknown unknowns?
Howdy!

This is the very first iteration of the Decentralized Application Security Project (or DASP) Top 10 of 2018

A group of security experts and consultants from the industry have gathered up and produced a top 10 of the different security vulnerabilities found in Ethereum smart contracts. This page is the first of its kind and will likely evolve along with Ethereum.

1. Reentrancy

also known as or related to race to empty, recursive call vulnerability, call to the unknown

this exploit was missed in review so many times by so many different people: reviewers tend to review functions one at a time, and assume that calls to secure subroutines will operate securely and as intended.

— Phil Daien

The Reentrancy attack, probably the most famous Ethereum vulnerability, surprised everyone when
the decentralized application security project:
www.dasp.co

follow us on
twitter.com/cryptodavidw
twitter.com/mah3mm

(we work here)