CS 177 - Computer Security

Web3 Security
Web3 Stack

**Application Layer**
- Higher-Level Apps
- Trading and Auctions
- Escrow and Notaries
- E-Voting
- Ecosystem
- Crypto-Tokens
- Exchanges and Wallets
- Oracles

**Compute Layer**
- Transactions
- Smart Contracts

**Consensus Layer**
- Byzantine Fault Tolerance
- Proof Of Work
- Proof of Stake

**Network Layer**
- IP / DNS
- Peer Discovery
- Authentication
BLOCKCHAINS
BLOCKCHAIN

I bought a dip, but it keeps dipping

I KNOW BLOCKCHAIN

SAY BLOCKCHAIN

DO YOU USE ETHEREUM?
YES
WHAT DID IT COST?
EVERYTHING

I WAS TOLD
THERE WOULD BE LAMBOS

ONE MORE TIME
Blockchains, Blocks, and Transactions

- Public append-only distributed data structure (ledger)
- Consensus protocol allows for agreement about state among mutually untrusted parties
- State is captured in a chain of blocks
- Typically, the contents of the block are a list of transactions

https://en.wikipedia.org/wiki/Blockchain
Nakamoto Consensus Algorithm

Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending.
Proof-of-Work

- Nodes (miners) compete on a puzzle to pick the node (leader) that decides which transactions to include and in which order (per round)
- Puzzle is to find a number (nonce) such that when the block content is hashed (SHA-256) along with the nonce, the result is numerically smaller than the network's difficulty target (that is, the result has a certain number of leading zeros)
- As a reward, miners collect fees + mint new coins
Longest Chain Rule

- A single view of history is necessary
- How does the blockchain resolve forks?
- Pick the longest chain!
- If majority of nodes is honest, system is secure and makes progress
Proof-Of-Stake (POS)

- Alternative to Proof-of-Work (POW) consensus
  - a particular concern with POW is environmental impact

- Validators (instead of miners) stake (lock up) some tokens
- Leader(s) selected at random among validators, with higher odds assigned to nodes that have larger stakes
-Validators receive transactions fees as rewards
- Typically, multiple validators/leaders work together (as a committee) to confirm a block
- If dishonest behavior is detected, the stake of offending node might be slashed
L1 Blockchains (and Their Coins)

- Bitcoin and Ethereum account for the vast majority of the market cap

- Ethereum moved from Proof-of-Work to Proof-of-Stake in September 2022 (in an event called “The Merge”)

<table>
<thead>
<tr>
<th>Layer-1 coins</th>
<th>Price</th>
<th>Market cap</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1</strong> Bitcoin BTC</td>
<td>$19,493.06</td>
<td>$373.74 billion</td>
</tr>
<tr>
<td><strong>2</strong> Ethereum ETH</td>
<td>$1,325.63</td>
<td>$162.20 billion</td>
</tr>
<tr>
<td><strong>3</strong> Binance Coin BNB</td>
<td>$281.25</td>
<td>$40.77 billion</td>
</tr>
<tr>
<td><strong>4</strong> Cardano ADA</td>
<td>$0.4233</td>
<td>$13.19 billion</td>
</tr>
<tr>
<td><strong>5</strong> Solana SOL</td>
<td>$32.78</td>
<td>$11.72 billion</td>
</tr>
<tr>
<td><strong>6</strong> Polkadot DOT</td>
<td>$6.333</td>
<td>$7.30 billion</td>
</tr>
<tr>
<td><strong>7</strong> TRON TRX</td>
<td>$0.06237</td>
<td>$5.76 billion</td>
</tr>
<tr>
<td><strong>8</strong> Avalanche AVAX</td>
<td>$16.82</td>
<td>$4.99 billion</td>
</tr>
<tr>
<td><strong>9</strong> Cosmos ATOM</td>
<td>$13.15</td>
<td>$4.10 billion</td>
</tr>
<tr>
<td><strong>10</strong> Litecoin LTC</td>
<td>$53.18</td>
<td>$3.77 billion</td>
</tr>
</tbody>
</table>
Attacks Against Consensus Protocol

- Double-spend on forks
- Feather Forks
- Selfish Mining, DDoS on Pools
- Miner’s Dilemma
- FlashBoys 2.0 (MEV)
- Power Adjusting + Bribery
- Fork After Withholding
- Stubborn Mining

Years:
- 2009
- 2012
- 2013
- 2014
- 2015
- 2016
- 2017
- 2018
- 2019
- 2020
- 2022
SMART CONTRACTS
Web3 Stack

Application Layer
- Higher-Level Apps
- Trading and Auctions
- Escrow and Notaries
- E-Voting
- Ecosystem
- Crypto-Tokens
- Exchanges and Wallets
- Oracles

Compute Layer
- Transactions
- Smart Contracts

Consensus Layer
- Byzantine Fault Tolerance
- Proof Of Work
- Proof of Stake

Network Layer
- IP / DNS
- Peer Discovery
- Authentication

Blockchains
Applications
Digital Currency as “Killer App”?  

Smart Contracts

• Digital currencies are just one blockchain application
• What about programs running on top of a blockchain?
  → Smart Contracts!
• In this lecture, we focus on Ethereum

• Addresses (users) in Ethereum have an account balance, and optionally (immutable) code and storage
Ethereum Smart Contracts

- An Ethereum smart contract is represented by code and storage associated with an Ethereum address.
- It executes as bytecode on top of *Ethereum VM (EVM)*.

- Programs are typically developed in higher-level languages such as Solidity, then compiled into EVM bytecode, then deployed (via a transaction).

- Power a wide range of applications, including decentralized finance (DeFi) apps, tokens, games, collectibles, ...
Ethereum Virtual Machine (EVM)

• The EVM is
  – single-threaded
  – stack-based
  – a distributed virtual machine

• Each instruction on the EVM incurs inherent gas cost

• Gas is converted (via gas price) to a transaction fee

• User who invokes a function must include fee to pay for execution

• Gas costs and maximum transaction fees avoid halting problem (and non-terminating code)
Sample EVM Opcodes

<table>
<thead>
<tr>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>STOP</td>
</tr>
<tr>
<td>ADD</td>
</tr>
<tr>
<td>MUL</td>
</tr>
<tr>
<td>LT</td>
</tr>
<tr>
<td>EQ</td>
</tr>
<tr>
<td>ISZERO</td>
</tr>
<tr>
<td>XOR</td>
</tr>
<tr>
<td>SHA3</td>
</tr>
<tr>
<td>CALLER</td>
</tr>
<tr>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>POP</td>
</tr>
<tr>
<td>MLOAD</td>
</tr>
<tr>
<td>MSTORE</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>SLOAD</td>
</tr>
<tr>
<td>SSTORE</td>
</tr>
<tr>
<td>LOG0</td>
</tr>
<tr>
<td>CALL</td>
</tr>
<tr>
<td>DELEGATECALL</td>
</tr>
<tr>
<td>STATICCALL</td>
</tr>
<tr>
<td>REVERT</td>
</tr>
<tr>
<td>SELFDESTRUCT</td>
</tr>
</tbody>
</table>
EVM Architecture

https://ethereum.org/en/developers/docs/evm/
Ethereum Transactions

• Each transaction in a block either
  – deploys a smart contract, or
  – invokes function of a smart contract, or
  – sends Ether (Ethereum currency) directly between addresses

• Transactions are broadcast by users into a public mempool

• Miners pick transactions from mempool to include in blocks
The EVM itself is part of the attack surface

*EVMFuzzer: Detect EVM Vulnerabilities via Fuzz Testing*
Ying Fu et al; ESEC/FSE ‘19

*Finding Consensus Bugs in Ethereum via Multi-transaction Differential Fuzzing*
Youngseok Yang et al; OSDI ‘21
Smart Contract Invocation Executes Atomically

UC Santa Barbara

```
a = 1
if (input == 1) {
    revert();
}
a = 2;
```

to: 0xfab3...
input: 00000004

```
0xfab3...
```

Input == 1

false

COMMIT

```
\_a = 2
```

true

REVERT

```
\_a unchanged
```
Smart Contracts Are Composable

UC Santa Barbara

to: 0xfab3...
input: 68656c6f
The code is the law

• What if the code is broken
• And the source code is not public
• And we cannot upgrade the code
• And the code handles (often large amounts of) money

What can possibly go wrong?
Smart Contract Security

- Popular Security Problems
  - Reentrancy
  - Write to Arbitrary Storage Location
  - Uninitialized Storage Location
  - Unprotected Ether Withdrawal
  - Integer Overflow or Underflow
  - Timestamp Dependence
  - Weak Source of Randomness
# Smart Contract Security

## SWC Registry

Smart Contract Weakness Classification and Test Cases

The following table contains an overview of the SWC registry. Each row consists of an SWC identifier (ID), weakness title, CWE parent and list of related code samples. The links in the ID and Test Cases columns link to the respective SWC definition. Links in the Relationships column link to the CWE Base or Class type.

<table>
<thead>
<tr>
<th>ID</th>
<th>Title</th>
<th>CWE</th>
<th>Relationships</th>
<th>Test cases</th>
</tr>
</thead>
</table>
| SWC-136 | Unencrypted Private Data On-Chain                | CWE-767: Access to Critical Private Variable via Public Method | • odd_even.sol  
• odd_even_fixed.sol  |
| SWC-135 | Code With No Effects                            | CWE-1164: Irrelevant Code | • deposit_box.sol  
• deposit_box_fixed.sol  
• wallet.sol  
• wallet_fixed.sol  |
| SWC-134 | Message call with hardcoded gas amount          | CWE-655: Improper Initialization | • hardcoded_gas_limits.sol  |
| SWC-133 | Hash Collisions With Multiple Variable Length Arguments | CWE-294: Authentication Bypass by Capture-replay | • access_control.sol  
• access_control_fixed_1.sol  
• access_control_fixed_2.sol  |
| SWC-132 | Unexpected Ether balance                        | CWE-667: Improper Locking | • Lockdrop.sol  |
| SWC-131 | Presence of unused variables                    | CWE-1164: Irrelevant Code | • unused_state_variables.sol  
• unused_state_variables_fixed.sol  
• unused_variables.sol  
• unused_variables_fixed.sol  |
| SWC-130 | Right-Left Override control                      | CWE-451: User Interface II: Misrepresentation of Critical | • users_the_number.sol  |
• Bank contract allows withdrawal of Ethers by users
• The `withdraw()` function transfers the amount through an external call
• After the call returns, user’s balance is updated.
• A user implements `getEthers()` to withdraw `100 Ethers` from the Bank
• She also invokes `bank.withdraw(100)` within her `fallback()` function as well
Reentrancy

**Attacker**

```solidity
getEthers() {
    bank.withdraw(100)
}

payable() {
    bank.withdraw(100)
}
```

**Bank**

```solidity
withdraw() {
    if (accounts[msg.sender] >= amount)
        msg.sender.call.value(amount)
    accounts[msg.sender] -= amount
}
```
Reentrancy

```javascript
bank.getEthers() {
    bank.withdraw(100)
}

payable() {
    bank.withdraw(100)
}

if (accounts[msg.sender] >= amount)
    msg.sender.call.value(amount)

accounts[msg.sender] -= amount
```
Reentrancy

```
getEthers() {
  bank.withdraw(100)
}

payable() {
  bank.withdraw(100)
}
```

```
withdraw() {
  if (accounts[msg.sender] >= amount)
    msg.sender.call.value(amount)
  accounts[msg.sender] -= amount
}
```
Reentrancy

Attacker

getEthers() {
    bank.withdraw(100)
}

payable() {
    bank.withdraw(100)
}

Bank

withdraw() {
    if (accounts[msg.sender] >= amount)
        msg.sender.call.value(amount)
    accounts[msg.sender] -= amount
}

Deferred update

1. Reentrant call
2. 3. 4. 5. 6. 7. 8.
The DAO Hack

Reentrancy Attacks and The DAO Hack

August 31, 2022  •  Zubin Pratap

Reentrancy attacks, like the one used in The DAO hack, are made possible by vulnerabilities in the way we structure Solidity code.

In this blog, we will explore one of the most infamous Solidity hacks of all time, which happened in the early days of Ethereum smart contract development. This attack used the reentrancy exploit to compromise a DAO (decentralized autonomous organization) called The DAO. The hack used an exploit commonly referred to as a reentrancy attack.

https://blog.chain.link/reentrancy-attacks-and-the-dao-hack/
address owner = ...;

function initWallet(address _owner) {
    owner = _owner;
}

function withdraw(uint amount) {
    if (msg.sender == owner) {
        owner.send(amount);
    }
}
Finding Security Vulnerabilities

Testing
Easy to implement, but very limited guarantees

Dynamic analysis
Symbolic execution
Better than testing, but can still miss vulnerabilities

Static analysis
Formal verification
Strong guarantees, but many false positives

Arthur Gervais; DeFi Security II; https://defi-learning.org/f21
Audits and Bug Bounties

- Ecosystem of companies that perform audits
  - Trail of Bits, ConsenSys Diligence, Certik, Hacken, OpenZeppelin, ...
- Provide no guarantees
- Bug bounties are a way to incentivize responsible disclosure
  - Immunefi, HackenProof

---

Last week, I discovered (and reported) a critical bug (which has been fully patched) in @optimismPBC (a "layer 2 scaling solution" for Ethereum) that would have allowed an attacker to print arbitrary quantity of tokens, for which I won a $2,000,042 bounty.

saurik.com
Attacking an Ethereum L2 with Unbridled Optimism

9:07 AM · Feb 10, 2022 · Twitter for iPhone
Automated Vulnerability Analysis

- Combination of static and dynamic analysis
- Small code sizes make it possible to adopt resource-heavy approaches
- Execution is limited by gas price
- Memory models are simpler
- All the (bytecode) code is available
2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)

Summary-Based Symbolic Evaluation for Smart Contracts

Yu Feng
yufeng@cs.ucsb.edu
University of California, Santa Barbara

Emina Torlak
emina@cs.washington.edu
University of Washington

Rastislav Bodik
bodik@cs.washington.edu
University of Washington

ABSTRACT

Automated analysis of programs whose states are altered by transactions, diagnosing the correctness or absence of a vulnerability, and understanding nontrivial vulnerabilities requires general-purpose tools. As demonstrated by recent attacks [2, 4, 6, 27], for instance, the code (simplified in Figure 1) illustrates the notorious REDEPLOYMENT flaw for Ethereum smart contracts. The details of such attacks are often

In this paper, we present SAFE, a symbolic evaluation tool for detecting REDEPLOYMENT and similar vulnerabilities in smart contracts. SAFE is a general-purpose symbolic evaluation engine that can be applied to a variety of languages and scenarios. Our evaluation of SAFE on a diverse set of benchmarks shows that it is effective at detecting a wide range of vulnerabilities, including those that have not been previously reported. We also demonstrate how SAFE can be used to develop secure smart contract specifications and to conduct formal verification of contract implementations.
Transaction Order Matters

• Front-running attacks
  – Identify a profit-making transaction
  – Attempt to put a similar transaction (with a different beneficiary) before the target transaction

• Sandwich attacks
  – Identify a target transaction that might change the valuation of an asset
  – Attempt to put a transaction before and one afterwards to take advantage of the change
DECENTRALIZED FINANCE (DEFI)
## DeFi Components

<table>
<thead>
<tr>
<th>ERC-20 Tokens</th>
<th>Oracles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERC-721 Tokens</td>
<td>Stablecoins</td>
</tr>
<tr>
<td>Automated Market Makers (AMMs)</td>
<td>Yield Farms</td>
</tr>
<tr>
<td>Loans &amp; Leverage</td>
<td>Bridges + Side-Chains</td>
</tr>
</tbody>
</table>
Smart Token Contracts

Token (ERC-20)

- balanceOf(address owner) -> uint256
- transferFrom(
  address a,
  address b,
  uint256 value
) -> bool

...  

NFT (ERC-721)

- ownerOf(uint256 id) -> address
- transferFrom(
  address a,
  address b,
  uint256 id
) -> bool

...
Distributed Exchanges

- Automated Market Makers (AMMs) are a popular type of DEX
- AMMs are based on liquidity pools
  - do not maintain an order book
- Exchange price is automatically determined
  - for example, the product of the two assets constant

UNISWAP  

SushiSwap
Centralized Exchanges

- **Ramp-in**
  - Transfer money from a bank account into Ether (or other coins)

- **Ramp-out**
  - Transfer Ether / coins into actual money in a bank account

https://coinmarketcap.com/rankings/exchanges/
DeFi Hacks

- Almost $5B stolen in the last two years

- Protocol hacks
  - Many DeFi protocols rely on oracles, which inject timely real-world (price) information into a blockchain. Attackers managed to distort oracle information, leading to contracts operating on incorrect data.

- Bridge hacks
  - A bridge connects two different L1 chains and allows to move coins from one to the other. A flaws might allow an attacker to mint tokens on one side without reducing (burning) the equivalent amount on the other side.

https://rekt.news/