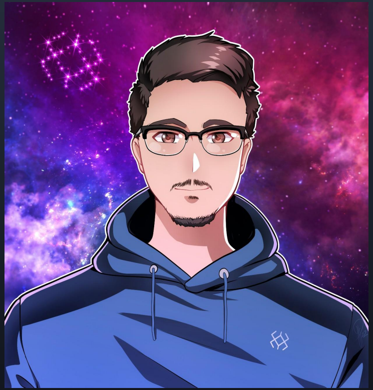
Tour of Common Web3 Vulnerabilities

Who are we

- Smart contract triagers at @immunefi
- Interested in investigating EVM based defi hacks.
- Previously worked as an appsec engineer in web2 space.
- Create educational content.


ArbazKiraak

0xrudrapratap

@immunefi-team/community-challenges

https://immunefi.medium.com

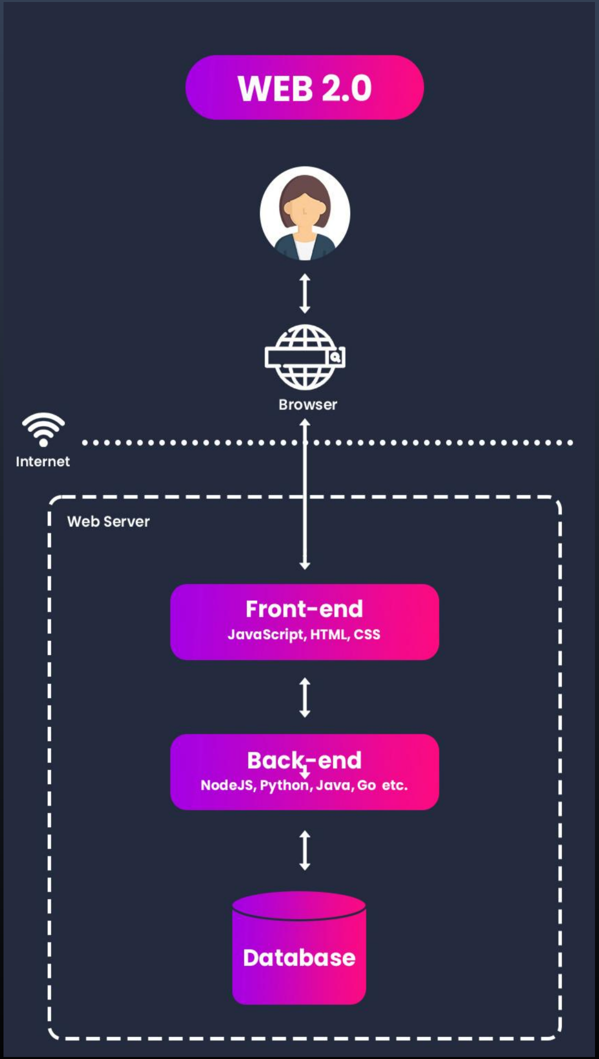
Agenda

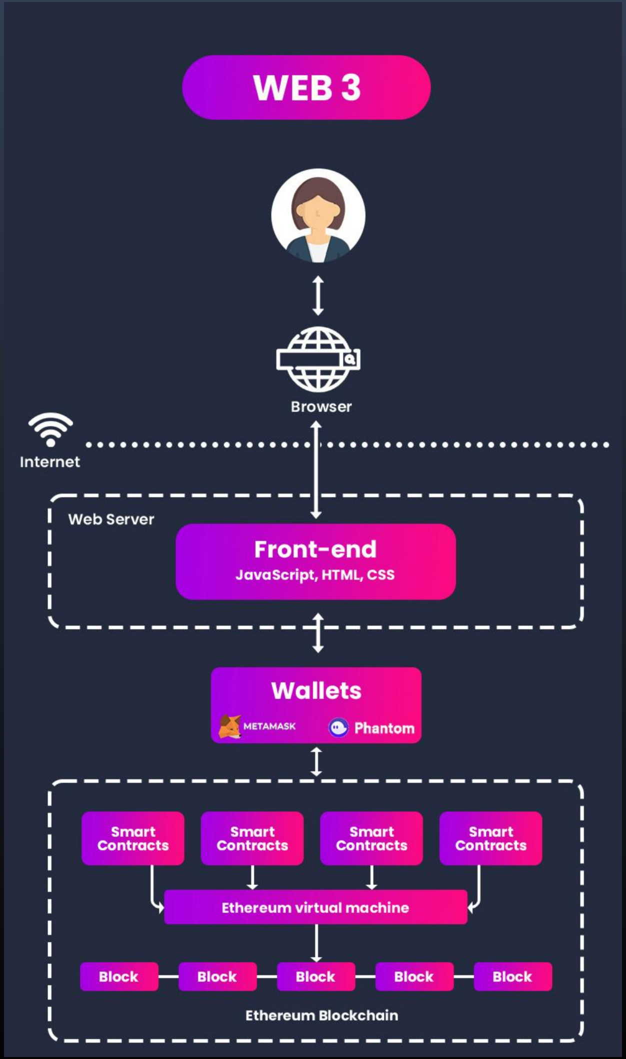
- What are smart contracts
- Web2 vs Web3 application architecture flow
- Most common decentralized application(DAPP) vulnerabilities
- Most common smart contract vulnerabilities
- Outro - Get started with smart contract hacking resources

What are smart contracts?

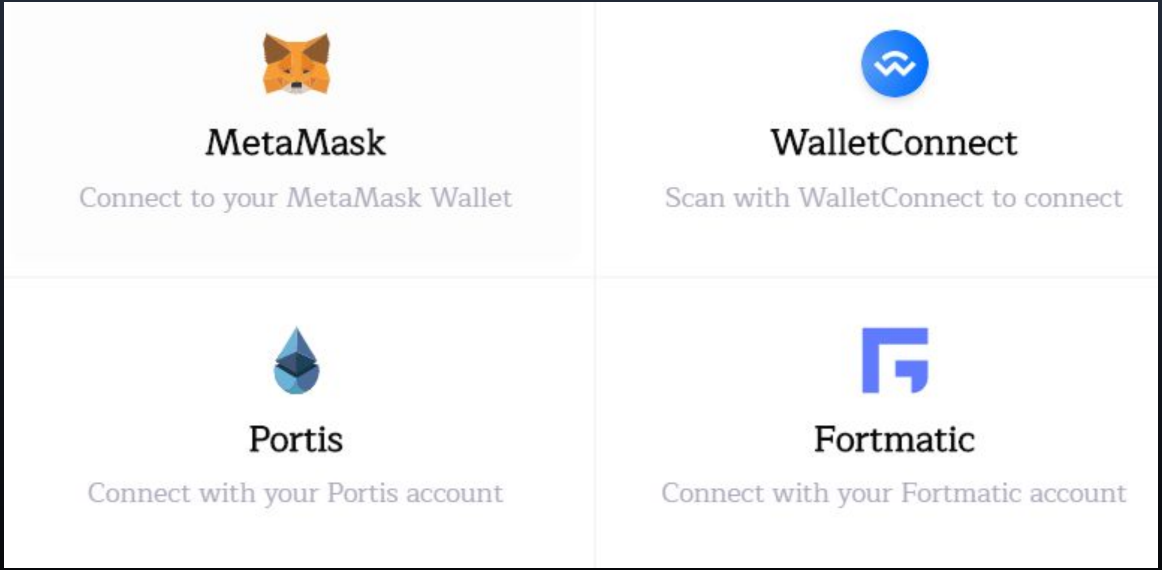
- Immutable program code containing set of instructions to be executed.
- Run on decentralized blockchain network such as Ethereum, Solana, Polkadot etc
- Extensively written in high level languages like Solidity, Vyper, Rust, Stacks etc
- Contains set of OPCODES which interacts with the EVM (Ethereum virtual Machine)

Web2 vs Web3 Architecture


WEB 2.0
Browser
Internet
Web Server
Front-end
JavaScript, HTML, CSS
Back-end
NodeJS, Python, Java, Go etc.
Database


WEB 3
Browser
Internet
Web Server
Front-end
JavaScript, HTML, CSS
Wallets
METAMASK
Phantom
Smart Contracts
Smart Contracts
Smart Contracts
Smart Contracts
Ethereum virtual machine
Block
Block
Block
Block
Block
Ethereum Blockchain

What are these wallets?



MetaMask

Connect to your MetaMask Wallet

WalletConnect

Scan with WalletConnect to connect

Portis

Connect with your Portis account

Fortmatic

Connect with your Fortmatic account

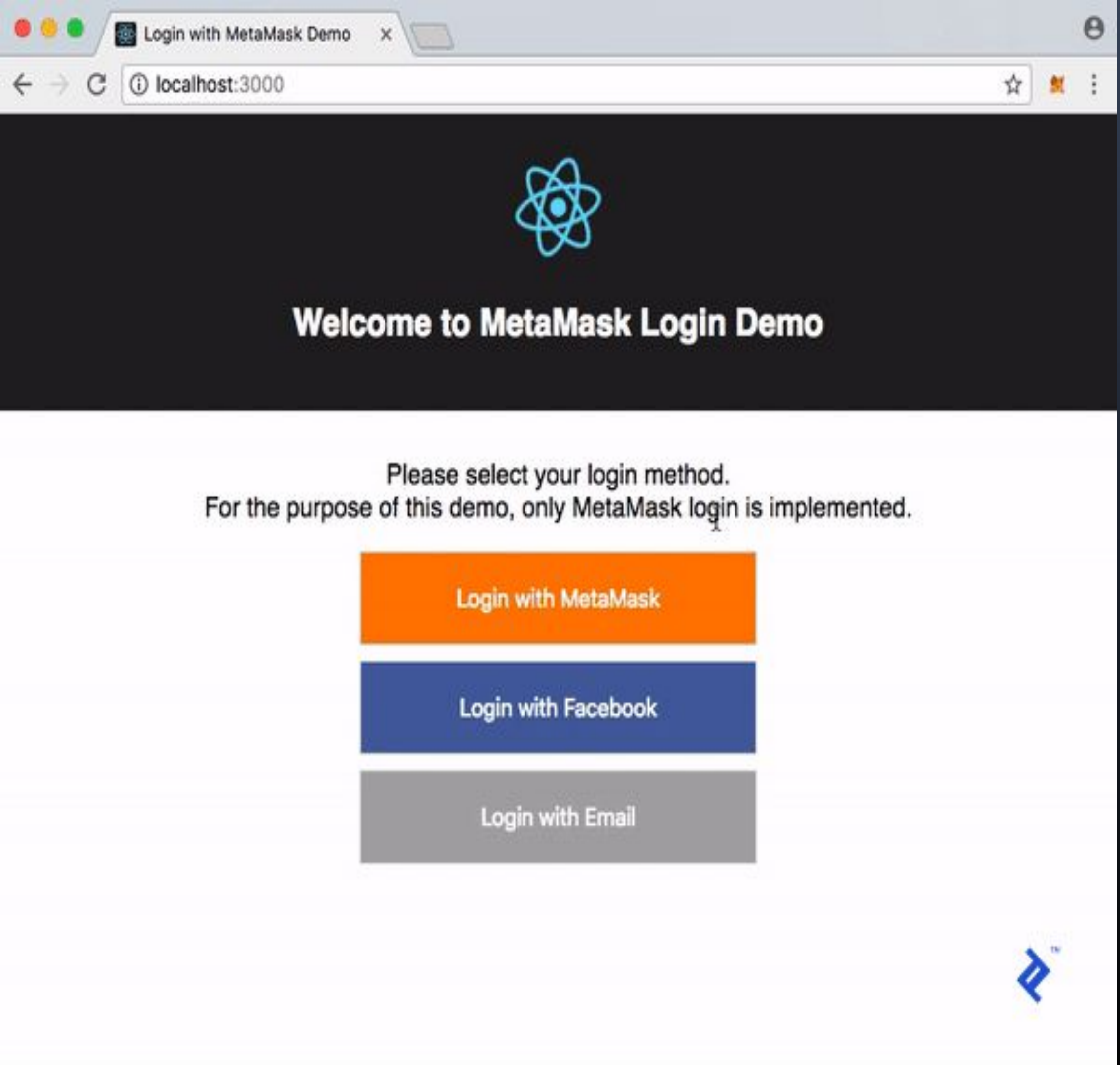
- Generates a pair of private key and public key
    - Private key(secp256k1) gives access to the wallet
    - Public key represents your address

- Handles the communication between smart contracts and the frontend
    - Read, Write, Execute instruction

- Store digital assets
    - Ethereum or ERC20 tokens like USDT etc
    - ERC721 (NFT) assets.
    - Other many variants.

How does authentication works?

Compared with current web2



Welcome to MetaMask Login Demo

Please select your login method.
For the purpose of this demo, only MetaMask login is implemented.

Why Signatures?

- Meant to be public in nature.
- Digital signatures are used to verify the ownership of an account.
- Use case: Signature owner can create a offline signature, then pass it to other user or contract that can use the signature to broadcast the transaction on behalf of the signer while paying the gas fee on behalf of the signer.

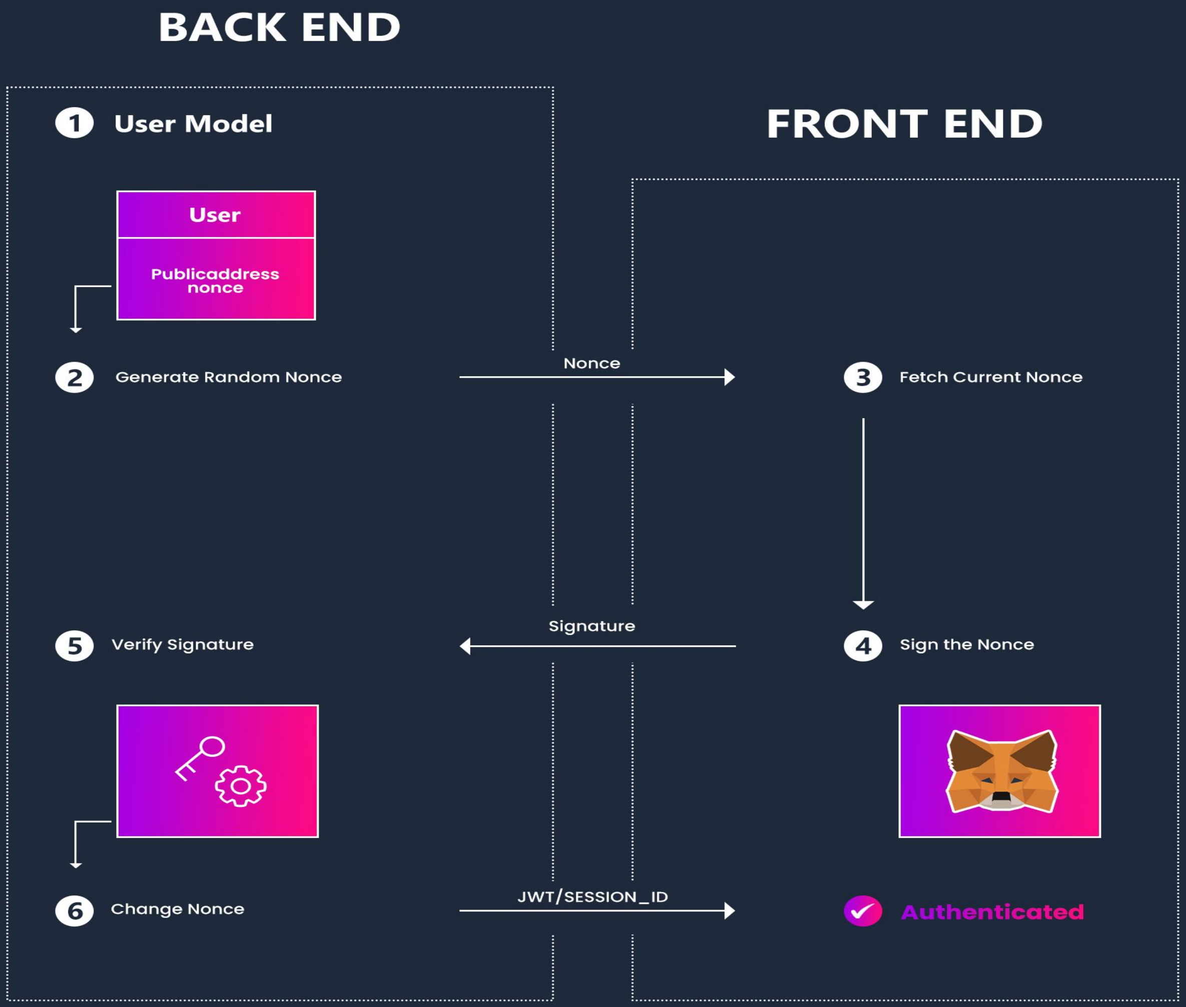
Auth Flow

1. User Initiates login and sends request to backend to create a random nonce.

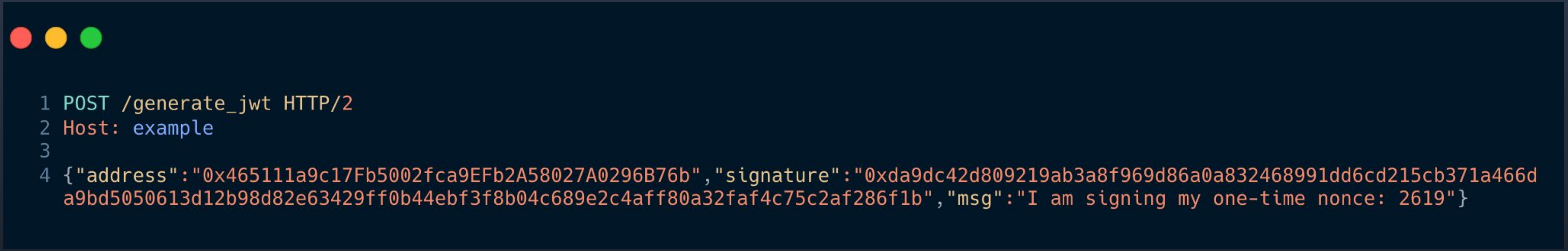
2. User signs a message which contains (message + nonce) with wallet to create a unique signature.

3. Backend verifies the signature by recovering the address of signer and generate the auth token.

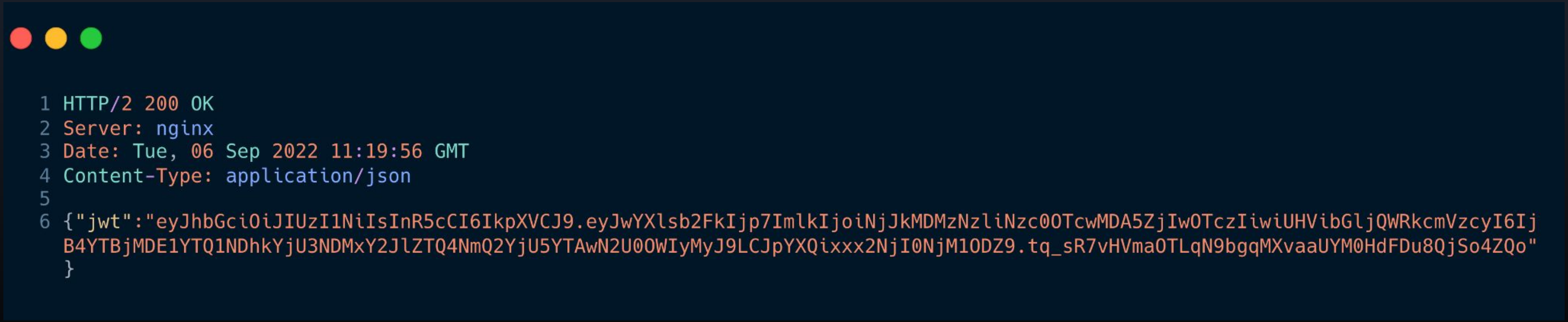
4. Backend expires the current nonce, so a unique nonce is created next time the user login.



Example of Auth Workflow



```
POST /generate_jwt HTTP/2
Host: example

{"address":"0x465111a9c17Fb5002fca9EFb2A58027A0296B76b","signature":"0xda9dc42d809219ab3a8f969d86a0a832468991dd6cd215cb371a466d
a9bd5050613d12b98d82e63429ff0b44ebf3f8b04c689e2c4aff80a32faf4c75c2af286f1b","msg":"I am signing my one-time nonce: 2619"}
```



```
HTTP/2 200 OK
Server: nginx
Date: Tue, 06 Sep 2022 11:19:56 GMT
Content-Type: application/json

{"jwt":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJwYXlsb2FkIjp7ImlkIjoiNjJkMDMzNzliNzc0OTcwMDA5ZjIwOTczIiwiUHVibGljQWRkcmVzcyI6Ij
B4YTBjMDE1YTQ1NDhkYjU3NDMxY2JlZTQ4NmQ2YjU5YTAwN2U0OWIyMyJ9LCJpYXQixxx2NjI0NjM1ODZ9.tq_sR7vHVmaOTLqN9bgqMXvaaUYM0HdFDu8QjSo4ZQo"
}
```

Authentication Vulnerabilities

1. Missing random nonce
   - Signature Replay

2. Validator accepts arbitrary message.

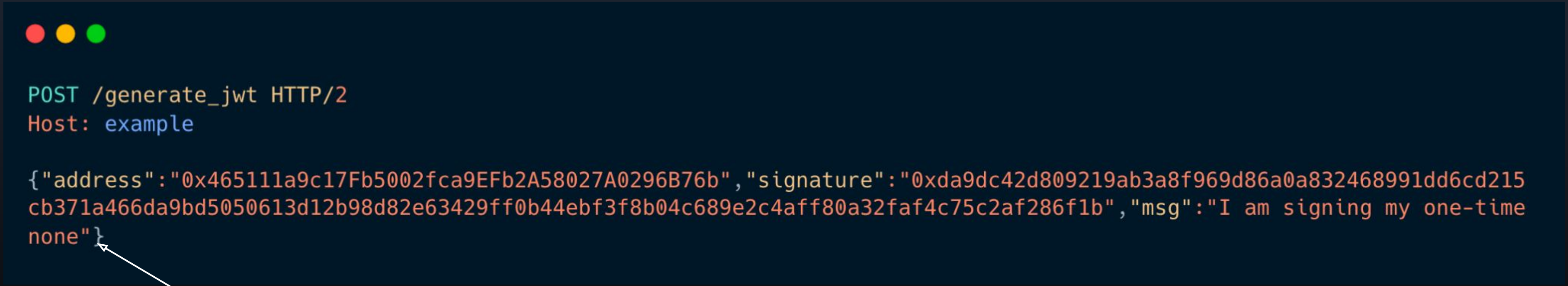
   ```
   {sigHash:"0xabc..",userAddr:"[victim address]"}
   ```

   - Logged in as victim.

Missing Random Nonce

When a cryptographic signature intended for a single use is permitted to be replayed repeatedly, leads to signature replay attacks.

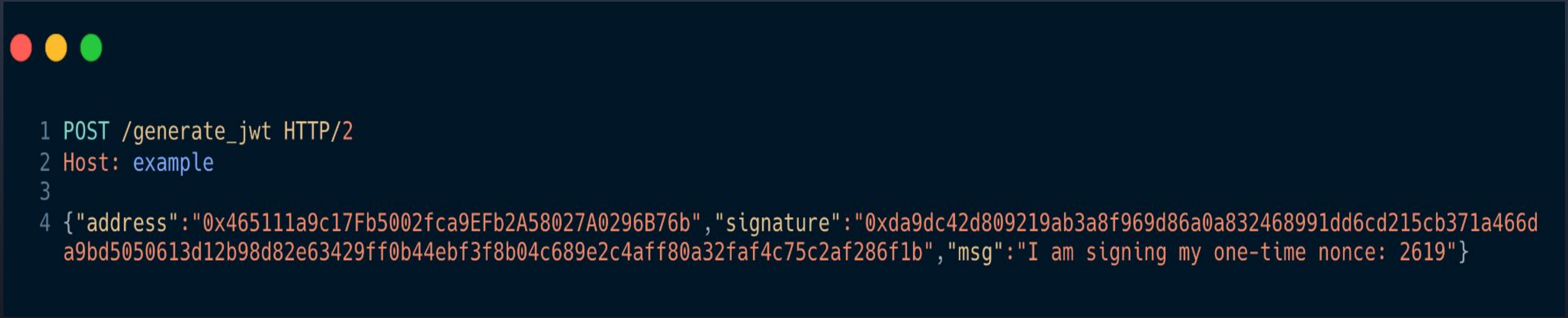
Applications that generate signatures but do not use a random nonce to generate the signatures are vulnerable to replay attacks.



```
POST /generate_jwt HTTP/2
Host: example

{"address":"0x465111a9c17Fb5002fca9EFb2A58027A0296B76b","signature":"0xda9dc42d809219ab3a8f969d86a0a832468991dd6cd215
cb371a466da9bd5050613d12b98d82e63429ff0b44ebf3f8b04c689e2c4aff80a32faf4c75c2af286f1b","msg":"I am signing my one-time
none"}
```

No nonce is used to generate a signature therefore making it vulnerable to signature replay attacks.

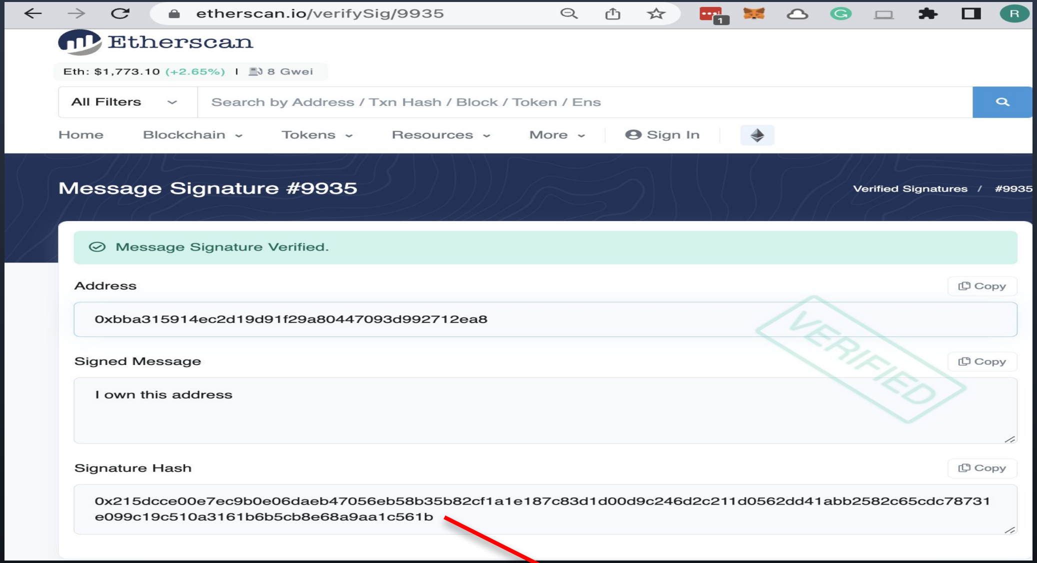
Validator Arbitrary accepts any message



```
POST /generate_jwt HTTP/2
Host: example

{"address":"0x465111a9c17Fb5002fca9EFb2A58027A0296B76b","signature":"0xda9dc42d809219ab3a8f969d86a0a832468991dd6cd215cb371a466d
a9bd5050613d12b98d82e63429ff0b44ebf3f8b04c689e2c4aff80a32faf4c75c2af286f1b","msg":"I am signing my one-time nonce: 2619"}
```

If an application only verifies the user-supplied signature without validating whether the provided message and signature are the same as those required by the application to generate JWT tokens, an **authentication bypass** could happen.

Validator Arbitrary accepts any message hash



etherscan.io/verifySig/9935

Etherscan

Eth: $1,773.10 (+2.65%) | 8 Gwei

All Filters | Search by Address / Txn Hash / Block / Token / Ens

Home | Blockchain | Tokens | Resources | More | Sign In

Message Signature #9935

Verified Signatures / #9935

Message Signature Verified.

Address

0xbba315914ec2d19d91f29a80447093d992712ea8

Signed Message

I own this address

Signature Hash

0x215dcce00e7ec9b0e06daeb47056eb58b35b82cf1a1e187c83d1d00d9c246d2c211d0562dd41abb2582c65cdc78731e099c19c510a3161b6b5cb8e68a9aa1c561b

> We substituted a random signature picked from the database of Ethereum Verified Signatures for each of the three parameter

1. Address
2. Signature
3. and message.

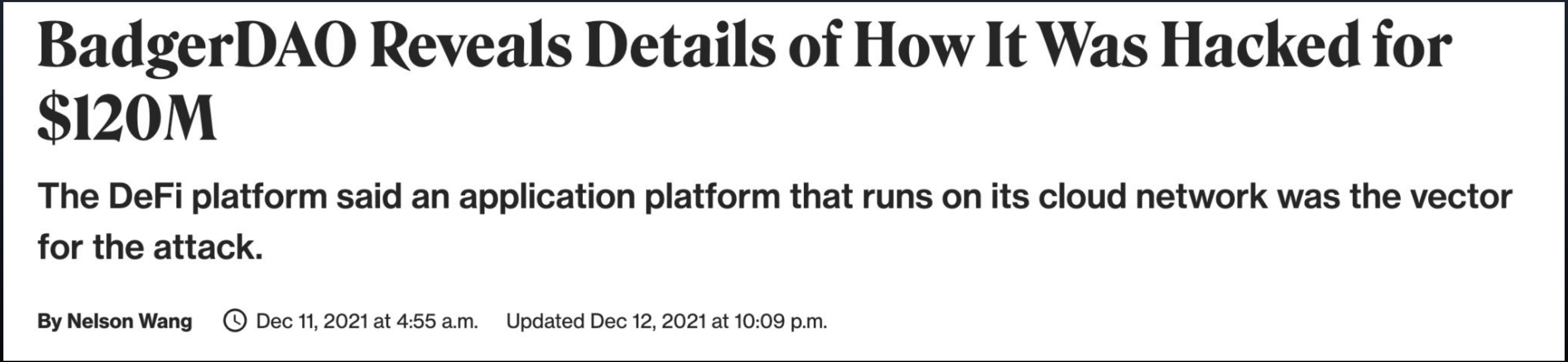
> If the application is not verifying that the message signed by the user is different from what the application asked the user to sign, an attacker could produce an auth token on the victim's behalf.

```
POST /generate_jwt HTTP/2
Host: example

{"address":"0xbba315914ec2d19d91f29a80447093d992712ea8","signature":"0x215dcce00e7ec9b0e06daeb47056eb58b35b82cf1a1e1
87c83d1d00d9c246d2c211d0562dd41abb2582c65cdc78731e099c19c510a3161b6b5cb8e68a9aa1c561b","message":"I own this
address"}
```

Client Side Injections

- Javascript injections (XSS)
- Substituting the contract addresses.
- Modifying transaction arguments or parameters.

Severity stands **critical** considering the digital assets at risk.



BadgerDAO Reveals Details of How It Was Hacked for $120M

The DeFi platform said an application platform that runs on its cloud network was the vector for the attack.

By Nelson Wang Dec 11, 2021 at 4:55 a.m. Updated Dec 12, 2021 at 10:09 p.m.

source: coindesk

Common smart contract vulnerabilities

01 Unsafe external calls

02 Insecure external dependencies

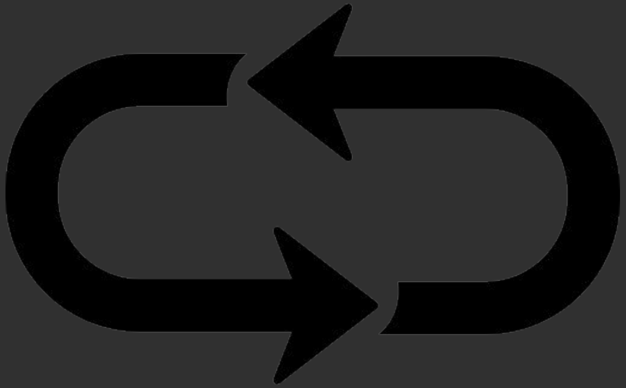
03 Access control issues

External Calls

Calls to 3rd party address that we do not control

- Calls to untrusted contracts can introduce several unexpected risks or errors.

- External calls controlled by an attacker may force your contract to transition into an undefined state.



Types of External Calls

01 STATIC - CALL

02 DELEGATE-CALL

Re-entrancy attack (call method)

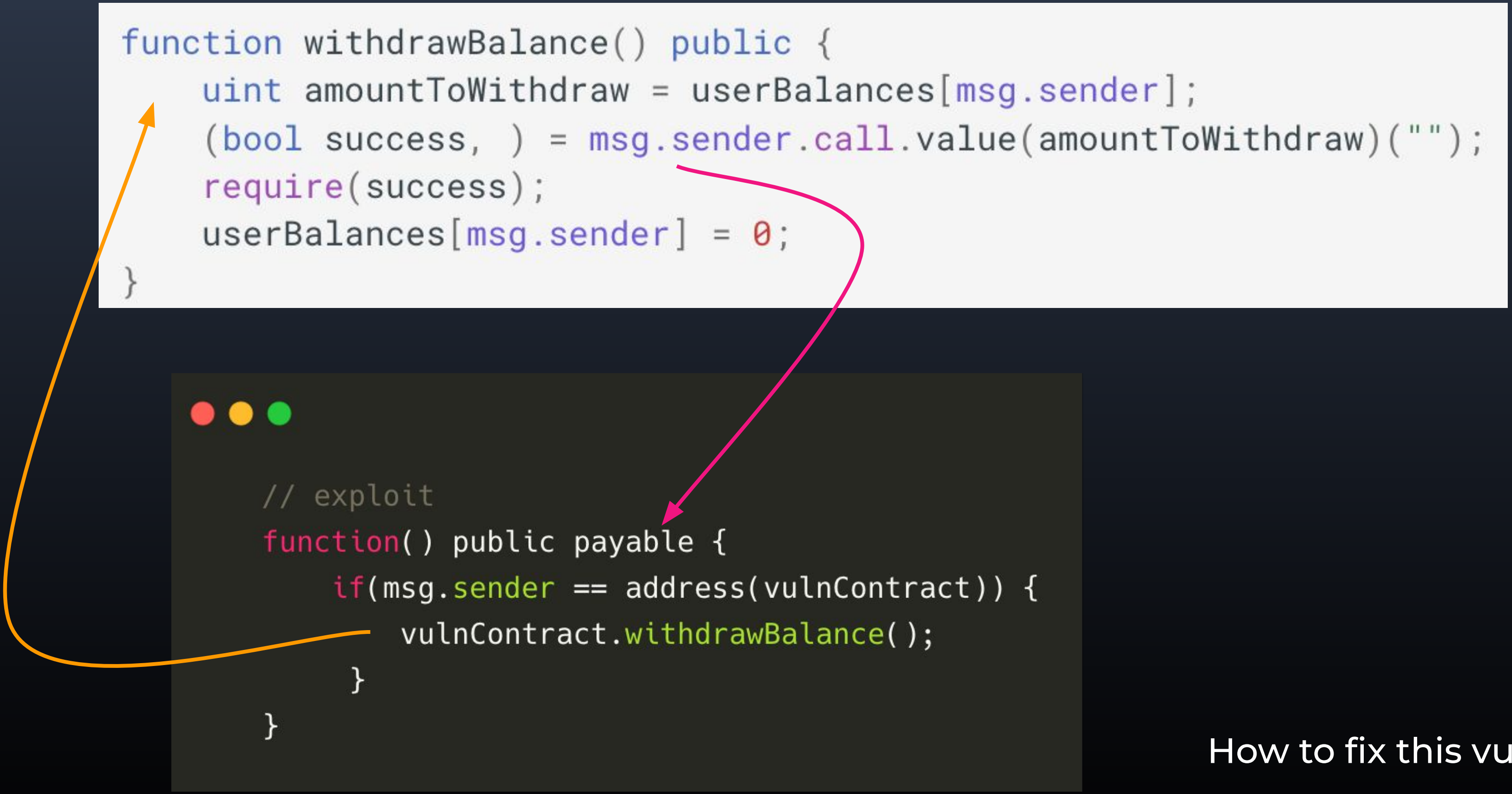
- A reentrancy attack occurs when a function makes an external call to another untrusted contract

- Then the untrusted contract makes a recursive callback to the vulnerable contract function to steal funds.

But first, Who can be the callers?

1. EOA (Externally Owned Accounts) 👥
2. Smart contracts themselves 📦

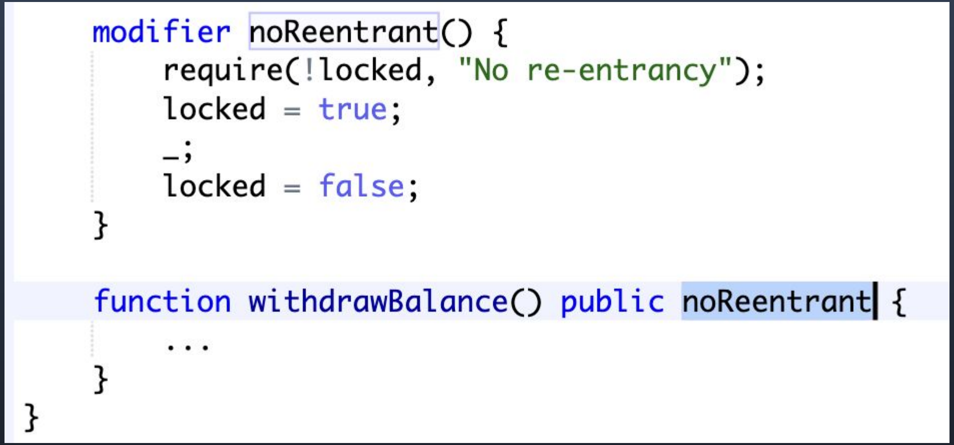
Example re-entrancy attack (call method)



```
function withdrawBalance() public {
    uint amountToWithdraw = userBalances[msg.sender];
    (bool success, ) = msg.sender.call.value(amountToWithdraw)("");
    require(success);
    userBalances[msg.sender] = 0;
}
```

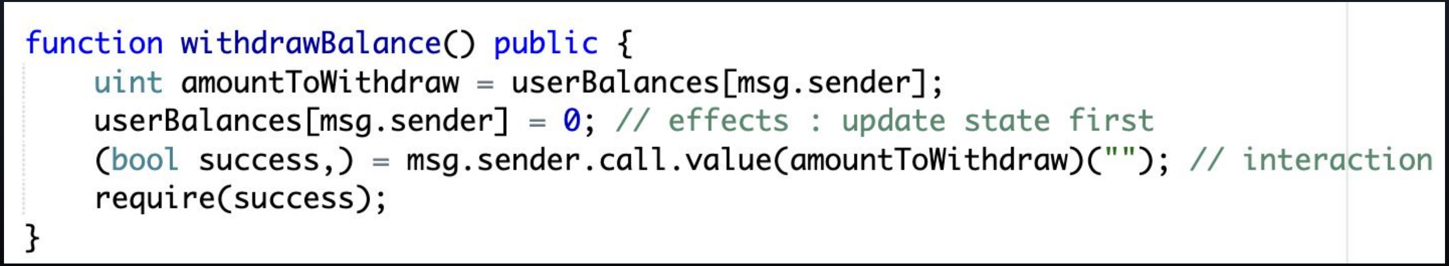
```
    // exploit
    function() public payable {
        if(msg.sender == address(vulnContract)) {
            vulnContract.withdrawBalance();
        }
    }
```

How to fix this vulnerability?



```
    modifier noReentrant() {
        require(!locked, "No re-entrancy");
        locked = true;
        _;
        locked = false;
    }

    function withdrawBalance() public noReentrant {
        ...
    }
}
```

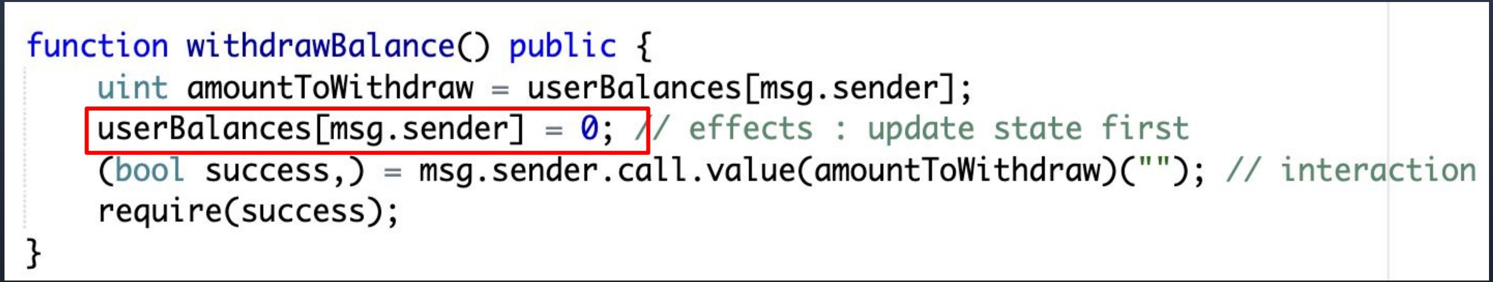
Mutex locking



```
function withdrawBalance() public {
    uint amountToWithdraw = userBalances[msg.sender];
    userBalances[msg.sender] = 0; // effects : update state first
    (bool success,) = msg.sender.call.value(amountToWithdraw)(""); // interaction
    require(success);
}
```

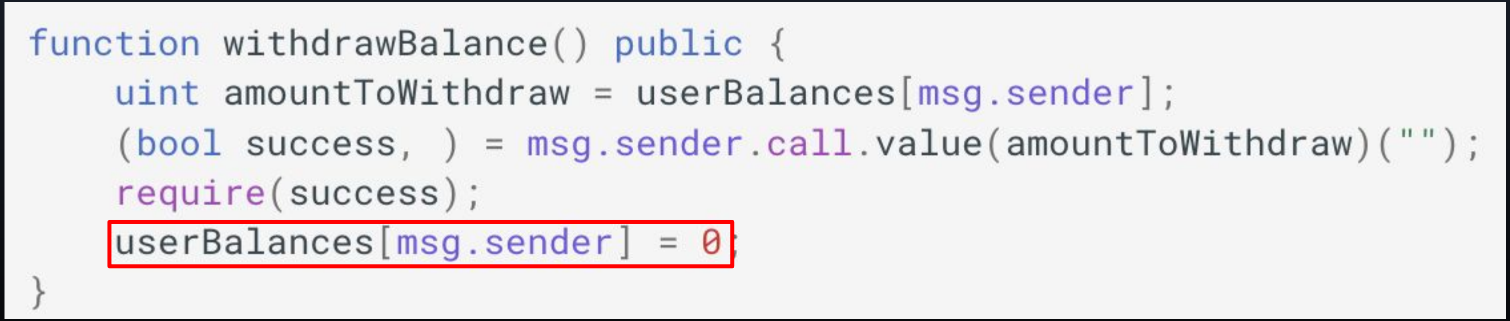
CEI (checks effects interaction) pattern

Comparison with CEI pattern



```
function withdrawBalance() public {
    uint amountToWithdraw = userBalances[msg.sender];
    userBalances[msg.sender] = 0; // effects : update state first
    (bool success,) = msg.sender.call.value(amountToWithdraw)(""); // interaction
    require(success);
}
```

CEI (checks effects interaction) pattern

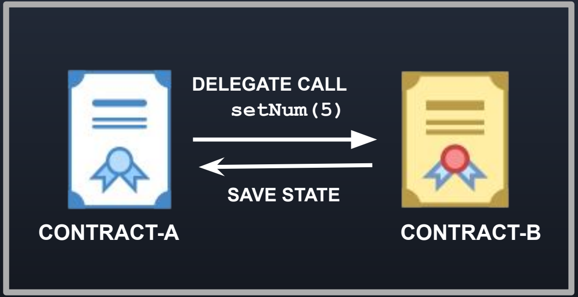


```
function withdrawBalance() public {
    uint amountToWithdraw = userBalances[msg.sender];
    (bool success, ) = msg.sender.call.value(amountToWithdraw)("");
    require(success);
    userBalances[msg.sender] = 0;
}
```

Reentrancy vulnerable pattern

Short intro to delegate(call)

- Example of the delegatecall



SLOT	Contract - A	Contract - B
0	5	0
1	.....	0

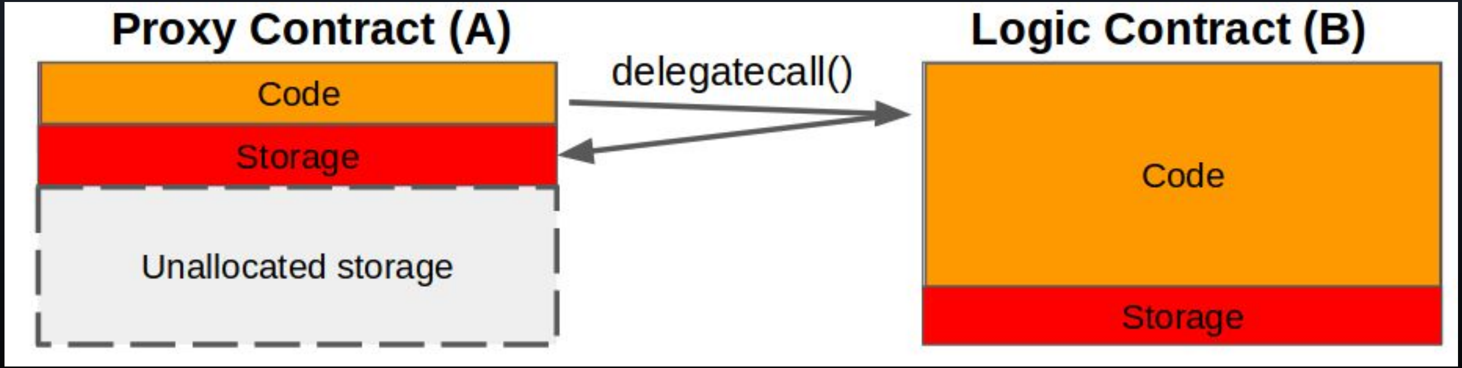
Storage layout

- Using this method, contract can preserve the storage state while using the logic of contract.

- Introduced the concept of Proxies.

Delegate(call) and proxies

- The proxy contract redirects all the calls it receives to an logic contract, whose address is stored in its (Contract A's) storage.
- The proxy contract runs Contract B's code as its own, modifying the storage and balance of Contract A.



Types of Proxies Patterns

1. **Transparent Proxy Pattern (TPP)**

   - upgrade logic is stored in **proxy** itself.
   - gas-inefficient.

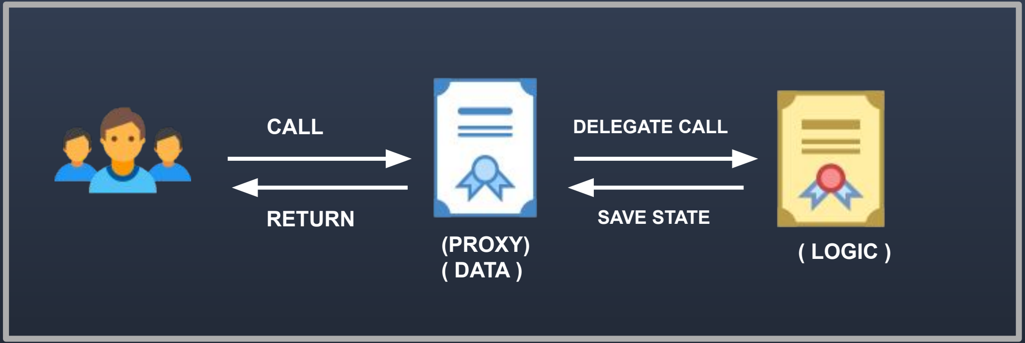
2. **Universal Upgradable Proxy Standard (UUPS)**

   - upgrade logic is stored in **logic** itself.
   - gas-efficient.

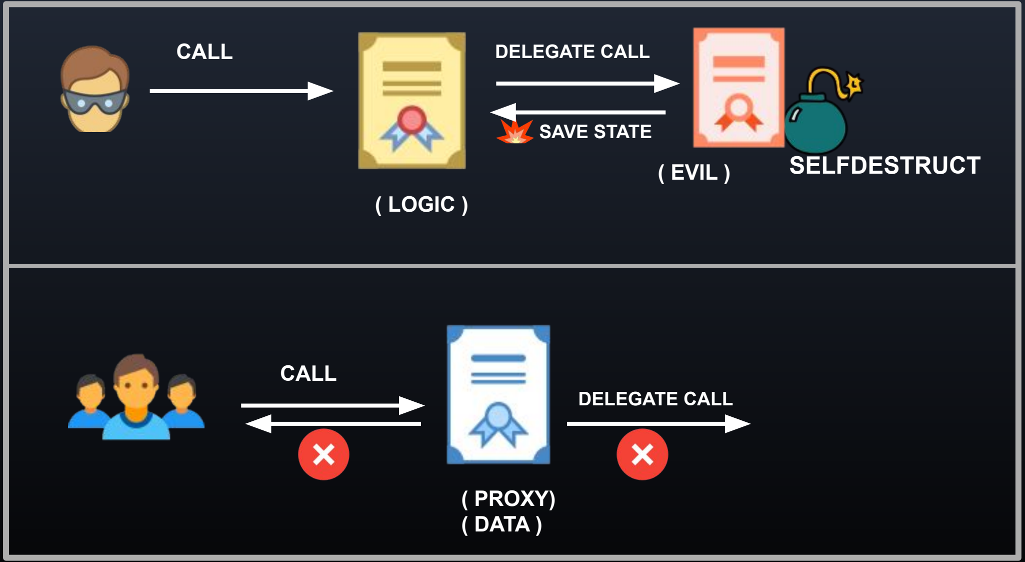
By calling the upgrade function, the storage slot on the proxy contract is updated to point to a new logic contract.

Uninitialized proxy bug

- Lot of developers often leave the contracts uninitialized. This is not an problem in most cases, but problematic when it leads to some major changes like: granting ownership to the caller.
- Owner of the contract can upgrade the implementation contract.
- This bug can lead to the self-destruction of the implementation contract, which could render the proxy contracts useless.



Normal Workflow.



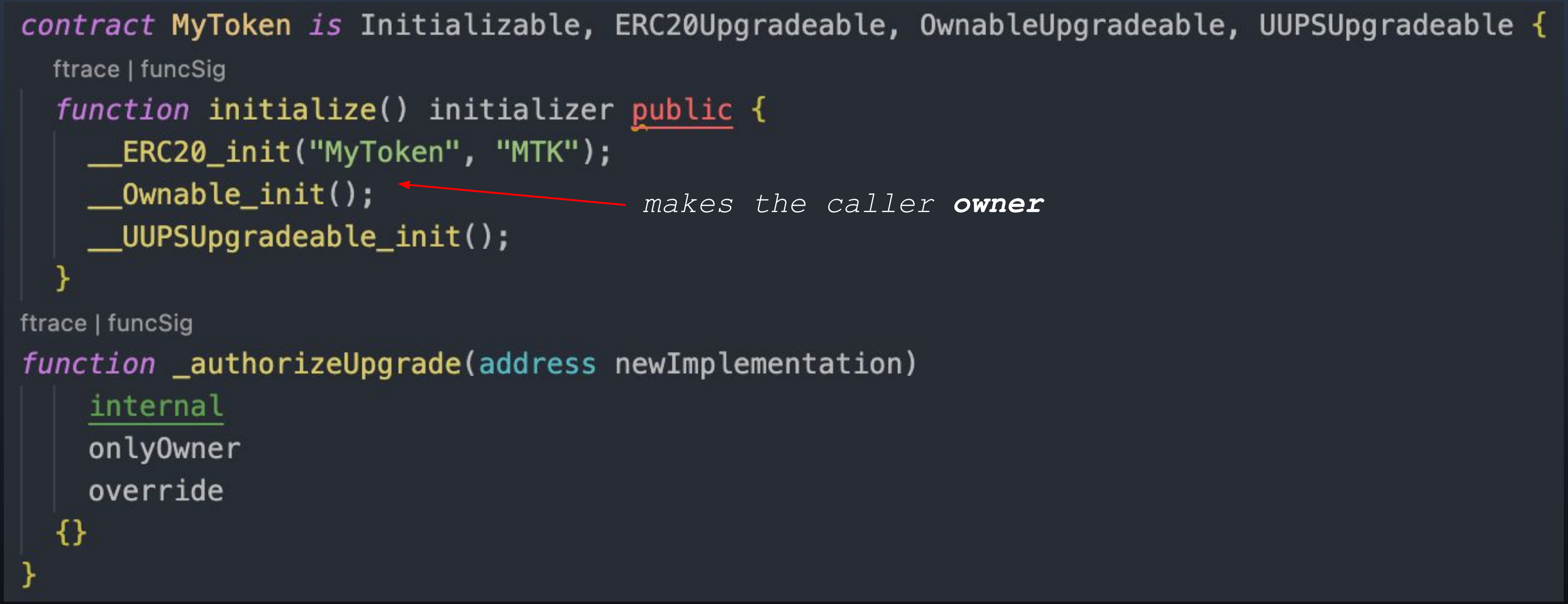
1. Malicious user deploys Evil contract containing SELFDESTRUCT opcode.

2. Delegate(call) to Evil contract.

Its storage and code are erased from the blockchain.

Proxy contract is bricked.

UUPS pattern uninitialized proxy bug



```
contract MyToken is Initializable, ERC20Upgradeable, OwnableUpgradeable, UUPSUpgradeable {
    ftrace | funcSig
    function initialize() initializer public {
        __ERC20_init("MyToken", "MTK");
        __Ownable_init();
        __UUPSUpgradeable_init();
    }
ftrace | funcSig
function _authorizeUpgrade(address newImplementation)
        internal
        onlyOwner
        override
    {}
}
```

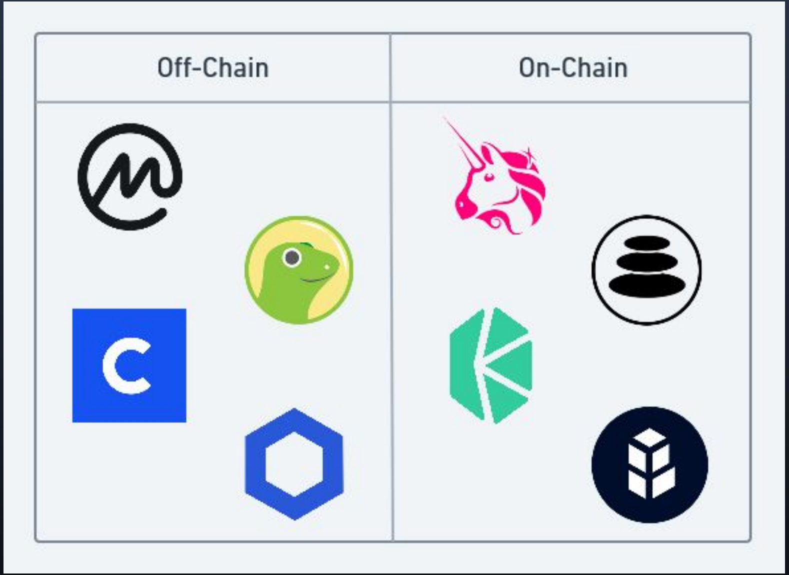
makes the caller **owner**

Wormhole bridge protocol : Attacker can held the entire protocol for ransom ($1.8 billion)
$10M Bounty : https://medium.com/immunefi/wormhole-uninitialized-proxy-bugfix-review-90250c41a43a
POC: https://github.com/immunefi-team/wormhole-uninitialized

Spot Price Dependency

Price Oracles

- A price oracle is a tool used to view the price information of a given asset.
- On-chain oracles rely on constant-product AMMs, like UniswapV2 or Balancer.
- Users rely on the current ratio of two tokens. For example, the ETH-DAI ratio gives us the current price of an ETH.



Onchain Spot Price

Finding the price of WBTC in ETH on Uniswap V2 pair for ETH/WBTC, grab the reserve balance of ETH and WBTC, then divide the two.

X = 20 WBTC , Y = 100 ETH

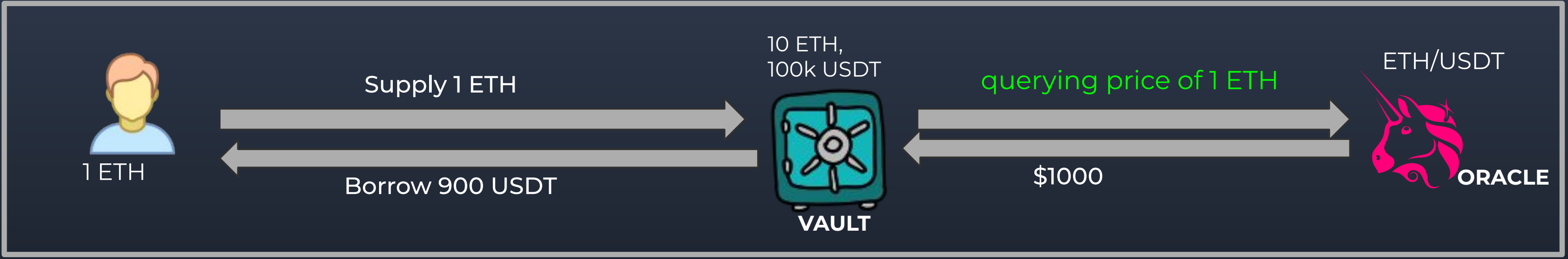
P(y) = Y / X

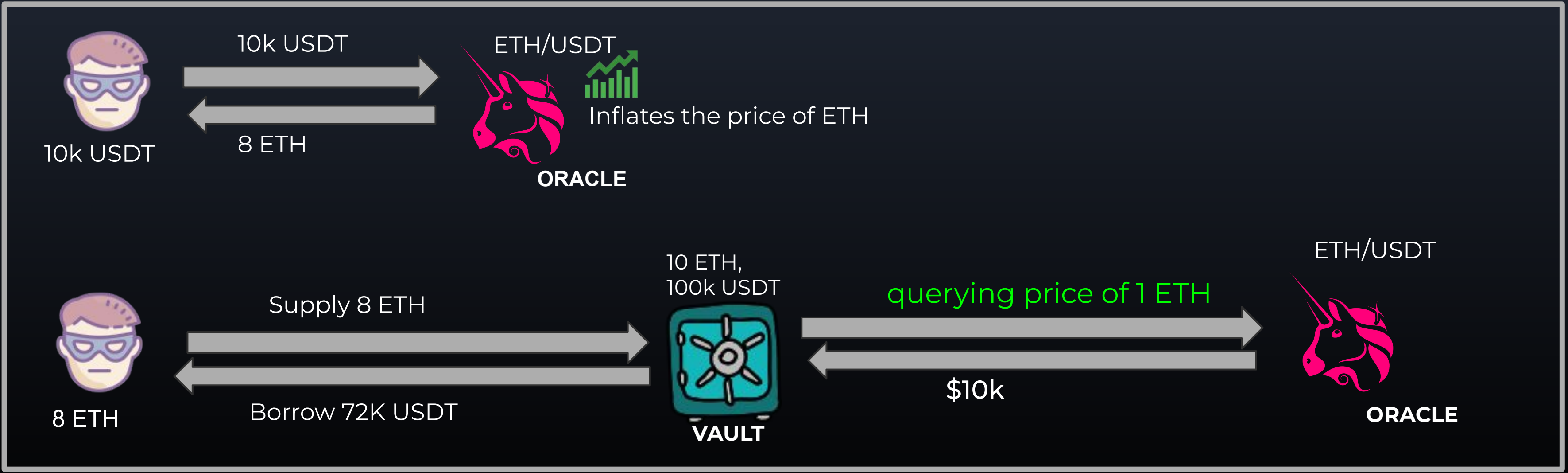
P(y) = 100 / 20 = 5 ETH per WBTC

(Easily impact the price movement by buying and selling)

- Manipulating the large volume with flash loan (considering high liquidity)
- Exploiting on borrowing platform as example. (spot price dependency)

Spot Price Dependency Example





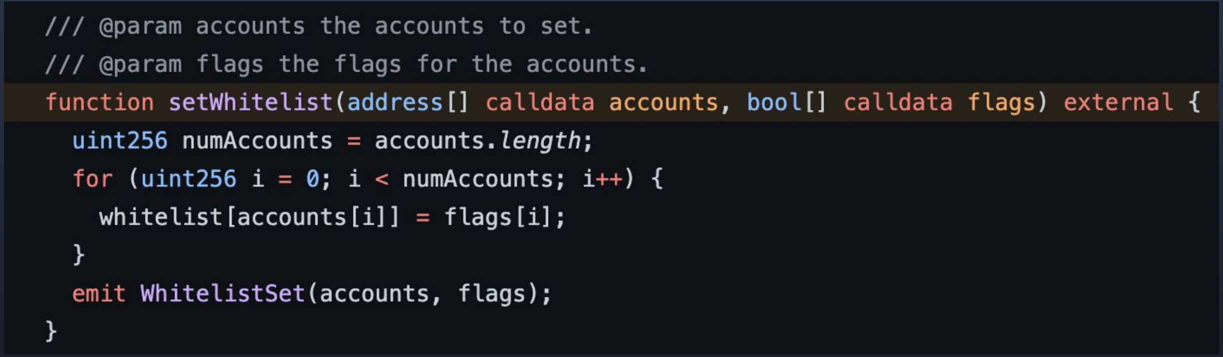
Hard choices, but better than spot price

- Relying on TWAP (Time Weighted Average Price)
  - Average price between the time intervals.

- M-of-N Reporters
  - Averaging the price between the multiple AMM products like Uniswap, MakerDAO, Balancer etc , and offchain oracle’s like chainlink.

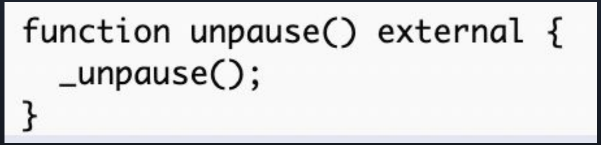
Lack of Authentication

Missing Authorization Check

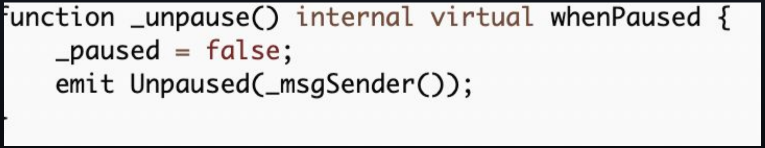


```
/// @param accounts the accounts to set.
/// @param flags the flags for the accounts.
function setWhitelist(address[] calldata accounts, bool[] calldata flags) external {
  uint256 numAccounts = accounts.length;
  for (uint256 i = 0; i < numAccounts; i++) {
    whitelist[accounts[i]] = flags[i];
  }
  emit WhitelistSet(accounts, flags);
}
```

Blacklist/Whitelist any user on the contract.



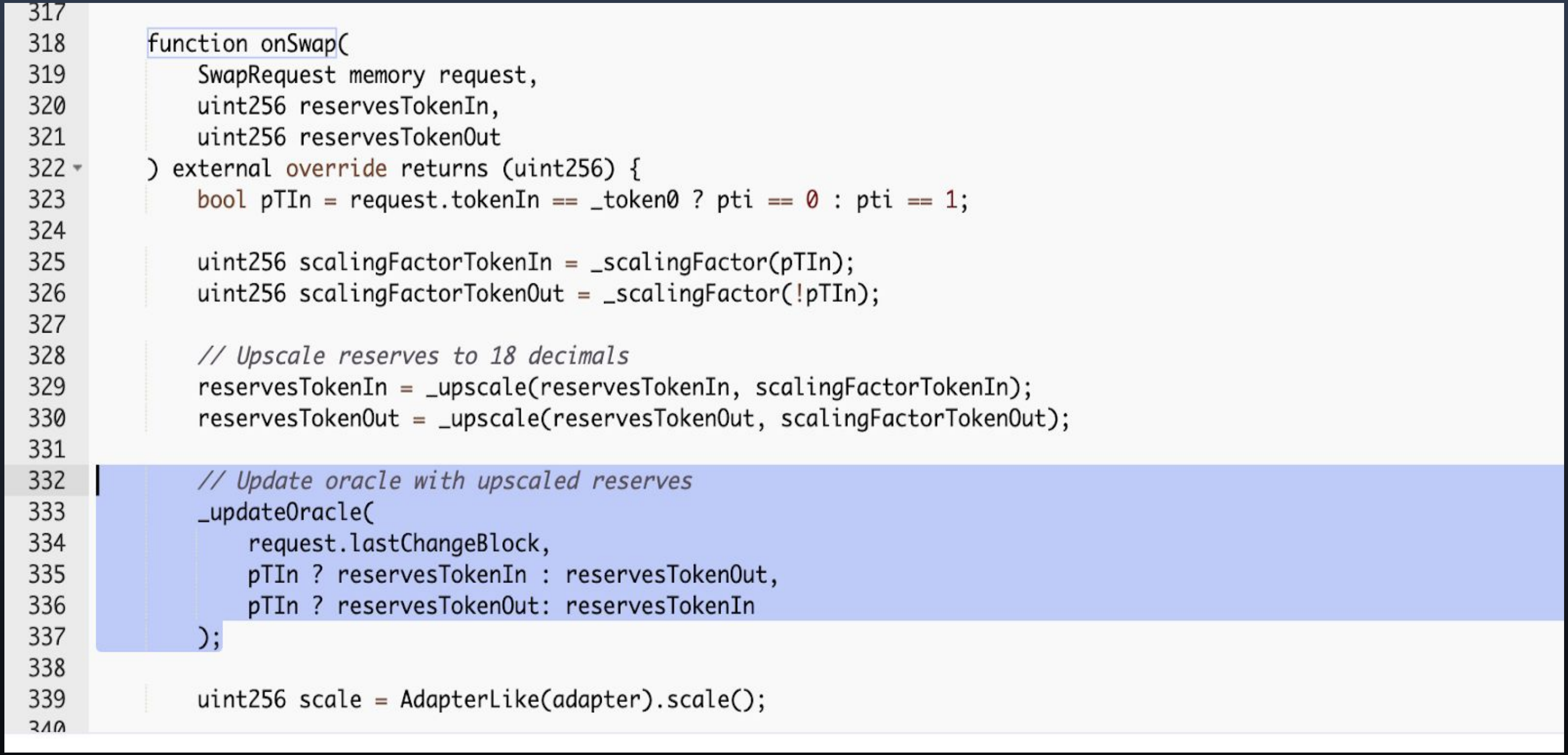
```
function unpause() external {
  _unpause();
}
```



```
unction _unpause() internal virtual whenPaused {
   _paused = false;
   emit Unpaused(_msgSender());
```

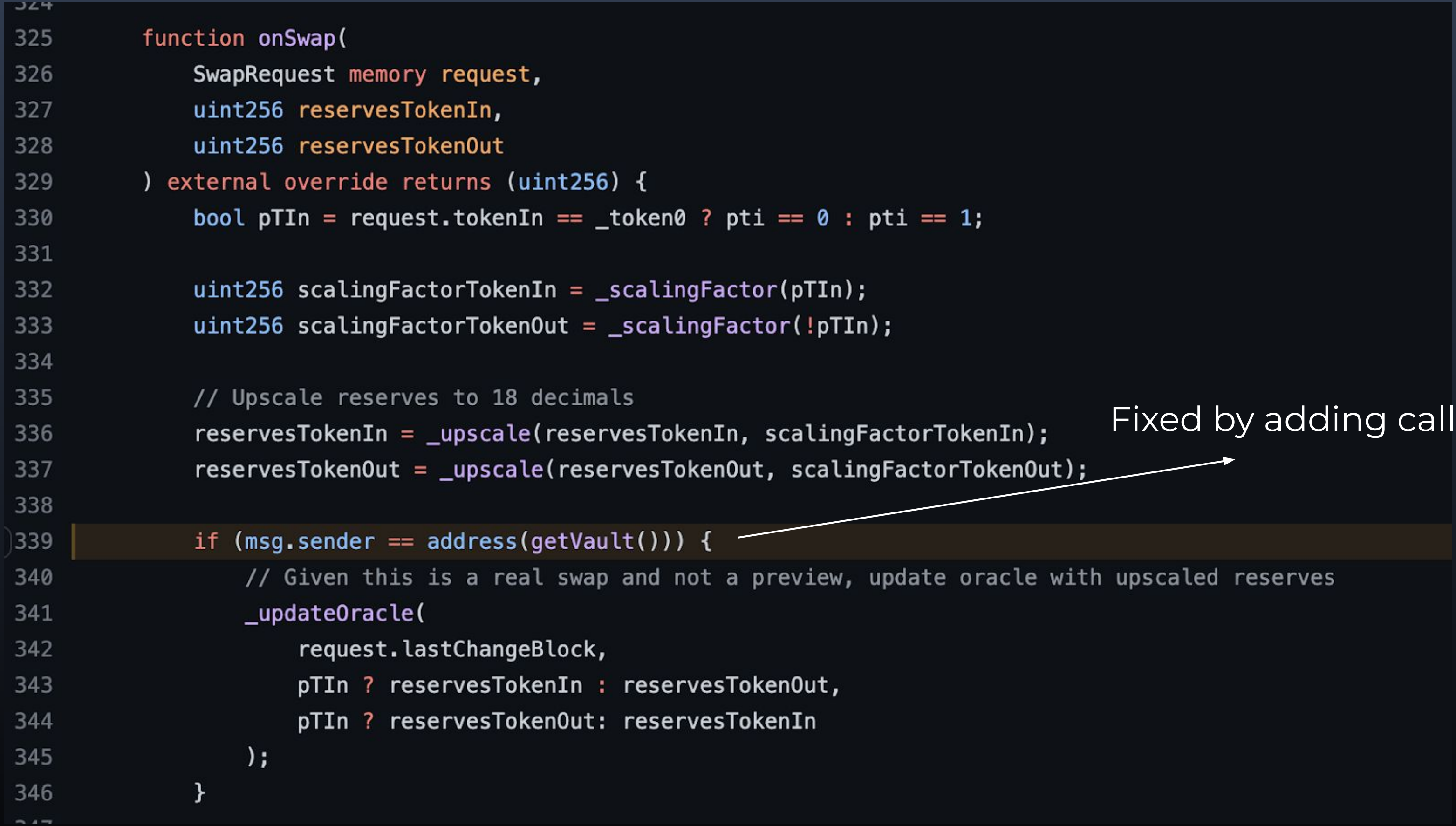
Creates a griefing opportunity by disrupting the operations of the protocol.

Sense Finance : Lack of access control updating oracle data



```
    function onSwap(
        SwapRequest memory request,
        uint256 reservesTokenIn,
        uint256 reservesTokenOut
    ) external override returns (uint256) {
        bool pTIn = request.tokenIn == _token0 ? pti == 0 : pti == 1;

        uint256 scalingFactorTokenIn = _scalingFactor(pTIn);
        uint256 scalingFactorTokenOut = _scalingFactor(!pTIn);

        // Upscale reserves to 18 decimals
        reservesTokenIn = _upscale(reservesTokenIn, scalingFactorTokenIn);
        reservesTokenOut = _upscale(reservesTokenOut, scalingFactorTokenOut);

        // Update oracle with upscaled reserves
        _updateOracle(
            request.lastChangeBlock,
            pTIn ? reservesTokenIn : reservesTokenOut,
            pTIn ? reservesTokenOut: reservesTokenIn
        );

        uint256 scale = AdapterLike(adapter).scale();
```

Anyone can call the **onSwap()** which updates the stored oracle information on the pool contract.



```
    function onSwap(
        SwapRequest memory request,
        uint256 reservesTokenIn,
        uint256 reservesTokenOut
    ) external override returns (uint256) {
        bool pTIn = request.tokenIn == _token0 ? pti == 0 : pti == 1;

        uint256 scalingFactorTokenIn = _scalingFactor(pTIn);
        uint256 scalingFactorTokenOut = _scalingFactor(!pTIn);

        // Upscale reserves to 18 decimals
        reservesTokenIn = _upscale(reservesTokenIn, scalingFactorTokenIn);
        reservesTokenOut = _upscale(reservesTokenOut, scalingFactorTokenOut);

        if (msg.sender == address(getVault())) {
            // Given this is a real swap and not a preview, update oracle with upscaled reserves
            _updateOracle(
                request.lastChangeBlock,
                pTIn ? reservesTokenIn : reservesTokenOut,
                pTIn ? reservesTokenOut: reservesTokenIn
            );
        }
```

Fixed by adding caller check

https://medium.com/immunefi/sense-finance-access-control-issue-bugfix-review-32e0c806b1a0

Why bug bounties and role of Immunefi?

- When code is law and code is money, then a bug that is exploited is just straight money for the exploiter.
- Protecting from the biggest threats of space.
- Largest bug bounty program for DeFi and blockchain in general.
- Attract the best talent with the best reward.
- We hope to turn blackhats into whitehats through alignments of incentives with projects.

Who wants to become a Web3 Hacker?

Useful links to get you started

- https://medium.com/immunefi/hacking-the-blockchain-an-ultimate-guide-4f34b33c6e8b
- https://solidity-by-example.org
- https://github.com/ethereumbook/ethereumbook
- https://github.com/OffcierCia/DeFi-Developer-Road-Map
- https://www.damnvulnerabledefi.xyz
- https://cmichel.io/how-to-become-a-smart-contract-auditor/
- https://ethernaut.openzeppelin.com/
- https://github.com/immunefi-team/community-challenges