Denial of Service (DoS)

Team-3:
Adit, Avi, & Emma

What is a DoS Attack?

- Attack where the contract is made inoperational temporarily or even permanently

ETH can be stuck forever!

Potential Vulnerabilities:

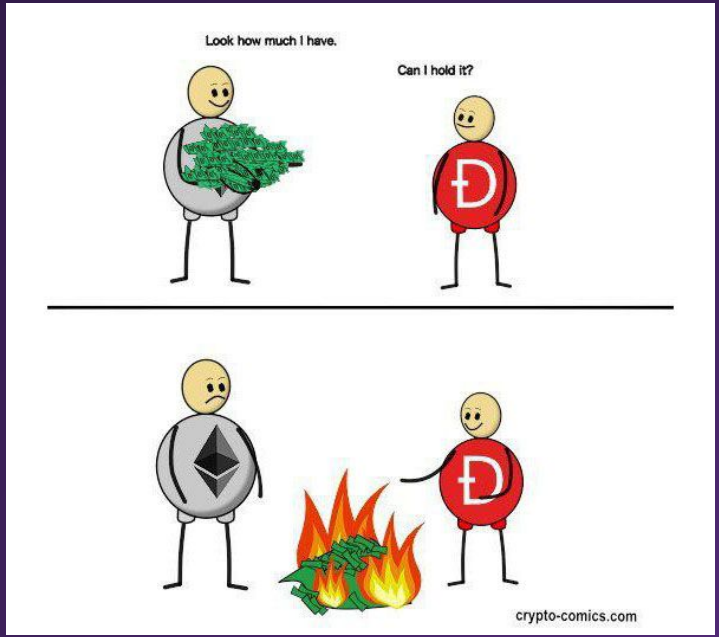
- Execution gas > block gas limit
- Owner loses pvt key
- Necessary call failure

https://gist.github.com/vasa-develop/32b7472991feaeda33e5eb96af980d7a#file-snippet-sol

```
bool public isFinalized = false;
address public owner; // gets set somewhere
function finalize() public {
    require(msg.sender == owner);
    isFinalized == true;
}
// ... extra ICO functionality
// overloaded transfer function
function transfer(address _to, uint _value) returns (bool) {
    require(isFinalized);
    super.transfer(_to,_value)
}
```

DoS vulnerability

Real World Example: GovernMental



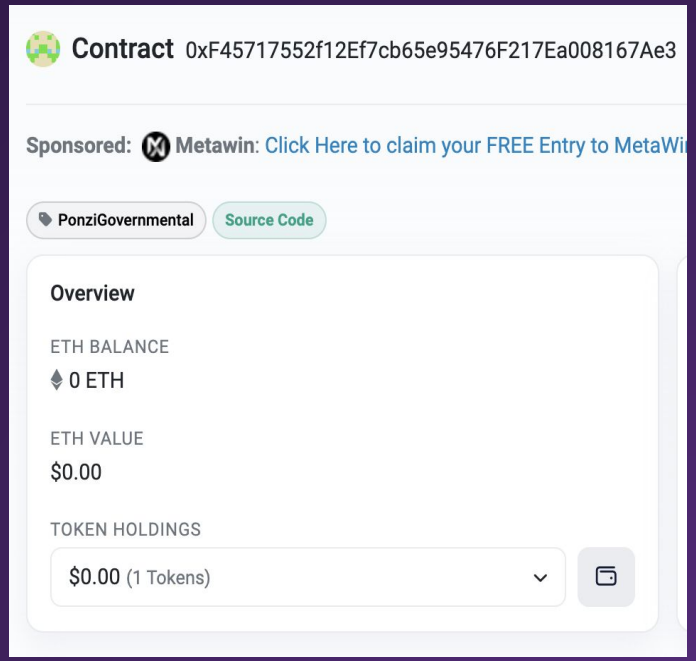
What is GovernMental?

http://governmental.github.io/GovernMental/

An educational game which simulates the finances of a government - in other words: It's a **Ponzi scheme.**

Four rules to play the game:

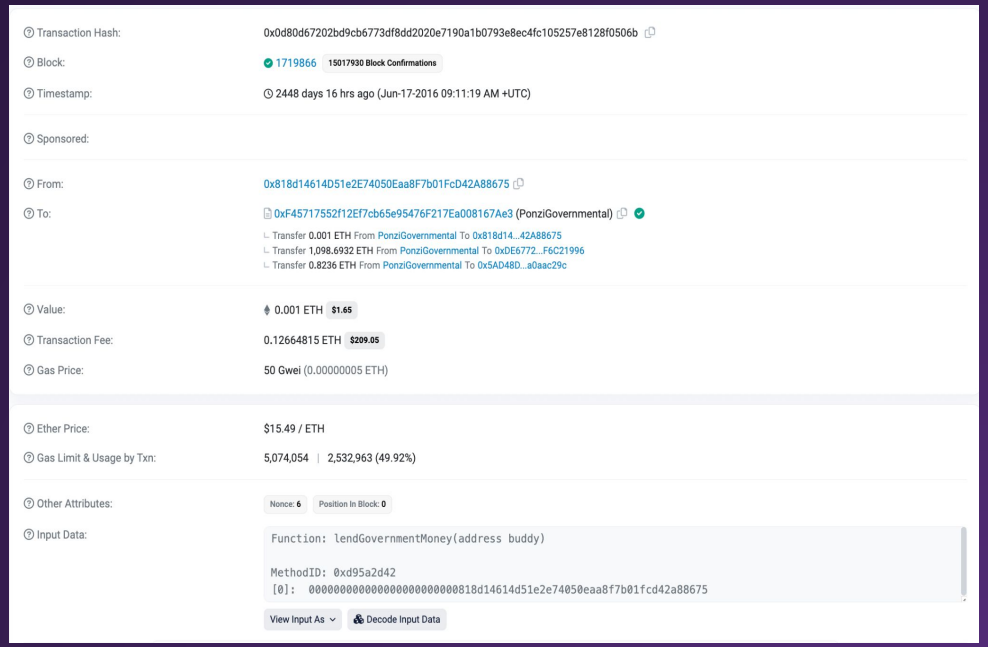
1. Lend the government money who promise to pay back + 10% interest
2. If the government does not receive money for 12 hours, it breaks down. Only the last creditor gets the jackpot. All others lose claim.
3. All incoming money is used in the following way: 5% goes into the "jackpot" (capped at 10k Ether), 5% goes to the corrupt elite that runs the government, 90% are used to pay out creditors in order of their date of credit. When the jackpot is full, 95% go toward the payout of creditors.
4. Creditors can share an affiliate link. Money deposited this way is distributed as follows: 5% go toward the linker directly, 5% to the corrupt elite, 5% into the jackpot (until full). The rest is used for payouts.



The Deployed Contract

PostMortem (what happened?)

- Governmental has a large array of creditors
- When the contract pays out the prize to the lucky winner, it clears the array.
- ENORMOUS AMOUNT OF GAS, > 5 million
- Block gas limit is ~ 4 million at the time



Transaction Hash: 0x0d80d67202bd9cb6773df8dd2020e7190a1b0793e8ec4fc105257e8128f0506b

Block: 1719866 15017930 Block Confirmations

Timestamp: 2448 days 16 hrs ago (Jun-17-2016 09:11:19 AM +UTC)

Sponsored:

From: 0x818d14614D51e2E74050Eaa8F7b01FcD42A88675

To: 0xF45717552f12Ef7cb65e95476F217Ea008167Ae3 (PonziGovernmental)

Transfer 0.001 ETH From PonziGovernmental To 0x818d14...42A88675
Transfer 1,098.6932 ETH From PonziGovernmental To 0xDE6772...F6C21996
Transfer 0.8236 ETH From PonziGovernmental To 0x5AD48D...a0aac29c

Value: 0.001 ETH $1.65

Transaction Fee: 0.12664815 ETH $209.05

Gas Price: 50 Gwei (0.00000005 ETH)

Ether Price: $15.49 / ETH

Gas Limit & Usage by Txn: 5,074,054 | 2,532,963 (49.92%)

Other Attributes: Nonce: 6 Position In Block: 0

Input Data:

```
Function: lendGovernmentMoney(address buddy)

MethodID: 0xd95a2d42
[0]:  000000000000000000000000818d14614d51e2e74050eaa8f7b01fcd42a88675
```

View Input As Decode Input Data

The Final Transaction to Withdraw

PostMortem (code)

```
contract Government {

    // Global Variables
    ...
    address[] public creditorAddresses;
    ...
    function lendGovernmentMoney(address buddy) returns (bool) {
        ...
        if (lastTimeOfNewCredit + TWELVE_HOURS < block.timestamp) {
           ...
           creditorAddresses = new address[](0);
           creditorAmounts = new uint[](0);
           ...
```

Preventative techniques

Contracts should not loop over data structures whose size can be changed by external users

In GovernMental:

- One option: keep track of live element instead of deleting

If privileged users are needed to change the state:

- Have multiple privileged users
- Time constraint alternative: unlocking either by owner or if current_time > presetTime

If external calls are needed to move forward:

- Account of their possible failures
- Again could have a time constraint alternative

https://ethereum.stackexchange.com/questions/3373/how-to-clear-large-arrays-without-blowing-the-gas-limit

```
uint numElements = 0;
uint[] array;

function insert(uint value) {
    if(numElements == array.length) {
        array.length += 1;
    }
    array[numElements++] = value;
}

function clear() {
    numElements = 0;
}
```

One Possible Alternative

The Challenges!!

Token Distributor

The owner wishes to distribute tokens amongst their investors. How can you prevent everyone from claiming their funds?

KickStarter

You will beat this level if you are able to prevent everyone from withdrawing their money, even the owner !!

Unstoppable

Stop this relentless lender from offering loans.You start out with 50 DVT tokens.

Tutorial: For Loops can be Gas Guzzlers

```
function testLoop1() public {
    for(uint i = 0; i < 3; i++) {
        value = value + i;
    }
}

function testLoop2() public {
    for(uint i = 0; i <= 2; i++) {
        value = value + i;
    }
}
```

Both Functions do 3 iterations

- Function 1 uses LESS THAN
- Function 2 uses LESS THAN OR EQUAL

Which function uses more gas?

Tutorial: For Loops can be Gas Guzzlers

```
function testLoop1() public {
    for(uint i = 0; i < 3; i++) {
        value = value + i;
    }
}
```

Gas Used by Transaction 27,779

```
function testLoop2() public {
    for(uint i = 0; i <= 2; i++) {
        value = value + i;
    }
}
```

Gas Used by Transaction 27,789

Function 2 uses ~10 extra gas

Why?

- There are only LT, GT, and EQ opcodes
- Each one costs a minimum 3 gas
- So function 2 needs to do a LT and an EQ operation everytime it checks loop condition

OPCODE	NAME	MINIMUM GAS
10	LT	3
11	GT	3
12	SLT	3
13	SGT	3
14	EQ	3

Tutorial: For Loops can be Gas Guzzlers

```
function testLoop1() public {
    for(uint i = 0; i < 3; i++) {
        value = value + i;
    }
}

function testLoop2() public {
    for(uint i = 0; i < 3;) {
        value = value + i;
        unchecked {
            i++;
        }
    }
}
```

Again, we have two functions, each does 3 iterations.

Which one is cheaper now?

Tutorial: For Loops can be Gas Guzzlers

```
function testLoop1() public {
    for(uint i = 0; i < 3; i++) {
        value = value + i;
    }
}
```

Gas Used by Transaction 27,779

```
function testLoop2() public {
    for(uint i = 0; i < 3;) {
        value = value + i;
        unchecked {
            i++;
        }
    }
}
```

Gas Used by Transaction 27,432

Function 2 is much cheaper now

Why?

After version 0.8 Solidity has safety checks for all integer arithmetic, including overflow and underflow guards.

If we know something will never over/underflow, we can use unchecked for gas savings

Tutorial: Token Distributor Challenge

```
pragma solidity ^0.8.11;

contract TokenDistributor {
    address public owner;
    address[] investors; // array of investors
    uint[] investorTokens; // the amount of tokens each investor gets

    function invest() public payable {
        investors.push(msg.sender);
        investorTokens.push(msg.value);
    }

    function distribute() public {
        require(msg.sender == owner);
        for(uint i = 0; i < investors.length; i++) {
            transferToken(i);
        }
    }

    function transferToken(uint index) private {
        address to = investors[index];
        uint amount = investorTokens[index];
        investorTokens[index] = 0;
    }

    constructor() {
        owner = msg.sender;
    }

    receive() external payable {}
}
```

This contract lets anyone invest into the Token. After people have invested, the contract owner can transfer everyone their tokens

What could go wrong?

Tutorial: Token Distributor Challenge

```
pragma solidity ^0.8.11;

contract TokenDistributor {
    address public owner;
    address[] investors; // array of investors
    uint[] investorTokens; // the amount of tokens each investor gets

    function invest() public payable {
        investors.push(msg.sender);
        investorTokens.push(msg.value);
    }

    function distribute() public {
        require(msg.sender == owner);
        for(uint i = 0; i < investors.length; i++) {
            transferToken(i);
        }
    }

    function transferToken(uint index) private {
        address to = investors[index];
        uint amount = investorTokens[index];
        investorTokens[index] = 0;
    }

    constructor() {
        owner = msg.sender;
    }

    receive() external payable {}
}
```

Anyone can invest into the token with as much money as they want

But, this function doesn't check if you are already an investor

Tutorial: Token Distributor Challenge

```
pragma solidity ^0.8.11;

contract TokenDistributor {
    address public owner;
    address[] investors; // array of investors
    uint[] investorTokens; // the amount of tokens each investor gets

    function invest() public payable {
        investors.push(msg.sender);
        investorTokens.push(msg.value);
    }

    function distribute() public {
        require(msg.sender == owner);
        for(uint i = 0; i < investors.length; i++) {
            transferToken(i);
        }
    }

    function transferToken(uint index) private {
        address to = investors[index];
        uint amount = investorTokens[index];
        investorTokens[index] = 0;
    }

    constructor() {
        owner = msg.sender;
    }

    receive() external payable {}
}
```

When it's time to give everyone their tokens, the owner calls distribute()

But, this function iterates over the whole investors array (which isn't a fixed size)

Tutorial: Token Distributor Challenge

```
pragma solidity ^0.8.11;

contract TokenDistributor {
    address public owner;
    address[] investors; // array of investors
    uint[] investorTokens; // the amount of tokens each investor gets

    function invest() public payable {
        investors.push(msg.sender);
        investorTokens.push(msg.value);
    }

    function distribute() public {
        require(msg.sender == owner);
        for(uint i = 0; i < investors.length; i++) {
            transferToken(i);
        }
    }

    function transferToken(uint index) private {
        address to = investors[index];
        uint amount = investorTokens[index];
        investorTokens[index] = 0;
    }

    constructor() {
        owner = msg.sender;
    }

    receive() external payable {}
}
```

distribute() makes a call to transferToken()

But, this function doesn't remove investors from the array after they have been paid

It just sets their token allocation to zero

Tutorial: Token Distributor Challenge

```
pragma solidity ^0.8.11;

contract TokenDistributor {
    address public owner;
    address[] investors; // array of investors
    uint[] investorTokens; // the amount of tokens each investor gets

    function invest() public payable {
        investors.push(msg.sender);
        investorTokens.push(msg.value);
    }

    function distribute() public {
        require(msg.sender == owner);
        for(uint i = 0; i < investors.length; i++) {
            transferToken(i);
        }
    }

    function transferToken(uint index) private {
        address to = investors[index];
        uint amount = investorTokens[index];
        investorTokens[index] = 0;
    }

    constructor() {
        owner = msg.sender;
    }

    receive() external payable {}
}
```

distribute() makes a call to transferToken()

But, this function doesn't remove investors from the array after they have been paid

It just sets their token allocation to zero

THANKS!

Happy Hacking ;)

Presentation template by SlidesCarnival