# D5: Denial of Service

Portland State
Computer Science

# #5: Denial of Service

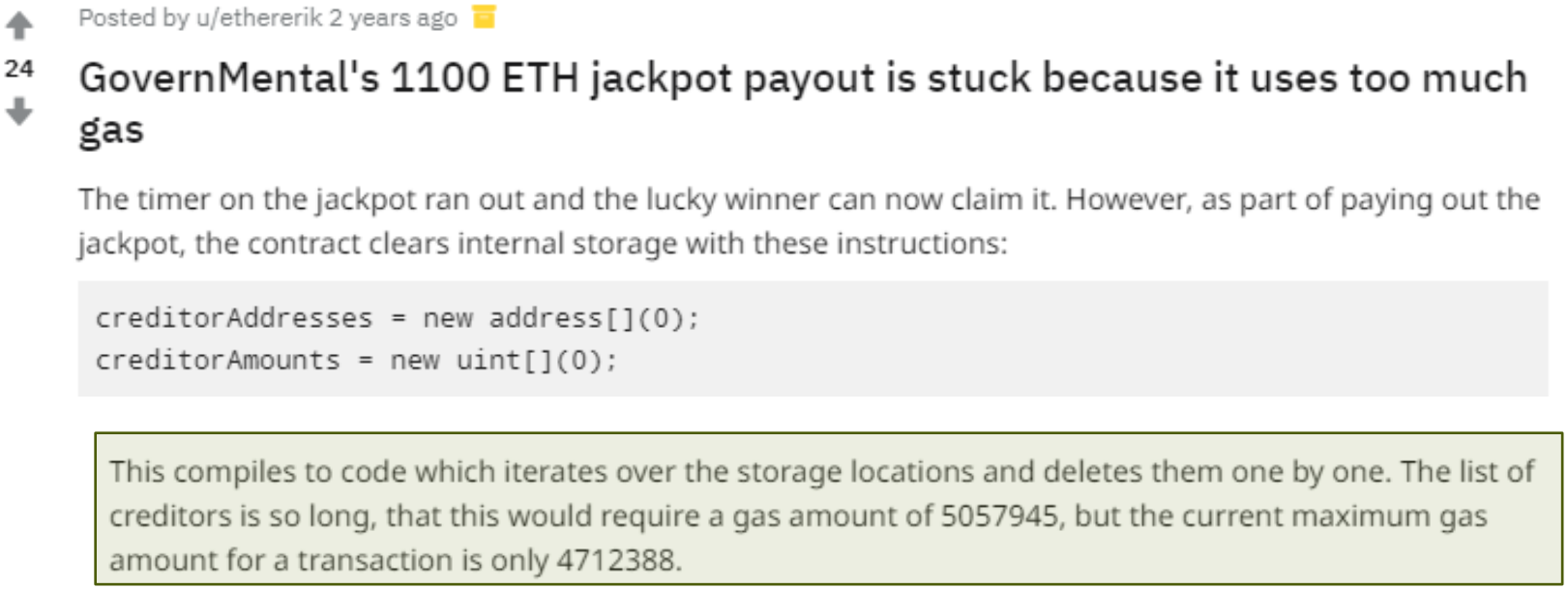
- a.k.a. **contract bricking**
  - Includes **gas limit reached**, **unexpected throw/kill, contract takeover, contract destruction**
- Examples
  - Trick contract into consuming arbitrary gas (to cause sender to fail)
  - Degrade smart contracts with data attacks
- **Loss**: estimated at 514,874 ETH (~300M USD at the time)

# Walkthrough scenario

- An **auction contract** allows users to bid on different assets.
- User calls **bid(uint object)** with the desired amount of ether.
  - Contract stores the ether in **escrow** until the object's owner accepts the bid or the bidder cancels it.
  - Contract holds full value of highest bid until auction ends
- Contract contains **withdraw(uint amount)** to allow admins to retrieve funds from the contract.
  - Function sends the **amount** to a hardcoded admin address
  - Developers have accidentally left the function public.
- An **attacker** calls **withdraw()** continuously, directing all the **contract**'s funds to its admins.
  - Blocks auctions from completing since contract has no ETH to pay object owner (until admins return escrowed ETH back to contract)
  - Fix is temporary as the **attacker** can continue to call **withdraw()**

# Example #1

- Smart contract unable to execute due to lack of gas.
  - GovernMental http://governmental.gihub.io/GovernMental



Posted by u/ethererik 2 years ago

24

**GovernMental's 1100 ETH jackpot payout is stuck because it uses too much gas**

The timer on the jackpot ran out and the lucky winner can now claim it. However, as part of paying out the jackpot, the contract clears internal storage with these instructions:

```
creditorAddresses = new address[](0);
creditorAmounts = new uint[](0);
```

This compiles to code which iterates over the storage locations and deletes them one by one. The list of creditors is so long, that this would require a gas amount of 5057945, but the current maximum gas amount for a transaction is only 4712388.

# Example #2 (Parity)

- Access control vulnerability leading to denial of service
  - Missing modifier leads to $30M in ETH lost (9/12/2017)


```
function initWallet(address[] _owners, uint _required, uint _daylimit) {
        initMultiowned(_owners, _required);
        initDaylimit(_daylimit) ;
}
```

???

Someone forgot to add the word 'internal' to this function, and that's what caused the 1st Parity hack of $30,000,000 to occur.

# Example #3 (Parity again!)

- Adversary compromising a smart contract library disabling all other contracts that depend upon it (11/6/2017)


  - Vulnerability in the "library" smart contract code, deployed as a shared component of all Parity multi-sig wallets.
    - Adversary made himself or herself the "owner" of the library contract, then subsequently self-destructed the component.
    - Parity multi-signature wallets depending on this component blocked from functioning
    - Funds in 587 wallets holding a total amount of 513,774.16 Ether as well as additional tokens. (~$200M in ETH)

# Code vulnerability example #1

- Pay to become president, price doubles on every transfer
- But, previous president can be a smart contract
  - Call to `transfer()` will execute its fallback function with 2300 gas (assumes a trivial fallback function!
  - Fallback function can be an infinite loop that can never succeed (e.g. caller will always run out of gas)
  - Failure of call allows previous, malicious president to remain forever

```
function becomePresident() payable {
  require(msg.value >= price);

  // Pay the previous president
  president.transfer(price);

  // Crown the new president
  president = msg.sender;

  // Double the price to become president
  price = price * 2;
}
```

# Code vulnerability example #2

- Vulnerable function iterates through all playerIDs to select a winner to reward

```
function selectNextWinners(uint256 _largestWinner)
{
  for(uint256 i = 0; i < _largestWinner, i++) {
    // some code
    playerIDs[i] ...
  }
}
```

- If adversary can inject a playerID that is extremely large, maximum gas level can effectively prevent for loop from ever being able to complete