Write to Arbitrary Storage Location

Description

A smart contract's data (e.g., storing the owner of the contract) is persistently stored at some storage location (i.e., a key or address) on the EVM level. The contract is responsible for ensuring that only authorized user or contract accounts may write to sensitive storage locations. If an attacker is able to write to arbitrary storage locations of a contract, the authorization checks may easily be circumvented. This can allow an attacker to corrupt the storage; for instance, by overwriting a field that stores the address of the contract owner.

Remediation

As a general advice, given that all data structures share the same storage (address) space, one should make sure that writes to one data structure cannot inadvertently overwrite entries of another data structure.

Example:

Code:

```
pragma solidity ^0.4.25;

contract Wallet {
    uint[] private bonusCodes;
    address private owner;

    constructor() public {
        bonusCodes = new uint[](0);
        owner = msg.sender;
    }

    function () public payable {
    }

    function PushBonusCode(uint c) public {
        bonusCodes.push(c);
    }

    function PopBonusCode() public {
        require(0 <= bonusCodes.length);
        bonusCodes.length--;
    }

    function UpdateBonusCodeAt(uint idx, uint c) public {
        require(idx < bonusCodes.length);
        bonusCodes[idx] = c;
    }

    function Destroy() public {
        require(msg.sender == owner);
        selfdestruct(msg.sender);
    }
}
```

Explanation:
In the example, any user may update the array's value without authorization in the shared storage slot and it is enabling the user to modify the other value as wants.

Occasionally, the dynamic array length value will be vulnerable to underflow or overflow, allowing another user to manipulate the other value in the shared storage slot.

Reference:

https://github.com/Arachnid/uscc/tree/master/submissions-2017/doughoyte