**Contract\_1.sol**

// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0;

contract Bank1{

address payable owner;

modifier onlyowner {

require(msg.sender==owner);

\_;

}

function setOwner()

public

{

owner = msg.sender;

}

function withdraw() public onlyowner

{

owner.transfer(address(this).balance);

}

}

**Issue:**  setOwner function should be called with onlyOwner modifier, as the owner should be changed by msg.sender

**Severity:** High

**Recommendation:**  Modifier should be added with setOwner function

**Contract\_2.sol**

// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0;

contract Bank{

mapping (address => uint) userBalance;

function getBalance(address u) public view returns(uint){

return userBalance[u];

}

function addToBalance() external payable {

userBalance[msg.sender] += msg.value;

}

function withdrawBalance() public{

(bool success, ) = msg.sender.call{value:userBalance[msg.sender]}("");

require(success);

userBalance[msg.sender] = 0;

}

}

**Issue:** At the withdrawBalance function, the caller's code is executed, and can call withdrawBalance againcausing the reentrancy attack

**Severity:** Critical

**Recommendation:** To prevent first the internal functionalities should be performed before making a call to the external function.

**Contract\_3.sol**

// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0;

contract Suicidal {

address payable owner;

function suicide() public returns (address) {

require(owner == msg.sender);

selfdestruct(owner);

}

}

contract C is Suicidal {

constructor() public {

owner = msg.sender;

}

}

**Issue:** Warning: This declaration shadows a builtin symbol. Got this warning on compiling for the suicide function

**Severity:** low

**Recommendation:** Changing the function name from suicide to suicides would solve the issue.

### Contract\_4.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0;

contract Auction {

address payable currentFrontrunner;

uint currentBid;

function bid() public payable {

require(msg.value > currentBid);

if (currentFrontrunner != address(0)) {

require(currentFrontrunner.send(currentBid));

}

currentFrontrunner = msg.sender;

currentBid = msg.value;

}

}

**Issue:**  This contract is having the issue of giving the control to calling contracts by sending the ether back to them. If a malicious contract calls the auction contract then it could create an issue.

**Severity:** Critical

**Recommendation:**  We can use balance withdrawal method to solve this issue and not sending the ether to other contracts