■ Spoon Audit Security Report

Contract Path: contracts
Analysis Date: 2025-08-06 06:46:46
Contracts: 2
Static Findings: 11
AI Findings: 10

Contract Summary

VulnerableContract: 11 functions, 2 events, 2 modifiers
VulnerableContract: 11 functions, 2 events, 2 modifiers

■ Static Analysis Findings

1. arbitrary-send-eth
Severity: HIGH
Tool: slither
Location: contracts/SafeContract.sol:46
Description: VulnerableContract.emergencyWithdraw() (contracts/SafeContract.sol#46-49) sends eth to arbitrary user Dangerous calls: - address(tx.origin).transfer(address(this).balance) (contracts/SafeContract.sol#48)

2. reentrancy-eth
Severity: HIGH
Tool: slither
Location: contracts/SafeContract.sol:33
Description: Reentrancy in VulnerableContract.withdraw(uint256) (contracts/SafeContract.sol#33-43): External calls: - (success) = msg.sender.call{value: amount}() (contracts/SafeContract.sol#37) State variables written after the call(s): - balances[msg.sender] -= amount (contracts/SafeContract.sol#40) VulnerableContract.balances (contracts/SafeContract.sol#9) can be used in cross function reentrancies: - VulnerableContract.balances (contracts/SafeContract.sol#9) - VulnerableContract.deposit() (contracts/SafeContract.sol#62-65) - VulnerableContract.receive() (contracts/SafeContract.sol#94-97) - VulnerableContract.secureWithdraw(uint256) (contracts/SafeContract.sol#81-91) - VulnerableContract.withdraw(uint256) (contracts/SafeContract.sol#33-43)

3. tx-origin
Severity: MEDIUM
Tool: slither
Location: contracts/SafeContract.sol:46
Description: VulnerableContract.emergencyWithdraw() (contracts/SafeContract.sol#46-49) uses tx.origin for authorization: require(bool,string)(tx.origin == owner,Not authorized) (contracts/SafeContract.sol#47)

4. unchecked-lowlevel
Severity: MEDIUM

Tool: slither
Location: contracts/SafeContract.sol:57
Description: VulnerableContract.unsafeTransfer(address,uint256) (contracts/SafeContract.sol#57-59) ignores return value by to.call{value: amount}() (contracts/SafeContract.sol#58)

5. missing-zero-check
Severity: LOW
Tool: slither
Location: contracts/SafeContract.sol:57
Description: VulnerableContract.unsafeTransfer(address,uint256).to (contracts/SafeContract.sol#57) lacks a zero-check on : - to.call{value: amount}() (contracts/SafeContract.sol#58)

6. missing-zero-check
Severity: LOW
Tool: slither
Location: contracts/SafeContract.sol:75
Description: VulnerableContract.changeOwner(address).newOwner (contracts/SafeContract.sol#75) lacks a zero-check on : - owner = newOwner (contracts/SafeContract.sol#77)

7. calls-loop
Severity: LOW
Tool: slither
Location: contracts/SafeContract.sol:68
Description: VulnerableContract.massTransfer(address[],uint256) (contracts/SafeContract.sol#68-72) has external calls inside a loop: address(recipients[i]).transfer(amount) (contracts/SafeContract.sol#70)

8. reentrancy-events
Severity: LOW
Tool: slither
Location: contracts/SafeContract.sol:81
Description: Reentrancy in VulnerableContract.secureWithdraw(uint256) (contracts/SafeContract.sol#81-91): External calls: - (success) = msg.sender.call{value: amount}() (contracts/SafeContract.sol#87) Event emitted after the call(s): - Withdrawal(msg.sender,amount) (contracts/SafeContract.sol#90)

9. reentrancy-events
Severity: LOW
Tool: slither
Location: contracts/SafeContract.sol:33
Description: Reentrancy in VulnerableContract.withdraw(uint256) (contracts/SafeContract.sol#33-43): External calls: - (success) = msg.sender.call{value: amount}() (contracts/SafeContract.sol#37) Event emitted after the call(s): - Withdrawal(msg.sender,amount) (contracts/SafeContract.sol#42)

10. timestamp
Severity: LOW
Tool: slither
Location: contracts/SafeContract.sol:52
Description: VulnerableContract.timeLock() (contracts/SafeContract.sol#52-54) uses timestamp for comparisons Dangerous comparisons: - block.timestamp > 1234567890 (contracts/SafeContract.sol#53)

11. solc-version
Severity: INFORMATIONAL
Tool: slither
Location: contracts/SafeContract.sol:2
Description: Pragma version^0.8.19 (contracts/SafeContract.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.8.18.

■ AI Analysis Findings

1. Reentrancy vulnerability in withdraw function
Severity: HIGH
Confidence: 1.0
Location: VulnerableContract:withdraw
Description: The withdraw function makes external calls before updating state, allowing for reentrancy attacks.
Reasoning: External call to msg.sender.call{value: amount}() occurs before balance update, which can lead to reentrancy attack.
Suggested Fix: Use checks-effects-interactions pattern: update state before external calls.

2. Improper use of tx.origin for authorization
Severity: HIGH
Confidence: 1.0
Location: VulnerableContract:emergencyWithdraw
Description: The emergencyWithdraw function uses tx.origin for authorization, which can be manipulated by a malicious contract.
Reasoning: tx.origin refers to the original sender of the transaction, not the current function caller, which can be manipulated by a malicious contract.
Suggested Fix: Replace tx.origin with msg.sender for proper authorization.

3. Unchecked external call in unsafeTransfer function
Severity: MEDIUM
Confidence: 1.0
Location: VulnerableContract:unsafeTransfer
Description: The unsafeTransfer function makes an external call without checking the return value.
Reasoning: External call to to.call{value: amount}() is not checked for success or failure, which can lead to loss of funds.
Suggested Fix: Always check the return value of external calls.

4. Block timestamp dependency in timeLock function
Severity: MEDIUM
Confidence: 0.8
Location: VulnerableContract:timeLock
Description: The timeLock function depends on block.timestamp, which can be manipulated by miners.
Reasoning: block.timestamp can be manipulated by miners to a certain degree, which can lead to unexpected behavior.
Suggested Fix: Avoid using block.timestamp for critical logic.

5. Potential integer overflow in deposit function
Severity: LOW
Confidence: 0.5
Location: VulnerableContract:deposit
Description: The deposit function does not check for integer overflow, although Solidity 0.8+ has built-in checks.
Reasoning: Integer overflow can lead to unexpected behavior, although Solidity 0.8+ has built-in

checks.
Suggested Fix: Always check for integer overflow and underflow.

6. Reentrancy vulnerability in withdraw function
Severity: HIGH
Confidence: 1.0
Location: VulnerableContract:withdraw
Description: The withdraw function makes external calls before updating state, allowing for reentrancy attacks.
Reasoning: External call to msg.sender.call{value: amount}() occurs before balance update.
Suggested Fix: Use checks-effects-interactions pattern: update state before external calls.

7. Use of tx.origin for authorization
Severity: HIGH
Confidence: 1.0
Location: VulnerableContract:emergencyWithdraw
Description: The emergencyWithdraw function uses tx.origin for authorization which can be manipulated by malicious contracts.
Reasoning: tx.origin refers to the original sender of the transaction and can be manipulated by malicious contracts.
Suggested Fix: Replace tx.origin with msg.sender for authorization.

8. Block timestamp dependency
Severity: MEDIUM
Confidence: 0.8
Location: VulnerableContract:timeLock
Description: The timeLock function depends on block.timestamp which can be manipulated by miners.
Reasoning: block.timestamp can be manipulated by miners to a certain degree.
Suggested Fix: Avoid using block.timestamp for critical logic.

9. Unchecked external call
Severity: MEDIUM
Confidence: 1.0
Location: VulnerableContract:unsafeTransfer
Description: The unsafeTransfer function makes an external call without checking the return value.
Reasoning: External calls can fail silently if the return value is not checked.
Suggested Fix: Always check the return value of external calls.

10. Potential integer overflow in deposit function
Severity: LOW
Confidence: 0.5
Location: VulnerableContract:deposit
Description: The deposit function could potentially have an integer overflow issue, although Solidity 0.8+ has built-in overflow checks.
Reasoning: Integer overflow can occur when a number is incremented beyond the maximum value that can be stored in its data type.
Suggested Fix: Use SafeMath library or similar to prevent integer overflow.