■■ HexSentinel

Smart Contract Security Analysis Report

■ EXECUTIVE SUMMARY

Report Generated	2025-07-05 20:35:54
Analysis Target	YieldFarmVault Smart Contract
Vulnerabilities Found	3
Raw Threat Score	328.00 points
Normalized Score	500.00/500
Overall Threat Level	■ CRITICAL

Vulnerability Type	Impact	Confidence	Threat Score
reentrancy-eth	High	Medium	288.00
solc-version	Informational	High	20.00
low-level-calls	Informational	High	20.00

■ DETAILED VULNERABILITY ANALYSIS

VULNERABILITY #1: REENTRANCY ETH

Detector ID	reentrancy-eth
Impact Level	High
Confidence Level	Medium
Threat Score	288.00 points
Scoring Breakdown	40.0 × 0.8 × 9 = 288.00
Affected Contract	YieldFarmVault
Affected Function	claimRewards()
File Location	contracts/YieldFarmVault.sol
Line Numbers	Lines 25-34

■ Severity Explanation:
This vulnerability is classified as High impact with Medium confidence because it allows recursive calls that can drain contract funds.

■ Technical Analysis:

REENTRANCY VULNERABILITY ANALYSIS:
Reentrancy in YieldFarmVault.claimRewards() (contracts/YieldFarmVault.sol#25-34):
External calls:
- (sent,None) = msg.sender.call{value: reward}() (contracts/YieldFarmVault.sol#30)
State variables written after the call(s):
- stakeBalance[msg.sender] = 0 (contracts/YieldFarmVault.sol#33)
YieldFarmVault.stakeBalance (contracts/YieldFarmVault.sol#10) can be used in cross function reentrancies:
- YieldFarmVault.claimRewards() (contracts/YieldFarmVault.sol#25-34)
- YieldFarmVault.stake() (contracts/YieldFarmVault.sol#19-22)
- YieldFarmVault.stakeBalance (contracts/YieldFarmVault.sol#10)

VULNERABILITY PATTERN IDENTIFIED:
- Function: claimRewards()
- External calls made before state updates
- State variables written after external calls
- This creates a window for recursive attacks

■ Code Analysis:
PROBLEMATIC CODE SEQUENCE:
1. External call on line 30: (sent,None) = msg.sender.call{value: reward}()
2. State update on line 33: stakeBalance[msg.sender] = 0

The state update happens AFTER the external call, creating reentrancy vulnerability.

■ Affected Code Locations:

Line(s)	Code	Type
30	(sent,None) = msg.sender.call{value: reward}()	external_calls
33	stakeBalance[msg.sender] = 0	variables_written

■ Attack Scenario:

ATTACK SCENARIO:
1. Attacker deploys malicious contract with fallback function
2. Attacker calls claimRewards() with legitimate stake

3. Contract calculates reward and makes external call
4. Attacker's fallback function calls claimRewards() again
5. Since state hasn't been updated, reward is calculated again
6. Process repeats until contract balance is drained

■■ Remediation:

IMMEDIATE FIXES REQUIRED:
1. Move state updates BEFORE external calls (Checks-Effects-Interactions pattern)
2. Add reentrancy guard using OpenZeppelin's ReentrancyGuard
3. Use pull payment pattern instead of push payments
4. Consider using transfer() instead of call() for simple Ether transfers

VULNERABILITY #2: SOLC VERSION

Detector ID	solc-version
Impact Level	Informational
Confidence Level	High
Threat Score	20.00 points
Scoring Breakdown	5.0 × 1.0 × 9 = 20.00

■ Severity Explanation:
Classified as Informational because it affects code reliability but doesn't directly create exploitable vulnerabilities.

■ Technical Analysis:
COMPILER ISSUE ANALYSIS:
Version constraint ^0.8.20 contains known severe issues
(https://solidity.readthedocs.io/en/latest/bugs.html)
- VerbatimInvalidDeduplication
- FullInlinerNonExpressionSplitArgumentEvaluationOrder
- MissingSideEffectsOnSelectorAccess.
It is used by:
- ^0.8.20 (contracts/YieldFarmVault.sol#2)

■ Code Analysis:
The pragma directive allows problematic compiler versions that contain known bugs.

■ Attack Scenario:
Not directly exploitable, but compiler bugs may create unexpected behavior that could be exploited.

■■ Remediation:
Upgrade to Solidity 0.8.21+ and pin to specific version without caret (^).

VULNERABILITY #3: LOW LEVEL CALLS

Detector ID	low-level-calls
Impact Level	Informational
Confidence Level	High
Threat Score	20.00 points
Scoring Breakdown	5.0 × 1.0 × 9 = 20.00
Affected Contract	YieldFarmVault
Affected Function	claimRewards()
File Location	contracts/YieldFarmVault.sol
Line Numbers	Lines 25-34

■ Severity Explanation:
Classified as Informational - not directly exploitable but enables other vulnerabilities.

■ Technical Analysis:
LOW-LEVEL CALL ANALYSIS:
Low level call in YieldFarmVault.claimRewards() (contracts/YieldFarmVault.sol#25-34):
- (sent,None) = msg.sender.call{value: reward}() (contracts/YieldFarmVault.sol#30)

■ Code Analysis:
Low-level call identified:
This bypasses Solidity safety checks and enables reentrancy.

■ Affected Code Locations:

Line(s)	Code	Type
30	(sent,None) = msg.sender.call{value: reward}()	general

■ Attack Scenario:
Enables reentrancy attacks by allowing unlimited gas forwarding to external contracts.

■■ Remediation:
Use transfer() for simple Ether transfers or implement proper gas limits and reentrancy protection.

■ THREAT SCORING METHODOLOGY

HexSentinel Dynamic Scoring System:

Base Severity Weights:
• High Impact: 40.0 points
• Medium Impact: 20.0 points
• Low Impact: 10.0 points
• Informational: 5.0 points

Confidence Multipliers:
• High Confidence: 1.0x (no reduction)
• Medium Confidence: 0.8x
• Low Confidence: 0.6x

Risk Amplification:
• High/Medium severity: 3x prevalence × 3x exploitability = 9x multiplier
• Low/Informational: 2x prevalence × 2x exploitability = 4x multiplier

Final Calculation:
Score = (Base Severity × Confidence Multiplier) × Risk Amplification

Your Contract Scores:
• reentrancy-eth: 288.00 points (High/Medium)
• solc-version: 20.00 points (Informational/High)
• low-level-calls: 20.00 points (Informational/High)

Total Raw Score: 328.00 points
Normalized Score: 500.00/500
Final Threat Level: ■ CRITICAL

■ PRIORITIZED RECOMMENDATIONS

■ CRITICAL PRIORITY (Immediate Action Required):
• Fix reentrancy-eth vulnerability in claimRewards()

■■ MEDIUM PRIORITY (Best Practices):
• Resolve solc-version for improved code quality
• Resolve low-level-calls for improved code quality

■ GENERAL RECOMMENDATIONS:
• Implement comprehensive unit and integration tests
• Consider professional security audit before mainnet deployment
• Establish continuous security monitoring
• Follow secure development best practices
• Regular dependency and compiler updates