> security_review · codehawks_first_flight

SNARKeling Treasure Hunt

Smart Contract & ZK Circuit Security Review — Full Audit Report

PROTOCOL
SNARKeling Treasure Hunt

PLATFORM
CodeHawks First Flight

DATE
May 2026

NSLOC
~220

RANKING
8th Place

HIGH	MEDIUM	LOW	INFO	GAS	TOTAL
2	1	5	2	3	13

AUDITOR
Prameya Singh Soni
blackburn

// Executive Summary

This report presents the findings of a security review conducted on the SNARKeling Treasure Hunt protocol — a real-world snorkeling treasure hunt with on-chain ETH reward claiming on an EVM blockchain. Participants find hidden physical treasures and submit a zero-knowledge proof to claim ETH rewards without revealing the treasure secret.

The audit covered the Solidity smart contract (TreasureHunt.sol), the Noir ZK circuit (main.nr), and the deploy script (Deploy.s.sol). A total of **13 findings** were identified. **Two independent HIGH severity vulnerabilities** each allow a single attacker to drain the entire 100 ETH prize pool — one via proof replay (broken claimed mapping key) and one via exposed secrets in the public deploy script.

All HIGH and MEDIUM findings were verified with Foundry PoC tests using a mock verifier. It is strongly recommended to resolve DS-1 and L-1 before deploying this protocol with real funds.

// Scope & Methodology

FILE	DESCRIPTION
contracts/src/TreasureHunt.sol	Main reward contract (Solidity 0.8.27)
circuits/src/main.nr	Noir ZK circuit
contracts/scripts/Deploy.s.sol	Deployment script

The audit followed an 8-pass structured methodology: function mapping → access control → state invariants → external interactions → economic attack simulation → edge cases → known vulnerability sweep → full attacker simulation (steal / lock / break).

// Findings Summary

ID	TITLE	SEVERITY	FILE
DS-1	Secret values exposed in public deploy script — ZK security model broken	HIGH	Deploy.s.sol

L-1	Wrong mapping key in replay guard — proof replay drains 100 ETH	HIGH	TreasureHunt.sol
AC-1	withdraw() missing onlyOwner — anyone can trigger withdrawal	MEDIUM	TreasureHunt.sol
EV-1	updateVerifier() missing zero-address check — can brick claims	LOW	TreasureHunt.sol
ST-2	Claimed event emits msg.sender instead of recipient	LOW	TreasureHunt.sol
C-1	Duplicate hash in ALLOWED_TREASURE_HASHES — only 9 unique treasures	LOW	main.nr
CEI-1	CEI violation in claim() — verifier called before state updates	LOW	TreasureHunt.sol
DS-2	Deploy script hash[8] differs from circuit — confirms C-1 is a bug	LOW	Deploy.s.sol
AC-2	receive() accepts ETH from anyone, contradicts design	INFO	TreasureHunt.sol
AC-3	Auth check order in updateVerifier() / emergencyWithdraw()	INFO	TreasureHunt.sol
G-2	9 custom errors defined but never thrown — require strings used instead	GAS	TreasureHunt.sol
G-3	onlyOwner modifier declared but never applied	GAS	TreasureHunt.sol
G-7	Redundant paused = false in constructor wastes deployment gas	GAS	TreasureHunt.sol

// Detailed Findings

DS-1

Secret Treasure Values Hardcoded in Public Deploy Script — ZK Security Model Broken

HIGH

The protocol's security model relies entirely on treasure secrets being unknown to participants. A valid ZK proof can only be generated by someone who knows the secret preimage of an allowed treasure hash. However, all 10 secret values are listed in plaintext in the public deploy script comments, alongside their corresponding hashes.

```
// contracts/scripts/Deploy.s.sol
// Secret Treasures for the snorkeling hunt (not revealed to the public):
//      1, 2, 3, 4, 5, 6, 7, 8, 9, 10   // ← exposed in public repo
// Treasures' hashes (revealed to the public):
//     [all 10 hashes listed alongside their secrets]
```

Impact: Anyone can read secrets 1–10, generate valid ZK proofs for all 10 treasures, and claim all 100 ETH without physical participation. The ZK proof system provides zero protection — the secret knowledge it is meant to protect is public.

Recommendation: Secrets must be stored in a secure off-chain vault and must never be committed to version control. Remove all secret values from the deploy script immediately.

FILE	LIKELIHOOD	IMPACT
Deploy.s.sol	High	Critical

L-1

Wrong Mapping Key in Replay Guard — Same Proof Drains Full 100 ETH Contract

HIGH

The claim() function's replay guard checks claimed[_treasureHash] where _treasureHash is a private immutable that is never assigned in the constructor, making it permanently bytes32(0). The actual claim recording uses claimed[treasureHash] (the function parameter). Since these are different mapping keys, the guard never fires and the same proof can be submitted up to MAX_TREASURES times.

```
bytes32 private immutable _treasureHash; // ← never assigned → always
bytes32(0)

function claim(..., bytes32 treasureHash, ...) external {
    if (claimed[_treasureHash]) revert AlreadyClaimed(...); // ← always
checks claimed[0x00]
    // ...
    _markClaimed(treasureHash); // ← correctly marks claimed[treasureHash],
never read by guard
}
```

Proof of Concept: With a single valid proof, an attacker can loop MAX_TREASURES times, each call passing the replay check (claimed[bytes32(0)] is never set) and receiving 10 ETH, draining 100 ETH total. claimsCount hits the maximum, permanently locking all other legitimate finders.

Recommendation:

```
- if (claimed[_treasureHash]) revert AlreadyClaimed(treasureHash);
+ if (claimed[treasureHash]) revert AlreadyClaimed(treasureHash);
```

Also remove the unused _treasureHash immutable entirely.

FILE	LIKELIHOOD	IMPACT
TreasureHunt.sol	High	Critical

AC-1

withdraw() Missing onlyOwner — Anyone Can Forcibly Trigger Owner's Withdrawal

MEDIUM

withdraw() is intended for the owner to reclaim funds after the hunt ends. However, the function has no access control — any address can call it once claimsCount == MAX_TREASURES. Funds always transfer to the hardcoded owner address so theft is impossible, but the owner loses all control over withdrawal timing.

```
// ← no onlyOwner modifier — callable by anyone
function withdraw() external {
    require(claimsCount == MAX_TREASURES, "HUNT_NOT_OVER");
    (bool sent, ) = owner.call{value: balance}(""); // funds go to owner
regardless
}
```

Recommendation: Add onlyOwner modifier to withdraw().

FILE	LIKELIHOOD	IMPACT
TreasureHunt.sol	High	Low

EV-1

updateVerifier() Missing Zero-Address Check — Owner Can Permanently Brick Claims

LOW

The constructor correctly guards against a zero-address verifier using InvalidVerifier. However, updateVerifier() has no equivalent check, allowing address(0) to be set accidentally or maliciously. Once set, all future claim() calls revert, permanently locking participants out.

Recommendation: Add if (address(newVerifier) == address(0)) revert InvalidVerifier(); to updateVerifier().

FILE	LIKELIHOOD	IMPACT
TreasureHunt.sol	Low	Medium

ST-2

Claimed Event Emits msg.sender Instead of recipient — Wrong Beneficiary Logged

LOW

The Claimed event logs msg.sender (the proof submitter) instead of recipient (the address that receives the ETH). Since the contract explicitly enforces recipient != msg.sender, these are always different addresses. Every emitted event is incorrect.

```
(bool sent, ) = recipient.call{value: REWARD}(""); // ETH goes to recipient
emit Claimed(treasureHash, msg.sender);             // ← but event logs msg.sender
```

Recommendation: Change to emit Claimed(treasureHash, recipient);

FILE	LIKELIHOOD	IMPACT
TreasureHunt.sol	High	Low

C-1

Duplicate Hash in ALLOWED_TREASURE_HASHES — Only 9 Unique Treasures Exist

LOW

The circuit defines 10 allowed treasure hashes matching MAX_TREASURES = 10. However, indices 8 and 9 contain identical values — only 9 unique physical secrets exist. This is confirmed by the deploy script which lists a distinct hash at index 8 that was incorrectly overwritten in the circuit. One 10 ETH reward slot is permanently unreachable by a unique finder.

```
// circuits/src/main.nr
global ALLOWED_TREASURE_HASHES: [Field; 10] = [
    // ...
    // indices 8 and 9 are identical — copy-paste error

-961435057317293580094826482786572873533235701183329831124091847635547871092
,

-961435057317293580094826482786572873533235701183329831124091847635547871092
];
```

Recommendation: Replace index 8 with the correct unique value from the deploy script.

FILE	LIKELIHOOD	IMPACT
circuits/src/main.nr	High	Low

CEI-1

claim() Violates CEI Pattern — Verifier Called Before State Updates

LOW

claim() follows a C-I-E-I pattern instead of the correct C-E-I. The external call to verifier.verify() occurs before claimsCount and claimed are updated. The nonReentrant modifier currently prevents exploitation, but it is load-bearing for two independent reasons simultaneously — reentrancy and the broken replay guard from L-1.

```
bool ok = verifier.verify(proof, publicInputs); // Interaction before
Effects
if (!ok) revert InvalidProof();
_incrementClaimsCount(); // Effects come after
_markClaimed(treasureHash);
```

Recommendation: Move state updates immediately after checks, before any external call.

FILE	LIKELIHOOD	IMPACT
TreasureHunt.sol	Low	Low

DS-2

Deploy Script hash[8] Differs From Circuit — Independently Confirms C-1 Is a Bug

LOW

The deploy script lists a distinct, unique value at index 8 that does not match the duplicated value in the circuit's ALLOWED_TREASURE_HASHES[8]. This cross-file discrepancy independently confirms that C-1 is an unintentional copy-paste error rather than a deliberate design choice. The deploy script is the ground truth for the intended unique hash values.

FILE	IMPACT
Deploy.s.sol	Low

AC-2

receive() Accepts ETH From Anyone, Contradicting OnlyOwnerCanFund Design

INFO

fund() restricts ETH deposits to the owner. receive() has no such guard, allowing any address to send ETH directly. This contradicts the OnlyOwnerCanFund error defined in the contract and the stated design intent.

FILE	IMPACT
TreasureHunt.sol	Informational

AC-3

Auth Check Order in updateVerifier() / emergencyWithdraw() Leaks State

INFO

Both functions check paused before msg.sender == owner. Convention is auth first, business logic second. The current order allows non-owners to trigger the paused revert, unnecessarily leaking whether the contract is paused.

FILE	IMPACT
TreasureHunt.sol	Informational

G-2

9 Custom Errors Defined but Never Thrown — require Strings Used Instead

GAS

The contract defines 9 custom errors but all admin functions use require() with string literals instead. Custom errors save ~50 gas per revert. The errors are already written — simply not used.

```
// Current (costly)
require(claimsCount == MAX_TREASURES, "HUNT_NOT_OVER");
// Recommended
if (claimsCount < MAX_TREASURES) revert HuntNotOver();
```

FILE	IMPACT
TreasureHunt.sol	Gas

G-3

onlyOwner Modifier Declared but Never Applied

GAS

The onlyOwner modifier is defined but never used. All 5 admin functions manually re-implement the same owner check inline with require, wasting deployment gas on dead bytecode and creating duplicated logic.

FILE	IMPACT
TreasureHunt.sol	Gas

G-7

Redundant paused = false in Constructor Wastes Deployment Gas

GAS

paused = false in the constructor is redundant — bool defaults to false in Solidity. This triggers an unnecessary SSTORE (~20,000 gas) during deployment for no effect.

Recommendation: Remove the redundant paused = false; line from the constructor.

FILE	IMPACT
TreasureHunt.sol	Gas