SNARKeling Treasure Hunt Audit Report

Khant Wai Yan Aung (SnavOhBurmaa)

16, 4, 2026

SNARKeling Treasure Hunt Audit Report

Prepared by: Khant Wai Yan Aung (SnavOhBurmaa) (https://www.linkedin.com/in/khant-wai-yan-ohburmaa/)

Table of contents

See table

About Me

Hi, I'm Khant Wai Yan Aung (SnavOhBurmaa). I'm currently focusing on smart contract security audit. Currently studying at Rangsit University(CS).

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

Audit Details

The findings described in this document correspond the following commit hash:

`https://github.com/CodeHawks-Contests/2026-04-snarkeling`

Scope

```
contracts/
├── scripts/
│   └── Deploy.s.sol
└── src/
    └── TreasureHunt.sol
```

```
circuits/
└── src/
    └── main.nr
```

Issues found

Severity	Count
High	2
Medium	2
Low	7
Informational	6
Gas	2
Total	**19**

Findings

High

[H-1] Wrong variable in "already claimed" check drains entire prize pool with one proof

Location: `contracts/src/TreasureHunt.sol:35, 88, 104`

Description: `TreasureHunt.claim` checks `claimed[_treasureHash]` instead of `claimed[treasureHash]`. `_treasureHash` is an `immutable` that is never initialized, so its value is always `bytes32(0)`.

```
    bytes32 private immutable _treasureHash;   // never set
    ...
@>  if (claimed[_treasureHash]) revert
        AlreadyClaimed(treasureHash);
    ...
    _markClaimed(treasureHash);                // correct mapping
        key written, but never checked
```

Impact: One valid proof can be replayed up to `MAX_TREASURES = 10` times, sending 10 ETH per call to the same recipient. The full **100 ETH** pool is drained by a single finder.

PoC: `test_PoC_H1_DrainContractWithOneProof` in `contracts/test/PoC.t.sol` drains the contract:

```
Contract balance before attack: 100 ETH
Contract balance after attack:  0
ETH stolen:                     100 ETH
```

Recommended Mitigation:

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

Also delete the unused `_treasureHash` immutable.

[H-2] Duplicate hash in `ALLOWED_TREASURE_HASHES` bricks one treasure

Location: `circuits/src/main.nr:64-65` (duplicate), `circuits/src/tests.nr:30` (workaround)

Description: `ALLOWED_TREASURE_HASHES[8]` (`main.nr:64`) and `ALLOWED_TREASURE_HASHES[9]` (`main.nr:65`) are the same value:

```
-961435057317293580094826482786572873533235701183329831124091847635
-961435057317293580094826482786572873533235701183329831124091847635
```

`Prover.toml.example` and `Deploy.s.sol` both expect index 8 to be `-4417726...`. The circuit's test file (`tests.nr:30`) works around this by using treasure `10` twice instead of `[9, 10]`.

Impact: The physical treasure whose real hash is `-4417726...` (treasure #9) cannot be redeemed. The finder of that treasure has no way to generate a proof the circuit will accept.

Recommended Mitigation: Replace index 8 with the correct hash and regenerate the verifier via `./build.sh`. Restore `tests.nr` to use `[1..10]`.

Medium

[M-1] `Claimed` event emits `msg.sender` instead of `recipient`

Location: `contracts/src/TreasureHunt.sol:44` (declaration), `contracts/src/TreasureHunt.sol:111` (wrong emission)

Description: The event is declared with a `recipient` parameter, but the code passes `msg.sender`:

```
    event Claimed(bytes32 indexed treasureHash, address indexed
        recipient);
    ...
    emit Claimed(treasureHash, msg.sender);  // wrong
```

Because M-2 forbids `recipient == msg.sender`, the two addresses are always different, so every `Claimed` event records a different address from the one that received the ETH.

Impact: Block explorers, subgraphs, and off-chain accounting see the submitter as the recipient. Future airdrops or analytics built on these events will reward the wrong addresses.

PoC: `test_PoC_M1_ClaimedEventEmitsWrongRecipient` in `contracts/test/PoC.t.sol`.

Recommended Mitigation:

```
-   emit Claimed(treasureHash, msg.sender);
+   emit Claimed(treasureHash, recipient);
```

Or add a third field to also track the submitter:

```
-   event Claimed(bytes32 indexed treasureHash, address indexed
        recipient);
+   event Claimed(bytes32 indexed treasureHash, address indexed
        recipient, address indexed submitter);
    ...
-   emit Claimed(treasureHash, msg.sender);
+   emit Claimed(treasureHash, recipient, msg.sender);
```

[M-2] `recipient == msg.sender` check adds no security but hurts UX

Location: `contracts/src/TreasureHunt.sol:86`

Description: `claim` rejects `recipient == msg.sender`. The proof-to-recipient binding is already provided by the public input, so this extra check does not prevent any attack. It only forces finders to use a second EOA or a relayer in order to pay themselves.

Recommended Mitigation: Remove the `recipient == msg.sender` clause.

```
-   if (recipient == address(0) || recipient == address(this) ||
        recipient == owner || recipient == msg.sender) revert
        InvalidRecipient();
+   if (recipient == address(0) || recipient == address(this) ||
        recipient == owner) revert InvalidRecipient();
```

Low

[L-1] Circuit public input `recipient` is unused, producing a warning

Location: `circuits/src/main.nr:28`

`nargo check` emits `warning: unused variable recipient`. Recipient binding today works because bb includes all `pub` inputs in the public-input vector, but a future Noir optimization could drop the unused input. Add an explicit no-op constraint:

```
+   assert(recipient == recipient);
```

[L-2] Custom errors declared but string reverts used everywhere

Location: `contracts/src/TreasureHunt.sol:7-25` (errors); string reverts at lines 54, 60, 108, 224, 227, 229, 237-238, 246, 255, 264-265, 274-277, 280

`TreasureHunt.sol` declares many custom errors (`OnlyOwnerCanFund`, `HuntNotOver`, `InvalidAmount`, ...) that are never used. String `require` messages are used instead, increasing gas and deploy size. Either delete the unused errors or switch all reverts to use them.

[L-3] `onlyOwner` modifier declared but never applied

Location: `contracts/src/TreasureHunt.sol:53-56` (dead modifier); inline duplicates at lines 237, 246, 255, 265, 275

The `onlyOwner` modifier exists but every admin function re-checks `msg.sender == owner` inline. Apply the modifier to `fund` (236), `pause` (245), `unpause` (254), `updateVerifier` (263), `emergencyWithdraw` (273).

[L-4] `receive()` bypasses `fund()`'s owner-only check

Location: `contracts/src/TreasureHunt.sol:236-241` (fund) vs `contracts/src/TreasureHunt.sol:287-289` (receive)

`fund()` is guarded by `ONLY_OWNER_CAN_FUND`, but `receive()` is public:

```
    function fund() external payable {
        require(msg.sender==owner, "ONLY_OWNER_CAN_FUND");
        ...
    }

    receive() external payable {
        emit Funded(msg.value, address(this).balance);
    }
```

Anyone can deposit ETH by sending it directly to the contract address. The owner restriction on `fund()` is meaningless. PoC: `test_PoC_L4_ReceiveBypassesOnlyOwnerFund`. Either restrict `receive()` with the same owner check, or delete `receive()`.

[L-5] `updateVerifier` accepts `address(0)`

Location: `contracts/src/TreasureHunt.sol:263-269`

The constructor (`TreasureHunt.sol:68`) rejects a zero verifier address; `updateVerifier` does not. Owner operator error can brick every future claim until they push a new update. PoC: `test_PoC_L5_UpdateVerifierAcceptsZeroAddress`. Add `if (address(newVerifier) == address(0)) revert InvalidVerifier();`.

[L-6] `emergencyWithdraw` forbids `recipient == owner`

Location: `contracts/src/TreasureHunt.sol:276`

```
require(recipient != address(0) && recipient != address(this) &&
        recipient != owner, "INVALID_RECIPIENT");
```

Owner cannot recover funds to their own wallet in an emergency. Remove the `recipient != owner` clause.

[L-7] `Funded` event omits sender

Location: `contracts/src/TreasureHunt.sol:45` (declaration); emits at lines 240 and 288

`event Funded(uint256 amount, uint256 newBalance)` has no `from` field. Because L-4 lets anyone deposit, downstream accounting cannot tell owner top-ups from stranger deposits. Add an indexed `from` field.

Info

[I-1] `withdraw()` has no caller restriction

Location: `contracts/src/TreasureHunt.sol:223-232`

`withdraw()` can be called by any address after `claimsCount >= MAX_TREASURES`. Funds always go to `owner` so no theft is possible, but the function should still be `onlyOwner` for clarity.

[I-2] Broad owner trust

Location: `contracts/src/TreasureHunt.sol:263-283`

Owner can replace the verifier with any contract while paused (line 263), then call `emergencyWithdraw` (line 273). A malicious or compromised owner can drain the contract. Use a multisig/timelock in production.

[I-3] Constructor does not require funding

Location: `contracts/src/TreasureHunt.sol:67-75`

The constructor accepts any `msg.value`, including zero. Deployments with less than `REWARD * MAX_TREASURES` break silently at first claim. Add `require(msg.value >= REWARD * MAX_TREASURES, ...)`.

[I-4] `pause()` / `unpause()` are not idempotence-guarded

Location: `contracts/src/TreasureHunt.sol:245-259`

Calling `pause()` while already paused (or `unpause()` while already unpaused) silently succeeds and emits a duplicate event. Add `require(!paused, ...)` / `require(paused, ...)`.

[I-5] `Nargo.toml` compiler version too loose

Location: `circuits/Nargo.toml:5`

`compiler_version = ">=1.0.0"` accepts any future major version. The README pins `1.0.0-beta.19`, and the generated Verifier/proof is version-sensitive. Narrow to `">=1.0.0-beta.19, <1.1.0"` to prevent accidental upgrades.

[I-6] Deploy script does not validate `INITIAL_FUNDING`

Location: `contracts/scripts/Deploy.s.sol:45-67`

`Deploy.s.sol` reads `INITIAL_FUNDING` from env and deploys with whatever value it gets. No assertion that `INITIAL_FUNDING >= REWARD * MAX_TREASURES`. A misconfigured deployment under-funds the hunt.

Gas

[G-1] Switch string `require` messages to custom errors

Location: See L-2 for full list of affected lines.

Custom errors are already declared and are cheaper than string reverts.

[G-2] Inline one-line internal functions

Location: `contracts/src/TreasureHunt.sol:209-215` (defs); called once each from `claim` at lines 103-104.

`_markClaimed` and `_incrementClaimsCount` each have a single caller and one line. Inline them in `claim` for clarity and a small gas save.