Aztec Soulbound

Security Review

Cantina Managed review by:
Desmond Ho, Lead Security Researcher
Arno, Associate Security Researcher

December 15, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Aztec Labs was founded in 2017, and has a team of +50 leading zero-knowledge cryptographers, engineers, and business experts. Aztec Labs is developing its namesake products: AZTEC (a privacy-first L2 on Ethereum) and NOIR (the universal ZK language).

From Aug 14th to Aug 16th the Cantina team conducted a review of ignition-monorepo on commit hash abfbcc19. The team identified a total of **5** issues:

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	1	1	0
Medium Risk	0	0	0
Low Risk	0	0	0
Gas Optimizations	0	0	0
Informational	4	4	0
Total	**5**	**5**	**0**

2.1 Scope

The security review had the following components in scope for ignition-monorepo on commit hash abfbcc19:

```
└── src
    ├── constants.sol
    ├── sale
    │   ├── GenesisSequencerSale.sol
    │   └── IGenesisSequencerSale.sol
    └── soulbound
        ├── IgnitionParticipantSoulbound.sol
        ├── IIgnitionParticipantSoulbound.sol
        └── providers
            ├── AttestationProvider.sol
            ├── IWhitelistProvider.sol
            ├── PredicateProvider.sol
            └── ZKPassportProvider.sol
```

3 Findings

3.1 High Risk

3.1.1 Missing On-Chain Enforcement of Passport Validity, Expiry, and Sanctions Exclusion Checks in ZKPassport Provider

Severity: High Risk

Context: ZKPassportProvider.sol#L89

Description: The ZKPassport provider lacks direct implementation of critical sanctions-related checks (valid passport, non-expired status, and exclusion from sanctions lists via nationality or name/passport number) within the smart contract.Leaving enforcement reliant on frontend or external assumptions, which violates the requirement for on-chain validation to prevent bypassing.

> - Users can privately prove that they are not on a sanctions list by performing a zk-passport sanctions check which ensures they:
>   - Have a valid passport.
>   - The passport has not expired.
>   - The name / passport number do not appear on a sanctions list.

> In the review these checks are implemented as `providers`. We are assuming that their dependencies will work as intended, such that invalid proofs of zkpassport can not be produced, and that its onchain verifier is implemented correctly. We can also assume that predicate will not issue fraudulent attestations.

> The frontend that purchasers will use to purchase tokens in the sale need to IP geo-block sanctions countries, and the participants need to comply with sanctions check requirements. It is not enough for these checks to be performed solely on the frontend, they must also be enforced within the sale smart contracts too. This is a key requirement.

As we can see, these checks are not implemented by the provider itself (pseudo code) reference:

```
// New: Check expiry date (inspired by example's attribute checks)

(uint256 currentDate, uint256 minDate, uint256 maxDate) =
↪   zkPassportVerifier.getDateProofInputs(

    params.committedInputs, params.committedInputCounts, ProofType.EXPIRY_DATE

);

require(maxDate > block.timestamp || maxDate == 0, "Document expired");  // Adjust logic
↪   based on circuit (0 for no upper bound)

// New: Check age (adapted from example's checkAge)

(uint256 ageCurrentDate, uint8 minAge, uint8 maxAge) =
↪   zkPassportVerifier.getAgeProofInputs(

    params.committedInputs, params.committedInputCounts

);

require(minAge >= 18, "User under minimum age");  // Example: Enforce 18+; make
↪   configurable

// New: Get nationality (adapted from example's getNationality using disclose data)

(bytes memory discloseMask, bytes memory discloseBytes) =
↪   zkPassportVerifier.getDiscloseProofInputs(
    params.committedInputs, params.committedInputCounts

);

(, , string memory nationality, , , , , ) =
↪  zkPassportVerifier.getDisclosedData(discloseBytes, isIDCard);

// New: Check nationality exclusion (adapted from example)

string[] memory excludedCountries = zkPassportVerifier.getCountryProofInputs(

    params.committedInputs, params.committedInputCounts, ProofType.NATIONALITY_EXCLUSION

);

bool isExcluded = false;

for (uint256 i = 0; i < excludedCountries.length; i++) {

    if (keccak256(bytes(excludedCountries[i])) == keccak256(bytes(nationality))) {

        isExcluded = true;

        break;

    }

}

require(!isExcluded, "Nationality excluded");
```

Impact Explanation: Without these checks enforced on-chain, malicious or non-compliant users could bypass frontend restrictions to participate in token sales or other activities, leading to potential sanctions violations, legal liabilities, and regulatory scrutiny for the protocol.

Recommendation: Integrate the suggested pseudo-code checks directly into the provider's `verify` function to enforce passport validity, expiry, and sanctions exclusion on-chain.

Aztec Labs: Fixed in commit f61d54b6.

Cantina Managed: Fix verified.

3.2 Informational

3.2.1 Test Coverage

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description/Recommendation: Modifications to the tests and trees to achieve 100% coverage (excluding the `tokenId` out of range checks). The more useful tests would be `test_FailsIfBeneficiaryAlreadyHasAnATP()` and the ones for `zkPassportProvider`.

```
  diff --git a/contracts/test/btt/sale/admin/withdrawEth.t.sol
  ↪  b/contracts/test/btt/sale/admin/withdrawEth.t.sol
  index 8210823..5bf0a02 100644
-  -- a/contracts/test/btt/sale/admin/withdrawEth.t.sol
+  ++ b/contracts/test/btt/sale/admin/withdrawEth.t.sol
  @@ -41,6 +41,14 @@ contract WithdrawEth is GenesisSequencerSaleBase {
           assertEq(address(_to).balance, toBalanceBefore + genesisSequencerSaleBalance);
       }

+      function test_WhenTheRecipientCannotReceiveETH() external {
+          uint256 genesisSequencerSaleBalance = address(genesisSequencerSale).balance;
+          // it reverts
+           vm.prank(FOUNDATION_ADDRESS);
+           vm.expectRevert(abi.encodeWithSelector(IGenesisSequencerSale.GenesisSequencer
↪  Sale__ETHTransferFailed.selector));
+           genesisSequencerSale.withdrawETH(address(saleToken),
↪  genesisSequencerSaleBalance);
+       }
+
        function test_sendToFoundationAddress(uint128 _amount) external {
            // it withdraws ETH from the contract
            // it emits a {ETHWithdrawn} event
  diff --git a/contracts/test/btt/sale/admin/withdrawEth.tree
   ↪  b/contracts/test/btt/sale/admin/withdrawEth.tree
  index 9470193..0354690 100644
-  -- a/contracts/test/btt/sale/admin/withdrawEth.tree
+  ++ b/contracts/test/btt/sale/admin/withdrawEth.tree
  @@ -2,5 +2,6 @@ WithdrawEth
   ├── when caller of withdrawEth is not owner
   │   └── it reverts
   └── when caller of withdrawEth is owner
-       ├── it withdraws ETH from the contract
+       ├── when the recipient cannot receive ETH
+       │   └── it reverts with {ETHTransferFailed()}
        └── it emits a {ETHWithdrawn} event
  \ No newline at end of file
  diff --git a/contracts/test/btt/sale/purchase/soulboundInteraction.t.sol
   ↪  b/contracts/test/btt/sale/purchase/soulboundInteraction.t.sol
  index e49a177..c3a9142 100644
-  -- a/contracts/test/btt/sale/purchase/soulboundInteraction.t.sol
+  ++ b/contracts/test/btt/sale/purchase/soulboundInteraction.t.sol
  @@ -3,6 +3,7 @@ pragma solidity ^0.8.28;

   import {GenesisSequencerSaleBase} from "../GenesisSequencerSaleBase.t.sol";
   import {IGenesisSequencerSale} from "src/sale/GenesisSequencerSale.sol";
+  import {IIgnitionParticipantSoulbound} from
↪  "src/soulbound/IIgnitionParticipantSoulbound.sol";

   contract SoulboundInteraction is GenesisSequencerSaleBase {
       function setUp() public override {
  @@ -22,4 +23,20 @@ contract SoulboundInteraction is GenesisSequencerSaleBase {
            vm.prank(_user);
            genesisSequencerSale.purchase{value: amount}(_beneficiary);
        }
+
+       function test_WhenTheCallerIsNotTheTokenSaleContract(address _user) external {
+           assumeAddress(_user);
+           vm.assume(_user != address(genesisSequencerSale));
+           // It reverts with {IgnitionParticipantSoulbound__CallerIsNotTokenSale()}
+           vm.prank(_user);
+           vm.expectRevert(IIgnitionParticipantSoulbound.IgnitionParticipantSoulbound__C
↪  allerIsNotTokenSale.selector);
+           soulboundToken.mintFromSale(
+               _user,
+               _user,
+               new bytes32[](0),
+               address(0),
+               "",
+               ""
+           );
+       }
   }
  diff --git a/contracts/test/btt/sale/purchase/soulboundInteraction.tree
   ↪  b/contracts/test/btt/sale/purchase/soulboundInteraction.tree
  index c425f13..2f8bf78 100644
-  -- a/contracts/test/btt/sale/purchase/soulboundInteraction.tree
+  ++ b/contracts/test/btt/sale/purchase/soulboundInteraction.tree
  @@ -1,3 +1,5 @@
```

```
   SoulboundInteraction
-  └── When the user does not have a soulbound token
-      └── It reverts with {NoSoulboundToken()}
 \ No newline at end of file
+  ├── When the user does not have a soulbound token
+  │   └── It reverts with {NoSoulboundToken()}
+  └── When the caller is not the token sale contract
+      └── It reverts with {CallerIsNotTokenSale()}
 \ No newline at end of file
 diff --git a/contracts/test/btt/soulbound/admin/adminBatchMint.t.sol
  ↪  b/contracts/test/btt/soulbound/admin/adminBatchMint.t.sol
 index 8f9b5fe..783b633 100644
-  -- a/contracts/test/btt/soulbound/admin/adminBatchMint.t.sol
+  ++ b/contracts/test/btt/soulbound/admin/adminBatchMint.t.sol
 @@ -11,7 +11,7 @@ contract AdminBatchMintTest is SoulboundBase {
      }

      /// forge-config: default.fuzz.runs = 10
-     function test_whenTheCallerIsNotTheAdmin(address _caller, address[] calldata _to,
↪  uint8[] calldata _tokenId)
+     function test_RevertWhen_TheCallerIsNotTheAdmin(address _caller, address[]
↪  calldata _to, uint8[] calldata _tokenId)
          public
      {
          vm.assume(_caller != address(this));
 @@ -34,24 +34,6 @@ contract AdminBatchMintTest is SoulboundBase {
          }
      }

-     /// forge-config: default.fuzz.runs = 10
-     function test_whenTheTokenIdIsOutOfRange(address[] calldata _to, uint8[] memory
↪  _tokenId) public {
-         vm.assume(_to.length == _tokenId.length);
-         vm.assume(_to.length > 0);
-
-         for (uint256 i = 0; i < _to.length; i++) {
-             _tokenId[i] =
-                 uint8(bound(_tokenId[i],
↪  uint8(IIgnitionParticipantSoulbound.TokenId.GENERAL) + 1, type(uint8).max));
-         }
-
-         // Use call in order to bypass solidity enum range check
-         address(soulboundToken).call(abi.encodeWithSelector(soulboundToken.adminBatch⌟
↪  Mint.selector, _to, _tokenId));
-
-         for (uint256 i = 0; i < _to.length; i++) {
-             assertEq(soulboundToken.balanceOf(_to[i], uint256(_tokenId[i])), 0);
-         }
-     }
-
      function assertUniqueAddresses(address[] memory _addresses) public pure returns
       ↪  (bool) {
          for (uint256 i = 0; i < _addresses.length; i++) {
              for (uint256 j = i + 1; j < _addresses.length; j++) {
 @@ -64,7 +46,21 @@ contract AdminBatchMintTest is SoulboundBase {
      }

      /// forge-config: default.fuzz.runs = 10
-     function test_whenTheTokenIdIsInRangeAndTheCallerIsTheAdmin(address[] memory _to,
↪  uint8[] calldata _tokenId)
+     function test_RevertWhen_TheTokenIdsAndAddressesAreOfDifferentLengths(address[]
↪  memory _to, uint8[] calldata _tokenId) external {
+         vm.assume(_to.length != _tokenId.length);
+         IIgnitionParticipantSoulbound.TokenId[] memory tokenIds =
+             new IIgnitionParticipantSoulbound.TokenId[](_tokenId.length);
+         for (uint256 i = 0; i < _tokenId.length; i++) {
```

```
+               tokenIds[i] = IIgnitionParticipantSoulbound.TokenId(
+                   bound(_tokenId[i], uint8(0),
↪   uint8(IIgnitionParticipantSoulbound.TokenId.GENERAL))
+               );
+           }
+           vm.expectRevert(abi.encodeWithSelector(IIgnitionParticipantSoulbound.Ignition⌟
↪   ParticipantSoulbound__InvalidInputLength.selector));
+           soulboundToken.adminBatchMint(_to, tokenIds);
+       }
+
+       /// forge-config: default.fuzz.runs = 10
+       function test_WhenTheTokenIdIsInRangeAndTheCallerIsTheAdmin(address[] memory _to,
↪   uint8[] calldata _tokenId)
            public
        {
            vm.assume(_to.length == _tokenId.length);
  @@ -92,7 +88,7 @@ contract AdminBatchMintTest is SoulboundBase {
        }

        /// forge-config: default.fuzz.runs = 10
-       function test_whenTheAddressHasAlreadyMinted(address[] memory _to, uint8[] memory
↪   _tokenId) public {
+       function test_RevertWhen_TheAddressHasAlreadyMinted(address[] memory _to, uint8[]
↪   memory _tokenId) public {
            vm.assume(_to.length == _tokenId.length);
            vm.assume(_to.length > 0);
            vm.assume(assertUniqueAddresses(_to));
  diff --git a/contracts/test/btt/soulbound/admin/adminBatchMint.tree
    ↪   b/contracts/test/btt/soulbound/admin/adminBatchMint.tree
  index 6a0ce17..b15c5d7 100644
-   -- a/contracts/test/btt/soulbound/admin/adminBatchMint.tree
+   ++ b/contracts/test/btt/soulbound/admin/adminBatchMint.tree
  @@ -1,9 +1,9 @@
   AdminBatchMintTest
   ├── When the caller is not the Admin
   │   └── It should revert
-  ├── When the tokenId is out of range
+  ├── When the address has already minted
   │   └── It should revert
-  ├── If the address has already minted
+  ├── When the tokenIds and addresses are of different lengths
   │   └── It should revert
   └── When the tokenId is in range and the caller is the admin
       └── It should mint the tokens
  \ No newline at end of file
  diff --git a/contracts/test/btt/soulbound/admin/adminMint.t.sol
    ↪   b/contracts/test/btt/soulbound/admin/adminMint.t.sol
  index 46c10e3..0bd5891 100644
-   -- a/contracts/test/btt/soulbound/admin/adminMint.t.sol
+   ++ b/contracts/test/btt/soulbound/admin/adminMint.t.sol
  @@ -10,7 +10,8 @@ contract AdminMintTest is SoulboundBase {
            super.setUp();
        }

-       function test_whenTheCallerIsNotTheAdmin(address _caller, address _to, uint8
↪   _tokenId) public {
+       function test_RevertWhen_TheCallerIsNotTheAdmin(address _caller, address _to,
↪   uint8 _tokenId) public {
+           // It should revert
            vm.assume(_caller != address(this));
            vm.assume(_tokenId <= uint8(IIgnitionParticipantSoulbound.TokenId.GENERAL));

  @@ -23,18 +24,8 @@ contract AdminMintTest is SoulboundBase {
            assertEq(soulboundToken.balanceOf(_to, uint256(_tokenId)), 0);
        }
```

```
-      function test_whenTheTokenIdIsOutOfRange(address _to, uint256 _tokenId) public {
-          vm.assume(_tokenId > uint256(IIgnitionParticipantSoulbound.TokenId.GENERAL));
-
-          // Use call in order to bypass solidity enum range check
-          (bool _success,) =
-              address(soulboundToken).call(abi.encodeWithSelector(soulboundToken.adminM⌋
↪  int.selector, _to, _tokenId));
-          assertEq(_success, false);
-
-          assertEq(soulboundToken.balanceOf(_to, uint256(_tokenId)), 0);
-      }
-
-      function test_whenTheTokenIdIsInRangeAndTheCallerIsTheAdmin(address _to, uint256
↪  _tokenId) public {
+      function test_WhenTheTokenIdIsInRangeAndTheCallerIsTheAdmin(address _to, uint256
↪  _tokenId) public {
+          // It should mint the token
           vm.assume(_tokenId <= uint256(IIgnitionParticipantSoulbound.TokenId.GENERAL));

           IIgnitionParticipantSoulbound.TokenId tokenId =
            ↪  IIgnitionParticipantSoulbound.TokenId(_tokenId);
  @@ -44,7 +35,8 @@ contract AdminMintTest is SoulboundBase {
           assertEq(soulboundToken.balanceOf(_to, uint256(_tokenId)), 1);
       }

-      function test_whenTheAddressHasAlreadyMinted(address _to, uint256 _tokenId) public
↪  {
+      function test_RevertWhen_TheAddressHasAlreadyMinted(address _to, uint256 _tokenId)
↪  public {
+          // It should revert
           vm.assume(_tokenId <= uint256(IIgnitionParticipantSoulbound.TokenId.GENERAL));

           IIgnitionParticipantSoulbound.TokenId tokenId =
            ↪  IIgnitionParticipantSoulbound.TokenId(_tokenId);
  diff --git a/contracts/test/btt/soulbound/admin/adminMint.tree
   ↪  b/contracts/test/btt/soulbound/admin/adminMint.tree
  index da5b684..c381aca 100644
-  -- a/contracts/test/btt/soulbound/admin/adminMint.tree
+  ++ b/contracts/test/btt/soulbound/admin/adminMint.tree
  @@ -1,9 +1,7 @@
   AdminMintTest
   ├── When the caller is not the Admin
   │   └── It should revert
-  ├── When the tokenId is out of range
-  │   └── It should revert
-  ├── If the address has already minted
+  ├── When the address has already minted
   │   └── It should revert
   └── When the tokenId is in range and the caller is the admin
       └── It should mint the token
  \ No newline at end of file
  diff --git a/contracts/test/btt/soulbound/providers/predicate/verify/verify.tree
   ↪  b/contracts/test/btt/soulbound/providers/predicate/verify/verify.tree
  index 6126e32..1d091f1 100644
-  -- a/contracts/test/btt/soulbound/providers/predicate/verify/verify.tree
+  ++ b/contracts/test/btt/soulbound/providers/predicate/verify/verify.tree
  @@ -1,6 +1,6 @@
   PredicateVerify
   ├── when caller is not the consumer
-  │   └── it reverts with {AuthorizationFailed}
+  │   └── it reverts with {InvalidConsumer}
   └── when caller is the consumer
       ├── when predicate succeeds
       │   └── it returns true
  diff --git a/contracts/test/btt/soulbound/providers/zkpassport/verify/verify.t.sol
   ↪  b/contracts/test/btt/soulbound/providers/zkpassport/verify/verify.t.sol
  index 6a48b6e..0553d1b 100644
-  -- a/contracts/test/btt/soulbound/providers/zkpassport/verify/verify.t.sol
+  ++ b/contracts/test/btt/soulbound/providers/zkpassport/verify/verify.t.sol
  @@ -3,7 +3,14 @@ pragma solidity >=0.8.27;

   import {ZKPassportBase} from "../ZKPassportBase.sol";
   import {ProofVerificationParams} from "@zkpassport/ZKPassportVerifier.sol";
-  import {IZKPassportProvider} from "src/soulbound/providers/ZKPassportProvider.sol";
+  import {IWhitelistProvider, IZKPassportProvider} from
 ↪  "src/soulbound/providers/ZKPassportProvider.sol";
+  import {IVerifier} from "@zkpassport/OuterCount5.sol";
+
+  contract AlwaysReturnFalseVerifier is IVerifier {
+      function verify(bytes calldata _proof, bytes32[] calldata _publicInputs) external
 ↪  view returns (bool) {
+          return false;
+      }
+  }

   contract ZKPassportVerify is ZKPassportBase {
       address public constant BOUND_ADDRESS =
        ↪  0x04Fb06E8BF44eC60b6A99D2F98551172b2F2dED8;
  @@ -23,15 +30,72 @@ contract ZKPassportVerify is ZKPassportBase {
           _;
       }

-      function test_WhenTheBoundAddressIsValid() external givenAValidZkpassportProof {
-          // It verifies the proof
-          // It returns true
+      /// forge-config: default.fuzz.runs = 10
+      function test_WhenCallerIsNotTheConsumer(address _user) external {
+          // it reverts with {InvalidConsumer}
+          vm.assume(_user != address(zkPassportProvider.consumer()));
+          vm.prank(_user);
+          vm.expectRevert(abi.encodeWithSelector(IWhitelistProvider.WhitelistProvider__⌋
 ↪  InvalidConsumer.selector));
+          zkPassportProvider.verify(_user, abi.encode(proof));
+      }

-          // Must come from the consumer
+      /// forge-config: default.fuzz.runs = 10
+      function test_WhenDomainDoesntMatch(string calldata _domain) external
 ↪  givenAValidZkpassportProof {
+          // it reverts with {InvalidProof}
+          vm.assume(keccak256(bytes(_domain)) !=
 ↪  keccak256(bytes(zkPassportProvider.domain())));
+          proof.domain = _domain;
+          vm.prank(address(zkPassportProvider.consumer()));
+          vm.expectRevert(abi.encodeWithSelector(IZKPassportProvider.ZKPassportProvider⌋
 ↪  __InvalidProof.selector));
+          zkPassportProvider.verify(BOUND_ADDRESS, abi.encode(proof));
+      }
+
+      /// forge-config: default.fuzz.runs = 10
+      function test_WhenScopeDoesntMatch(string calldata _scope) external
 ↪  givenAValidZkpassportProof {
+          // it reverts with {InvalidProof}
+          vm.assume(keccak256(bytes(_scope)) !=
 ↪  keccak256(bytes(zkPassportProvider.scope())));
+          proof.scope = _scope;
+          vm.prank(address(zkPassportProvider.consumer()));
+          vm.expectRevert(abi.encodeWithSelector(IZKPassportProvider.ZKPassportProvider⌋
 ↪  __InvalidProof.selector));
+          zkPassportProvider.verify(BOUND_ADDRESS, abi.encode(proof));
+       }
+
+       function test_WhenDevModeIsTrue() external givenAnInvalidZkpassportProof {
+           // it reverts with {InvalidProof}
+           proof.devMode = true;
+           vm.prank(address(zkPassportProvider.consumer()));
+           vm.expectRevert(abi.encodeWithSelector(IZKPassportProvider.ZKPassportProvider⌋
↪   __InvalidProof.selector));
+           zkPassportProvider.verify(BOUND_ADDRESS, abi.encode(proof));
+       }
+
+       function test_WhenVerifierChecksTheProofAndReturnsFalse() external
↪   givenAValidZkpassportProof {
+           AlwaysReturnFalseVerifier falseVerifier = new AlwaysReturnFalseVerifier();
+
+           // Replace the verifier
+           bytes32[] memory vkeyHashes = new bytes32[](1);
+           vkeyHashes[0] = VKEY_HASH;
+
+           address[] memory verifiers = new address[](1);
+           verifiers[0] = address(falseVerifier);
+
+           zkPassportVerifier.addVerifiers(vkeyHashes, verifiers);
+
+           // it reverts with {InvalidProof}
+           vm.prank(address(zkPassportProvider.consumer()));
+           vm.expectRevert(abi.encodeWithSelector(IZKPassportProvider.ZKPassportProvider⌋
↪   __InvalidProof.selector));
+           zkPassportProvider.verify(BOUND_ADDRESS, abi.encode(proof));
+       }
+
+       /// forge-config: default.fuzz.runs = 10
+       function test_WhenTheChainIdDoesntMatch(uint64 _chainId) external
↪   givenAValidZkpassportProof {
+           // it reverts with {InvalidProof}
+           vm.assume(_chainId != block.chainid);
+           vm.chainId(_chainId);
            vm.prank(address(zkPassportProvider.consumer()));
+           vm.expectRevert(abi.encodeWithSelector(IZKPassportProvider.ZKPassportProvider⌋
↪   __InvalidProof.selector));
            zkPassportProvider.verify(BOUND_ADDRESS, abi.encode(proof));
        }

+       /// forge-config: default.fuzz.runs = 10
        function test_WhenTheBoundAddressIsInvalid(address _addr) external
         ↪  givenAValidZkpassportProof {
            // It reverts with {InvalidProof}
            vm.assume(_addr != BOUND_ADDRESS);
  @@ -41,11 +105,23 @@ contract ZKPassportVerify is ZKPassportBase {
            zkPassportProvider.verify(_addr, abi.encode(proof));
        }

-       function test_GivenAnInvalidZkpassportProof(address _addr) external
↪   givenAnInvalidZkpassportProof {
-           // It returns false
-
+       function test_ValidProof() external givenAValidZkpassportProof {
+           // it returns true
+           // Must come from the consumer
            vm.prank(address(zkPassportProvider.consumer()));
-           vm.expectRevert(abi.encodeWithSelector(IZKPassportProvider.ZKPassportProvider⌋
↪   __InvalidProof.selector));
-           zkPassportProvider.verify(_addr, abi.encode(proof));
+           zkPassportProvider.verify(BOUND_ADDRESS, abi.encode(proof));
+       }
+
+      function test_WhenTheProofIsAlreadyUsed() external givenAValidZkpassportProof {
+          // it reverts with {SybilDetected}
+          vm.startPrank(address(zkPassportProvider.consumer()));
+          zkPassportProvider.verify(BOUND_ADDRESS, abi.encode(proof));
+          vm.expectRevert(
+              abi.encodeWithSelector(
+                  IZKPassportProvider.ZKPassportProvider__SybilDetected.selector,
+                  proof.publicInputs[proof.publicInputs.length - 1]
+              )
+          );
+          zkPassportProvider.verify(BOUND_ADDRESS, abi.encode(proof));
       }
   }
  diff --git a/contracts/test/btt/soulbound/providers/zkpassport/verify/verify.tree
   ↪  b/contracts/test/btt/soulbound/providers/zkpassport/verify/verify.tree
  index 990d21b..37f2fbc 100644
-  -- a/contracts/test/btt/soulbound/providers/zkpassport/verify/verify.tree
+  ++ b/contracts/test/btt/soulbound/providers/zkpassport/verify/verify.tree
  @@ -1,9 +1,24 @@
   ZKPassportVerify
-  ├── given a valid zkpassport proof
-  │   ├── When the bound address is valid
-  │   │   ├── It verifies the proof
-  │   │   └── It returns true
-  │   └── When the bound address is invalid
-  │       └── It reverts with {InvalidProof}
-  └── given an invalid zkpassport proof
-      └── It returns false
  \ No newline at end of file
+  ├── when caller is not the consumer
+  │   └── it reverts with {InvalidConsumer}
+  └── when caller is the consumer
+      ├── given a valid zkpassport proof
+      │   ├── when the proof is already used
+      │   │   └── it reverts with {SybilDetected}
+      │   ├── when the proof is not used
+      │   │   ├── when the bound address is the user
+      │   │   │   └── when the chainId matches the on-chain chainId
+      │   │   │       └── it returns true
+      │   │   │   └── when the chainId doesn't match the on-chain chainId
+      │   │   │       └── it reverts with {InvalidProof}
+      │   │   └── when the bound address is not the user
+      │   │       └── it reverts with {InvalidProof}
+      └── given an invalid zkpassport proof
+          ├── when devMode is true
+          │   └── it reverts with {InvalidProof}
+          ├── when domain doesn't match
+          │   └── it reverts with {InvalidProof}
+          ├── when scope doesn't match
+          │   └── it reverts with {InvalidProof}
+          └── when verifier checks the proof and returns false
+              └── it reverts with {InvalidProof}
  \ No newline at end of file
  diff --git a/contracts/test/btt/soulbound/transfer.t.sol
   ↪  b/contracts/test/btt/soulbound/transfer.t.sol
  index 889a689..10e48f9 100644
-  -- a/contracts/test/btt/soulbound/transfer.t.sol
+  ++ b/contracts/test/btt/soulbound/transfer.t.sol
  @@ -14,6 +14,14 @@ contract TransferSoulboundToken is SoulboundBase, MerkleTreeGetters
   ↪  {
          initTrees();
      }

+      /// forge-config: default.fuzz.runs = 10
+      function test_RevertWhen_SetApprovalForAllIsCalled(address _user) external {
+          assumeAddress(_user);
+          vm.prank(_user);
+          vm.expectRevert(IIgnitionParticipantSoulbound.IgnitionParticipantSoulbound__T⌋
↪  okenIsSoulbound.selector);
+          soulboundToken.setApprovalForAll(address(1), true);
+      }
+
       /// forge-config: default.fuzz.runs = 10
       function test_shouldNotTransferSoulboundToken(uint8 _userIndex, address
        ↪  _beneficiary, address _to, uint8 _tokenId)
           public
  diff --git a/contracts/test/integration/staking-aligned-participant/StakingAlignedPart⌋
   ↪  icipant.t.sol
   ↪  b/contracts/test/integration/staking-aligned-participant/StakingAlignedParticipant⌋
   ↪  .t.sol
  index 7531243..1ad8bbf 100644
-  -- a/contracts/test/integration/staking-aligned-participant/StakingAlignedParticipant⌋
↪  .t.sol
+  ++ b/contracts/test/integration/staking-aligned-participant/StakingAlignedParticipant⌋
↪  .t.sol
  @@ -29,6 +29,8 @@ import {StakingRegistry} from
   ↪  "src/staking-registry/StakingRegistry.sol";
   // External contracts - Splits
   import {PullSplitFactory} from "@splits/splitters/pull/PullSplitFactory.sol";

+  import {Errors} from "@oz/utils/Errors.sol";
+
   import {ProofVerificationParams} from "@zkpassport/ZKPassportVerifier.sol";

   import {IntegrationTestBase} from "./IntegrationTestBase.sol";
  @@ -376,4 +378,26 @@ contract StakingAlignedParticipantTest is IntegrationTestBase {

           // TODO: Grant rewards to be claimed by the participants
       }
+
+      function test_FailsIfBeneficiaryAlreadyHasAnATP() public {
+          // 1. Admin mints a soulbound token to 2 participants
+          address participant2 = makeAddr("participant2");
+          address beneficiary = makeAddr("beneficiary");
+          soulboundToken.adminMint(participant,
↪  IIgnitionParticipantSoulbound.TokenId.GENESIS_SEQUENCER);
+          soulboundToken.adminMint(participant2,
↪  IIgnitionParticipantSoulbound.TokenId.GENESIS_SEQUENCER);
+
+          vm.deal(participant, 100 ether); // Participant is rich
+          vm.deal(participant2, 100 ether); // Participant 2 is rich
+
+          // Set the sale to be active
+          vm.warp(block.timestamp + 1 days + 1);
+          genesisSequencerSale.startSale();
+
+          uint256 purchaseCostInEth = genesisSequencerSale.getPurchaseCostInEth();
+          vm.prank(participant);
+          genesisSequencerSale.purchase{value: purchaseCostInEth}(beneficiary);
+          vm.prank(participant2);
+          vm.expectRevert(Errors.FailedDeployment.selector);
+          genesisSequencerSale.purchase{value: purchaseCostInEth}(beneficiary);
+      }
   }
```

3.2.2 Redundancies

Severity: Informational

Context: GenesisSequencerSale.sol#L106-L113, IGenesisSequencerSale.sol#L27, IGenesisSequencerSale.sol#L31, IGenesisSequencerSale.sol#L39-L42, IgnitionParticipantSoulbound.sol#L363, IgnitionParticipantSoulbound.sol#L418

Description:

- IGenesisSequencerSale.sol#L39-L42, GenesisSequencerSale.sol#L106-L113: Redundant `receive()` function, as the expected ETH inflow is from the `purchase*()` functions.
- IgnitionParticipantSoulbound.sol#L363, IgnitionParticipantSoulbound.sol#L418: Redundant checks on `tokenId` range as the compiler handles illegal enum values at runtime, and is implicitly conducted. Note: `test_RevertWhen_TheTokenIdIsOutOfRange()` doesn't bypass this check, the tests revert even in the absence of the require statement.
- IGenesisSequencerSale.sol#L27, IGenesisSequencerSale.sol#L31): Unused errors `GenesisSequencerSale__InsufficientTokensRemaining` & `GenesisSequencerSale__TokenTransferFailed`.

Recommendation: Remove the referenced lines.

Aztec Labs: Fixed in commit 833c1b3a.

Cantina Managed: Fix verified.

3.2.3 Minor Recommendations

Severity: Informational

Context: GenesisSequencerSale.sol#L66, IgnitionParticipantSoulbound.sol#L392, IgnitionParticipantSoulbound.sol#L403

Description/Recommendation:

- GenesisSequencerSale.sol#L66-L72: Missing natspec for `_saleToken` & `_milestoneId`.
- IgnitionParticipantSoulbound.sol#L392:

```
- nonReentract
+ nonReentrant
```

- IgnitionParticipantSoulbound.sol#L403: Replace `_beneficiary` with `_soulboundRecipient` for clarity, to avoid mixing up with the ATP beneficiary.

```
- // Perform sanctions check on the _beneficiary address
+ // Perform sanctions check on the _soulboundRecipient address
```

Aztec Labs: Fixed in commit 971a777c.

Cantina Managed: Fix verified.

3.2.4 Misordered Minting in `_internalMint`

Severity: Informational

Context: IgnitionParticipantSoulbound.sol#L393

Description: In `_internalMint`, the `_mint` call happens before the final `IWhitelistProvider.verify` sanctions check, breaking the checks-effects-interactions pattern. Even with nonReentrant protection, this risks state inconsistencies if reentrancy sneaks in via hooks or callbacks during minting. No immediate exploit, but it's a potential issue for extensions or if guards change.

Recommendation: Move `_mint` to the end, after all checks. Updated code:

```
function _internalMint(
    address _identityAddress,
    TokenId _tokenId,
    address _soulboundRecipient,
    bytes32[] calldata _merkleProof,
    address _identityProvider,
    bytes calldata _identityData,
    bytes calldata _soulboundRecipientScreeningData
) internal {
    // TokenID must be between 0 and 2 (inclusive)
    require(_tokenId <= TokenId.GENERAL,
     ↪  IgnitionParticipantSoulbound__InvalidTokenId(_tokenId));

    require(!hasMinted[_identityAddress], IgnitionParticipantSoulbound__AlreadyMinted());
    hasMinted[_identityAddress] = true;

    // Verify through identity provider
    require(identityProviders[_identityProvider],
     ↪  IgnitionParticipantSoulbound__InvalidAuth(_identityProvider));

    if (_tokenId == TokenId.GENESIS_SEQUENCER) {
        // Verify merkle proof for genesis sequencer whitelist
        require(genesisSequencerMerkleRoot != bytes32(0),
         ↪  IgnitionParticipantSoulbound__NoMerkleRootSet());

        bytes32 leaf = keccak256(bytes.concat(keccak256(abi.encode(_identityAddress))));
        require(
            MerkleProof.verify(_merkleProof, genesisSequencerMerkleRoot, leaf),
            IgnitionParticipantSoulbound__MerkleProofInvalid()
        );
    } else if (_tokenId == TokenId.CONTRIBUTOR) {
        // Verify merkle proof for contributor whitelist
        require(contributorMerkleRoot != bytes32(0),
         ↪  IgnitionParticipantSoulbound__NoMerkleRootSet());

        bytes32 leaf = keccak256(bytes.concat(keccak256(abi.encode(_identityAddress))));
        require(
            MerkleProof.verify(_merkleProof, contributorMerkleRoot, leaf),
            IgnitionParticipantSoulbound__MerkleProofInvalid()
        );
    }
    // Further steps required for all cases


    // Ext call
    // Perform sanctions check on the msg.sender address
    require(
        IWhitelistProvider(_identityProvider).verify(_identityAddress, _identityData),
        IgnitionParticipantSoulbound__InvalidAuth(_identityProvider)
    );

    // Ext call
    // Perform sanctions check on the _beneficiary address
    require(
        IWhitelistProvider(addressScreeningProvider).verify(_soulboundRecipient,
         ↪  _soulboundRecipientScreeningData),
        IgnitionParticipantSoulbound__InvalidAuth(addressScreeningProvider)
    );
      // Ext call - with possible reentrancy on acceptance check - nonReentract added to
       ↪  prevent
    _mint(_soulboundRecipient, uint256(_tokenId), 1, "");


    emit IgnitionParticipantSoulboundMinted(_soulboundRecipient, _identityAddress,
     ↪  _tokenId);
}
```

Aztec Labs: Fixed in PR 71.

Cantina Managed: Fix verified.