// Private Smart Contract Security Assessment 06.24.2024 - 07.10.2024

Champions Tactics: Grimoria Chronicles

Ubisoft

HALBORN

Champions Tactics: Grimoria Chronicles - Ubisoft

Prepared by: HALBORN

Last Updated 08/22/2024

Date of Engagement by: June 24th, 2024 - July 10th, 2024

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
11	1	0	0	5	5

TABLE OF CONTENTS

8. Automated Testing

1. Introduction

Ubisoft, in collaboration with LayerZero engaged Halborn to conduct a security assessment on their smart contracts beginning on 2024-06-24 and ending on 2024-07-10. The security assessment was scoped to the smart contracts provided in directly by the team.

Commit hashes and further details can be found in the Scope section of this report.

2. Assessment Summary

The team at Halborn was provided two weeks and three days for the engagement and assigned one full-time security engineers to check the security of the smart contract. The security engineers are blockchain and smart-contract security experts with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified several security concerns that were mostly addressed by the Ubisoft team.

3. Test Approach And Methodology

Halborn performed a combination of manual review of the code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the smart contract assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the assessment:

- Research into the architecture, purpose, and use of the platform.
- Smart contract manual code review and walkthrough to identify any logic issue.
- Thorough assessment of safety and usage of critical Solidity variables and functions in scope that could led to arithmetic related vulnerabilities.
- Manual testing by custom scripts.
- Graphing out functionality and contract logic/connectivity/functions (solgraph).
- Static Analysis of security for scoped contract, and imported functions. (Slither).
- Local or public testnet deployment (Foundry, Remix IDE).

4. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

4.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

4.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

4.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4

SEVERITY	SCORE VALUE RANGE
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY

(a) Repository: -

(b) Assessed Commit ID: genesis

(c) Items in scope:

- src/abstracts/GenesisBaseV2.sol
- src/abstracts/GenesisUpgradeable.sol
- src/CrafterRules/GenesisCrafterRule.sol
- src/interfaces/IGenesisChampion.sol
- src/interfaces/IGenesisChampionFactory.sol
- src/interfaces/IGenesisCrafter.sol
- src/interfaces/IGenesisCrafterRule.sol
- src/interfaces/IGenesisMinter.sol
- src/interfaces/IGenesisRewardDistributor.sol
- src/librairies/Errors.sol
- src/types/CraftCounter.sol
- src/types/CraftData.sol
- src/types/GenesisChampionArgs.sol
- src/types/MintDataV2.sol
- src/types/RewardClaim.sol
- src/types/SeasonReward.sol
- src/types/SupplyConfig.sol
- src/GenesisChampion.sol
- src/GenesisChampionBridged.sol
- src/GenesisChampionFactory.sol
- src/GenesisCrafter.sol
- src/GenesisMinter.sol
- src/GenesisRewardDistributor.sol
- lib/authenticated-relay/src/AuthenticatedRelay.sol
- src/ERC721Psi/ERC721Psi.sol
- src/ERC721Psi/extensions/ERC721PsiAddressData.sol

Out-of-Scope:

REMEDIATION COMMIT ID:

- 81d3660
- 451a6fd
- c91ba15

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	0	0	5	5

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
LACK OF PERMISSION CHECKS ALLOW THEFT OF BRIDGED TOKENS	CRITICAL	SOLVED - 07/04/2024
POTENTIAL NONCES MISMATCH	LOW	RISK ACCEPTED
UNSAFE NFT MINTING FOR BRIDGED TOKEN	LOW	SOLVED - 07/04/2024
USE OF OWNABLE LIBRARY WITH SINGLE-STEP OWNERSHIP TRANSFER	LOW	RISK ACCEPTED
USE OF UNSAFE TRANSFERFROM FUNCTION	LOW	SOLVED - 07/04/2024
LACK OF STORAGE GAP IN UPGRADEABLE CONTRACTS	LOW	SOLVED - 07/05/2024

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
UNNECESSARY NATIVE CURRENCY TRANSFER	INFORMATIONAL	ACKNOWLEDGED
CONSIDER USING NAMED MAPPINGS	INFORMATIONAL	SOLVED - 07/04/2024
UPDATE OR REMOVE UNUSED COMMENTS TO IMPROVE CODE CLARITY	INFORMATIONAL	SOLVED - 07/04/2024
UNUSED IMPORTS	INFORMATIONAL	SOLVED - 07/05/2024
INEFFICIENT CONDITIONAL CHECKS	INFORMATIONAL	SOLVED - 07/10/2024

7. FINDINGS & TECH DETAILS

7.1 LACK OF PERMISSION CHECKS ALLOW THEFT OF BRIDGED TOKENS

// CRITICAL

Description

The `GenesisChampionBridged` implements logic allowing to represent a `GenesisChampion` tokens after bridging them through LayerZero. The NFT bridged from the main chain is locked within the `GenesisChampion` contract, and the same token ID is minted on the chosen destination chain. When the user intends to send the NFT to the other chain or back to the main chain, they need to call the `send` function.

```
54      function send(uint32 _dstEid, address _dstTo, uint256 _id, bytes call
55          external
56          payable
57          returns (MessagingReceipt memory receipt)
58      {
59          _burn(_id);
60          bytes memory _payload = abi.encode(_dstTo, _id);
61          receipt = _lzSend(_dstEid, _payload, _options, MessagingFee(msg.v
62      }
```

It burns the intended NFT and calls the `_lzSend` to continue the process of bridging. However, the internal `_burn` function defined in the ERC721 contract, does not check whether the caller is the owner or is approved to perform this operation on the specified NFT. This allows anyone to steal all the bridged NFTs minted by the `GenesisChampionBridge` contracts.

Proof of Concept

The following `Foundry` test was used in order to prove the aforementioned issue:

```
    function test_BridgedNFTTheft() public mintBefore {
        vm.deal(attacker, 1 ether);
        vm.startPrank(bob);

        uint256 targetToken = 3;

        //Estimate message gas fees via the quote function.
        bytes memory opts = OptionsBuilder.newOptions().addExecutorLzReceiveOp
        MessagingFee memory fee = champion.quote(eid2, bob, targetToken, opts,
```

```
        //Bridge NFT via the _lzSend() method.
        MessagingReceipt memory receipt = champion.send{value: fee.nativeFee}(

        //Deliver packet to GenesisChampionBridged manually.
        verifyPackets(eid2, addressToBytes32(address(champion2)));

        //User received its nft
        assertEq(champion2.ownerOf(targetToken), bob);
        vm.stopPrank();

        //Attacker initiate the briding process
        vm.startPrank(attacker);
        bytes memory opts2 = OptionsBuilder.newOptions().addExecutorLzReceive(
        MessagingFee memory fee2 = champion2.quote(eid1, bob, targetToken, opt
        MessagingReceipt memory receipt2 = champion2.send{value: fee2.nativeFe
        vm.stopPrank();

        // Fulfill the request
        verifyPackets(eid1, addressToBytes32(address(champion)));

        //Attacker has stolen the token
        assertEq(champion.ownerOf(targetToken), attacker);
    }
```

```
Ran 1 test for test/genesis-champion/bridge.t.sol:GenesisChampion_Bridge_Test
[PASS] test_BridgedNFTTheft() (gas: 1540131)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 13.03ms (3.54ms CPU time)

Ran 1 test suite in 148.23ms (13.03ms CPU time): 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:C/Y:N/R:N/S:C (10.0)

Recommendation

Implement a check in the send function to ensure only the token owner or an approved operator can initiate the bridging process.

Remediation Plan

SOLVED: The **Ubisoft team** solved the issue as recommended.

Remediation Hash

81d3660b332c4662f605ff97e7777c5b1107390f

7.2 POTENTIAL NONCES MISMATCH

// LOW

Description

The `relay` function from the `AuthenticatedRelay` contract is responsible for authorizing and forwarding transactions using EIP-712 signatures. The nonce parameter from the `RelayData` struct received as an argument, is checked against previous usage, and the value in the `_usedNonces` mapping is set to true if it has not been used. Then, the call to the indicated address is executed.

```
38      function relay(RelayData calldata data, bytes memory signature) exter
39          bytes32 _hash = _hashTypedDataV4(hashStruct(data));
40
41          if (block.timestamp < data.validityStart || block.timestamp > dat
42          {
43              revert InvalidSignature();
44          }
45
46          address recovered = ECDSA.recover(_hash, signature);
47          if (!hasRole(OPERATOR_ROLE, recovered)) {
48              revert Unauthorized();
49          }
50
51          if (_usedNonces[data.nonce]) revert AlreadyUsed();
52          _usedNonces[data.nonce] = true;
53
54          emit SignatureUsed(data.nonce, false);
55
56          (bool success, bytes memory result) = data.to.call{value: msg.val
57          if (!success) {
58              revert CallFailed();
59          }
60          return result;
61      }
```

The `AuthenticatedRelay` is intended to work as an entry point for users to execute designated operations by passing the struct. Each struct has a defined `nonce` value, which is emitted by the event in each function. However, this `nonce` is not verified anywhere, potentially allowing a scenario where the nonce verified by the `AuthenticatedRelay` contract differs from the nonce passed in calldata. This discrepancy may lead to further issues with proper off-chain accounting.

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:L/D:N/Y:N/R:N/S:C (3.1)

Recommendation

Consider adding the value of nonce verified by the AuthenticatedRelay to the calldata attached to the call.

Remediation Plan

RISK ACCEPTED: The **Ubisoft team** accepted the risk of the issue.

7.3 UNSAFE NFT MINTING FOR BRIDGED TOKEN

// LOW

Description

The `GenesisChamptionBridge` contract implements the internal `_lzReceive` method to handle incoming messages and mint the `GenesisChampion` NFT demanded for bridging to another chain. It uses the `_mint` function defined in ERC721.

```
94      function _lzReceive(
95          Origin calldata, /*_origin*/
96          bytes32, /*_guid*/
97          bytes calldata payload,
98          address, /*_executor*/
99          bytes calldata /*_extraData*/
100     ) internal override {
101         (address to, uint256 id) = abi.decode(payload, (address, uint256)
102         require(_ownerOf(id) == address(0), "token already exists");
103         _mint(to, id);
104     }
```

This practice is discouraged and can be dangerous if the user specifies a contract address that is not set up to handle NFTs (which is properly checked in the main `GenesisChampion` contract), as the NFT will get stuck.

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:L/D:N/Y:N/R:N/S:C (3.1)

Recommendation

To keep the consistency between the main `GenesisChampion` implementation and bridged NFTs, use the `_safeMint` function instead of `_mint`.

Remediation Plan

SOLVED: The **Ubisoft team** solved the issue as recommended.

Remediation Hash

8id3660b332c4662f605ff97e7777c5b1107390f

7.4 USE OF OWNABLE LIBRARY WITH SINGLE-STEP OWNERSHIP TRANSFER

// LOW

Description

The ownership of the contracts can be lost as the `GenesisChampion`, `GenesisChampionBridge`, and `GenesisChampionFactory` contracts inherited from the Ownable contract and their ownership can be transferred in a single-step process. The address the ownership is changed to should be verified to be active or willing to act as the owner.

BVSS

AO:A/AC:L/AX:L/C:N/I:L/A:L/D:N/Y:N/R:N/S:U (3.1)

Recommendation

It is recommended to implement a two-step process where the owner nominates an account and the nominated account needs to call an `acceptOwnership` function for the transfer of the ownership to fully succeed. This ensures the nominated EOA account is a valid and active account. This can be achieved by using OpenZeppelin's Ownable2Step contract instead of the Ownable.

Remediation Plan

RISK ACCEPTED: The **Ubisoft team** accepted the risk of the issue.

7.5 USE OF UNSAFE TRANSFERFROM FUNCTION

// LOW

Description

The `_processERC20Payment` function calls the `transferFrom` function on the token inherited from the IERC20 interface.

```
335     function _processERC20Payment(address payer, address token, uint256 v
336         IERC20 erc20 = IERC20(token);
337         bool success = erc20.transferFrom(payer, vault, value);
338         if (!success) revert Errors.TransferCraftFees(token, value, vault
339         emit CraftFees(vault, token, value, payer);
340     }
```

Some tokens do not fully comply with the ERC20 standard but remain compatible with most code designed to handle ERC20 tokens. For instance, the USDT `transferFrom` functions on the ETH mainnet do not return booleans as required by the specification, instead, they lack any return value. When during the crafting, the user provides a USDT address as a payment token, the call to the `transferFrom` function reverts when it attempts to decode the expected boolean return value.

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:L/D:N/Y:N/R:N/S:C (3.1)

Recommendation

Consider using OpenZeppelin's SafeERC20 `safeTransferFrom` functions.

Remediation Plan

SOLVED: The **Ubisoft team** solved the issue.

Remediation Hash

81d3660b332c4662f605ff97e7777c5b1107390f

7.6 LACK OF STORAGE GAP IN UPGRADEABLE CONTRACTS

// LOW

Description

The `GenesisMinter` and `GenesisCrafter` contracts are designed to be used with the UUPS proxy pattern. However, it lacks storage gaps. Storage gaps are essential for ensuring that new state variables can be added in future upgrades without affecting the storage layout of inheriting child contracts. Without it, any addition of new state variables in future contract versions can lead to storage collisions.

BVSS

AO:A/AC:L/AX:L/C:N/I:L/A:L/D:N/Y:N/R:N/S:U (3.1)

Recommendation

Consider adding a storage gap as the last storage variable to the mentioned contracts.

Remediation Plan

SOLVED: The **Ubisoft team** solved the issue.

Remediation Hash

45ia6fd7684d5e72b77a784b61f43fbea4d3fab3

7.7 UNNECESSARY NATIVE CURRENCY TRANSFER

// INFORMATIONAL

Description

The `relay` function implemented in the `AuthenticatedRelay` contract calls the intended contract with the calldata provided by the user, if all the data is properly authorized and verified. However, along with the calldata, it passes the `msg.value` attached to the transaction, but none of the functions that it is designated to operate with, are intended to receive the internal chain's currency, nor these contracts have implemented receive/fallback function.

```
56          (bool success, bytes memory result) = data.to.call{value: msg.val
57          if (!success) {
58              revert CallFailed();
59          }
```

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:L/Y:N/R:P/S:C (1.6)

Recommendation

Consider removing the redundant transfer.

Remediation Plan

ACKNOWLEDGED: The **Ubisoft team** acknowledged this finding.

7.8 CONSIDER USING NAMED MAPPINGS

// INFORMATIONAL

Description

The project is using Solidity version greater than 0.8.18, which supports named mappings. Using named mappings can improve the readability and maintainability of the code by making the purpose of each mapping clearer. This practice will enhance code readability and make the purpose of each mapping more explicit.

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation

Consider refactoring the mappings to use named arguments.
For example, on `GenesisChampionFactory`, instead of declaring:

```
mapping(address => uint256) public deployedVersions;
```

The mapping could be declared as:

```
mapping(address collection => uint256 index) public deployedVersions;
```

Remediation Plan

SOLVED: The **Ubisoft team** solved the issue.

Remediation Hash

81d3660b332c4662f605ff97e7777c5b1107390f

7.9 UPDATE OR REMOVE UNUSED COMMENTS TO IMPROVE CODE CLARITY

// INFORMATIONAL

Description

Three wrong or unused comments were identified in the codebase:
On GenesisMinter:

- `/// @notice emitted after a successful claimWithSignatureevent Claim(address indexed collection, bytes32 indexed nonce, uint256 amount);`

On GenesisCrafter: /
`// @notice craftCounters holds the CraftData of all Champions from any deployed GenesisChampion contract mapping(address => mapping(uint256 => CraftCounter)) public craftCounters;`
On GenesisChampionFactory:

- `// /// @notice Reference implementation address of GenesisChampion// address immutable public referenceContract;`

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation

Update the two incorrect comments to accurately reflect their respective functions, and remove the unused comment to enhance code clarity.

Remediation Plan

SOLVED: The **Ubisoft team** solved the issue.

Remediation Hash

81d3660b332c4662f605ff97e7777c5b1107390f

7.10 UNUSED IMPORTS

// INFORMATIONAL

Description

Throughout the code in scope, there are several instances where the imports are declared but never used.

GenesisBaseV2

- `import {IAccessControl} from "openzeppelinV4/access/IAccessControl.sol";`
- `import {IERC721Metadata} from "openzeppelinV4/token/ERC721/extensions/IERC721Metadata.sol";`
- `import {EIP712} from "openzeppelinV4/utils/cryptography/EIP712.sol";`
- `import {MintData} from "src/types/MintData.sol";`

GenesisChampion

- `import {ERC721PsiAddressData} from "src/ERC721Psi/extension/ERC721PsiAddressData.sol";`

GenesisChampionFactory

- `import {Errors} from "src/librairies/Errors.sol";`

GenesisCrafter

- `import {IERC20Errors} from "@openzeppelin/contracts/interfaces/draft-IERC6093.sol";`

GenesisRewardDistributor

- `import {GenesisChampion} from "src/GenesisChampion.sol";`

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation

Consider removing all unused imports

Remediation Plan

SOLVED: The **Ubisoft team** solved the issue.

Remediation Hash

451a6fd7684d5e72b77a784b61f43fbea4d3fab3

7.11 INEFFICIENT CONDITIONAL CHECKS

// INFORMATIONAL

Description

The `claim` function from `GenesisRewardDistributor` contract is responsible for minting rewards in the form of ERC20, ERC721, or ERC1155 tokens based on the intended season's reward type.

```
144         // Mint the reward following its type
145         if (currentSeason.rewardType == uint8(RewardType.ERC20)) {
146             IERC20MintableReward(currentSeason.collection).mint(request.t
147         } else if (currentSeason.rewardType == uint8(RewardType.ERC721))
148             IERC721MintableReward(currentSeason.collection).mint(request.
149         } else {
150             IERC1155MintableReward(currentSeason.collection).mint(request
151         }
```

However, as the ERC721 tokens are expected to be minted most frequently throughout the protocol's liveness, the current code structure requires an additional check for each transaction, increasing gas consumption. By placing the ERC721 minting logic first, the number of checks can be reduced when this condition is met, optimizing gas usage.

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation

Consider reordering the conditional checks to prioritize the ERC721 minting path.

Remediation Plan

SOLVED: The **Ubisoft team** solved the issue.

Remediation Hash

`c91ba1572af10c4241919f1c2f933b742c1d15ae`

8. AUTOMATED TESTING

Static Analysis Report

Description

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their abi and binary formats, Slither was run on the all-scoped contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Slither results

GenesisChampion.sol

```
GenesisChampion.tokenURI(uint256) (src/GenesisChampion.sol#96-100) calls abi.encodePacked() with multiple dynamic arguments:
        - string(abi.encodePacked(_baseURI(),tokenId.toString(),.json)) (src/GenesisChampion.sol#99)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#abi-encodePacked-collision
INFO:Detectors:
ERC721Psi.balanceOf(address).count (src/ERC721Psi/ERC721Psi.sol#89) is a local variable never initialized
ERC721Psi.tokensOfOwner(address).tokenIdsIdx (src/ERC721Psi/ERC721Psi.sol#403) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
```

GenesisCrafter.sol

```
GenesisChampion.tokenURI(uint256) (src/GenesisChampion.sol#96-100) calls abi.encodePacked() with multiple dynamic arguments:
        - string(abi.encodePacked(_baseURI(),tokenId.toString(),.json)) (src/GenesisChampion.sol#99)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#abi-encodePacked-collision
INFO:Detectors:
Reentrancy in GenesisChampionFactory.deploy(GenesisChampionArgs) (src/GenesisChampionFactory.sol#34-47):
        External calls:
        - impl = new GenesisChampion(_args) (src/GenesisChampionFactory.sol#37)
        State variables written after the call(s):
        - lastVersion = newVersion (src/GenesisChampionFactory.sol#42)
        GenesisChampionFactory.lastVersion (src/GenesisChampionFactory.sol#25) can be used in cross function reentrancies:
        - GenesisChampionFactory.deploy(GenesisChampionArgs) (src/GenesisChampionFactory.sol#34-47)
        - GenesisChampionFactory.lastVersion (src/GenesisChampionFactory.sol#25)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
ERC721Psi.balanceOf(address).count (src/ERC721Psi/ERC721Psi.sol#89) is a local variable never initialized
ERC721Psi.tokensOfOwner(address).tokenIdsIdx (src/ERC721Psi/ERC721Psi.sol#403) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
GenesisChampionFactory.deploy(GenesisChampionArgs).newDeployment (src/GenesisChampionFactory.sol#39) lacks a zero-check on :
                - lastDeployment = newDeployment (src/GenesisChampionFactory.sol#41)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in GenesisChampionFactory.deploy(GenesisChampionArgs) (src/GenesisChampionFactory.sol#34-47):
        External calls:
        - impl = new GenesisChampion(_args) (src/GenesisChampionFactory.sol#37)
        State variables written after the call(s):
        - deployedVersions[newDeployment] = newVersion (src/GenesisChampionFactory.sol#40)
        - lastDeployment = newDeployment (src/GenesisChampionFactory.sol#41)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in GenesisChampionFactory.deploy(GenesisChampionArgs) (src/GenesisChampionFactory.sol#34-47):
        External calls:
        - impl = new GenesisChampion(_args) (src/GenesisChampionFactory.sol#37)
        Event emitted after the call(s):
        - ContractCreated(newDeployment,newVersion) (src/GenesisChampionFactory.sol#44)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

GenesisMinter.sol

```
Reentrancy in GenesisMinter.claim(MintData) (src/GenesisMinter.sol#102-126):
        External calls:
        - (firstId,lastId) = _mint(request.collection,request.to,allocation) (src/GenesisMinter.sol#124)
                - (firstId,lastId) = champion.mint(to,amount) (src/GenesisMinter.sol#171)
        Event emitted after the call(s):
        - Claim(request.collection,request.nonce,allocation) (src/GenesisMinter.sol#125)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

GenesisRewardDistributor.sol

```
Reentrancy in GenesisRewardDistributor.claim(RewardClaim) (src/GenesisRewardDistributor.sol#121-153):
        External calls:
        - IERC20MintableReward(currentSeason.collection).mint(request.to,request.amount) (src/GenesisRewardDistributor.sol#146)
        - IERC721MintableReward(currentSeason.collection).mint(request.to,request.amount) (src/GenesisRewardDistributor.sol#148)
        - IERC1155MintableReward(currentSeason.collection).mint(request.to,currentSeason.tokenId,request.amount,) (src/GenesisRewardDistributor.sol#150)
        Event emitted after the call(s):
        - ClaimReward(request.nonce,request.to) (src/GenesisRewardDistributor.sol#152)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
GenesisRewardDistributor.claim(RewardClaim) (src/GenesisRewardDistributor.sol#121-153) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp < currentSeason.claimStart || block.timestamp > currentSeason.claimEnd (src/GenesisRewardDistributor.sol#129)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.