SALUS SECURITY

APR 2025

CODE SECURITY ASSESSMENT

JUTSU WORLD

Overview

Project Summary

- Name: Jutsu - ursas
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/admin-jutsuinc/-ursas
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Jutsu - ursas
Version	v2
Type	Solidity
Dates	Apr 09 2025
Logs	Mar 03 2025, Apr 09 2025

Vulnerability Summary

Total High-Severity issues	3
Total Medium-Severity issues	4
Total Low-Severity issues	4
Total informational issues	4
Total	15

Contact

E-mail: support@salusec.io

SALUS

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

SALUS

Content

SALUS

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

SALUS

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Reroll card results can be predictive	High	Business Logic	Resolved
2	Reroll price can be manipulated	High	Business Logic	Resolved
3	Possible multiple overflow in price calculation	Medium	Business Logic	Resolved
4	The remaining ETH will not be refunded to the user in the reRollCards function.	Medium	Business Logic	Resolved
5	Cards can be removed from deck when deck is joining a game	Medium	Business Logic	Resolved
6	Centralization risk	Medium	Centralization	Acknowledged
7	Seeds may be used beyond maxSeedUsage	Low	Business Logic	Resolved
8	Zero seed will be used for the first several reroll	Low	Business Logic	Resolved
9	The owner cannot directly call erc20TransferFrom to transfer token	Low	Business Logic	Resolved
10	Not using safeErc721TransferMultiFrom to transfer cards to the deck may result in NFT loss	Low	Business Logic	Resolved
11	Missing update card data in erc20TransferFrom	Low	Business Logic	Acknowledged
12	Implementation contract could be initialized by everyone	Informational	Business Logic	Resolved
13	Insufficient maxPlayers check	Informational	Business Logic	Resolved
14	Redundant code	Informational	Gas optimization	Resolved
15	Missing valid id check in ERC721Approve and ERC721TransferFrom	Informational	Business Logic	Resolved

SALUS

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Reroll card results can be predictive

Severity: High	Category: Business Logic

Target:
- contracts/Jutsu/MainnetV1/CardsLibraryMainnetV1.sol

Description

To win a game, users have the option to reroll their cards by paying a reroll fee. The outcome of each reroll is determined by a random seed generated by the Gelato VRF, along with the current block's timestamp.

While the seed itself is random, users can take advantage of the block.timestamp to predict the result. This allows them to continue rerolling their tokens within a specific block until they obtain a rare card.

contracts/Jutsu/MainnetV1/CardsLibraryMainnetV1.sol: L93-L107

```
function getPseudoRandom(
  bytes32 randomness,
  uint32 counter,
  uint32 maxValue
) internal view returns (uint32) {
  // Using block.timestamp as a simple randomness source
  bytes memory input = abi.encodePacked(randomness, block.timestamp, counter);
  // Get the hash
  bytes32 hash = sha256(input);
  // Convert the first 4 bytes of the hash into a uint32
  uint32 hashU32 = uint32(bytes4(hash));
  // Get a random number in the range [0, maxValue]
  uint32 number = hashU32 % (maxValue + 1);
  return number;
}
```

Recommendation

Make sure that the final reroll result is not predictive.

Status

This issue has been resolved by the team with commit f2309e6.

SALUS

2. Reroll price can be manipulated

Severity: High	Category: Business Logic

Target:
- contracts/Jutsu/MainnetV1/CardsLibraryMainnetV1.sol

Description

To win a game, users can reroll their cards by paying a reroll fee. The reroll price is dynamically fetched from a Uniswap V3 liquidity pool and calculated based on the sqrtPriceX96.

However, since sqrtPriceX96 reflects the real-time price, it is susceptible to manipulation. Malicious users could artificially alter the token's price, allowing them to reroll at an exceptionally low cost.

contracts/Jutsu/MainnetV1/CardsLibraryMainnetV1.sol: L369-L393

```
function getReRollPricePerCard(CardsStorage storage $) public view returns (uint256) {
  uint160 sqrtPriceX96;
  try $.jutsuToNativeLPAddress.slot0() returns (
    uint160 _sqrtPriceX96, int24, uint16,
    Uint16, uint16, uint8, bool
  ) {
    sqrtPriceX96 = _sqrtPriceX96;
  } catch {
    return $.reRollPricePerCard;
  }
  uint256 price;
  if (token0 == address(this)) {
    // Price of 1 token1 in terms of token0
    price = (uint256(sqrtPriceX96) * uint256(sqrtPriceX96) * (10 ** 18)) >> (96 * 2);
  } else if (token1 == address(this)) {
    // Price of 1 token0 in terms of token1
    price = ((uint256(1) << (96 * 2)) * (10 ** 18)) / (uint256(sqrtPriceX96) *
uint256(sqrtPriceX96));
  } else {
    return $.reRollPricePerCard;
  }
```

Recommendation

Chainlink oracle or Twap price in Uniswap is suggested.

Status

This issue has been resolved by the team with commit f2309e6.

SALUS

3. Possible multiple overflow in price calculation

Severity: Medium	Category: Business Logic

Target:
- contracts/Jutsu/MainnetV1/CardsLibraryMainnetV1.sol

Description

To win a game, users can reroll their cards by paying a reroll fee, with the price determined by a Uniswap V3 liquidity pool.

When calculating the token price, we use the formula `sqrtPriceX96^2 * 10^18`. Since `uint256.max` has 78 decimals, and `sqrtPriceX96` is scaled by `2^96` (approximately 29 decimals), the multiplication can easily exceed the limit, triggering multiple overflows. As a result, the reroll process may revert.

contracts/Jutsu/MainnetV1/CardsLibraryMainnetV1.sol: L369-L393

```
 function getReRollPricePerCard(CardsStorage storage $) public view returns (uint256) {
   uint256 price;
   if (token0 == address(this)) {
     // Price of 1 token1 in terms of token0
     price = (uint256(sqrtPriceX96) * uint256(sqrtPriceX96) * (10 ** 18)) >> (96 * 2);
   } else if (token1 == address(this)) {
     // Price of 1 token0 in terms of token1
     price = ((uint256(1) << (96 * 2)) * (10 ** 18)) / (uint256(sqrtPriceX96) *
uint256(sqrtPriceX96));
   } else {
     return $.reRollPricePerCard;
   }

   return (price * $.reRollPriceCardPartsPerMillion) / 1_000_000;
 }
```

Recommendation

Suggest using safeMath to avoid the possible overflow.

Status

This issue has been resolved by the team with commit f2309e6.

SALUS

4. The remaining ETH will not be refunded to the user in the reRollCards function.

Severity: High	Category: Business Logic

Target:
- contracts/Jutsu/MainnetV1/JutsuMainnetV1.sol

Description

The `reRollCards` function is used to extract NFTs associated with cards, requiring a fee that may be dynamically calculated based on the pool price.

However, if a user sends more native token than the required fee, the excess amount is not refunded but remains in the contract. This could result in unintended user losses.

contracts/Jutsu/MainnetV1/JutsuMainnetV1.sol: L72-L87

```
  function reRollCards(
    uint256[] memory tokenIds,
    uint256 beliefPricePerToken
  ) public payable whenNotPaused {
    ERC404V4Storage storage $ = _getERC404V4Storage();
    RopeTreeAVLLibraryMainnetV1.RopeTreeAVLStorage
    storage $2 = _getRopeTreeAVLStorage();
    uint32[] memory convertedTokenIds = new uint32[](tokenIds.length);
    for (uint64 i = 0; i < tokenIds.length; i++) {
      convertedTokenIds[i] = uint32(tokenIds[i]);
    }
    if (!$2.isOwnerOfMultiAVLTokens(convertedTokenIds, msg.sender)) {
      revert Unauthorized();
    }
    executeReRollCards(convertedTokenIds, beliefPricePerToken);
  }
```

Recommendation

It is recommended to refund any unused ETH to the user at the end of the `reRollCards` function.

Status

This issue has been resolved by the team with commit f2309e6.

SALUS

5. Cards can be removed from deck when deck is joining a game

Severity: Medium	Category: Business Logic
Target: - contracts/Deck/Deck.sol	

Description

Users can transfer their Card NFTs to the Deck to mint a Deck NFT, which is then used to start or participate in a game.

However, a critical issue arises: Card NFTs can be removed from a Deck even while it is actively engaged in a game. This allows malicious users to exploit the system by joining a game with a Deck NFT, removing its cards to create another Deck NFT, and re-entering the same game multiple times.

The impact will be as below:

1. Malicious users can use one Card NFT to join the same game multiple times to block normal users' join.
2. claimBackTime limitation can be bypassed.

contracts/Jutsu/MainnetV1/CardsLibraryMainnetV1.sol: L369-L393

```
function removeTokenIdsFromDeck(uint64 deckId, uint32[] memory tokenIdsToRemove) public
{
    DeckStorage storage $ = _getDeckStorage();
    DeckInfo storage deck = $._deckInfo[deckId];

    if (deck.owner != _msgSender()) revert AuthenticationError(_msgSender());
    ...
    assembly {
        mstore(removeTokenIds, cnt)
    }

    if (removeTokenIds.length != tokenIdsToRemove.length) revert InvalidTokenIds();

    IERC721MultiTransfer($.validJutsuAddress).erc721TransferMulti(
        deck.owner,
        removeTokenIds
    );
    if (deck.tokenIds.length == 0) burn(deckId);
}
```

Recommendation

When the Deck NFT is in one game, lock the Card NFTs.

Status

This issue has been resolved by the team with commit f2309e6.

SALUS

6. Centralization risk

Severity: Medium	Category: Centralization
Target: - contracts/Jutsu/MainnetV1/CardsMainnetV1.sol	

Description

Contracts in Ursas have one privileged account: owner. The owner role can modify many critical parameters, upgrade contracts, etc.

If the private keys of these accounts are compromised, it could have a devastating impact on the system.

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

This issue has been acknowledged by the team.

SALUS

7. Seeds may be used beyond maxSeedUsage

Severity: Medium	Category: Business Logic

Target:
- contracts/Jutsu/MainnetV1/CardsLibraryMainnetV1.sol

Description

For each seed, if the usage count exceeds `maxSeedUsage`, a new seed is requested. However, since the seed update is asynchronous, calls to `reRollCards` may still rely on the previous seed until the update is complete. As a result, the actual usage count of a seed may exceed the defined `maxSeedUsage`, potentially leading to unintended behavior.

contracts/Jutsu/MainnetV1/CardsLibraryMainnetV1.sol: L369-L393

```
function executeReRollCards(
  CardsStorage storage $,
  GelatoLibraryMainnetV1.GelatoStorage storage $2,
  uint256 amountRequired,
  uint32[] memory tokenIds,
  uint256 beliefPricePerToken
) public {
  ...
  $2.seedUsage += 1;
  if ($2.VRFAddress != address(0) && $2.seedUsage % $2.maxSeedUsage == 0){
    $2.requestRandomness(new bytes(0));
  }

  emit CardsRerollSuccess(msg.sender, result);
}
```

Recommendation

It is recommended to pause user calls to `reRollCards` when the seed has not been updated. Additionally, if the random number request fails due to external factors, privileged accounts should be allowed to request the seed again.

Status

This issue has been resolved by the team with commit f2309e6.

SALUS

8. Zero seed will be used for the first several reroll

Severity: Low	Category: Business Logic

Target:
- contracts/Jutsu/MainnetV1/CardsLibraryMainnetV1.sol

Description

Users can reroll their Card NFT by paying a reroll fee, with the outcome determined by a random seed from Gelato VRF.

However, the first random seed request is only sent after reaching the `maxSeedUsage` threshold. This means that for the initial rerolls, the contract relies on the default seed value (0), which is insecure and could compromise the fairness of the reroll process.

contracts/Jutsu/MainnetV1/CardsLibraryMainnetV1.sol: L400-L452

```
function executeReRollCards(
  CardsStorage storage $,
  GelatoLibraryMainnetV1.GelatoStorage storage $2,
  uint256 amountRequired,
  uint32[] memory tokenIds,
  uint256 beliefPricePerToken
) public {
  uint32[] memory cards = generateRandomNCards(
    $,
    $2.seed,
    uint32(tokenIds.length)
  );
  $2.seedUsage += 1;
  if ($2.VRFAddress != address(0) && $2.seedUsage % $2.maxSeedUsage == 0){
    $2.requestRandomness(new bytes(0));
  }
}
```

Recommendation

Send one random request when we deploy the contract to initialize the seed.

Status

This issue has been resolved by the team with commit f2309e6.

SALUS

9. The owner cannot directly call erc20TransferFrom to transfer token

Severity: Low	Category: Business Logic

Target:
- contracts/Jutsu/MainnetV1/ERC404MainnetV1.sol

Description

When the owner calls `erc20TransferFrom` to transfer their own tokens, the transaction cannot succeed directly. Instead, the owner must first approve to transfer the tokens on their behalf by granting an allowance.

contracts/Jutsu/MainnetV1/ERC404MainnetV1.sol: L453-L479

```
function erc20TransferFrom(
  address from_,
  address to_,
  uint256 value_
) public virtual returns (bool) {
  ERC404V4Storage storage $ = _getERC404V4Storage();
  // Prevent minting tokens from 0x0.
  if (from_ == address(0)) {
    revert InvalidSender();
  }

  // Prevent burning tokens to 0x0.
  if (to_ == address(0)) {
    revert InvalidRecipient();
  }

  uint256 allowed = $.allowance[from_][msg.sender];
  // Check that the operator has sufficient allowance.
  if (allowed != type(uint256).max) {
    $.allowance[from_][msg.sender] = allowed - value_;
  }

  // Transferring ERC-20s directly requires the _transferERC20WithERC721 function.
  // Handles ERC-721 exemptions internally.
  return _transferERC20WithERC721(from_, to_, value_);
}
```

Recommendation

It is recommended that when `msg.sender == from`, the allowance check is skipped.

Status

This issue has been resolved by the team with commit f2309e6.

SALUS

10. Not using safeErc721TransferMultiFrom to transfer cards to the deck may result in NFT loss

Severity: Low	Category: Business Logic
Target: - contracts/Deck/Deck.sol	

Description

The deck contract is used to create a deck NFT containing multiple cards, but the creation logic is executed within the `onERC721MultiReceived` function. This means that if users transfer their cards to the deck contract using methods other than `safeErc721TransferMultiFrom`, their NFTs will be lost and cannot be recovered.

contracts/Deck/Deck.sol: L175-L230

```
function onERC721MultiReceived(
    address operator,
    address from,
    uint256[] memory tokenIds,
    bytes calldata data
) external override returns (bytes4) {
    ...
}
```

Recommendation

It is recommended to implement a function in the deck contract to handle the recovery of incorrectly transferred card NFTs.

Status

This issue has been resolved by the team with commit f2309e6.

SALUS

11. Missing update card data in erc20TransferFrom

Severity: Low	Category: Business Logic
Target: - contracts/Jutsu/MainnetV1/ERC404MainnetV1.sol	

Description

The `JutsuMainnetV1` contract overrides the `_transferERC721` and `erc721TransferMultiFrom` functions to ensure that card data is updated whenever NFT ownership changes.

However, when users call `erc20TransferFrom`, both ERC20 and ERC721 tokens are transferred to the specified address, but the associated card data for the ERC721 tokens remains unchanged. This discrepancy could result in an inconsistent contract state.

contracts/Jutsu/MainnetV1/ERC404MainnetV1.sol: L453-L479

```
function erc20TransferFrom(
  address from_,
  address to_,
  uint256 value_
) public virtual returns (bool) {
  ...
  // Transferring ERC-20s directly requires the _transferERC20WithERC721 function.
  // Handles ERC-721 exemptions internally.
  return _transferERC20WithERC721(from_, to_, value_);
}
```

Recommendation

It is recommended to similarly override the `_transferERC20WithERC721` function to ensure that the associated card data is updated with a higher state after the NFT transfer.

Status

This issue has been acknowledged by the team.

SALUS

2.3 Informational Findings

12. Implementation contract could be initialized by everyone

Severity: Information	Category: Business Logic
Target: - contracts/Deck/Deck.sol - contracts/Game/Game.sol - Jutsu/MainnetV1/JutsuMainnetV1.sol	

Description

According to OpenZeppelin, the implementation contract should not be left uninitialized.

An uninitialized implementation contract can be taken over by an attacker, which may impact the proxy. There is nothing preventing the attacker from calling the functions in contracts' implementation contracts.

Recommendation

To prevent the implementation contract from being used, consider invoking the _disableInitializers function in the constructor of the Deck/Game/JutsuMainnetV1 contract to automatically lock it when it is deployed.

Status

This issue has been resolved by the team with commit f2309e6.

SALUS

13. Insufficient maxPlayers check

Severity: Information	Category: Business Logic
Target: - contracts/Game/Game.sol	

Description

In the `startGame` function, it checks that the `maxPlayers` quantity is not zero. It would be better to check that `maxPlayers` is greater than 1, as there should be at least two players in the game; otherwise, the account initiating the game will inevitably become the winner.

contracts/Game/Game.sol: L98-L118

```
function startGame(uint64 gameTypeId, uint32 maxPlayers, uint64 deckId) external {
    GameStorage storage $ = _getGameStorage();
    if (maxPlayers == 0) revert InvalidMaxPlayers();
    ...
}
```

Recommendation

It is recommended to check that `maxPlayers` is greater than 1.

Status

This issue has been resolved by the team with commit f2309e6.

SALUS

14. Redundant code

Severity: Information	Category: Gas optimization

Target:
- contracts/Jutsu/MainnetV1/GelatoMainnetV1.sol
- contracts/Jutsu/MainnetV1/GelatoLibraryMainnetV1.sol

Description

There is some unnecessary code in the protocol that can be removed to reduce both interaction gas costs and contract deployment costs.

In the `GelatoMainnetV1` contract, there is an error that is never used.

contracts/Jutsu/MainnetV1/GelatoMainnetV1.sol: L21-L22

```
abstract contract GelatoMainnetV1 is IGelatoVRFConsumer, Initializable,
OwnableUpgradeable {
    ...
    error OnlyDedicatedMsgSender();
```

In the `fulfillRandomness` function, the `requestedHash` is deleted multiple times

contracts/Jutsu/MainnetV1/GelatoLibraryMainnetV1.sol: L111-L114

```
function fulfillRandomness(
    GelatoStorage storage $,
    uint256 randomness,
    bytes calldata dataWithRound
) public {
    ...
        delete $.requestedHash[requestId];
    }
    delete $.requestedHash[requestId];
}
```

Recommendation

It is recommended to remove the redundant code.

Status

This issue has been resolved by the team with commit f2309e6.

SALUS

15. Missing valid id check in ERC721Approve and ERC721TransferFrom

Severity: Information	Category: Gas optimization

Target:
- contracts/Jutsu/MainnetV1/ERC404MainnetV1.sol

Description

The `ERC721Approve` and `ERC721TransferFrom` functions are public functions. However, they lack a valid ID check, which could result in events being emitted with invalid or non-existent token IDs.

contracts/Jutsu/MainnetV1/GelatoMainnetV1.sol: L218-L228

```
function erc721Approve(address spender_, uint256 id_) public virtual {
  ...
  emit ERC721Events.Approval(erc721Owner, spender_, id_);
}
```

contracts/Jutsu/MainnetV1/GelatoMainnetV1.sol: L299-L337

```
function erc721TransferFrom(
  address from_,
  address to_,
  uint256 id_
) public virtual {
  ...
}
```

Recommendation

It is recommended to add a valid ID check in the `ERC721Approve` and `ERC721TransferFrom` functions.

Status

This issue has been resolved by the team with commit f2309e6.

SALUS

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 18dbaee:

File	SHA-1 hash
Deck.sol	19baafbc8dcd706e8813a2febb04bed291a51d4b
Game.sol	879116de4ab951621c92396a4f2ac900a237acdc
MerkleClaimDrop.sol	4b50a54e513153559858dfa48342257786ea6bce
TokenVesting.sol	d5c7aa46fc594acb4c7830ac56e9fc60f69063c3
SolmateERC20.sol	4c47917b707df0a0fa03ad2100b8ca34c1531ced
SolmateSafeTransfer.sol	09d408caef56f95218cc0e06d0fbde6052e40aa6
CardsLibraryMainnetV1.sol	b02bbfbf261a30298ef1b55ff5bb41b9c762176c
CardsMainnetV1.sol	0a3b98be51f8553564a19d86822a274136c61d1e
ERC404MainnetV1.sol	8c55b68dd7dba711909f7c42c8d1078cc1fa8b0a
ERC404V4StorageMainnetV1.sol	5d72e193e4e1a57a0e4a17c52453b5a434f5fc9e
GelatoLibraryMainnetV1.sol	5ca2af82a34c3da4bdc8eecc3f916a8465594125
GelatoMainnetV1.sol	2b320af80472d27a4b9e98b5e8f37505f0ae7b41
JutsuMainnetV1.sol	f18f0f74cc625773625af4e133a595d968d965fd
RopeTreeAVLLibraryMainnetV1.sol	577e8d8ddeba8d92c4ae76973010328e22c196d9
RopeTreeAVLStorageMainnetV1.sol	f9760db3690a8e29b15a181d5383bc580524b5bc

SALUS