SUB7
Web3 Security

Gangster Arena S5 Audit Final Report
Security Assessment Findings Report

Date: September 20, 2024
Version 0.1

Contents

1 Confidentiality statement

This document is the exclusive property of Uncharted and Sub7 Security. This document contains proprietary and confidential information. Duplication, redistribution, or use, in whole or in part, in any form, requires consent of both Uncharted and Sub7 Security.

2 Disclaimer

A smart contract security audit is considered a snapshot in time. The findings and recommendations reflect the information gathered during the assessment and not any changes or modifications made outside of that period.

Time-limited engagements do not allow for a full evaluation of all security controls. Sub7 Security prioritized the assessment to identify the weakest security controls an attacker would exploit. Sub7 Security recommends conducting similar assessments on an annual basis by internal or third-party assessors to ensure the continued success of the controls

3 About Sub7

Sub7 is a Web3 Security Agency, offering Smart Contract Auditing Services for blockchain-based projects in the DeFi, Web3 and Metaverse space.

Learn more about us at https://sub7.xyz

4 Project Overview

Gangster Arena is a cryptonomic game for web3 natives. Play as a mob boss and expand your gang. Earn passive rewards and climb the leaderboard. Risk it all for the ETH prizes which get paid out at the end of each season.

The Gangster NFT is the core unit of the game that provides players with daily FIAT yield and reputation to climb the leaderboard.

Website: https://gangsterarena.com

5 Executive Summary

Sub7 Security has been engaged to what is formally referred to as a Security Audit of Solidity Smart Contracts, a combination of automated and manual assessments in search for vulnerabilities, bugs, unintended outputs, among others inside deployed Smart Contracts.

The goal of such a Security Audit is to assess project code (with any associated specification, and documentation) and provide our clients with a report of potential security-related issues that should be addressed to improve security posture, decrease attack surface and mitigate risk.

As well general recommendations around the methodology and usability of the related project are also included during this activity

1 (One) Security Auditors/Consultants were engaged in this activity.

5.1 Scope

Audit game contract and presale contract. Repository:
https://github.com/wearedayone/GangsterArena2/tree/ga5 Commit:
562eca74ce528dad243bc1908cdd084df53bf3b9
https://github.com/wearedayone/GangsterArena2/contract/GA5/contracts/GangsterArena.sol
https://github.com/wearedayone/GangsterArena2/contract/GA5/contracts/Minter.sol

5.2 Timeline

12.09.2024 - 20.09.2024

5.3 Summary of Findings Identified

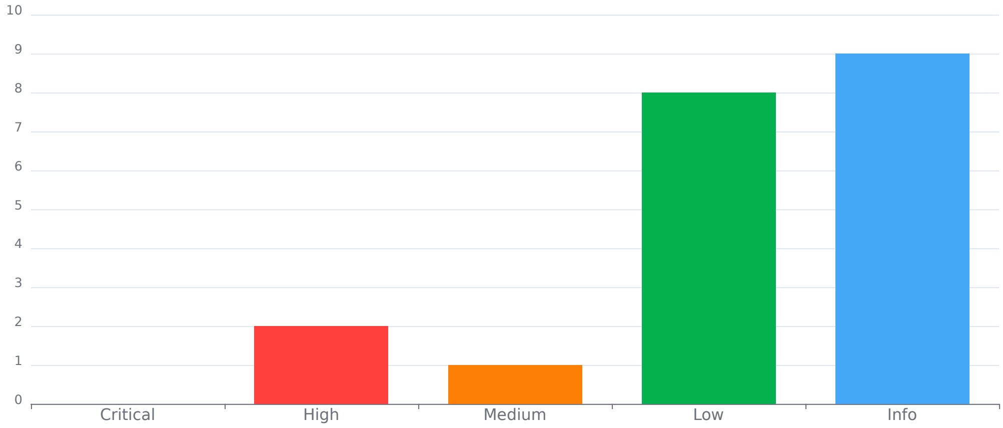

Figure 1: Executive Summary

# 1 High Signature replay possible in `buyStartPack` – ***Fixed***

# 2 High The `updateProbability` aggregates propabilities – ***Fixed***

# 3 Medium The `buyStartPack` uses `ethPrice` as input parameter – ***Acknowledged***

# 4 Low The `buyStartPack` can revert due to transfer usage – ***Fixed***

# 5 Low The `_refAddr` lacks input validation – ***Acknowledged***

# 6 Low The `_calculateMintResult` can revert due to integer overflow – ***Fixed***

# 7 Low Cross-chain signature replay possible – ***Fixed***

# 8 Low Insufficient fees input validation – ***Fixed***

# 9 Low Lack of two-step ownership pattern – ***Acknowledged***

# 10 Low Lack of singer update possibility – ***Fixed***

# 11 Low Lack of two-step ownership transfer – ***Acknowledged***

# 12 Info Mint can be called with 0 amount – ***Fixed***

# 13 Info Transfer is used instead of low-level call function – ***Fixed***

# 14 Info Lack of zero address input validation – ***Acknowledged***

# 15 Info Unused properties in `Phase` structure – ***Fixed***

# 16 Info `RevealMint` is emitted always with the same phase – ***Fixed***

# 17 Info The `encodePacked` is used for signature based authorization – ***Acknowledged***

# 18 Info Blast-testnet Entropy is in use – ***Acknowledged***

# 19 Info The `ReentrancyGuard` is inherited but not used – ***Acknowledged***

# 20 Info The `keccak256` usage with `abi.encodePacked` – ***Acknowledged***

5.4 Methodology

SUB7's audit methodology involves a combination of different assessments that are performed to the provided code, including but not limited to the following:

Specification Check

Manual assessment of the assets, where they are held, who are the actors, privileges of actors, who is allowed to access what and when, trust relationships, threat model, potential attack vectors, scenarios, and mitigations. Well-specified code with standards such as NatSpec is expected to save time.

Documentation Review

Manual review of all and any documentation available, allowing our auditors to save time in inferring the architecture of the project, contract interactions, program constraints, asset flow, actors, threat model, and risk mitigation measures

Automated Assessments

The provided code is submitted via a series of carefully selected tools to automatically determine if the code produces the expected outputs, attempt to highlight possible vulnerabilities within non-running code (Static Analysis), and providing invalid, unexpected, and/or random data as inputs to a running code, looking for exceptions such as crashes, failing built-in code assertions, or potential memory leaks.

Examples of such tools are Slither, MythX, 4naly3er, Sstan, Natspec-smells, and custom bots built by partners that are actively competing in Code4rena bot races.

Manual Assessments

Manual review of the code in a line-by-line fashion is the only way today to infer and evaluate business logic and application-level constraints which is where a majority of the serious vulnerabilities are being found. This intensive assessment will check business logics, intended functionality, access control & authorization issues, oracle issues, manipulation attempts and multiple others.

Security Consultants make use of checklists such as SCSVS, Solcurity, and their custom notes to ensure every attack vector possible is covered as part of the assessment

6 Findings and Risk Analysis

6.1 Signature replay possible in `buyStartPack`

Severity: High

Status: Fixed

Description

The `buyStartPack` function uses signature based authorisation to allow a particular user to purchase the NFT. To prevent the replay attack the signature is protected by the `nonce`. However, the implementation lacks an update of the `usedNonces` array with nonce used. Thus, a single signature can be reused multiple times until `MAX_PER_WALLET` limit is reached.

```
function buyStartPack(
  uint256 packId,
  uint256 ethPrice,
  uint256 _sTime,
  uint256 _nonce,
  bytes memory _sig
) public payable {
  require(!usedNonces[_nonce], 'Nonce is used');
  require(packs[packId].MAX_PER_WALLET > 0, 'Invalid Id');
  require(msg.value >= ethPrice, 'Need to send more ether');    require(block.timestamp <
       _sTime + vtd, 'Invalid timestamp');
  require(packs[packId].MAX_PER_WALLET > boughtPacks[msg.sender][packId], 'Over wallet
      limit');
  bytes32 message = SignerLib.prefixed(keccak256(abi.encodePacked(msg.sender, ethPrice,
      _sTime, _nonce)));
  require(signer.verifyAddressSigner(message, _sig), 'INVALID_SIGN');


  StartPack memory pack = packs[packId];
  boughtPacks[msg.sender][packId] += 1;
  address payable receiver = payable(devAddr_);
  receiver.transfer(msg.value);
  mintNFT(msg.sender, pack.numberOfGangster);


  emit BuyStartPack(msg.sender, packId, pack.numberOfGangster);
}
```

Location

`./contracts/GangsterArena.sol`

Recommendation

It is recommended to update the `usedNonces` array whenever the nonce is consumed by the signature verification.

Comments

6.2 The `updateProbability` aggregates propabilities

Severity: High
Status: Fixed

Description

The `updateProbability` function is supposed to update the array of `probabilities` used in the further processing of `Gangster` minting. However, it is written in such a way it always adds new records of `probabilities`, but it does not reset the `probabilities` array. Thus, only the first set of probabilities will be used for every protocol's iteration. Any subsequent update may have no further effects of the processing.

```
function updateProbability(uint256[] memory _probability, uint256[] memory
    _gangsterReward) public onlyOwner {
  require(sumOf(_probability) == 1000, 'Invalid input');
  // probabilities = new Probability[](_probability.length);
  for (uint256 i = 0; i < _probability.length; i++) {
    probabilities.push(Probability(_probability[i], _gangsterReward[i], 1000));
  }
}
```

Location

`./contracts/Minter.sol`

Recommendation

It is recommended to reset the `probabilities` array in every `updateProbability` function call.

Comments

6.3 The `buyStartPack` uses `ethPrice` as input parameter

Severity: Medium
Status: Acknowledged

Description

The `buyStartPack` function uses `ethPrice` from the function's input, despite the fact that the `ethPrice` is set in the packs array record within the `setPack` function. Still, the `ethPrice` parameter is included in the signed message, thus manipulation possibility is decreased. However, the final price is under the control of the signer’s address. Whenever this account is compromised, this weakness can be abused to mint NFTs free of charge.

```
 function buyStartPack(
...
   uint256 ethPrice,
...
 ) public payable {
...
   require(msg.value >= ethPrice, 'Need to send more ether');
...
   bytes32 message = SignerLib.prefixed(keccak256(abi.encodePacked(msg.sender, ethPrice,
       _sTime, _nonce)));
...
 }
```

Location

`./contracts/GangsterArena.sol`

Recommendation

It is recommended to use `ethPrice` set by the protocol owner within the `setPack` function.

Comments

6.4 The `buyStartPack` can revert due to transfer usage

Severity: Low

Status: Fixed

Description

Within the `buyStartPack` function the transfer is used to send native tokens. This function has limited Gas provided to the receiver. Thus, in certain circumstances, the transfer may revert due to `Out Of Gas` error if only the `devAddr_ receive` function contains any business logic to execute. As a result of this weakness the NFT purchase via the `buyStartPack` function may be not possible.

```
function buyStartPack(
   uint256 packId,
   uint256 ethPrice,
   uint256 _sTime,
   uint256 _nonce,
   bytes memory _sig
 ) public payable {
...
   address payable receiver = payable(devAddr_);
   receiver.transfer(msg.value);
   mintNFT(msg.sender, pack.numberOfGangster);

```

```

    emit BuyStartPack(msg.sender, packId, pack.numberOfGangster);
  }
```

Location

`./contracts/GangsterArena.sol`

Recommendation

It is recommended to use a low-level `call` function which does now include this weakness.

Comments

6.5 The `_refAddr` lacks input validation

Severity: Low

Status: Acknowledged

Description

The `buyAsset` function allows now to set the `_refAddr` address along with the `_refValue` that represent the amount of Greed tokens that should be transferred to this address. However, this particular address lacks input validation against zero address. Thus, it is possible that a referral amount will be sent to an existing account due to human error or mistake in off-chain processing.

The function transfers tokens to other accounts as well, however, these accounts are set in the contract's constructor by the protocol owner.

```
function buyAsset(
    uint256 _typeA,
    uint256 _amount,
    uint256 _value,
    uint256 _lastB,
    uint256 _sTime,
    address _refAddr,
    uint256 _refValue,
    uint256 _nonce,
    bytes memory _sig
  ) public {
...


    if (burnValue > 0) greedToken.burn(burnValue);
    if (auctionValue > 0) greedToken.transfer(auctionTreasury_, auctionValue);
    if (devValue > 0) greedToken.transfer(devAddr_, devValue);
    if (_refValue > 0) greedToken.transfer(_refAddr, _refValue);
...
}
```

Location

`./contracts/GangsterArena.sol`

Recommendation

It is recommended to implement input validation against zero address set.

Comments

6.6 The `_calculateMintResult` can revert due to integer overflow

Severity: Low
Status: Fixed

Description

The `_calculateMintResult` function suffers from the integer overflow weakness, that depends on the `amount` input parameter. The `1000 **` i calculation can revert with integer overflow error whenever the `amount` is equal to 27. Such instances will prevent the execution of `entropyCallback` and `manualCallback` functions. However, the vulnerability surface is limited and it depends on the `MAX_PER_BATCH` parameter for a particular phase. If only it is set for 27 or more, it enables the possibility to trigger vulnerable scenarios.

The Client's team confirmed that it plans to set the `MAX_PER_BATCH` parameter to around 10-20, thus the likelihood is significantly decreased.

```
function _calculateMintResult(uint256 randomNumber, uint256 amount) public view returns (uint256[] memory) {
  uint256[] memory arr = new uint256[](amount);
  for (uint16 i = 0; i < amount; i++) {
    uint256 rad = uint256((randomNumber / (1000 ** i)) % 1000);
    arr[i] = _calculateProbability(rad);
  }
  return arr;
}
```

Location

`./contracts/Minter.sol`

Recommendation

It is recommended to adjust the calculation of the rad parameter to eliminate the integer overflow possibility.

Comments

6.7 Cross-chain signature replay possible

Severity: Low

Status: Fixed

Description

Within the `mint` function, for the whitelisted phase the signature-based authentication is used to allow request minting. However, the signed message includes only business-related input parameters, and it lacks any security related parameters. As the solution is firstly deployed on the blast-testnet, all signatures used for minting transactions in this blockchain can be replayed on the production blockchain.

```
 function mint(uint256 phaseId_, uint16 amount_, bytes32 userRandomNumber_, bytes memory sig_) public payable {
...
   if (mintPhase[phaseId_].whitelistedOnly) {
     bytes32 message = SignerLib.prefixed(
       keccak256(abi.encodePacked(msg.sender, phaseId_, amount_, userRandomNumber_))
     );
     require(signer.verifyAddressSigner(message, sig_), 'Invalid signature'); // validate signature //signature EIP-712, deadline, chainId, nonce?
   }
...


   emit RequestMint(msg.sender, amount_, phaseId_, sequenceNumber);
 }
```

Location

`./contracts/Minter.sol`

Recommendation

It is recommended to add cross-chain signature replay protection, e.g. include `block.chainId` in signed message.

Comments

6.8 Insufficient fees input validation

Severity: Low

Status: Fixed

Description

Within the mint function the input validation checks whether msg.value is sufficient for calculated totalFee value However, it does not check whether it is sufficient for sole pythFee whenever BASE_PRICE is 0 and protocol offers free of charge minting. Thus, it can revert with unexpected errors.

```
 function mint(uint256 phaseId_, uint16 amount_, bytes32 userRandomNumber_, bytes memory
     sig_) public payable {
...
   uint256 mintFee = amount_ * mintPhase[phaseId_].BASE_PRICE;
   uint pythFee = entropy.getFee(provider);
   uint totalFee = pythFee + mintFee;
   if (mintPhase[phaseId_].BASE_PRICE > 0) {
     require(msg.value >= totalFee, 'Send more eth');
   }
...
 }
```

Location

./contracts/Minter.sol

Recommendation

It is recommended to improve the fees input validation to handle the aforementioned scenario.

Comments

6.9 Lack of two-step ownership pattern

Severity: Low

Status: Acknowledged

Description

The Minter contract implements Ownable which implements step ownership transfer. In the event of the ownership transfer to the incorrect address, the access to all functions protected by the onlyOwner modifier will be permanently lost.

```
 address private signer; // Signer address
```

Location

./contracts/Minter.sol

Recommendation

It is recommended to introduce a two-step ownership pattern, where the new owner must confirm the transfer in a separate transaction.

Comments

6.10 Lack of singer update possibility

Severity: Low
Status: Fixed

Description

The `Minter` contract uses signature-based authorisation. It sets the signer address in the constructor. However, it does not contain any function to update this address. In the event of private key compromisation, the address hijackers would be capable of generating valid signatures permanently.

```
address private signer; // Signer address
```

Location

`./contracts/Minter.sol`

Recommendation

It is recommended to introduce signer address update functionality.

Comments

6.11 Lack of two-step ownership transfer

Severity: Low
Status: Acknowledged

Description

The protocol facilitates the authorization by means of `AccessControl` and `AccessControlUpgradeable` contracts. These extensions allow the use of role-based authorization with the possibility of enumerating the members of each role. However, OpenZepplin offers the

AccessControlDefaultAdminRules and AccessControlDefaultAdminRulesUpgradeable extensions as well, which have additional security-related benefits implemented:

- Only one account can hold the DEFAULT_ADMIN_ROLE since deployment until it is potentially renounced.
- Enforces a 2-step process to transfer the DEFAULT_ADMIN_ROLE to another account.
- Enforces a configurable delay between the two steps, with the ability to cancel before the transfer is accepted. The delay is also configurable.

Thus, contracts are missing a two-step ownership-transfer process among the others. As a result, in case of mistaken ownership transfer to the invalid account, all administrative functionalities may become unavailable for contract owners.

```
contract GangsterArena is AccessControl, ReentrancyGuard, IGangsterArena {
[...]
```

Location

./contracts/GangsterArena.sol

Recommendation

We suggest inheriting, implementing and configuring the superior AccessControlDefaultAdminRules and AccessControlDefaultAdminRulesUpgradeable extensions.

Comments

6.12 Mint can be called with 0 amount

Severity: Info

Status: Fixed

Description

Within the mint function the amount_ input parameter must be provided. However, whenever the BASE_PRICE is equal to 0, the algorithm allows to call this function with amount_ set to 0, which eventually results in reverted processing.

```
function mint(uint256 phaseId_, uint16 amount_, bytes32 userRandomNumber_, bytes memory sig_) public payable {
  require(mintPhase[phaseId_].status, 'Mint phase is not available'); // check phase status
  require(mintPhase[phaseId_].MAX_PER_BATCH >= amount_, 'Over max per batch'); // Limit max per batch
  require(requestedRandomNumber[userRandomNumber_] == 0, 'Rad is used');
```

```
...
```

Location

`./contracts/Minter.sol`

Recommendation

It is recommended to assert whether `amount_` is above the 0.

Comments

6.13 Transfer is used instead of low-level call function

Severity: Info

Status: Fixed

Description

Within the `withdraw` and `emegencyWithdraw` functions use the transfer function to send native tokens. This function has limited Gas provided to the receiver. Thus, in certain circumstances, the transfer may revert due to `Out Of Gas` error if only the receiver's `receive` function contains any business logic to execute. The finding was reported as a deviation from the leading security practices.

```
 function withdraw() public onlyOwner {
   require(address(this).balance > 0, 'Nothing to withdraw');
   require(treasuryAddr != address(0), 'Treasury is not set');
   address payable receiver = payable(treasuryAddr);
   receiver.transfer(address(this).balance);
 }
...
 function emegencyWithdraw() public onlyOwner {
   require(address(this).balance > 0, 'Nothing to withdraw');
   address payable receiver = payable(msg.sender);
   receiver.transfer(address(this).balance);
 }
```

Location

`./contracts/Minter.sol`

Recommendation

It is recommended to use a low-level `call` function which does not include this weakness.

Comments

6.14 Lack of zero address input validation

Severity: Info

Status: Acknowledged

Description

It was identified that constructor and multiple functions update the state properties of the `address` type, however, there is no check whether the provided input value is not a zero address. Such validation might be important for e.g. `signer` parameters.

The finding was reported as a deviation from the leading security practices.

```
 function withdraw() public onlyOwner {
   require(address(this).balance > 0, 'Nothing to withdraw');
   require(treasuryAddr != address(0), 'Treasury is not set');
   address payable receiver = payable(treasuryAddr);
   receiver.transfer(address(this).balance);
 } constructor(address initialOwner, address _gangster, address _signer) Ownable(
     initialOwner) {
   gangster = Gangster(_gangster);
   signer = _signer;
   BLAST.configureClaimableGas();
 }
...
 function setGasFeeOperator(address _gasFeeOperator) public onlyOwner {
   gasFeeOperator = _gasFeeOperator;
 }


 function setTreasuryAddress(address _addr) public onlyOwner {
   treasuryAddr = _addr; // Max supply for presale
 }


 function setWorker(address _addr) public onlyOwner {
   worker = _addr; // Max supply for presale
 }
```

Location

`./contracts/Minter.sol`

Recommendation

It is recommended to implement input validation against zero-address value.

Comments

6.15 Unused properties in Phase structure

Severity: Info
Status: Fixed

Description

The `Phase` struct has two unused properties anywhere in the code: `MIN_VALUE` and `MAX_VALUE`.

The `Probability` struct has a single unused property anywhere in the code: `MAX_PROBAILITY`. It is only used to hold `1000` value within the `updateProbability` function.

```
struct Phase {
  uint32 MAX_PER_BATCH;
  uint256 BASE_PRICE;
  bool status;
  bool whitelistedOnly;
  uint32 MIN_VALUE;
  uint32 MAX_VALUE;
}


struct Probability {
  uint probability;
  uint gangsterReward;
  uint MAX_PROBAILITY;
}
```

Location

`./contracts/Minter.sol`

Recommendation

It is recommended to remove unused properties from struct to save some Gas.

Comments

6.16 `RevealMint` is emitted always with the same phase

Severity: Info
Status: Fixed

Description

The `RevealMint` event is supposed to be emitted with `phaseId` related to the current mint. However, it is always emitted with `1` instead. This weakness may have a negative impact on the off-chain processing.

```
  event RevealMint(address addr, uint256 amount, uint256 phaseId, uint64 sequenceNumber);


  function entropyCallback(uint64 _sequence, address _provider, bytes32 _randomNumber)
      internal override {
...
    emit RevealMint(playerAddress, total, 1, _sequence);
  }


  function manualCallback(uint64 _sequence, bytes32 _randomNumber) public {
...
    emit RevealMint(playerAddress, total, 1, _sequence);
  }
```

Location

`./contracts/Minter.sol`

Recommendation

It is recommended to emit the `phaseId` event with valid `phaseId` value every time.

Comments

6.17 The encodePacked is used for signature based authorization

Severity: Info

Status: Acknowledged

Description

The `mint` function uses `abi.encodePacked` to encode input data for signed messages before signature check. The Solidity documentation warns:

`If you use abi.encodePacked for signatures, authentication or data integrity, make sure to always use the same types and check that at most one of them is dynamic. Unless there is a compelling reason, abi.encode should be preferred.`

The finding was reported as a deviation from leading security practices.

```
...
    if (mintPhase[phaseId_].whitelistedOnly) {
      bytes32 message = SignerLib.prefixed(
        keccak256(abi.encodePacked(msg.sender, phaseId_, amount_, userRandomNumber_))
```

```
     );
     require(signer.verifyAddressSigner(message, sig_), 'Invalid signature');
   }
...
```

Location

`./contracts/Minter.sol`

Recommendation

It is recommended to consider usage of `abi.encode` instead of `abi.encodePacked` when verifying signatures.

Comments

6.18 Blast-testnet Entropy is in use

Severity: Info

Status: Acknowledged

Description

The `Minter` contract has hard-coded the Entropy address that points to Blast-testnet blockchain. Reference: https://docs.pyth.network/entropy/contract-addresses

```
IEntropy entropy = IEntropy(0x98046Bd286715D3B0BC227Dd7a956b83D8978603);
```

Location

`./contracts/Minter.sol`

Recommendation

It is recommended to redesign solution architecture so that it takes the valid address from the deployment configuration file.

Comments

6.19 The `ReentrancyGuard` is inherited but not used

Severity: Info

Status: Acknowledged

Description

The `GangsterArena` contract inherits from the `ReentrancyGuard` library, but it does not use it in any function.

```
contract GangsterArena is AccessControl, ReentrancyGuard, IGangsterArena {
```

Location

`./contracts/GangsterArena.sol`

Recommendation

It is recommended to remove the usage of unused libraries to save Gas.

Alternatively, it is recommended to apply the `nonReentrant` modifier to functions that process native tokens.

Comments

6.20 The `keccak256` usage with `abi.encodePacked`

Severity: Info
Status: Acknowledged

Description

The solution uses signature based authorisation for `spin`, `pickCrew` and `buyAsset` functions. It was identified that to generate the `keccak256` hash the result of the `abi.encodePacked` function is used. The `abi.encodePacked`is considered an ambiguous encoding (https://docs.soliditylang.org/en/latest/abi-spec.html#non-standard-packed-mode). Thus, it is not recommended for use with signatures, authentication or data integrity, as it can lead to hash collisions.

In the implementation, only variables of the `uint256` type were used, so each time all 32 bytes were encoded separately. This feature mitigates the possibility of hash collision instances. Nevertheless, in the future updates the dynamic types might be used, which may lead to security vulnerabilities.

```
[...]
   bytes32 message = SignerLib.prefixed(
     keccak256(abi.encodePacked(msg.sender, _typeA, _amount, _value, _lastB, _sTime,
         _nonce))
   );

[...]
   bytes32 message = SignerLib.prefixed(
     keccak256(abi.encodePacked(msg.sender, _spinType, _amount, _value, _lastSpin, _sTime, _nonce))
   );

[...]
   bytes32 message = SignerLib.prefixed(keccak256(abi.encodePacked(msg.sender, _value, _sTime, _nonce)));

[...]
```

Location

`./contracts/GangsterArena.sol`

Recommendation

It is recommended to use abi.encode whenever a `keccak256` hash is created for signing purposes.

Comments