Competitive Security Assessment

Tevaera

Apr 13th, 2023

Secure3

secure3.io

Summary 4

Overview 5

Audit Scope 6

Code Assessment Findings 7

TVA-1:Claim Contract doesn't update core contract addresses 10

TVA-2:Functional risk in `CitizenIDV1` contract `blacklistAddresses` function 11

TVA-3:Gas Optimization: Use call data instead of memory 14

TVA-4:Gas optimization: Cache array length outside for loop 17

TVA-5:In the contract CitizenIDV1 function claim() allows claimContract to mint tokens even when claiming is disabled 19

TVA-6:Inconsistent comment in `_transfer` function 21

TVA-7:Incorrect usage of solidity builtin funciton `blockhash` 22

TVA-8:Integer overflow risk in `KarmaPointV1` contract `buy` function 24

TVA-9:KarmaPointV1 _transfer function Data input boolean logic error 28

TVA-10:Meaningless judgment 29

TVA-11:Missing `zksloc` configuration 30

TVA-12:Missing parameter check in `CitizenIDV1::updateRep` 32

TVA-13:Missing 0 address check 34

TVA-14:Missing array length check in `CitizenIDV1::updateRep` 36

TVA-15:Multiple tokens can be minted by an EOA in the contract NomadicYetiV1 39

TVA-16:Overwriting Unclaimed Karma Points in sync function 40

TVA-17:Redundant nonReentrant modifier 41

TVA-18:Unlock pragma used in multiple contracts 43

TVA-19:Unnecessary modifier -- `whenNotPaused` 45

TVA-20:Unused Variables 46

TVA-21:Unused `_burn` function 48

TVA-22:Use `send` may cause out of gas 49

TVA-23:Use unsafe and outdated function(`.send` . `transfer`) to transfer ETH 51

TVA-24:Weak Sources of Randomness 52

TVA-25:`CitizenIDV1::setTokenPrice` need more restrictions 58

TVA-26:`KarmaPointV1::updatePrice` need more restrictions 61

TVA-27:`mint()` will revert with very high probability due to the wrong implementation of `getRandomAvailableTokenId` 66

TVA-28:`msg.value` is not strictly checked in `claim()` 70

TVA-29:`send` method is not supported in zkSync 72

Disclaimer 74

Summary

Tevaera is a gaming ecosystem unlocking the next era of on-chain games on zkSync Era. Tevaera ecosystem features On-chain Gaming Infrastructure and Teva Games. A one-stop shop for game developers to launch their onchain games that keeps player experience on the forefront without compromising on security and fair gameplay. Tevaera's vision is to empower and onboard the next 100 million gamers to crypto with a simplified user experience, while driving the adoption of on-chain multiplayer gaming.

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

Project Detail

Project Name	Tevaera
Platform & Language	Solidity
Codebase	• https://github.com/tevaera-labs/contracts • audit commit - 69107a13161632915a625837fd84bed08d92f4a2 • final commit - 16466d7caa910538f70ff81e726afb32488fa747
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

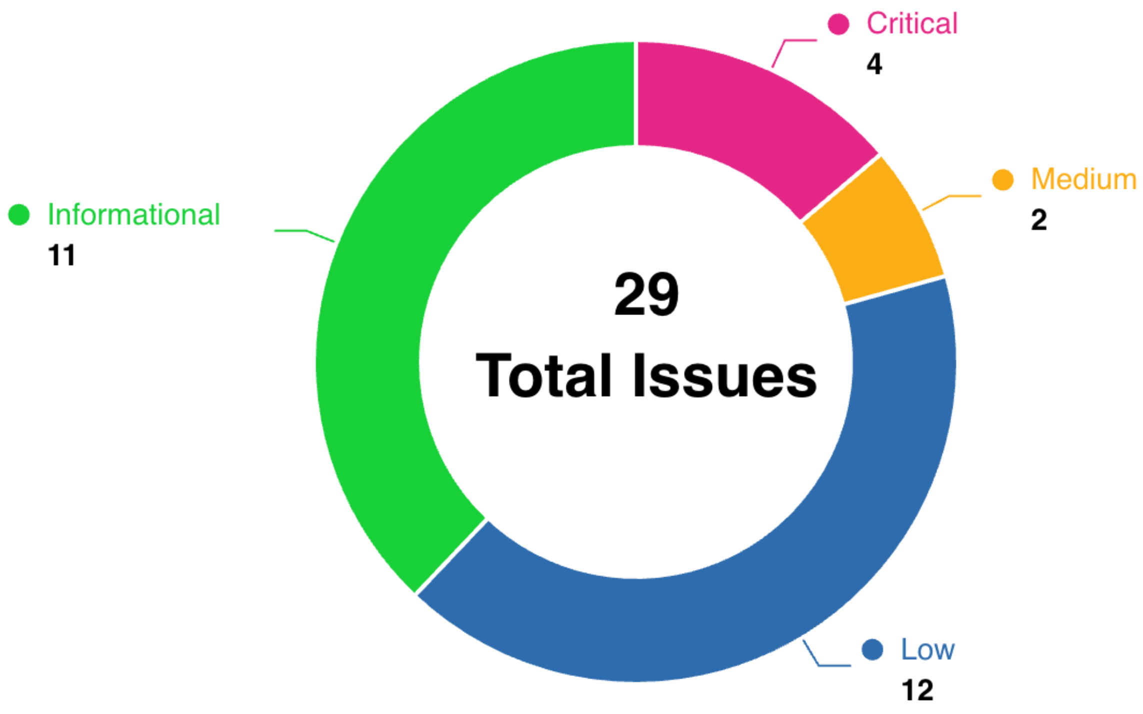
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	4	0	0	4	0	0
Medium	2	0	0	2	0	0
Low	12	0	3	9	0	0
Informational	11	0	0	10	0	1

Audit Scope

File	Commit Hash
contracts/citizenid/CitizenIDV1.sol	69107a13161632915a625837fd84bed08d92f4a2
contracts/guardians/MagicalPhoenixV1.sol	69107a13161632915a625837fd84bed08d92f4a2
contracts/guardians/NomadicYetiV1.sol	69107a13161632915a625837fd84bed08d92f4a2
contracts/karmapoint/KarmaPointV1.sol	69107a13161632915a625837fd84bed08d92f4a2
contracts/guardians/ReformistSphinxV1.sol	69107a13161632915a625837fd84bed08d92f4a2
contracts/proxy/transparent/TransparentUpgradeableProxy.sol	69107a13161632915a625837fd84bed08d92f4a2
contracts/proxy/transparent/ProxyAdmin.sol	69107a13161632915a625837fd84bed08d92f4a2
contracts/karmapoint/Claim.sol	69107a13161632915a625837fd84bed08d92f4a2

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
TVA-1	Claim Contract doesn't update core contract addresses	Logical	Low	Fixed	zeroxvee
TVA-2	Functional risk in `CitizenIDV1` contract `blacklistAddresses` function	Logical	Informational	Fixed	newway55
TVA-3	Gas Optimization: Use call data instead of memory	Gas Optimization	Informational	Fixed	newway55, Hupixiong3
TVA-4	Gas optimization: Cache array length outside for loop	Gas Optimization	Informational	Fixed	Hupixiong3

TVA-5	In the contract CitizenIDV1 function claim() allows claimContract to mint tokens even when claiming is disabled	Access Control	Medium	Fixed	n16h7m4r3
TVA-6	Inconsistent comment in `_transfer` function	Logical	Low	Fixed	jayphbee
TVA-7	Incorrect usage of solidity builtin funciton `blockhash`	Language Specific	Low	Fixed	jayphbee
TVA-8	Integer overflow risk in `KarmaPointV1` contract `buy` function	Integer Overflow	Critical	Fixed	Secure3
TVA-9	KarmaPointV1 _transfer function Data input boolean logic error	Logical	Critical	Fixed	zeroxvee
TVA-10	Meaningless judgment	Logical	Informational	Fixed	Hupixiong3
TVA-11	Missing `zksloc` configuration	Logical	Informational	Fixed	Secure3
TVA-12	Missing parameter check in `CitizenIDV1::updateRep`	Logical	Low	Fixed	zeroxvee
TVA-13	Missing 0 address check	Logical	Low	Fixed	Hupixiong3
TVA-14	Missing array length check in `CitizenIDV1::updateRep`	Logical	Low	Fixed	Hupixiong3, zeroxvee
TVA-15	Multiple tokens can be minted by an EOA in the contract NomadicYetiV1	Logical	Low	Fixed	n16h7m4r3
TVA-16	Overwriting Unclaimed Karma Points in sync function	Logical	Low	Fixed	jayphbee
TVA-17	Redundant nonReentrant modifier	Gas Optimization	Informational	Fixed	zeroxvee
TVA-18	Unlock pragma used in multiple contracts	Code Style	Informational	Fixed	n16h7m4r3
TVA-19	Unnecessary modifier -- `whenNotPaused`	Gas Optimization	Informational	Fixed	Hupixiong3
TVA-20	Unused Variables	Gas Optimization	Informational	Fixed	jayphbee, Hupixiong3, n16h7m4r3
TVA-21	Unused `_burn` function	Gas Optimization	Informational	Declined	jayphbee
TVA-22	Use `send` may cause out of gas	Code Style	Informational	Fixed	jayphbee

TVA-23	**Use unsafe and outdated function(`.send` `.transfer`) to transfer ETH**	**Logical**	**Low**	**Fixed**	**zeroxvee**
TVA-24	**Weak Sources of Randomness**	**Weak Sources of Randomness**	**Low**	**Acknowledged**	**newway55, Hupixiong3, zeroxvee**
TVA-25	**`CitizenIDV1::setTokenPrice` need more restrictions**	**Flashloan attack.**	**Low**	**Acknowledged**	**newway55**
TVA-26	**`KarmaPointV1::updatePrice` need more restrictions**	**Governance Manipulation**	**Low**	**Acknowledged**	**newway55**
TVA-27	**`mint()` will revert with very high probability due to the wrong implementation of `getRandomAvailableTokenId`**	**Logical**	**Critical**	**Fixed**	**jayphbee**
TVA-28	**`msg.value` is not strictly checked in `claim()`**	**Logical**	**Medium**	**Fixed**	**jayphbee**
TVA-29	**`send` method is not supported in zkSync**	**Language Specific**	**Critical**	**Fixed**	**Hupixiong3**

TVA-1:Claim Contract doesn't update core contract addresses

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/contracts/karmapoint/Claim.sol#L25	Fixed	zeroxvee

Code

```
25:    constructor(CitizenIDV1 _citizenIdContract, KarmaPointV1 _kpContract) {
```

Description

zeroxvee : In `Claim` contract, both `CitizenIDV1` and `KarmaPoint` contracts are defined during contract creation. `Claim` contract doesn't introduce upgradeability or update functions for core contracts `CitizenIDV1` and `KarmaPoint`.

Recommendation

zeroxvee : Either make Claim upgradeable or add `setCitizenIdContract` and `setKpContract` functions.

```
    function setCitizenIdContract(address _newCitizenIdContract) {
        citizenIdContract = _newCitizenIdContract;
    }

        function setKpContract(address _newKpContract) {
        kpContract = _newKpContract;
    }
```

Client Response

Fixed by adding new functions to set citizen id and kp contracts.

TVA-2:Functional risk in `CitizenIDV1` contract `blacklistAddresses` function

Category	Severity	Code Reference	Status	Contributor
Logical	**Informational**	• **code/contracts/citizenid/CitizenIDV1.sol#L87**	**Fixed**	**newway55**

Code

```
87:    function blacklistAddresses(address[] memory addresses) external onlyOwner {
```

Description

newway55 : The address(0) can be blacklisted, and this implies that it will block some use cases for address(0) such as :

- usage of function `_beforeTokenTransfer()` for example. This function is used in the contract **ReformistSphinxV1.sol**. The `_beforeTokenTransfer()` has these particularities : *When from and to are both non-zero, from's tokenId will be transferred to. When from is zero, tokenId will be minted for to.*
- Address(0) is in this case very important for the right execution of the function.

It is preferable to verify that address(0) cannot be blacklisted.

Consider below POC contract

Foundry testing :

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "forge-std/Test.sol";
import "../src/CitizenIDV1.sol";

contract CitizenTest is Test {
    CitizenIDV1 public citizen;

    function setUp() public {
        citizen = new CitizenIDV1();
        citizen.initialize("Uri", 200);
    }


    function test_blacklist_addressZero() public {

        address[] memory addresses = new address[](1);
        addresses[0] = address(0);
        citizen.blacklistAddresses(addresses);

        assertEq(citizen.blacklisted(addresses[0]), true);
    }
}
```

Recommendation

newway55 : Check that `address(0)` cannot be blacklisted.

Consider below fix in the `blacklistAddresses` function

```
function blacklistAddresses(address[] memory addresses) external onlyOwner {
        for (uint256 i = 0; i < addresses.length; i++) {
        require(addresses[i] != address(0), "Invalid address");
            if (!blacklisted[addresses[i]]) {
                blacklisted[addresses[i]] = true;
            }
        }
    }
```

Client Response

Added necessary checks

TVA-3:Gas Optimization: Use call data instead of memory

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	**Informational**	• **code/contracts/citizenid/CitizenIDV1.sol#L69** • **code/contracts/citizenid/CitizenIDV1.sol#L87** • **code/contracts/citizenid/CitizenIDV1.sol#L98** • **code/contracts/citizenid/CitizenIDV1.sol#L198** **code/contracts/citizenid/CitizenIDV1.sol#L226** • **code/contracts/citizenid/CitizenIDV1.sol#L226** • **code/contracts/citizenid/CitizenIDV1.sol#L227**	**Fixed**	**newway55, Hupixiong3**

Code

```
69:        string memory _tokenImageUri,

87:    function blacklistAddresses(address[] memory addresses) external onlyOwner {

98:        address[] memory addresses

198:        string memory _tokenImageUri

226:        uint256[] memory _tokenIds,

226:        uint256[] memory _tokenIds,

227:        uint256[] memory _reps
```

Description

newway55 : Use calldata instead of memory for function parameters that represent variables that will not be modified.

Consider below POC contract :

run : `forge test --gas-report`

```
function updateRep(
        uint256[] memory _tokenIds,
        uint256[] memory _reps
    ) external onlyRepAdmin {
        for (uint256 i = 0; i < tokens.length; i++) {
            uint256 tokenId = _tokenIds[i];
            uint256 rep = _reps[i];

            // update token uri (metadata)
            _setTokenURI(tokenId, getTokenURI(tokenId, rep));

            tevanRep[ownerOf(tokenId)] = rep;

            emit RepScoreUpdated(tokenId, rep);
        }
    }
```

Hupixiong3 : Using calldata saves GAS compared to memory.

Recommendation

newway55 : Consider below fix in the `updateRep` function

```
function updateRep(
        uint256[] calldata _tokenIds,
        uint256[] memory _reps
    ) external onlyRepAdmin {
        for (uint256 i = 0; i < _tokenIds.length; i++) {
            uint256 tokenId = _tokenIds[i];
            uint256 rep = _reps[i];

            // update token uri (metadata)
            _setTokenURI(tokenId, getTokenURI(tokenId, rep));

            tevanRep[ownerOf(tokenId)] = rep;

            emit RepScoreUpdated(tokenId, rep);
        }
    }
```

Hupixiong3 : Using calldata

Client Response

Fixed

TVA-4:Gas optimization: Cache array length outside for loop

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	**Informational**	• **code/contracts/citizenid/CitizenIDV1.sol#L88-L91** • **code/contracts/citizenid/CitizenIDV1.sol#L100-L104** • **code/contracts/citizenid/CitizenIDV1.sol#L229-L239**	**Fixed**	**Hupixiong3**

Code

```
88:         for (uint256 i = 0; i < addresses.length; i++) {
89:             if (!blacklisted[addresses[i]]) {
90:                 blacklisted[addresses[i]] = true;
91:             }

100:         for (uint256 i = 0; i < addresses.length; i++) {
101:             if (blacklisted[addresses[i]]) {
102:                 blacklisted[addresses[i]] = false;
103:             }
104:         }

229:         for (uint256 i = 0; i < _tokenIds.length; i++) {
230:             uint256 tokenId = _tokenIds[i];
231:             uint256 rep = _reps[i];
232:
233:             // update token uri (metadata)
234:             _setTokenURI(tokenId, getTokenURI(tokenId, rep));
235:
236:             tevanRep[ownerOf(tokenId)] = rep;
237:
238:             emit RepScoreUpdated(tokenId, rep);
239:         }
```

Description

Hupixiong3 : There is GAS optimization space in the cycle structure.

Recommendation

Hupixiong3 : GAS optimization of the cycle structure was carried out.

Consider below fix in the `CitizenIDV1.updateRep()` function

```
    function updateRep(
        uint256[] calldata _tokenIds,
        uint256[] calldata _reps
    ) external onlyRepAdmin {
        uint256 len = _tokenIds.length;
        for (uint256 i = 0; i < len; ) {
            uint256 tokenId = _tokenIds[i];
            uint256 rep = _reps[i];

            // update token uri (metadata)
            _setTokenURI(tokenId, getTokenURI(tokenId, rep));

            tevanRep[ownerOf(tokenId)] = rep;

            emit RepScoreUpdated(tokenId, rep);
            unchecked { ++i; }
        }
    }
```

Client Response

Fixed

TVA-5:In the contract CitizenIDV1 function claim() allows claimContract to mint tokens even when claiming is disabled

Category	Severity	Code Reference	Status	Contributor
Access Control	**Medium**	• **code/contracts/citizenid/CitizenIDV1.sol#L125-L128** • **code/contracts/citizenid/CitizenIDV1.sol#L211-L220**	**Fixed**	**n16h7m4r3**

Code

```
125:    /// @dev Mints the Citizen ID
126:    function claim(address tevan) external payable onlyClaim whenNotPaused {
127:        mintToken(tevan);
128:    }

211:    /// @dev Allows owner to update claim capability
212:    /// @param _claimContract the claim contract address
213:    /// @param _canClaim a flag to indicate whether to enable or disable the capability
214:    function updateClaimCapability(
215:        address _claimContract,
216:        bool _canClaim
217:    ) external onlyOwner {
218:        claimContract = _claimContract;
219:        canClaim = _canClaim;
220:    }
```

Description

n16h7m4r3 : Boolean state of the variable `canClaim` is set by the `owner` using the function `updateClaimCapability()`, allowing `claimContract` to mint a `TEVAN` token for a user. The function `claim()` does not check the state of `canClaim` allowing `claimContract` to mint tokens for a user when `canClaim` is set to `False`.

Recommendation

n16h7m4r3 : Implement checks to ensure that minting of `TEVAN` tokens by the `claimContract` contract is only possible when claiming is enabled i.e `canClaim` is set to `True`.

Client Response

Fixed

TVA-6:Inconsistent comment in `_transfer` function

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/contracts/karmapoint/KarmaPointV1.sol#L231-L235	Fixed	jayphbee

Code

```
231:        // sender should have citizenship
232:        require(
233:            citizenIdContract.balanceOf(to) > 0,
234:            "Receiver is not a Tevan!"
235:        );
```

Description

jayphbee : In the `_transfer` function, the comment "sender should have citizenship" is not consistent with the actual code. The code checks if the receiver has citizenship, not the sender.

```
// sender should have citizenship
        require(
            citizenIdContract.balanceOf(to) > 0,
            "Receiver is not a Tevan!"
        );
```

Recommendation

jayphbee : The protocol developer should have a check if the code does follow the design spec.

Client Response

Fixed

TVA-7:Incorrect usage of solidity builtin funciton `blockhash`

Category	Severity	Code Reference	Status	Contributor
Language Specific	Low	• code/contracts/guardians/NomadicYetiV1.sol#L196 • code/contracts/guardians/MagicalPhoenixV1.sol#L196	Fixed	jayphbee

Code

```
196:                                blockhash(block.number),

196:                                blockhash(block.number),
```

Description

jayphbee : `blockhash` function will return the hash of the most recent 256 blocks, excluding the current block, otherwise it will return zero. The blockhash is used as entropy to derive the `randomNum` in the `getRandomAvailableTokenId` function.

```
uint256 randomNum = uint256(
            keccak256(
                abi.encodePacked(
                    keccak256(
                        abi.encode(
                            to,
                            nonce,
                            tx.gasprice,
                            block.number,
                            block.timestamp,
                            block.prevrandao,
                            blockhash(block.number),
                            address(this)
                        )
                    )
                )
            )
        ) % MAX_YETIS;
```

This code use the current block to get the block hash, so it will return zero. That is to say the `randomNum` is less "random" than expected because it always returns zero.

Recommendation

jayphbee : Use the previous blockhash or anyother of the recent 256 blocks.

```
blockhash(block.number - 1)
```

Client Response

Fixed

TVA-8:Integer overflow risk in `KarmaPointV1` contract `buy` function

Category	Severity	Code Reference	Status	Contributor
Integer Overflow	Critical	• code/contracts/karmapoint/KarmaPointV1.sol#L125 • code/contracts/karmapoint/KarmaPointV1.sol#L181	Fixed	Secure3

Code

```
125:        require(msg.value == getPrice(kpAmount), "Invalid amount");

181:            return (price * kpAmount);
```

Description

Secure3 :

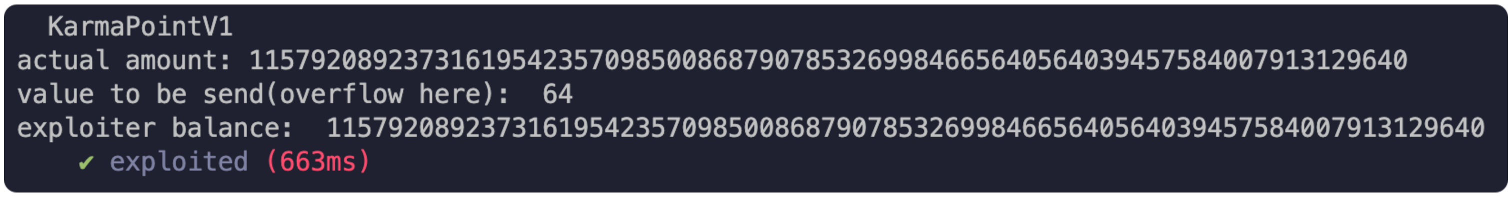
```
function buy(
        uint256 kpAmount
    ) external payable isTevan isNotBlacklisted nonReentrant whenNotPaused {
        // make sure if doesn't exceed the total cap
        require(totalSupply() + kpAmount <= cap(), "Exceeds total cap");

        // make sure if doesn't exceed the individual buying cap
        uint256 kpBalance = kpAmount + boughtKP[msg.sender];
        require(kpBalance <= buyCap, "Exceeds buying cap");
        // make sure the amount passed is matching the kp value
        require(msg.value == getPrice(kpAmount), "Invalid amount"); // could be an integer overflow
here

        boughtKP[msg.sender] += kpAmount;
        _mint(msg.sender, kpAmount);
    }
```

In `buy` function, the price a player need to pay is calculated from `getPrice` function, which is as below:

```
    function getPrice(uint256 kpAmount) public view returns (uint256) {
            unchecked {
                return (price * kpAmount);
            }
        }
```

as you can see, it has an integer overflow risk, if a user pass a specific value to the function, after multiplied by the price, if the result>=2**256, the it will overflow。

Back to the contract, hacker may gain too much value just to pay a little price。



```
  KarmaPointV1
actual amount: 115792089237316195423570985008687907853269984665640564039457584007913129640
value to be send(overflow here):  64
exploiter balance:  115792089237316195423570985008687907853269984665640564039457584007913129640
    ✔ exploited (663ms)
```

Here is the hardhat exploit script:

```
const { expect } = require("chai");
const { ethers } = require("hardhat");


describe("KarmaPointV1", function () {
    it("exploited", async function () {
        const [owner, safe, user] = await ethers.getSigners();
        // Deploy contract
        const CitizenIDV1 = await ethers.getContractFactory("contracts/citizenid/CitizenIDV1.sol:Cit
izenIDV1");
        const CitizenIDV1Contract = await CitizenIDV1.connect(owner).deploy();
        const KarmaPointV1 = await ethers.getContractFactory("contracts/karmapoint/KarmaPointV1.sol:
KarmaPointV1");
        const KarmaPointV1Contract = await KarmaPointV1.connect(owner).deploy();
        const Claim = await ethers.getContractFactory("contracts/karmapoint/Claim.sol:Claim");
        const ClaimContract = await Claim.connect(owner).deploy(CitizenIDV1Contract.address,
KarmaPointV1Contract.address);

        // Initialize contracts
        await CitizenIDV1Contract.connect(owner).initialize("http://example.com", 1);
        const bigMaxValue = ethers.BigNumber.from(2).pow(256).sub(1); // 2**256 - 1
        const price = 1000;
        const evilPrice = bigMaxValue.div(price).add(1);
        await KarmaPointV1Contract.connect(owner).initialize(CitizenIDV1Contract.address,
safe.address, price, bigMaxValue, bigMaxValue);

        //Functional Check
        await CitizenIDV1Contract.connect(user).mintCitizenId({ value: ethers.utils.parseUnits("1.
0", "wei") });
        await KarmaPointV1Contract.connect(user).buy(evilPrice, { value: ethers.utils.parseUnits("6
4.0", "wei") });
        const balance = await KarmaPointV1Contract.connect(user).balanceOf(user.address);
        console.log("exploiter balance: ",balance.toString())
        expect(await balance).to.above(0);
    });
});
```

Recommendation

Secure3 : remove the `unchecked` tag.

Client Response

Fixed

TVA-9:KarmaPointV1 _transfer function Data input boolean logic error

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	• code/contracts/karmapoint/KarmaPointV1.sol#L224	Fixed	zeroxvee

Code

```
224:        require(
```

Description

zeroxvee : In `KarmaPoint` contract _transfer function first require check doesn't follow proper boolean logic and wrong equals equation.

```
        // this require allow me to bypass with @from and @to being equal address(0) or
        // with canTransfer = false, but @from == address(0) and
        // will throw "Unauthorized" if @from and @to anything else but address(0)
        require(
            canTransfer || from == address(0) || to == address(0),
            "Unauthorized"
        );
```

Recommendation

zeroxvee : Consider this fix, which addresses all the issues.

```
        require(
            canTransfer && from != address(0) && to != address(0),
            "Unauthorized"
        );
```

Client Response

Fixed

TVA-10:Meaningless judgment

Category	Severity	Code Reference	Status	Contributor
Logical	**Informational**	• **code/contracts/guardians/ReformistSphinxV1.sol#L81** • **code/contracts/guardians/MagicalPhoenixV1.sol#L109** • **code/contracts/guardians/NomadicYetiV1.sol#L109**	**Fixed**	**Hupixiong3**

Code

```
81:        require(balanceOf(msg.sender) == 0, "already minted");

109 :        require(balanceOf(msg.sender) == 0, "already minted");

109 :        require(balanceOf(msg.sender) == 0, "already minted");
```

Description

Hupixiong3 : The nft transfer is allowed, which causes the mint function's first check to fail.

Recommendation

Hupixiong3 : Learning the CitizenIDV1 contract allows only address 0 to be transferred.

Client Response

Added a mapper to track mins by the wallet. Using this mapper to make sure one user mints only once without impacting transfer functionality

TVA-11:Missing `zksloc` configuration

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	• code/contracts/karmapoint/KarmaPointV1.sol#L133 • code/contracts/guardians/NomadicYetiV1.sol#L135 • code/contracts/guardians/MagicalPhoenixV1.sol#L138 • code/contracts/citizenid/CitizenIDV1.sol#L263	Fixed	Secure3

Code

```
133:        require(safeAddress.send(address(this).balance));

135:        require(charityAddress.send(charityAmount));

138:        require(safeAddress.send(amountAfterCharity));

263:        require(safeAddress.send(address(this).balance));
```

Description

Secure3 : This project will be deployed on zkSync. Due to the differences between zkevm and Ethereum EVM, we need to use `zksolc` to compile the solidity files. However, the `hardhat.config.ts` configuration file is not included in the audit project, which is a potential risk if the configuration is wrong as we cannot verify it. We used version `1.3.8` of zksolc to compile and found some warnings, below just showing one of them for illustration:

```
import "@matterlabs/hardhat-zksync-deploy";
import "@matterlabs/hardhat-zksync-solc";

module.exports = {
  zksolc: {
    version: "1.3.8",
    compilerSource: "binary",
    settings: {},
  },
  solidity: {
    version: "0.8.18",
  },
};


┌────────────────────────────────────────────────────────────────────────────────────┐
| Warning: It looks like you are using '<address payable>.send/transfer(<X>)' without providing |
| the gas amount. Such calls will fail depending on the pubdata costs.                          |
| This might be a false positive if you are using some interface (like IERC20) instead of the   |
| native Solidity send/transfer                                                                 |
| Please use 'payable(<address>).call{value: <X>}("")' instead.                                 |
└────────────────────────────────────────────────────────────────────────────────────┘
--> contracts/citizenid/CitizenIDV1.sol
```

According the zkSync official document https://era.zksync.io/docs/dev/troubleshooting/changelog.html#compiler-local-setup-update-feb-20th-2023, using `send` method may failed to transfer ETH. This has also been identified in the other issues in the report.

Recommendation

Secure3 : 1. Use the newer version zksolc such as `1.3.8` in `hardhat.config.ts`
2.Use `payable(<address>).call{value: <X>}("")` instead of `<address payable>.send/transfer(<X>)`

Client Response

Added the config

TVA-12:Missing parameter check in `CitizenIDV1::updateRep`

Category	Severity	Code Reference	Status	Contributor
Logical	**Low**	• **code/contracts/citizenid/CitizenIDV1.sol#L229**	**Fixed**	**zeroxvee**

Code

```
229:        for (uint256 i = 0; i < _tokenIds.length; i++) {
```

Description

zeroxvee : function function sync(address[] memory tevans,uint256[] memory amounts) - has no input data validation. Due to possibly unhandled error during fetch on the frontend, can lead to errors in the work of the contract.

```
    function updateRep(
        uint256[] memory _tokenIds, //[2,4]
        uint256[] memory _reps      //["OG"]
    ) external onlyRepAdmin {}
    // as result token 2 will get OG and token 4 no rep at all which can quickly break the project
mechanics
    ->
```

Recommendation

zeroxvee : Consider the fix.

```
    function sync(
        address[] memory tevans,
        uint256[] memory amounts
    ) external onlyOwner {
        require(tevans.length == amounts.length, "Different arrays lengths");
        for (uint256 i = 0; i < tevans.length; i++) {
            require(tevans[i] != address(0), "Zero Address");
            require(amounts[i] > 0 address(0), "KP <= 0");
            toBeClaimedKP[tevans[i]] = amounts[i];
        }
    }
```

Client Response

Added checks

TVA-13:Missing 0 address check

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/contracts/guardians/MagicalPhoenixV1.sol#L155-L159 • code/contracts/citizenid/CitizenIDV1.sol#L214-L220	Fixed	Hupixiong3

Code

```
155:    function updateSafeAddress(
156:        address payable _safeAddress
157:    ) external onlyOwner whenNotPaused {
158:        safeAddress = _safeAddress;
159:    }

214:    function updateClaimCapability(
215:        address _claimContract,
216:        bool _canClaim
217:    ) external onlyOwner {
218:        claimContract = _claimContract;
219:        canClaim = _canClaim;
220:    }
```

Description

Hupixiong3 : The updateClaimCapability function lacks a 0 address check, and claimContract being set to 0 will render the claim function functionality unusable.

Recommendation

Hupixiong3 : Add 0 address check.

Consider below fix in the `CitizenIDV1.updateClaimCapability()` function

```
    function updateClaimCapability(
        address _claimContract,
        bool _canClaim
    ) external onlyOwner {
        require(_claimContract!= address(0), "Invalid address!");
        claimContract = _claimContract;
        canClaim = _canClaim;
    }
```

Client Response

Added checks

TVA-14:Missing array length check in `CitizenIDV1::updateRep`

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/contracts/citizenid/CitizenIDV1.sol#L225-L240 • code/contracts/citizenid/CitizenIDV1.sol#L229	Fixed	Hupixiong3, zeroxvee

Code

```
225:    function updateRep(
226:        uint256[] memory _tokenIds,
227:        uint256[] memory _reps
228:    ) external onlyRepAdmin {
229:        for (uint256 i = 0; i < _tokenIds.length; i++) {
230:            uint256 tokenId = _tokenIds[i];
231:            uint256 rep = _reps[i];
232:
233:            // update token uri (metadata)
234:            _setTokenURI(tokenId, getTokenURI(tokenId, rep));
235:
236:            tevanRep[ownerOf(tokenId)] = rep;
237:
238:            emit RepScoreUpdated(tokenId, rep);
239:        }
240:    }

229:        for (uint256 i = 0; i < _tokenIds.length; i++) {
```

Description

Hupixiong3 : In the updateRep function, when the array arguments _tokenIds and _reps are passed in different lengths, a setup error may occur.
zeroxvee : function updateRep(uint256[] memory _tokenIds,uint256[] memory _reps) - has no arrays length validation check. It can lead to wrong token given wrong Rep levels.

```
    //
    function updateRep(
        uint256[] memory _tokenIds, //[2,4]
        uint256[] memory _reps      //["OG"]
    ) external onlyRepAdmin {}
    // as result token 2 will get OG and token 4 no rep at all which can quickly break the project
mechanics
    ->
```

Recommendation

Hupixiong3 : Verify the length of the passed parameter.

Consider below fix in the `CitizenIDV1.updateRep()` function

```
    function updateRep(
        uint256[] calldata _tokenIds,
        uint256[] calldata _reps
    ) external onlyRepAdmin {
       require(_tokenIds.length==_reps.length,"Invalid length");
        uint256 len = _tokenIds.length;
        for (uint256 i = 0; i < len; ) {
            uint256 tokenId = _tokenIds[i];
            uint256 rep = _reps[i];

            // update token uri (metadata)
            _setTokenURI(tokenId, getTokenURI(tokenId, rep));

            tevanRep[ownerOf(tokenId)] = rep;

            emit RepScoreUpdated(tokenId, rep);
            unchecked { ++i; }
        }
    }
```

zeroxvee : Consider the fix. Ideally I would also check each `_tokenIds[i] != 0` and `_reps[i] != ""`,

```
    function updateRep(
        uint256[] memory _tokenIds,
        uint256[] memory _reps
    ) external onlyRepAdmin {
        require(_tokenIds.length == _reps.length, "Data input length mismatch")
        for (uint256 i = 0; i < _tokenIds.length; i++) {
            uint256 tokenId = _tokenIds[i];
            uint256 rep = _reps[i];
            //ideal variant
            require(tokenId != 0, "TokenId can't be 0")
            require(rep != "", "Empty Rep input")
        }
```

Client Response

Added checks

TVA-15:Multiple tokens can be minted by an EOA in the contract NomadicYetiV1

Category	Severity	Code Reference	Status	Contributor
Logical	**Low**	• **code/contracts/guardians/NomadicYetiV1.sol#L109** • **code/contracts/guardians/NomadicYetiV1.sol#L121**	**Fixed**	**n16h7m4r3**

Code

```
109:        require(balanceOf(msg.sender) == 0, "already minted");

121:        _mint(msg.sender, tokenId);
```

Description

n16h7m4r3 : Based on the requirements in the function `mint()` an wallet is allowed to mint only one `YETI` token in exchange for ETH. But an user can mint arbitrary amount of token to a wallet by transferring the previously minted token to an controlled wallet allowing a wallet to own multiple number of tokens.

Recommendation

n16h7m4r3 : Based on the requirements in the function mint() an wallet is allowed to mint only one YETI token in exchange for ETH. But an user can mint arbitrary amount of token to a wallet by transferring the previously minted token to an controlled wallet allowing a wallet to own multiple number of tokens.However, the caller address has been judged in L109 to prevent this state.

Client Response

Fixed

TVA-16:Overwriting Unclaimed Karma Points in sync function

Category	Severity	Code Reference	Status	Contributor
Logical	**Low**	• **code/contracts/karmapoint/KarmaPointV1.sol#L172**	**Fixed**	**jayphbee**

Code

```
172:            toBeClaimedKP[tevans[i]] = amounts[i];
```

Description

jayphbee : In the `sync` function, the unclaimed `KP` balance for each user is being overwritten. This might cause users to lose their unclaimed `KP` balance if this function is called again with new values for the same users.

Recommendation

jayphbee :

```
    function sync(
        address[] memory tevans,
        uint256[] memory amounts
    ) external onlyOwner {
        require(tevans.length == amounts.length, "Different arrays lengths");
        for (uint256 i = 0; i < tevans.length; i++) {
            toBeClaimedKP[tevans[i]] += amounts[i];
        }
    }
```

Client Response

Fixed

TVA-17:Redundant nonReentrant modifier

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/contracts/guardians/ReformistSphinxV1.sol#L51 • code/contracts/guardians/MagicalPhoenixV1.sol#L57 • code/contracts/guardians/NomadicYetiV1.sol#L57 • code/contracts/citizenid/CitizenIDV1.sol#L71	Fixed	zeroxvee

Code

```
51:    function initialize(

57:    function initialize(

57:    function initialize(

71:    ) external initializer nonReentrant {
```

Description

zeroxvee : initialize function `nonReentrant` is redundant in MagicalPhoenixV1,CitizenIDV1,ReformistSphinxV1,NomadicYetiV1 In initialize function, `nonReentrant` is redundant, even if we imagine the worst case scenario, the way `initializer` modifier works, even if reentered by a malicious actor, won't let initialize another time.

Recommendation

zeroxvee : Remove `nonReentrant` modifier.

```
function initialize(
        address payable _safeAddress,
        address payable _charityAddress,
        CitizenIDV1 _citizenIdContract,
        string memory _tokenBaseUri
    ) external initializer {}
```

Client Response

Fixed

TVA-18:Unlock pragma used in multiple contracts

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/contracts/citizenid/CitizenIDV1.sol#L2 • code/contracts/guardians/MagicalPhoenixV1.sol#L2 • code/contracts/guardians/NomadicYetiV1.sol#L2 • code/contracts/guardians/ReformistSphinxV1.sol#L2 • code/contracts/karmapoint/Claim.sol#L2 • code/contracts/karmapoint/KarmaPointV1.sol#L2 • code/contracts/proxy/transparent/ProxyAdmin.sol#L4 • code/contracts/proxy/transparent/TransparentUpgradeableProxy.sol#L4	Fixed	n16h7m4r3

Code

```
2:pragma solidity ^0.8.18;

2:pragma solidity ^0.8.18;

2:pragma solidity ^0.8.18;

2:pragma solidity ^0.8.18;

2:pragma solidity ^0.8.18;

2:pragma solidity ^0.8.18;

4:pragma solidity ^0.8.0;

4:pragma solidity ^0.8.0;
```

Description

n16h7m4r3 : Contracts should be deployed using the same compiler version/flags with which they have been tested. Locking the floating pragma, i.e. by not using ^ in pragma solidity ^0.8.0, ensures that contracts do not accidentally get deployed using an older compiler version with unfixed bugs.

Recommendation

n16h7m4r3 : In most contracts, the pragma statements are declared as pragma solidity >=0.6.0 <0.8.0;, which are unlocked and could cause the contracts to accidentally be compiled or deployed using an outdated or buggy compiler version.

Client Response

Fixed

TVA-19:Unnecessary modifier -- `whenNotPaused`

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	**Informational**	• **code/contracts/citizenid/CitizenIDV1.sol#L108** • **code/contracts/citizenid/CitizenIDV1.sol#L280**	**Fixed**	**Hupixiong3**

Code

```
108:    function mintToken(address tevan) internal whenNotPaused nonReentrant {

280:        whenNotPaused
```

Description

Hupixiong3 : The internal functions mintToken and _beforeTokenTransfer are already in modifier whenNotPaused when called.

Recommendation

Hupixiong3 : Delete redundant modifier whenNotPaused.

Client Response

Fixed

TVA-20:Unused Variables

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	**Informational**	• **code/contracts/citizenid/CitizenIDV1.sol#L30** • **code/contracts/guardians/MagicalPhoenixV1.sol#L45-L46** • **code/contracts/guardians/NomadicYetiV1.sol#L45-L46** • **code/contracts/guardians/MagicalPhoenixV1.sol#L46** • **code/contracts/guardians/NomadicYetiV1.sol#L46** • **code/contracts/citizenid/CitizenIDV1.sol#L219**	**Fixed**	**jayphbee, Hupixiong3, n16h7m4r3**

Code

```
30 :    bool public canClaim = false;

45:    uint16 private index;
46:    uint16[10000] private ids;

45:    uint16 private index;
46:    uint16[10000] private ids;

46:    uint16[10000] private ids;

46:    uint16[10000] private ids;

219 :        canClaim = _canClaim;
```

Description

jayphbee : `ids` and `index` are defined but not used in `MagicalPhoenixV1.sol` and `NomadicYetiV1.sol` contracts.
Hupixiong3 : In the CitizenIDV1 contract,bool variable canClaim, not used. The functionality associated with it has not been implemented or has been abandoned.A similar situation exists in other contracts.

n16h7m4r3 : The private variable `ids` declared in the contracts `MagicalPhoenixV1` and `NomadicYetiV1` is never used.

Recommendation

jayphbee : remove unused variable `ids` and `index` in `MagicalPhoenixV1.sol` and `NomadicYetiV1.sol` contracts.
Hupixiong3 : Delete the Unused variables or implement the corresponding function.
n16h7m4r3 : delete unused state variable can save gas fees.

Client Response

Fixed

TVA-21:Unused `_burn` function

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/contracts/guardians/ReformistSphinxV1.sol#L114 • code/contracts/guardians/MagicalPhoenixV1.sol#L241 • code/contracts/guardians/NomadicYetiV1.sol#L241 • code/contracts/citizenid/CitizenIDV1.sol#L266	Declined	jayphbee

Code

```
114:    function _burn(

241:    function _burn(

241:    function _burn(

266:    function _burn(
```

Description

jayphbee : The `_burn` function is internal, overridden, and not called anywhere in the contract. If burning tokens is not desired, consider removing the `_burn` function.

Recommendation

jayphbee : If the protocol want to allow burning tokens under specific circumstances, create a public function that calls `_burn` with necessary access controls (e.g., onlyOwner).

Client Response

If we remove burn(), we get the “Derived contract must override function "_burn" error. Just to avoid this error we override it as an internal method.

TVA-22:Use `send` may cause out of gas

Category	Severity	Code Reference	Status	Contributor
Code Style	**Informational**	• **code/contracts/karmapoint/KarmaPointV1.sol#L133** • **code/contracts/guardians/MagicalPhoenixV1.sol#L135** • **code/contracts/guardians/NomadicYetiV1.sol#L135** • **code/contracts/guardians/MagicalPhoenixV1.sol#L138** • **code/contracts/guardians/NomadicYetiV1.sol#L138** • **code/contracts/citizenid/CitizenIDV1.sol#L263**	**Fixed**	**jayphbee**

Code

```
133:        require(safeAddress.send(address(this).balance));

135:        require(charityAddress.send(charityAmount));

135:        require(charityAddress.send(charityAmount));

138:        require(safeAddress.send(amountAfterCharity));

138:        require(safeAddress.send(amountAfterCharity));

263:        require(safeAddress.send(address(this).balance));
```

Description

jayphbee : `gnosis-safe` is a smart contract wallet, its `receive()` function implemented like this

```
abstract contract NativeCurrencyPaymentFallback {
    event SafeReceived(address indexed sender, uint256 value);

    /**
     * @notice Receive function accepts native currency transactions.
     * @dev Emits an event with sender and received value.
     */
    receive() external payable {
        emit SafeReceived(msg.sender, msg.value);
    }
}
```

We can't promise that this implementation will not change, if a storage writes happened in the `receive()` function, `safeAddress.send(address(this).balance)` always reverts due to out of gas, because `send` only forwards 2300 gas to the callee.

Recommendation

jayphbee : use `call` to send ether.

Client Response

Fixed

TVA-23:Use unsafe and outdated function(`.send` `.transfer`) to transfer ETH

Category	Severity	Code Reference	Status	Contributor
Logical	**Low**	• **code/contracts/karmapoint/KarmaPointV1.sol#L132** • **code/contracts/citizenid/CitizenIDV1.sol#L261**	**Fixed**	**zeroxvee**

Code

```
132:    function withdrawFunds() external onlyOwner {

261:    function withdrawFunds() external onlyOwner {
```

Description

zeroxvee : In `withdrawFunds` - `.send` or `.transfer` are both considered unsafe and outdated.

Recommendation

zeroxvee : Use `.call` to transfer ETH and remove first require, since can be only called by owner = msg.sender

```
    function withdrawFunds() external onlyOwner {
        (bool sent, bytes memory data) = _to.call{value: address(this).balance}("");
        require(sent, "Failed to send Ether");
    }
```

Client Response

Fixed

TVA-24:Weak Sources of Randomness

Category	Severity	Code Reference	Status	Contributor
Weak Sources of Randomness	Low	• code/contracts/guardians/MagicalPhoenixV1.sol#L118 • code/contracts/guardians/NomadicYetiV1.sol#L118 • code/contracts/guardians/MagicalPhoenixV1.sol#L181 • code/contracts/guardians/NomadicYetiV1.sol#L181	Acknowledged	newway55, Hupixiong3, zeroxvee

Code

```
118 :        uint256 tokenId = getRandomAvailableTokenId(msg.sender, randomNonce);

118:        uint256 tokenId = getRandomAvailableTokenId(msg.sender, randomNonce);

181:    function getRandomAvailableTokenId(

181:    function getRandomAvailableTokenId(
```

Description

newway55 : The keccak256 function is used to generate pseudo-random numbers based on the values of various inputs. However, the code provided does not use a secure source of randomness, as it relies on block timestamp and other easily guessable values, which can potentially be manipulated by miners.
Consider below POC contract

```
function initialize(
        address payable _safeAddress,
        address payable _charityAddress,
        CitizenIDV1 _citizenIdContract,
        string memory _tokenBaseUri
    ) external initializer nonReentrant {
        __ERC721_init("MagicalPhoenix", "PHOENIX");
        __ERC721Enumerable_init();
        __ERC721Royalty_init();
        __Ownable_init();
        __Pausable_init();
        __ReentrancyGuard_init();

        safeAddress = _safeAddress;
        charityAddress = _charityAddress;
        citizenIdContract = _citizenIdContract;
        tokenBaseUri = _tokenBaseUri;

        // set default royalty to 5%
        _setDefaultRoyalty(msg.sender, 500);

        // set random nonce starting index
        randomNonce = uint256(
            keccak256(
                abi.encodePacked(
                    keccak256(
                        abi.encode(
                            tx.gasprice,
                            block.number,
                            block.timestamp,
                            block.prevrandao,
                            blockhash(block.number),
                            address(this)
                        )
                    )
                )
            )
        );

        numAvailableTokens = MAX_PHOENIXES;
    }
```

Hupixiong3 : Tokenid is generated using a random number based on chain predictability. If the tokenid is valuable, it will be pre-minted by the hacker.
zeroxvee : In function `getRandomAvailableTokenId` depending on the further game logic, weak source of randomness can play a breaking mechanics or malicious role. If tokenId in some way defines token traits, etc. Consider well-known oracles.

Recommendation

newway55 : - Use an **external source of randomness**, such as a Chainlink Oracle.
Consider below fix in the contract `MagicalPhoenixV1.sol`

```
// Import Chainlink VRF
import "@chainlink/contracts/src/v0.8/interfaces/VRFCoordinatorV2Interface.sol";
import "@chainlink/contracts/src/v0.8/VRFConsumerBaseV2.sol";


contract  MagicalPhoenixV1 is
    ERC721RoyaltyUpgradeable,
    ERC721EnumerableUpgradeable,
    OwnableUpgradeable,
    PausableUpgradeable,
    ReentrancyGuardUpgradeable
    VRFConsumerBaseV2 {

    // Chainlink PART for Randomness with zkSync compatibility :

     //---zkSync
    VRFCoordinatorV2Interface COORDINATOR;

    // Your subscription ID.
    uint64 public s_subscriptionId;


    // see https://docs.chain.link/docs/vrf-contracts/#configurations
    address vrfCoordinator = <>;


    // see https://docs.chain.link/docs/vrf-contracts/#configurations
    bytes32 keyHash =<>;


    uint32 callbackGasLimit = 100000;
    uint16 requestConfirmations = 3;
    uint32 numNonce =  2;

    uint256[] public s_randomWords;
    uint256 public s_requestId;
    address public s_owner;

   function initialize(
        address payable _safeAddress,
        address payable _charityAddress,
        CitizenIDV1 _citizenIdContract,
string memory _tokenBaseUri
    ) external initializer nonReentrant {
        __ERC721_init("MagicalPhoenix", "PHOENIX");
        VRFConsumerBaseV2(vrfCoordinator);
        __ERC721Enumerable_init();
        __ERC721Royalty_init();
        __Ownable_init();
        __Pausable_init();
        __ReentrancyGuard_init();

        safeAddress = _safeAddress;
        charityAddress = _charityAddress;
        citizenIdContract = _citizenIdContract;
        tokenBaseUri = _tokenBaseUri;

        // set default royalty to 5%
        _setDefaultRoyalty(msg.sender, 500);
        s_subscriptionId = subscriptionId;


        numAvailableTokens = MAX_PHOENIXES;
    }


    }

    // Assumes the subscription is funded sufficiently.
    function requestRandomNonce() external onlyOwner {
        // Will revert if subscription is not set and funded.
        s_requestId = COORDINATOR.requestRandomNonce(
        keyHash,
        s_subscriptionId,
        requestConfirmations,
        callbackGasLimit,
        numNonce
        );
    }

    function fulfillRandomNonce(
        uint256, /* requestId */
        uint256[] memory randomNonce
    ) internal override {
        s_randomNonce = randomNonce;
}


}
```

Hupixiong3 : Advised to use random numbers generated by oracle.
zeroxvee : Add Chainlink VRF (Verifiable Random Function). https://vrf.chain.link/

Client Response

The comment is valid and we’re aware of it. Since Chainlink is not yet available on zkSync, we’re using the pseudo-random number.

TVA-25: `CitizenIDV1::setTokenPrice` need more restrictions

Category	Severity	Code Reference	Status	Contributor
Flashloan attack.	**Low**	• **code/contracts/citizenid/CitizenIDV1.sol#L205-L209**	**Acknowledged**	**newway55**

Code

```
205:    function setTokenPrice(
206:        uint256 _tokenPrice
207:    ) external onlyOwner whenNotPaused {
208:        tokenPrice = _tokenPrice;
209:    }
```

Description

newway55 : SetTokenPrice can be changed while a transfer/mint function is occurring. This can be exploited in many ways. The `tokenPrice` is present as `msg.value` in the `mintToken` function so a risk of Flashloan attack using a higher tokenPrice is a possibility.

Consider below POC contract

```
pragma solidity ^0.8.13;

import "forge-std/Test.sol";
import "../src/CitizenIDV1.sol";

contract CitizenTest is Test {
    CitizenIDV1 public citizen;
    address internal spender;
    uint256 internal spenderPrivateKey;
    using stdStorage for StdStorage;


    function setUp() public {
        citizen = new CitizenIDV1();
        citizen.initialize("Uri", 200);
        spenderPrivateKey = 0xB0B;
        spender = vm.addr(spenderPrivateKey);
    }

   function test_setPrice() public {
        citizen.setTokenPrice(0.01 ether);
        citizen.mintCitizenId{value: 0.01 ether}();
        citizen.setTokenPrice(0.02 ether);
        assertEq(citizen.tokenPrice(), 0.02 ether);
  }

}
```

Recommendation

newway55 : - Change tokenPrice variable to private and add a getter function, which can retrieve the current value of tokenPrice without directly accessing the private variable from outside the contract.

- Include a time delay or require multiple signatures to approve any changes to the token price. It's important to ensure that the token price cannot be changed during a minting process or any other transaction that could be exploited in a flash loan attack.

Consider below fix in the function

```
/// @dev the token price in ETH
    uint256 private tokenPrice;

function getTokenPrice() external view returns (uint256) {
    return tokenPrice;
}

function setTokenPrice(
        uint256 _tokenPrice
    ) external onlyOwner whenNotPaused {
         require(block.timestamp > lastPriceChangeTimestamp + priceChangeCooldown, "Price can only b
e changed once per cooldown period");
        require(_tokenPrice > 0, "Price must be greater than zero");

        // Prevent flash loan attacks by requiring a minimum amount of time to pass before the new
price can be used for minting
        uint256 currentPrice = tokenPrice;
        uint256 timeSinceLastPriceChange = block.timestamp - lastPriceChangeTimestamp;
        if (timeSinceLastPriceChange < priceChangeCooldown) {
            currentPrice = lastTokenPrice; // Use the old price if we're still within the cooldown
period
        } else {
            lastTokenPrice = currentPrice; // Save the current price as the last price
            lastPriceChangeTimestamp = block.timestamp; // Update the timestamp for the last price
change
            currentPrice = _tokenPrice; // Use the new price if we're past the cooldown period
        }

        tokenPrice = currentPrice;
}
```

Client Response

We’re deferring it this time.

TVA-26: `KarmaPointV1::updatePrice` need more restrictions

Category	Severity	Code Reference	Status	Contributor
Governance Manipulation	Low	• code/contracts/karmapoint/KarmaPointV1.sol#L186	Acknowledged	newway55

Code

```
186:    /// @param _price a price of karma points in stable coins
```

Description

newway55 : updatePrice can be changed multiple times without restrictions and lead to many issues.

- `sync` function allows owner to airdrop karma points which price is updated in updatePrice based on the off-chain data can be exploited.

If the price of karma points is set too low, it can lead to an inflationary environment where users can accumulate large amounts of karma points with minimal effort and specifically because this is a game and require players to play and spend time accumulating points leading to a devaluation of the currency.

The distribution of rewards is based on off-chain data. This means unfair distribution of rewards also if data manipulated.

Consider below POC contract

```
pragma solidity ^0.8.13;

import "forge-std/Test.sol";
import "../src/KarmaPointV1.sol";

contract KarmaPointV1Test is Test {
    KarmaPointV1 public karma;
    CitizenIDV1 public citizen;


    function setUp() public {
        address payable safe ;
        citizen = new CitizenIDV1();
        citizen.initialize("Uri", 200);
        karma = new KarmaPointV1();
        karma.initialize(citizen, safe, 20, 2000, 100);
    }

   function test_updatePrice() public {
        karma.updatePrice(0.02 ether);
        address tevan1 = 0x839B878873998F02cE2f5c6D78d1B0842e58F192;
        address tevan2 = 0x8Cb9C0b4060Ec96E73e2d4f63D1E4b72f2499c7F;
        address[] memory addresses = new address[](2);
        addresses[0] = tevan1;
        addresses[1] = tevan2;

        uint256[] memory amounts = new uint256[](2);
        amounts[0] = 10;
        amounts[1] = 10;

        karma.sync( addresses, amounts);
        karma.updatePrice(10 ether);
    }

}

}
```

Recommendation

newway55 : - Change price variable to private and add a getter function, which can retrieve the current value of price without directly accessing the private variable from outside the contract.

- Implement a mechanism that requires multiple parties to agree on any changes to the price of karma points. This could be achieved through a multi-signature scheme or a DAO-based governance model, where changes to the system are subject to a vote by a group of stakeholders It's important to ensure that the price of karma points cannot be changed during an airdrop.

Consider below fix in the function

```
// Define a struct to represent a signature
struct Signature {
    address signer;
    bool signed;
}

// Define a mapping to keep track of signatures for each update proposal
mapping(uint256 => mapping(address => Signature)) private signatures;

// Define a function to update the price of karma points
function updatePrice(uint256 _price, uint256 _proposalId) external {
    require(_price > 0, "Invalid price");

    // Check that the caller has not already signed this proposal
    require(!signatures[_proposalId][msg.sender].signed, "You have already signed this proposal");

    // Add the signature to the mapping
    signatures[_proposalId][msg.sender] = Signature({
        signer: msg.sender,
        signed: true
    });

    // Check if the proposal has received enough signatures to be executed
    uint256 signatureCount;
    for (uint256 i = 0; i < SIGNERS.length; i++) {
        if (signatures[_proposalId][SIGNERS[i]].signed) {
            signatureCount++;
        }
    }
    require(signatureCount >= MIN_SIGNATURES, "Not enough signatures");

    // Update the price of karma points
    price = _price;
}
```

Client Response

We’re deferring it this time.

TVA-27: `mint()` will revert with very high probability due to the wrong implementation of `getRandomAvailableTokenId`

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	• code/contracts/guardians/MagicalPhoenixV1.sol#L202 • code/contracts/guardians/NomadicYetiV1.sol#L202	Fixed	jayphbee

Code

```
202:        ) % MAX_PHOENIXES;

202:        ) % MAX_YETIS;
```

Description

jayphbee : The `mint()` function in MagicalPhoenixV1.sol and NomadicYetiV1.sol will always revert with very high probability.
Here is the test for `MagicalPhoenixV1.mint()` function:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.0;

import "forge-std/Test.sol";
import "src/MagicalPhoenixV1.sol";
import "src/CitizenIDV1.sol";

contract TestMagical is Test {
    MagicalPhoenixV1 magical;
    CitizenIDV1 citizen;

    function setUp() public {
        citizen = new CitizenIDV1();
        citizen.initialize("", 0);

        magical = new MagicalPhoenixV1();
        magical.initialize(
            payable(address(1337)), payable(address(1338)), citizen, ""
        );
    }

    function testMint() public {
        for (uint i = 1; i <= magical.MAX_PHOENIXES(); ++i) {
            address minter = address(uint160(uint256(keccak256(abi.encode(i)))));
            vm.deal(minter, 0.025 ether);
            vm.startPrank(minter);
            citizen.mintCitizenId();
            magical.mint{value: 0.025 ether}();
            vm.stopPrank();
        }
    }
}
```

run `forge test --mt testMint -vvv`

```
Failing tests:
Encountered 1 failing test in test/magical.t.sol:TestMagical
[FAIL. Reason: ERC721: token already minted] testMint() (gas: 4026089)
```

In this test after 536 mints, the duplicate `tokenId` arises, as this piece of code in the `mint()` function implies

```
        // get the random token id
        uint256 tokenId = getRandomAvailableTokenId(msg.sender, randomNonce);

        // mint the guardian nft
        _mint(msg.sender, tokenId);
```

The problem here is that `randomNum` generated in `getRandomAvailableTokenId` function is out of bound. The value range of `randomNum` should be reduced by one once a `mint` is successed.

The impact is that in order for the users to mint sucessfully when the `mint()` reverts, they have to retry hundred or thousand of new accounts.

Recommendation

jayphbee :

```
    function getRandomAvailableTokenId(
        address to,
        uint256 nonce
    ) internal returns (uint256) {
        uint256 randomNum = uint256(
            keccak256(
                abi.encodePacked(
                    keccak256(
                        abi.encode(
                            to,
                            nonce,
                            tx.gasprice,
                            block.number,
                            block.timestamp,
                            block.prevrandao,
                            blockhash(block.number),
                            address(this)
                        )
                    )
                )
            )
        ) % numAvailableTokens; // modify here

        return getAvailableTokenAtIndex(randomNum);
    }
```

Apply the similar change in NomadicYetiV1.sol.

Client Response

Fixed

TVA-28: `msg.value` is not strictly checked in `claim()`

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/contracts/karmapoint/Claim.sol#L32-L40	Fixed	jayphbee

Code

```
    function claim() external payable isNotBlacklisted whenNotPaused {
        if (citizenIdContract.balanceOf(msg.sender) == 0) {
            citizenIdContract.claim{value: msg.value}(msg.sender);
        }

        if (kpContract.toBeClaimedKP(msg.sender) > 0) {
            kpContract.claim(msg.sender);
        }
    }
```

Description

jayphbee : In the `claim` function, if user's balance of `CitizenID` is zero then `msg.value` amount of ether will be forward to `CitizenID` contract to mint a new ID. However if his balance is not zero, `msg.value` amount of ether will be stucked in the `Claim` contract and there's no way to withdraw the stucked ethers in the `Claim` contract.

Recommendation

jayphbee : restrict the `msg.value` to zero when user's `CitizenID` balance is not zero.

```
    function claim() external payable isNotBlacklisted whenNotPaused {
        if (citizenIdContract.balanceOf(msg.sender) == 0) {
            citizenIdContract.claim{value: msg.value}(msg.sender);
        } else {
            require(msg.value == 0, "unexpected ether sent");
        }

        if (kpContract.toBeClaimedKP(msg.sender) > 0) {
            kpContract.claim(msg.sender);
        }
    }
```

Client Response

Fixed

TVA-29: `send` method is not supported in zkSync

Category	Severity	Code Reference	Status	Contributor
Language Specific	Critical	• code/contracts/karmapoint/KarmaPointV1.sol#L133 • code/contracts/guardians/NomadicYetiV1.sol#L135 • code/contracts/guardians/MagicalPhoenixV1.sol#L135 • code/contracts/guardians/NomadicYetiV1.sol#L138 • code/contracts/guardians/MagicalPhoenixV1.sol#L138 • code/contracts/citizenid/CitizenIDV1.sol#L263	Fixed	Hupixiong3

Code

```
133:        require(safeAddress.send(address(this).balance));

135:        require(charityAddress.send(charityAmount));

135:        require(charityAddress.send(charityAmount));

138:        require(safeAddress.send(amountAfterCharity));

138 :        require(safeAddress.send(amountAfterCharity));

263 :        require(safeAddress.send(address(this).balance));
```

Description

Hupixiong3 : The `send` method is not recommended in zkSync. zkSync gas is a dynamic handling fee and does not support the call method of fixed gas. Please read more details in - https://era.zksync.io/docs/dev/troubleshooting/changelog.html#compiler-local-setup-update-feb-20th-2023

Recommendation

Hupixiong3 : Transfer by call.

Consider below fix in the `CitizenIDV1.withdrawFunds()` function

```
    function withdrawFunds() external onlyOwner {
        require(safeAddress != address(0), "Missing safe address!");
        (bool success, ) =payable(safeAddress).call{value:(address(this).balance)}("");
        require(success,"Transfer failed.");
    }
```

Client Response

Fixed

Disclaimer