EVMNS Audit Report

Audit Report - EVMNS

Date	Oct 09, 2023
Auditor	0xBytes

About EVMNS

EVMNS is a name registrar platform build on EOS EVM, which allows anyone to register names and bind it to specific address.

Table of Contents

- Disclaimer
- Severity classification
- Scope
- Summary
- Findings

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

Severity classification

Severity	Description
Critical	A **directly** exploitable security vulnerability that leads to stolen/lost/locked/compromised assets or catastrophic denial of service.
High	A security vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service. It may not be directly exploitable or may require certain, external conditions in order to be exploited.
Medium	Assets not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but

Severity	Description
	external requirements.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.

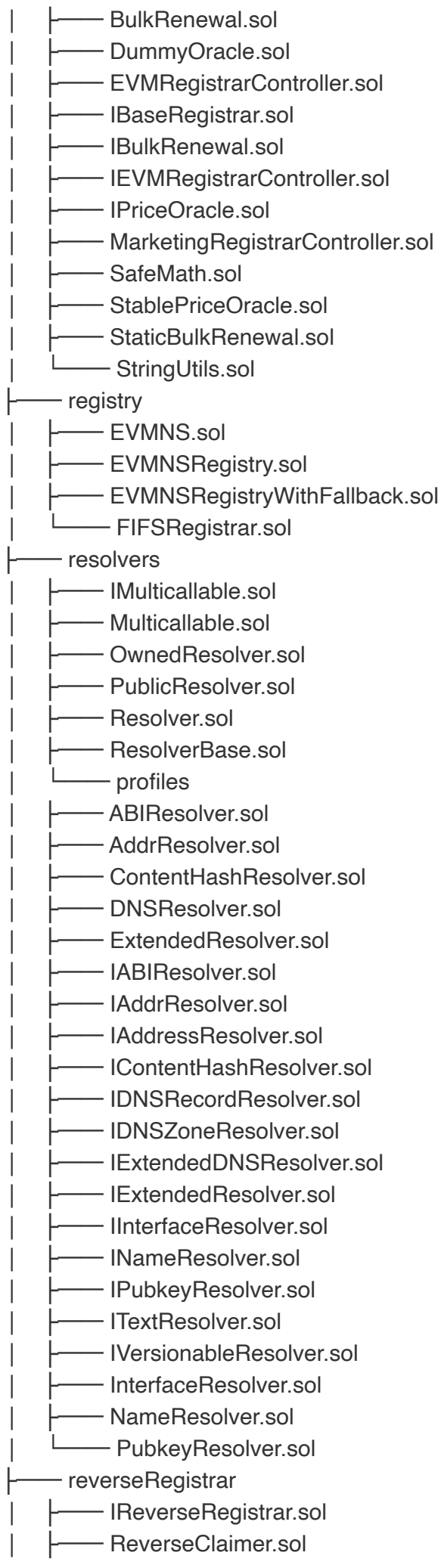
Scope

The review focused on the file hash of original files, as following:

- name: evmns合约包.zip
- hash(sha256): d7bec3d32879ac71cb874eaecec29ef6d48ec509bd31aeb32d3bf71d987f44c4

Files

```
├── dnsregistrar
│   ├── DNSClaimChecker.sol
│   ├── DNSRegistrar.sol
│   ├── IDNSRegistrar.sol
│   ├── OffchainDNSResolver.sol
│   ├── PublicSuffixList.sol
│   ├── RecordParser.sol
│   ├── SimplePublicSuffixList.sol
│   └── TLDPublicSuffixList.sol
├── dnssec-oracle
│   ├── BytesUtils.sol
│   ├── DNSSEC.sol
│   ├── DNSSECImpl.sol
│   ├── Owned.sol
│   ├── RRUtils.sol
│   ├── SHA1.sol
│   ├── algorithms
│   │   ├── Algorithm.sol
│   │   ├── DummyAlgorithm.sol
│   │   ├── EllipticCurve.sol
│   │   ├── ModexpPrecompile.sol
│   │   ├── P256SHA256Algorithm.sol
│   │   ├── RSASHA1Algorithm.sol
│   │   ├── RSASHA256Algorithm.sol
│   │   └── RSAVerify.sol
│   └── digests
│   ├── Digest.sol
│   ├── DummyDigest.sol
│   ├── SHA1Digest.sol
│   └── SHA256Digest.sol
├── evmregistrar
│   ├── ActivitiesRegistrarController.sol
│   ├── AirdropRegistrarController.sol
│   ├── BaseRegistrarImplementation.sol
```


BulkRenewal.sol
DummyOracle.sol
EVMRegistrarController.sol
IBaseRegistrar.sol
IBulkRenewal.sol
IEVMRegistrarController.sol
IPriceOracle.sol
MarketingRegistrarController.sol
SafeMath.sol
StablePriceOracle.sol
StaticBulkRenewal.sol
StringUtils.sol
registry
EVMNS.sol
EVMNSRegistry.sol
EVMNSRegistryWithFallback.sol
FIFSRegistrar.sol
resolvers
IMulticallable.sol
Multicallable.sol
OwnedResolver.sol
PublicResolver.sol
Resolver.sol
ResolverBase.sol
profiles
ABIResolver.sol
AddrResolver.sol
ContentHashResolver.sol
DNSResolver.sol
ExtendedResolver.sol
IABIResolver.sol
IAddrResolver.sol
IAddressResolver.sol
IContentHashResolver.sol
IDNSRecordResolver.sol
IDNSZoneResolver.sol
IExtendedDNSResolver.sol
IExtendedResolver.sol
IInterfaceResolver.sol
INameResolver.sol
IPubkeyResolver.sol
ITextResolver.sol
IVersionableResolver.sol
InterfaceResolver.sol
NameResolver.sol
PubkeyResolver.sol
reverseRegistrar
IReverseRegistrar.sol
ReverseClaimer.sol

```
│   └── ReverseRegistrar.sol
├── root
│   ├── Controllable.sol
│   ├── Ownable.sol
│   └── Root.sol
├── utils
│   ├── ERC20Recoverable.sol
│   ├── HexUtils.sol
│   ├── LowLevelCallUtils.sol
│   ├── NameEncoder.sol
│   └── UniversalResolver.sol
└── wrapper
├── BytesUtils.sol
├── Controllable.sol
├── ERC1155Fuse.sol
├── IMetadataService.sol
├── INameWrapper.sol
├── INameWrapperUpgrade.sol
├── NameWrapper.sol
└── StaticMetadataService.sol
```

Summary

The audited contracts contain 1 **critical** issues, **2 medium** severity issues, **3 minor** issues.

#	Title	Severity	Status
1	Malicious users can consume other users register quota	Critical	Resolved
2	Emoji name and normal names share same quota storage	Medium	Confirm
3	Limited name bypass	Medium	Confirmed
4	Missing check on addresses	Minor	Confirmed
5	Redundancy Logic	Minor	Resolved
6	Redundancy events	Minor	Resolved

Findings

Critical Findings(1)

1. Malicious users can consume other users register quota Critical

Loc:

ActivitiesRegistrarController #L358

Description:

In the `ActivitiesRegistrarController` contract, `register` and `registerWithEmoji` functions use `availableWithOwner` and `availableWithEmojiAndOwner` respectively to check the user's whitelist quota. But there is no restriction `owner == msg.sender` here. As a result, malicious users can set a name with a small renting price for other users, maliciously consuming the user's whitelist quota

```
function register(
        string calldata name,
        address owner,
        uint256 duration,
        bytes32 secret,
        address resolver,
        bytes[] calldata data,
        bool reverseRecord,
        uint16 ownerControlledFuses
    ) public payable override {
        uint256 price = rentPrice(name, duration);
        if (msg.value < price) {
            revert InsufficientValue();
        }
        // @audit Don't restrict owner must be msg.sender, which allows any one to consume other
        // users whitelist quota
        if (!availableWithOwner(name, owner)) {
            revert NameNotAvailable(name);
        }
        _consumeCommitment(
            name,
            duration,
            makeCommitment(
                name,
                owner,
                duration,
                secret,
                resolver,
                data,
                reverseRecord,
                ownerControlledFuses
            ),
            owner
        );
        ......

function availableWithOwner(
        string memory name,
        address owner
    ) public view returns (bool) {
        if (whitelist[owner] < 1) revert("Invalid address");
        if (whitelist[owner] == registers[owner])
            revert("Max allowed!");
        if (limites[name] != address(0) && limites[name] != owner)
```

```
45              revert("Limited name!!!");
46          bytes32 label = keccak256(bytes(name));
47          return valid(name) && !hasZeroWidthChar(name) &&
    base.available(uint256(label));
48      }
49
50      function availableWithEmojiAndOwner(string memory name,address
    owner) public view returns (bool) {
51          if (whitelist[owner] < 1) revert("Invalid address");
52          if (whitelist[owner] == registers[owner])
53              revert("Max allowed!");
54          if (limites[name] != address(0) && limites[name] != owner)
55              revert("Limited name!!!");
56          bytes32 label = keccak256(bytes(name));
57          return valid(name) && !hasZeroWidthEmoji(name) &&
    base.available(uint256(label));
58      }
```

Impact: This vulnerability allows anyone to consume other users whitelist quota.

Status: Resolved.

Medium Findings(2)

Emoji name and normal names share same quota storage medium

Loc:

ActivitiesRegistrarController#L177, #L187

Description:

In the `ActivityRegistrarController` contract, there is emoji name and the normal name. From the logic of the contract, there are two interfaces to query their respective whitelist quota, but they use a same variable

```
1  function availableWithOwner(
2          string memory name,
3          address owner
4      ) public view returns (bool) {
5          if (whitelist[owner] < 1) revert("Invalid address");
6          //@audit query whitelist variable same as
    availableWithEmojiAndOwner
7          if (whitelist[owner] == registers[owner])
8              revert("Max allowed!");
9          if (limites[name] != address(0) && limites[name] != owner)
10             revert("Limited name!!!");
11         bytes32 label = keccak256(bytes(name));
12         return valid(name) && !hasZeroWidthChar(name) &&
    base.available(uint256(label));
13     }
```

```
14
15      function availableWithEmojiAndOwner(string memory name,address
    owner) public view returns (bool) {
16          if (whitelist[owner] < 1) revert("Invalid address");
17          if (whitelist[owner] == registers[owner])
18              revert("Max allowed!");
19          if (limites[name] != address(0) && limites[name] != owner)
20              revert("Limited name!!!");
21          bytes32 label = keccak256(bytes(name));
22          return valid(name) && !hasZeroWidthEmoji(name) &&
    base.available(uint256(label));
23      }
```

Impact:: This vulnerability will result user comsume emoji quota while register normal name and and vice versa.

Status: Confirmed, the team confirmed that this is the design.

Limited name bypass medium

Loc::

ActivitiesRegistrarController #L180

Description:

In the `ActivityRegistrarController` contract, `availableWithOwner` and `availableWithEmojiAndOwner` function will check whether the name is a limited name, but the `valid` function does not check the case of the letter, resulting in bypassing the case of the name. Assuming the definition is named demo, the user can register DEMO, DEMo, dEMO, etc. to bypass the name check.

```
1  function availableWithOwner(
2          string memory name,
3          address owner
4      ) public view returns (bool) {
5          if (whitelist[owner] < 1) revert("Invalid address");
6          if (whitelist[owner] == registers[owner])
7              revert("Max allowed!");
8          if (limites[name] != address(0) && limites[name] != owner)
9              revert("Limited name!!!");
10         bytes32 label = keccak256(bytes(name));
11         //@audit name can be bypass by upper case
12         return valid(name) && !hasZeroWidthChar(name) &&
    base.available(uint256(label));
13     }
14
15 function availableWithEmojiAndOwner(string memory name,address owner)
    public view returns (bool) {
16         if (whitelist[owner] < 1) revert("Invalid address");
17         if (whitelist[owner] == registers[owner])
```

```
18              revert("Max allowed!");
19          if (limites[name] != address(0) && limites[name] != owner)
20              revert("Limited name!!!");
21          bytes32 label = keccak256(bytes(name));
22          //@audit name can be bypass by upper case
23          return valid(name) && !hasZeroWidthEmoji(name) &&
    base.available(uint256(label));
24      }
25  function valid(string memory name) public pure returns (bool) {
26          return
27              name.strlen() >= 3 &&
28              name.strlen() <= 250 &&
29              !hasInvalidChar(name) ;
30      }
```

Impact: This vulnerability will allow users to use different case of names to bypass the limited name check

Status: Confirmed, the team confirms that they only show lower case name on UI and only allow user to register lowercase name from UI

Minor Findings(3)

Missing check on addresses Minor

Loc::

ActivitiesRegistrarController#L106-11

Description:

In the `ActivitiesRegistrarController` contract, in the process of setting the user's whitelist, no check is made for duplicate addresses, resulting in duplicate address assignments

```
1  constructor(
2          BaseRegistrarImplementation _base,
3          IPriceOracle _prices,
4          uint256 _minCommitmentAge,
5          uint256 _maxCommitmentAge,
6          ReverseRegistrar _reverseRegistrar,
7          INameWrapper _nameWrapper,
8          EVMNS _evmns,
9          uint160[][] memory _whitelist,
10         string[][] memory _limits
11     ) ReverseClaimer(_evmns, msg.sender) {
12         if (_maxCommitmentAge <= _minCommitmentAge) {
13             revert MaxCommitmentAgeTooLow();
14         }
15
16         if (_maxCommitmentAge > block.timestamp) {
17             revert MaxCommitmentAgeTooHigh();
```

```
        }
        base = _base;
        prices = _prices;
        minCommitmentAge = _minCommitmentAge;
        maxCommitmentAge = _maxCommitmentAge;
        reverseRegistrar = _reverseRegistrar;
        nameWrapper = _nameWrapper;
        for (uint i = 0; i < _whitelist.length; i++) {
              //@audit No checks for duplicate addresses
            whitelist[address(uint160(_whitelist[i][0]))] = uint8(
                _whitelist[i][1]
            );
            discounts[address(uint160(_whitelist[i][0]))] = uint8(
                _whitelist[i][2]
            );
        }
        for (uint i = 0; i < _limits.length; i++) {
            limites[_limits[i][0]] = stringToAddress(_limits[i][1]);
        }
    }
```

Impact: This vulnerability will results in duplicate address assignments.

Status: Confirmed, the team confirms that they will ensure all addresses are unique when deploy

Redundancy logic Minor

Loc:

ActivitiesRegistrarController#L106-111

Description:
In the `ActivitiesRegistrarController` contract, when the `addEmojis` function sets the `currentNodes.next` variable, there is a repeated assignment behavior

```
function addEmojis(bytes[] calldata _emojis) public onlyOwner {
        for(uint i = 0; i < _emojis.length; i++){
            //emojis.push(_emojis[i]);
            if(emojilink[bytes1(_emojis[i][0])].length == 0){
                emojilink[bytes1(_emojis[i][0])].length = 1;
                emojilink[bytes1(_emojis[i][0])].end = false;
            }
            Emoji storage currentNode = emojilink[bytes1(_emojis[i]
[0])];
            for(uint8 j = 1; j < _emojis[i].length; j++){
                if(currentNode.nexts[_emojis[i][j]].length == 0){
                    currentNode.nexts[_emojis[i][j]].length = j+1;
                    //@audit Redundancy logic
                    currentNode.nexts[_emojis[i][j]].end = false;
                    if(j == _emojis[i].length-1){
                        currentNode.nexts[_emojis[i][j]].end = true;
```

```
16                      } else {
17                          currentNode.nexts[_emojis[i][j]].end = false;
18                      }
19                  }else{
20                      if(j == _emojis[i].length-1){
21                          currentNode.nexts[_emojis[i][j]].end = true;
22                      }
23                  }
24                  currentNode = currentNode.nexts[_emojis[i][j]];
25              }
26          }
27      }
```

Impact: Redundancy logic cost more gas

Status: Resolved

Redundancy event Minor

Loc:

ActivitiesRegistrarController #L74

Description:

In the `ActivitiesRegistrarControlle` contract, since the `renew()` function is deleted, the NameRenewed event is no longer used and can be deleted.

Impact: Unused code in contract

Status: Resolved