ZerO - zNS

Date	May 2021
Lead Auditor	David Oz Kashi
Co-auditors	Martin Ortner

1 Executive Summary

This report is part of a series of reports presenting the results of our engagement with **zer0** to review **zNS, zAuction, and zBanc, zDAO Token**.

The review was conducted over four weeks, from **19 April 2021** to **21 May 2021**. A total of 2x4 person-weeks were spent.

1.1 Layout

It was requested to present the results for the four code-bases under review in individual reports. Links to the individual reports can be found below.

The Executive Summary and Scope sections are shared amongst the individual reports. They provide a general overview of the engagement and summarize scope changes and insights into how time was spent during the audit. The section Recommendations and Findings list the respective findings for the component under review.

The following reports were delivered:

- zNS

- zAuction
- zBanc
- zDAO-Token

1.2 Assessment Log

In the first week, the assessment team focussed its work on the `zNS` and `zAuction` systems. Details on the scope for the components was set by the client and can be found in the next section. A walkthrough session for the systems in scope was requested, to understand the fundamental design decisions of the system as some details were not found in the specification/documentation. Initial security findings were also shared with the client during this session. It was agreed to deliver a preliminary report sharing details of the findings during the end-of-week sync-up. This sync-up is also used to set the focus/scope for the next week.

In the second week, the assessment team focussed its work on `zBanc` a modification of the bancor protocol solidity contracts. The initial code revision under audit (zBanc `48da0ac1eebbe31a74742f1ae4281b156f03a4bc`) was updated half-way into the week on Wednesday to zBanc (`3d6943e82c167c1ae90fb437f9e3ed1a7a7a94c4`). Preliminary findings were shared during a sync-up discussing the changing codebase under review. Thursday morning the client reported that work on the `zDAO Token` finished and it was requested to put it in scope for this week as the token is meant to be used soon. The assessment team agreed to have a brief look at the codebase, reporting any obvious security issues at best effort until the end-of-week sync-up meeting (1day). Due to the very limited left until the weekly sync-up meeting, it was recommended to extend the review into next week as. Finally it was agreed to update and deliver the preliminary report sharing details of the findings during the end-of-week sync-up. This sync-up is also used to set the focus/scope for the next week.

In the third week, the assessment team continued working on `zDAO Token` on Monday. We provided a heads-up that the snapshot functionality of zDAO Token was not working the same day. On Tuesday focus shifted towards reviewing changes to zAuction (`135b2aaddcfc70775fd1916518c2cc05106621ec` , remarks). On the same day the client provided an updated review commit for zDAO Token (`81946d451e8a9962b0c0d6fc8222313ec115cd53`) addressing the issue we reported on

Monday. The client provided an updated review commit for zNS (
`ab7d62a7b8d51b04abea895e241245674a640fc1`) on Wednesday and zNS (
`bc5fea725f84ae4025f5fb1a9f03fb7e9926859a`) on Thursday.

As can be inferred from this timeline various parts of the codebases were undergoing changes while the review was performed which introduces inefficiencies and may have an impact on the review quality (reviewing frozen codebase vs. moving target). As discussed with the client we highly recommend to plan ahead for security activities, create a dedicated role that coordinates security on the team, and optimize the software development lifecycle to explicitly include security activities and key milestones, ensuring that code is frozen, quality tested, and security review readiness is established ahead of any security activities. It should also be noted that code-style and quality varies a lot for the different repositories under review which might suggest that there is a need to better anchor secure development practices in the development lifecycle.

After a one-week hiatus the assessment team continued reviewing the changes for `zAuction` and `zBanc` . The findings were initially provided with one combined report and per client request split into four individual reports.

2 Scope

Our review focused on the following components and code revisions:

2.1 Objectives

Together with the zer0 team, we identified the following priorities for our review:

1. Ensure that the system is implemented consistently with the intended functionality, and without unintended edge cases.
2. Identify known vulnerabilities particular to smart contract systems, as outlined in our Smart Contract Best Practices, and the Smart Contract Weakness Classification Registry.

2 Week - 1

- zNS (`b05e503ea1ee87dbe62b1d58426aaa518068e395`) (scope doc) (1, 2)
- zAuction (`50d3b6ce6d7ee00e7181d5b2a9a2eedcdd3fdb72`) (scope doc) (1, 2)

Original Scope overview document

2.3 Week - 2

- zBanc (`48da0ac1eebbe31a74742f1ae4281b156f03a4bc`) initial commit under review
- zBanc (`3d6943e82c167c1ae90fb437f9e3ed1a7a7a94c4`) updated commit under review (mid of week) (scope doc) (1)
  - Files in Scope:
    - `contracts/converter/types/dynamic-liquid-token/DynamicLiquidTokenConverter`
    - `contracts/converter/types/dynamic-liquid-token/DynamicLiquidTokenConverterFactory`
    - `contracts/converter/ConverterUpgrader.sol` (added handling new converterType 3)
- zDAO token provided on thursday (scope doc) (1)
  - Files in Scope:
    - `ZeroDAOToken.sol`
    - `MerkleTokenAirdrop.sol`
    - `MerkleTokenVesting.sol`
    - `MerkleDistributor.sol`
    - `TokenVesting.sol`
    - And any relevant Interfaces / base contracts

The `zDAO` review in week two was performed best effort from Thursday to Friday attempting to surface any obvious issues until the end-of-week sync-up meeting.

2.4 Week - 3

- Continuing on zDAO token (`1b678cb3fc4a8d2ff3ef2d9c5625dff91f6054f6`)
- Updated review commit for zAuction (`135b2aaddcfc70775fd1916518c2cc05106621ec` , 1) on Monday
- Updated review commit for zDAO Token (`81946d451e8a9962b0c0d6fc8222313ec115cd53`) on Tuesday

- Updated review commit for zNS (`ab7d62a7b8d51b04abea895e241245674a640fc1`) on Wednesday
- Updated review commit for zNS (`bc5fea725f84ae4025f5fb1a9f03fb7e9926859a`) on Thursday

2.5 Hiatus - 1 Week

The assessment continues for a final week after a one-week long hiatus.

2.6 Week - 4

- Updated review commit for zAuction (`2f92aa1c9cd0c53ec046340d35152460a5fe7dd0` , 1)
- Updated review commit for zAuction addressing our remarks
- Updated review commit for zBanc (`ff3d91390099a4f729fe50c846485589de4f8173` , 1)
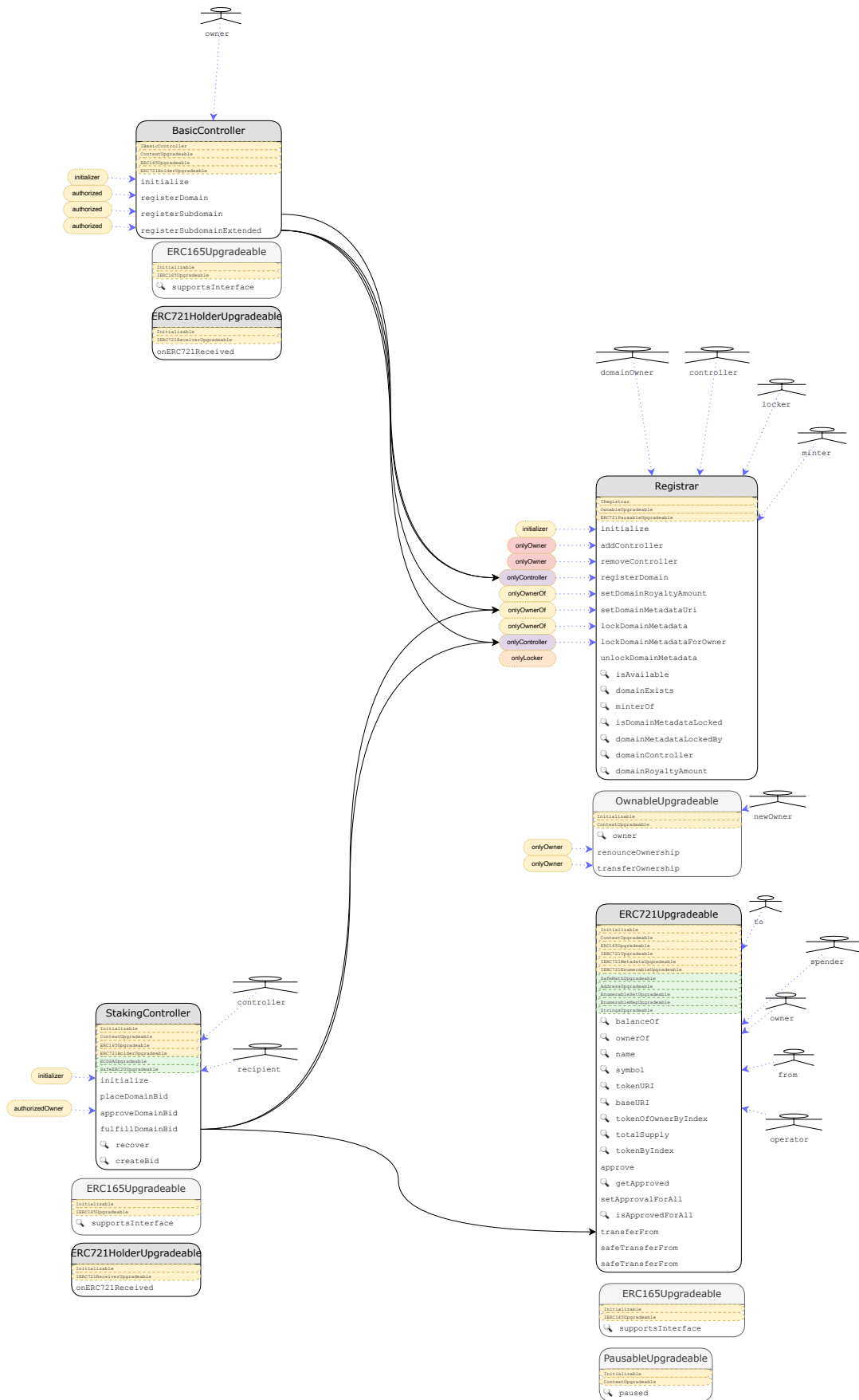
3 System Overview

This section describes the top-level/deployable contracts, their inheritance structure and interfaces, actors, permissions and important contract interactions of the initial system under review. This section does not take any fundamental changes into account that were introduced during or after the review was conducted.

Contracts are depicted as boxes. Public reachable interface methods are outlined as rows in the box. The 🔍 icon indicates that a method is declared as non-state-changing (view/pure) while other methods may change state. A yellow dashed row at the top of the contract shows inherited contracts. A green dashed row at the top of the contract indicates that that contract is used in a usingFor declaration. Modifiers used as ACL are connected as yellow bubbles in front of methods.

zNS

owner

BasicController
IBasicController
ContextUpgradeable
ERC165Upgradeable
ERC721HolderUpgradeable
- initializer — initialize
- authorized — registerDomain
- authorized — registerSubdomain
- authorized — registerSubdomainExtended

ERC165Upgradeable
Initializable
IERC165Upgradeable
🔍 supportsInterface

ERC721HolderUpgradeable
Initializable
IERC721ReceiverUpgradeable
onERC721Received

domainOwner   controller   locker   minter

Registrar
IRegistrar
OwnableUpgradeable
ERC721PausableUpgradeable
- initializer — initialize
- onlyOwner — addController
- onlyOwner — removeController
- onlyController — registerDomain
- onlyOwnerOf — setDomainRoyaltyAmount
- onlyOwnerOf — setDomainMetadataUri
- onlyOwnerOf — lockDomainMetadata
- onlyController — lockDomainMetadataForOwner
- onlyLocker — unlockDomainMetadata
- 🔍 isAvailable
- 🔍 domainExists
- 🔍 minterOf
- 🔍 isDomainMetadataLocked
- 🔍 domainMetadataLockedBy
- 🔍 domainController
- 🔍 domainRoyaltyAmount

OwnableUpgradeable
Initializable
ContextUpgradeable
🔍 owner
- onlyOwner — renounceOwnership
- onlyOwner — transferOwnership

newOwner

ERC721Upgradeable
Initializable
ContextUpgradeable
ERC165Upgradeable
IERC721Upgradeable
IERC721MetadataUpgradeable
IERC721EnumerableUpgradeable
SafeMathUpgradeable
AddressUpgradeable
EnumerableSetUpgradeable
EnumerableMapUpgradeable
StringsUpgradeable
- 🔍 balanceOf
- 🔍 ownerOf
- 🔍 name
- 🔍 symbol
- 🔍 tokenURI
- 🔍 baseURI
- 🔍 tokenOfOwnerByIndex
- 🔍 totalSupply
- 🔍 tokenByIndex
- approve
- 🔍 getApproved
- setApprovalForAll
- 🔍 isApprovedForAll
- transferFrom
- safeTransferFrom
- safeTransferFrom

to   spender   owner   from   operator

controller   recipient

StakingController
Initializable
ContextUpgradeable
ERC165Upgradeable
ERC721HolderUpgradeable
IERC20Upgradeable
SafeERC20Upgradeable
- initializer — initialize
- authorizedOwner — placeDomainBid
- approveDomainBid
- fulfillDomainBid
- 🔍 recover
- 🔍 createBid

ERC165Upgradeable
Initializable
IERC165Upgradeable
🔍 supportsInterface

ERC721HolderUpgradeable
Initializable
IERC721ReceiverUpgradeable
onERC721Received

ERC165Upgradeable
Initializable
IERC165Upgradeable
🔍 supportsInterface

PausableUpgradeable
Initializable
ContextUpgradeable
🔍 paused

zNS

`zNS` is the zer0 name system. It is comprised of three main components. A `BaseController` that allows owners of domains to create subdomains. It was a design decision that an owner has full control of their sub-namespace and can create sub-domains at will. A domain name is represented by an NFT implemented in `Registrar` . Ownership can be freely transferred but domain NFT's cannot be burned. Domains cannot be recovered if the owner loses access to its account. Domains do not expire or require renewal. Other accounts can bid on subdoimains on a 2nd layer. The owner can then accept bids and transfer complete ownership of a sub-namespace to that account. Domain NFT's can contain metadata that can be locked by the current owner of the token. An owner may lock domain metadata before transferring ownership and the new owner will not be able to unlock it. However, the original locking account can at any time unlock the metadata remotely even without currently owning the account. Depending on the use-case and system design this may have security/trust implications together with the fact that the owner - if the nft is unlocked - might frontrun transactions to change the royaltyAmount in `%` which might have unpredictable consequences depending on how the royalty system works. Domain information is made available in a subgraph. The subgraph's domain separator is a `.` .

4 Recommendations

4.1 Ensure that implementations of upgradeable contracts are initialized

Description

It is recommended to check whether implementations/logic contracts used with the OZ upgradability pattern may be left uninitialized. While these logic contracts are typically not consumed directly (they are only delegated to) they may still be claimable by anyone as the `initialize` function is not access protected. This is usually not a problem unless there's a way to self-destruct the contract. However, there is a risk of reputational damage if someone initialized the implementation in an attempt to carry out a malicious campaign potentially ~~ky~~ users into believing this is the legitimate contract while it's only the logic ~~ntract~~ for an upgradeable contract.

4.2 zNS - prepare for domain name canonicalisation and potential injection vectors

Description

The off-chain components are not in scope for this review, however, it should be noted that there is a significant risk that domain names may include non-printable or control characters. Depending on the presentation layer of the application this may allow someone to register names that are visually indistinguishable from other domains in a console or web application (e.g. by adding zero bytes, spaces, delete character, newlines, non-printables, ...) or contain payloads that exploit injection vectors in consuming applications.

It is therefore highly recommended to canonicalize domain names or properly encode them before displaying or consuming them with other applications.

4.3 zNS - potential gas optimizations

Description

- the check for `registrar.domainExists(domain)` might be redundant as `registrar.ownerOf(domain) == _msgSender()` should fail for unknown nft ids anyway

zNS/contracts/BasicController.sol:L21-L28

```
modifier authorized(uint256 domain) {
  require(registrar.domainExists(domain), "Zer0 Controller: Invalid Domain");
  require(
    registrar.ownerOf(domain) == _msgSender(),
    "Zer0 Controller: Not Authorized"
  );
  _;
}
```

- rootdomain owner might directly call `registerSubdomain` instead of `registerDomain` to save gas on the public call and duplicate authorization check

s/contracts/BasicController.sol:L38-L55

```solidity
function registerDomain(string memory domain, address owner)
  public
  override
  authorized(rootDomain)
{
  registerSubdomain(rootDomain, domain, owner);
}

function registerSubdomain(
  uint256 parentId,
  string memory label,
  address owner
) public override authorized(parentId) {
  address minter = _msgSender();
  uint256 id = registrar.registerDomain(parentId, label, owner, minter);

  emit RegisteredDomain(label, id, parentId, owner, minter);
}
```

- `StakingController` - `address controller storage` holds the value of address(this) throughout the entire contract, thus could simply be removed to save gas costs by avoiding unnecessary storage operations.

4.4 Where possible, a specific contract type should be used rather than `address` ` Acknowledged `

Description

Consider using the best type available in the function arguments and declarations instead of accepting `address` and later casting it to the correct type.

Examples

This is only one of many examples.

zAuction/contracts/zAuction.sol:L22-L26

```
function init(address accountantaddress) external {
    require(!initialized);
    initialized = true;
    accountant = zAuctionAccountant(accountantaddress);
}
```

zAuction/contracts/zAuction.sol:L52-L54

```
IERC721 nftcontract = IERC721(nftaddress);
weth.transferFrom(bidder, msg.sender, bid);
nftcontract.transferFrom(msg.sender, bidder, tokenid);
```

zAuction/contracts/zAuction.sol:L40-L42

```
IERC721 nftcontract = IERC721(nftaddress);
accountant.Exchange(bidder, msg.sender, bid);
nftcontract.transferFrom(msg.sender, bidder, tokenid);
```

zAuction/contracts/zAuctionAccountant.sol:L60-L63

```
function SetZauction(address zauctionaddress) external onlyAdmin{
    zauction = zauctionaddress;
    emit ZauctionSet(zauctionaddress);
}
```

4.5 zNS, zAuction - check for inconsistent use of ERC-721 `safe*` family of methods ✓ Fixed

> ### Resolution
>
> Addressed with the following changes switching to the `safe*` family of
> methods (note that for erc721 this can introduce potentially reentrancy
> ...ectory):

- zer0-os/zAuction@ `135b2aa` . The client provided the following statement:

  4.9 safeTransferFrom used in Zauction and Zsale

- zer0-os/ZNS@ `ab7d62a`

Description

Domains are minted using `safeMint` which checks that the recipient accepts the token if it is a contract. This is basically to avoid that tokens get locked up in contracts by accident.

When transferring the token the system is not using the `safeTransferFrom` method which would perform the same checks. this appears to be inconsistent as creating a domain would check this condition, but transferring the token won't.

This may be a design decision, however, it is recommended to document the rationale behind when checks are to be performed. It should be noted that token retrieval can be rejected by the recipient when using the `safe*` method family.

Examples

zNS/contracts/Registrar.sol:L257-L264

```solidity
function _createDomain(
  uint256 domainId,
  address domainOwner,
  address minter,
  address controller
) internal {
  // Create the NFT and register the domain data
  _safeMint(domainOwner, domainId);
```

zNS/contracts/BasicController.sol:L71-L71

```solidity
registrar.transferFrom(controller, owner, id);
```

zNS/contracts/StakingController.sol:L140-L140

```
registrar.transferFrom(controller, recoveredBidder, id);
```

zAuction/contracts/zAuction.sol:L42-L42

```
nftcontract.transferFrom(msg.sender, bidder, tokenid);
```

5 Findings

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.

- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.

- Major issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.

- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

5.1 zNS - Domain bid might be approved by non owner account Critical ✓ Fixed

Resolution
Addressed with zer0-os/zNS@ab7d62a by storing the domain request data on-chain.

Description

The spec allows anyone to place a bid for a domain, while only parent domain owners are allowed to approve a bid. Bid placement is actually enforced and purely informational. In practice, `approveDomainBid` allows any parent domain owner to approve bids (signatures) for any other domain even if they do not own it. Once approved, anyone can call `fulfillDomainBid` to create a domain.

Examples

zNS/contracts/StakingController.sol:L95-L103

```solidity
function approveDomainBid(
    uint256 parentId,
    string memory bidIPFSHash,
    bytes memory signature
) external authorizedOwner(parentId) {
  bytes32 hashOfSig = keccak256(abi.encode(signature));
  approvedBids[hashOfSig] = true;
  emit DomainBidApproved(bidIPFSHash);
}
```

Recommendation

Consider adding a validation check that allows only the parent domain owner to approve bids on one of its domains. Reconsider the design of the system introducing more on-chain guarantees for bids.

5.2 zAuction, zNS - Bids cannot be cancelled, never expire, and the auction lifecycle is unclear Major ✓ Fixed

Resolution
• Addressed with zer0-os/zNS@ab7d62a by refactoring the `StakingController` to control the lifecycle of bids instead of handling this off-chain.

- Addressed with zer0-os/zAuction@ `135b2aa` for `zAuction` by adding a bid/saleOffer expiration for bids. The client also provided the following statement:

  > 5.6 added expireblock and startblock to zauction, expireblock to zsale Decided not to add a cancel function. Paying gas to cancel isn't ideal, and it can be used as a griefing function. though that's still possible to do by moving weth but differently

The stateless nature of auctions may make it hard to enforce bid/sale expirations and it is not possible to cancel a bid/offer that should not be valid anymore. The expiration reduces the risk of old offers being used as they now automatically invalidate after time, however, it is still likely that multiple valid offers may be present at the same time. As outlined in the recommendation, one option would be to allow someone who signed a commitment to explicitly cancel it in the contract. Another option would be to create a stateful auction where the entity that puts up something for "starts" an auction, creating an auction id, requiring bidders to bid on that auction id. Once a bid is accepted the auction id is invalidated which invalidates all bids that might be floating around.

UPDATE Fixed with zer0-os/zAuction@ `2f92aa1` for `zAuction/zSale` by allowing the seller (`zSale`) to cancel their offer, and by allowing the bidder (`zAuction`) to cancel bids (pot. more than one per auction) up to a certain price threshold. Since `auctionId` can only be used once, all other bids for an auction are automatically invalidated after a bid is accepted. Note that the current version is using a unique number as an auction id. There can be concurrent auctions that by chance or maliciously use the same auction id. The first auction to pass will cancel the competing auction that was using the same id. This fact can be used as a griefing vector to terminate running auctions by reusing the other auctions id and self-accepting the bid. The other auction cannot be fulfilled anymore.

Description

The lifecycle of a bid both for `zAuction` and `zNS` is not clear, and has many flaws.

- `zAuction` - Consider the case where a bid is placed, then the underlying asset in being transferred to a new owner. The new owner can now force to sell the asset even though it's might not be relevant anymore.

- `zAuction` - Once a bid was accepted and the asset was transferred, all other bids need to be invalidated automatically, otherwise and old bid might be accepted even after the formal auction is over.

- `zAuction`, `zNS` - There is no way for the bidder to cancel an old bid. That might be useful in the event of a significant change in market trend, where the old pricing is no longer relevant. Currently, in order to cancel a bid, the bidder can either withdraw his ether balance from the `zAuctionAccountant`, or disapprove `WETH` which requires an extra transaction that might be front-runned by the seller.

Examples

zAuction/contracts/zAuction.sol:L35-L45

```solidity
function acceptBid(bytes memory signature, uint256 rand, address bidder, uint25
    address recoveredbidder = recover(toEthSignedMessageHash(keccak256(abi.enco
    require(bidder == recoveredbidder, 'zAuction: incorrect bidder');
    require(!randUsed[rand], 'Random nonce already used');
    randUsed[rand] = true;
    IERC721 nftcontract = IERC721(nftaddress);
    accountant.Exchange(bidder, msg.sender, bid);
    nftcontract.transferFrom(msg.sender, bidder, tokenid);
    emit BidAccepted(bidder, msg.sender, bid, nftaddress, tokenid);
}
```

zNS/contracts/StakingController.sol:L120-L152

```
  function fulfillDomainBid(
  uint256 parentId,
  uint256 bidAmount,
  uint256 royaltyAmount,
  string memory bidIPFSHash,
  string memory name,
  string memory metadata,
  bytes memory signature,
  bool lockOnCreation,
  address recipient
) external {
  bytes32 recoveredBidHash = createBid(parentId, bidAmount, bidIPFSHash, name);
  address recoveredBidder = recover(recoveredBidHash, signature);
  require(recipient == recoveredBidder, "ZNS: bid info doesnt match/exist");
  bytes32 hashOfSig = keccak256(abi.encode(signature));
  require(approvedBids[hashOfSig] == true, "ZNS: has been fullfilled");
  infinity.safeTransferFrom(recoveredBidder, controller, bidAmount);
  uint256 id = registrar.registerDomain(parentId, name, controller, recoveredBi
  registrar.setDomainMetadataUri(id, metadata);
  registrar.setDomainRoyaltyAmount(id, royaltyAmount);
  registrar.transferFrom(controller, recoveredBidder, id);
  if (lockOnCreation) {
    registrar.lockDomainMetadataForOwner(id);
  }
  approvedBids[hashOfSig] = false;
  emit DomainBidFulfilled(
    metadata,
    name,
    recoveredBidder,
    id,
    parentId
  );
}
```

Recommendation

Consider adding an expiration field to the message signed by the bidder both for `zAuction` and `zNS`. Consider adding auction control, creating an `auctionId`, and have users bid on specific auctions. By adding this id to the signed message, all other bids are invalidated automatically and users would have to place new bids for a new auction. Optionally allow users to cancel bids licitly.

5.3 zNS - Insufficient protection against replay attacks Major

✓ Fixed

> ## Resolution
>
> Addressed with zer0-os/zNS@ab7d62a by avoiding the use of digital signatures and storing the domain request data on-chain.

Description

There is no dedicated data structure to prevent replay attacks on `StakingController`. `approvedBids` mapping offers only partial mitigation, due to the fact that after a domain bid is fulfilled, the only mechanism in place to prevent a replay attack is the `Registrar` contract that might be replaced in the case where `StakingController` is being re-deployed with a different `Registrar` instance. Additionally, the digital signature used for domain bids does not identify the buyer request uniquely enough. The bidder's signature could be replayed in future similar contracts that are deployed with a different registrar or in a different network.

Examples

zNS/contracts/StakingController.sol:L176-L183

```solidity
function createBid(
  uint256 parentId,
  uint256 bidAmount,
  string memory bidIPFSHash,
  string memory name
) public pure returns(bytes32) {
  return keccak256(abi.encode(parentId, bidAmount, bidIPFSHash, name));
}
```

Recommendation

Consider adding a dedicated mapping to store the a unique identifier of a bid, as well as adding `address(this)` , `block.chainId` , `registrar` and `nonce` to the message that is being signed by the bidder.

5.4 zNS - domain name collisions Major ✓ Fixed

Resolution

Addressed with zer0-os/ZNS@ `ab7d62a` by disallowing empty names for domain registrations. The name validation in off-chain components (e.g. subgraph components) has not been verified.

Description

Domain registration accepts an empty (zero-length) name. This may allow a malicious entity to register two different NFT's for the same visually indistinguishable text representation of a domain. Similar to this the domain name is mapped to an NFT via a subgraph that connects parent names to the new subdomain using a domain separation character (dot/slash/...). Someone might be able to register `a.b` to `cats.cool` which might resolve to the same domain as if someone registers `cats.cool.a` and then `cats.cool.a.b` .

Examples

- `0/cats/` = `0xfe`
- `0/cats/<empty-string/` = `0xfe.keccak("")`

zNS/contracts/Registrar.sol:L76-L96

```
function registerDomain(
  uint256 parentId,
  string memory name,
  address domainOwner,
  address minter
) external override onlyController returns (uint256) {
  // Create the child domain under the parent domain
  uint256 labelHash = uint256(keccak256(bytes(name)));
  address controller = msg.sender;

  // Domain parents must exist
  require(_exists(parentId), "Zer0 Registrar: No parent");

  // Calculate the new domain's id and create it
  uint256 domainId =
    uint256(keccak256(abi.encodePacked(parentId, labelHash)));
  _createDomain(domainId, domainOwner, minter, controller);

  emit DomainCreated(domainId, name, labelHash, parentId, minter, controller);

  return domainId;
```

Recommendation

Disallow empty subdomain names. Disallow domain separators in names (in the offchain component or smart contract).

5.5 zAuction, zNS - gas griefing by spamming offchain fake bids Medium Acknowledged

Resolution

Addressed and acknowledged with changes from zer0-os/zAuction@ `135b2aa` . The client provided the following remark:

> 5.19 I have attempted to order the requires sensibly, putting the least expensive first. Please advise if the ordering is optimal. gas griefing will be mitigated in the dapp with off-client checks

Description

The execution status of both `zAuction.acceptBid` and `StakingController.fulfillDomainBid` transactions depend on the bidder, as his approval is needed, his signature is being validated, etc. However, these transactions can be submitted by accounts that are different from the bidder account, or for accounts that do not have the required funds/deposits available, luring the account that has to perform the on-chain call into spending gas on a transaction that is deemed to fail (gas griefing). E.g. posting high-value fake bids for zAuction without having funds deposited or `WETH` approved.

Examples

zNS/contracts/StakingController.sol:L120-L152

```solidity
    function fulfillDomainBid(
    uint256 parentId,
    uint256 bidAmount,
    uint256 royaltyAmount,
    string memory bidIPFSHash,
    string memory name,
    string memory metadata,
    bytes memory signature,
    bool lockOnCreation,
    address recipient
  ) external {
    bytes32 recoveredBidHash = createBid(parentId, bidAmount, bidIPFSHash, name);
    address recoveredBidder = recover(recoveredBidHash, signature);
    require(recipient == recoveredBidder, "ZNS: bid info doesnt match/exist");
    bytes32 hashOfSig = keccak256(abi.encode(signature));
    require(approvedBids[hashOfSig] == true, "ZNS: has been fullfilled");
    infinity.safeTransferFrom(recoveredBidder, controller, bidAmount);
    uint256 id = registrar.registerDomain(parentId, name, controller, recoveredBi
    registrar.setDomainMetadataUri(id, metadata);
    registrar.setDomainRoyaltyAmount(id, royaltyAmount);
    registrar.transferFrom(controller, recoveredBidder, id);
    if (lockOnCreation) {
      registrar.lockDomainMetadataForOwner(id);
    }
    approvedBids[hashOfSig] = false;
    emit DomainBidFulfilled(
      metadata,
      name,
      recoveredBidder,
      id,
      parentId
    );
  }
```

zAuction/contracts/zAuction.sol:L35-L44

```
function acceptBid(bytes memory signature, uint256 rand, address bidder, uint25
    address recoveredbidder = recover(toEthSignedMessageHash(keccak256(abi.enco
    require(bidder == recoveredbidder, 'zAuction: incorrect bidder');
    require(!randUsed[rand], 'Random nonce already used');
    randUsed[rand] = true;
    IERC721 nftcontract = IERC721(nftaddress);
    accountant.Exchange(bidder, msg.sender, bid);
    nftcontract.transferFrom(msg.sender, bidder, tokenid);
    emit BidAccepted(bidder, msg.sender, bid, nftaddress, tokenid);
}
```

Recommendation

- Revert early for checks that depend on the bidder before performing gas-intensive computations.
- Consider adding a dry-run validation for off-chain components before transaction submission.

5.6 zNS - anyone can front-run `fulfillDomainBid` to lock the domain setting or set different metadata Medium ✓ Fixed

> ### Resolution
>
> Addressed with zer0-os/ZNS@ `ab7d62a` by restricting the method to only be callable by the requester.

Description

Anyone observing a call to `fulfillDomainBid` can front-run this call for the original bidder, provide different metadata/royalty amount, or lock the metadata, as these parameters are not part of the bidder's signature. The impact is limited as both metadata, royalty amount, and lock state can be changed by the domain owner after creation.

mples

zNS/contracts/StakingController.sol:L120-L143

```solidity
  function fulfillDomainBid(
  uint256 parentId,
  uint256 bidAmount,
  uint256 royaltyAmount,
  string memory bidIPFSHash,
  string memory name,
  string memory metadata,
  bytes memory signature,
  bool lockOnCreation,
  address recipient
) external {
  bytes32 recoveredBidHash = createBid(parentId, bidAmount, bidIPFSHash, name);
  address recoveredBidder = recover(recoveredBidHash, signature);
  require(recipient == recoveredBidder, "ZNS: bid info doesnt match/exist");
  bytes32 hashOfSig = keccak256(abi.encode(signature));
  require(approvedBids[hashOfSig] == true, "ZNS: has been fullfilled");
  infinity.safeTransferFrom(recoveredBidder, controller, bidAmount);
  uint256 id = registrar.registerDomain(parentId, name, controller, recoveredBi
  registrar.setDomainMetadataUri(id, metadata);
  registrar.setDomainRoyaltyAmount(id, royaltyAmount);
  registrar.transferFrom(controller, recoveredBidder, id);
  if (lockOnCreation) {
    registrar.lockDomainMetadataForOwner(id);
  }
```

Recommendation

Consider adding `metadata` , `royaltyAmount` , and `lockOnCreation` to the message signed by the bidder if the parent should have some control over metadata and lockstatus and restrict access to this function to `msg.sender==recoveredbidder` .

5.7 zNS- Using a digital signature as a hash preimage `Medium`

✓ Fixed

Resolution

ddressed with zer0-os/zNS@ab7d62a by avoiding the use of digital signatures.

Description

Using the encoded signature (r,s,v) or the hash of the signature to prevent replay or track if signatures have been seen/used is not recommended in general, as it may introduce signature malleability issues, as two different signature params (r,s,v) may be producable that validly sign the same data.

The impact for this codebase, however, is limited, due to the fact that openzeppelins `ECDSA` wrapper library is used which checks for malleable ECDSA signatures (high s value). We still decided to keep this as a medium issue to raise awareness, that it is bad practice to rely on the hash of signatures instead of the hash of the actual signed data for checks.

In another instance in zAuction, a global random nonce is used to prevent replay attacks. This is suboptimal and instead, the hash of the signed data (including a nonce) should be used.

Examples

zNS/contracts/StakingController.sol:L120-L152

```
  function fulfillDomainBid(
  uint256 parentId,
  uint256 bidAmount,
  uint256 royaltyAmount,
  string memory bidIPFSHash,
  string memory name,
  string memory metadata,
  bytes memory signature,
  bool lockOnCreation,
  address recipient
) external {
  bytes32 recoveredBidHash = createBid(parentId, bidAmount, bidIPFSHash, name);
  address recoveredBidder = recover(recoveredBidHash, signature);
  require(recipient == recoveredBidder, "ZNS: bid info doesnt match/exist");
  bytes32 hashOfSig = keccak256(abi.encode(signature));
  require(approvedBids[hashOfSig] == true, "ZNS: has been fullfilled");
  infinity.safeTransferFrom(recoveredBidder, controller, bidAmount);
  uint256 id = registrar.registerDomain(parentId, name, controller, recoveredBi
  registrar.setDomainMetadataUri(id, metadata);
  registrar.setDomainRoyaltyAmount(id, royaltyAmount);
  registrar.transferFrom(controller, recoveredBidder, id);
  if (lockOnCreation) {
    registrar.lockDomainMetadataForOwner(id);
  }
  approvedBids[hashOfSig] = false;
  emit DomainBidFulfilled(
    metadata,
    name,
    recoveredBidder,
    id,
    parentId
  );
}
```

zAuction/contracts/zAuction.sol:L35-L39

```
  function acceptBid(bytes memory signature, uint256 rand, address bidder, uint25
    address recoveredbidder = recover(toEthSignedMessageHash(keccak256(abi.enc
    require(bidder == recoveredbidder, 'zAuction: incorrect bidder');
    require(!randUsed[rand], 'Random nonce already used');
    randUsed[rand] = true;
```

Recommendation

Consider creating the bid identifier by hashing the concatenation of all bid parameters instead. Ensure to add replay protection https://github.com/ConsenSys/zer0-zns-audit-2021-05/issues/19. Always check for the hash of the signed data instead of the hash of the encoded signature to track whether a signature has been seen before.

Consider implementing Ethereum typed structured data hashing and signing according to EIP-712.

5.8 zNS - Registrar skips `__ERC721Pausable_init()` Minor

✓ Fixed

Resolution

Addressed with zer0-os/ZNS@ `ab7d62a`

Description

The initialization function of registrar skips the chained initializer `__ERC721Pausable_init` to initialize `__ERC721_init("Zer0 Name Service", "ZNS")`. This basically skips the following initialization calls:

```
abstract contract ERC721PausableUpgradeable is Initializable, ERC721Upgradeable
    function __ERC721Pausable_init() internal initializer {
        __Context_init_unchained();
        __ERC165_init_unchained();
        __Pausable_init_unchained();
        __ERC721Pausable_init_unchained();
    }
```

Examples

zNS/contracts/Registrar.sol:L39-L45

```
function initialize() public initializer {
  __Ownable_init();
  __ERC721_init("Zer0 Name Service", "ZNS");

  // create the root domain
  _createDomain(0, msg.sender, msg.sender, address(0));
}
```

Recommendation

consider calling the missing initializers to register the interface for erc165 if
needed.

5.9 zNS - `Registrar` is `ERC721PausableUpgradeable` but there is no way to actually pause it [Minor] [✓ Fixed]

Resolution

Addressed with zer0-os/ZNS@ `ab7d62a` by exposing pausable functionality
to the contract owner.

Description

The registrar is ownable and pausable but the functionality to pause the
contract is not implemented.

zNS/contracts/Registrar.sol:L8-L12

```
contract Registrar is
  IRegistrar,
  OwnableUpgradeable,
  ERC721PausableUpgradeable
{
```

commendation

Simplification is key. Remove the pausable functionality if the contract is not meant to be paused or consider implementing an external `pause()` function decorated `onlyOwner` .

5.10 zNS - Avoid no-ops Minor   ✓ Fixed

Resolution
Addressed with zer0-os/ZNS@ `ab7d62a` .

Description

Code paths that are causing transactions to be ended with an ineffective outcome or no-operation (no actual state changes) are not advisable, as they consume more gas, hide misconfiguration or error cases (e.g. adding the same controller multiple times), and may impact other processes that rely upon transaction's logs.

Examples

- Reject adding an already existing controller, and removing non existing controller.

zNS/contracts/Registrar.sol:L51-L67

```
/**
  @notice Authorizes a controller to control the registrar
  @param controller The address of the controller
 */
function addController(address controller) external override onlyOwner {
  controllers[controller] = true;
  emit ControllerAdded(controller);
}

/**
  @notice Unauthorizes a controller to control the registrar
  @param controller The address of the controller
 */
function removeController(address controller) external override onlyOwner {
  controllers[controller] = false;
  emit ControllerRemoved(controller);
}
```

Recommendation

Consider reverting code paths that end up in ineffective outcomes (i.e. no-operation) as early as possible.

Appendix 1 - Disclosure

ConsenSys Diligence ("CD") typically receives compensation from one or more clients (the "Clients") for performing the analysis contained in these reports (the "Reports"). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such

business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of code and only the code we note as being within the scope of our review within this report. Any Solidity code itself presents unique and unquantifiable risks as the Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond specified code that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. In some instances, we may perform penetration testing or infrastructure assessments depending on the scope of the particular engagement.

CD makes the Reports available to parties other than the Clients (i.e., "third parties") – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no

responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.
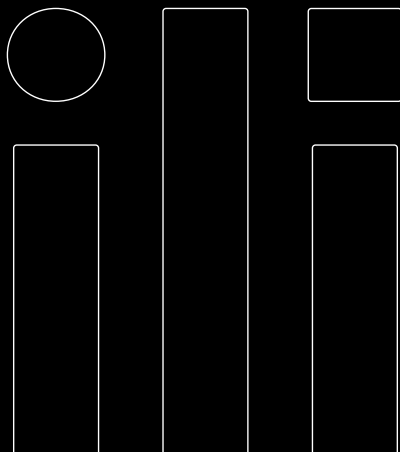
TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.

Request a Security Review Today

Get in touch with our team to request a quote for a smart contract audit.

CONTACT US

AUDITS

FUZZING

SCRIBBLE

BLOG

TOOLS

RESEARCH

ABOUT

CONTACT

CAREERS

PRIVACY POLICY

Subscribe to Our Newsletter

Stay up-to-date on our latest offerings, tools, and the world of blockchain security.

Email*

liz.daldalian@gmail.com

→