CONSENSYS Diligence

Zer0 - zAuction

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.

2.2 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.

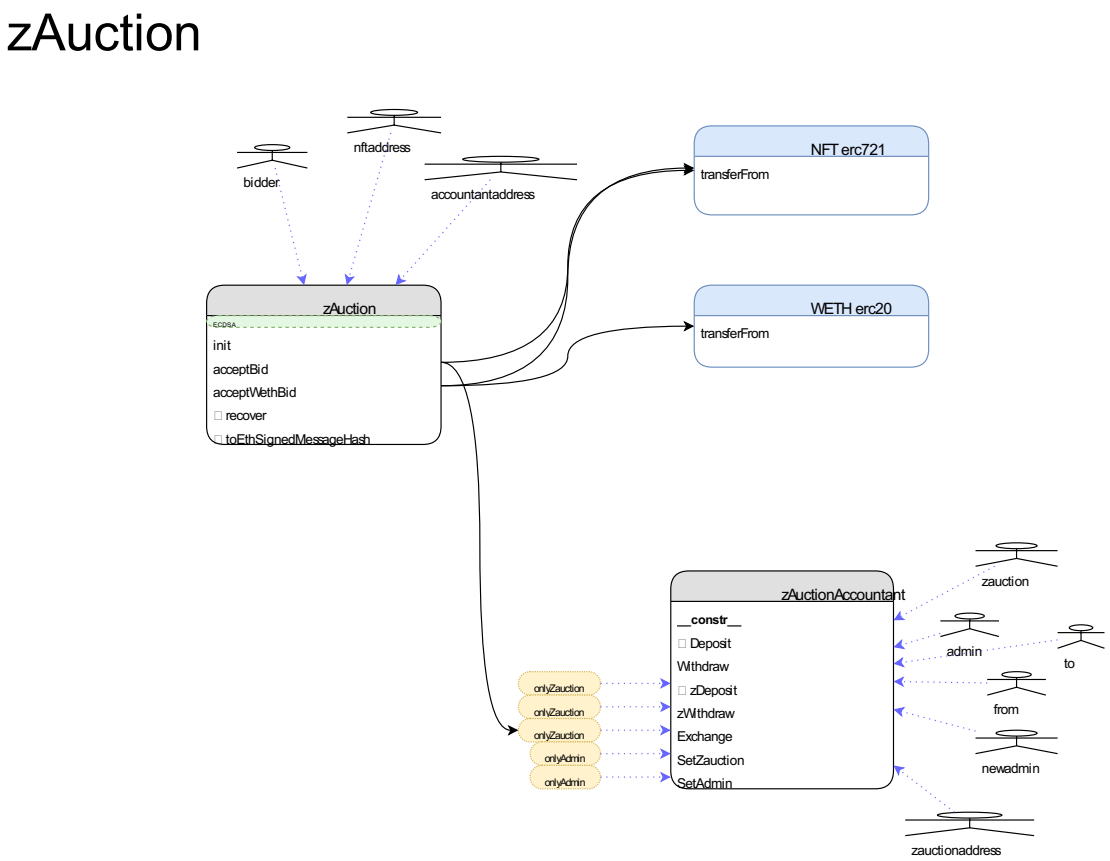
2.6 Week - 4

- 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.



zAuction

`zAuction` is a simple general purpose auction system for NFT's allowing bidders to share bids on an async 2nd layer. Bid price is paid in `WETH` by approving it to the auction contract or in `ETH` by depositing it in a `WETH`-like custom `zAuctionAccountat` contract. Auctions are state-less. Bids are not tracked on the ethereum chain, they don't expire, there is no functionality to cancel them which increases the attack surface for the system. The user journe starts with a bidder sharing a bid on a 2nd layer. The current holder of an NFT can then call one of the `accept*` functions to accept a specific bid and transfer ownership to the recipient. Due to the state-less nature of this system, auctions are not explicitly created by an nft owner. Unaccepted bids for an auction may still be valid after one bid was accepted which might allow an nft owner to force the transfer of ownership by accepting outdated bids.

4 Recommendations

4.1 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);
}
```

4.2 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 vectory):

- 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

```
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

```
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);
```

4.3 zAuction - consider replacing zAuctionAccountant with a call to wrap ETH Fixed

Resolution

`zAuctionAccountant` has been removed with zer0-os/zAuction@`135b2aa`. The client provided the following statement:

> 4.5 accountant deprecated, but address is still used in the hashed data inputs
> 4.6 zAuctionAccountant is deprecated, Zauction and Zsale now have single functions expecting a pre-approved weth balance

Description

`zAuctionAccountant` is essentially implementing a simplified version of `WETH` raising the question of whether it is worth re-creating a variant of `WETH` which may just be increasing the overall complexity of the system.

One way to tackle this would be to provide a method that accepts `ETH` bids that automatically wraps submissions to `WETH` or only allow `WETH` deposits/approvals.

4.4 zAuction - separate test artifacts from system contracts

Fixed

Resolution

Test artifacts are now in a separate folder with zer0-os/zAuction@`135b2aa`. The client provided the following statement:

> 4.4 test artifacts separated into tokentest folder

Description

Consider moving `ERC721TestToken` to a dedicated `test` subdirectory.

zAuction/contracts/ERC721TestToken.sol:L28-L28

```
contract ERC721TestToken is Context, AccessControlEnumerable, ERC721Enumerable, ERC721Burnable, ERC721
```

4.5 zAuction - Consider using the openzeppelin npm packages instead of copying their contracts

Fixed

Resolution

Addressed with zer0-os/zAuction@`135b2aa` and the following statement:

> 4.3 now using openzeppelin npm packages

Description

It is considered best practice to reference external contracts that are commonly available via an integrity checked dependency and package management system from that system instead of copying contracts manually into the projects source directory.

see `zAuction/contracts/oz`. Consider importing the contracts via npm.

4.6 zAuction - consider switching to usingFor notation Fixed

Resolution

The affected contract has been removed with zer0-os/zAuction@`135b2aa`. The client provided the following statement:

> 4.2 usingFor notation no longer needed, functions deprecated

Description

Declaring `SafeMath usingFor uint256` may improve code readability.

`ethbalance[msg.sender] = SafeMath.add(ethbalance[msg.sender], msg.value);`

to

`ethbalance[msg.sender] = ethbalance[msg.sender].add(msg.value)`

Examples

zAuction/contracts/zAuctionAccountant.sol:L34-L34

```
ethbalance[msg.sender] = SafeMath.add(ethbalance[msg.sender], msg.value);
```

4.7 zAuction - adhere to commonly used code style and naming conventions Fixed

Resolution

Addressed with zer0-os/zAuction@`135b2aa` and the following statement:

> 4.1 code naming conventions fixed, functions deprecated

Description

Consider following the official solidity code style guidelines. Consider enforcing code style guidelines in the CI pipeline using style checker/linter tools.

Examples

- the initialization function is named `initialize` in zNS/openzeppelin but `init` in zAuction

zAuction/contracts/zAuction.sol:L22-L22

```
function init(address accountantaddress) external {
```

- Methods in `zAuctionAccountant` are written in PascalCase which makes them less easily distinguishable from Events/Contracts.

zAuction/contracts/zAuctionAccountant.sol:L33-L33

```
function Deposit() external payable {
```

- lowercase starting events may be hard to distinguish from function calls

zAuction/contracts/zAuctionAccountant.sol:L14-L16

```
event zDeposited(address indexed depositer, uint256 amount);
event zWithdrew(address indexed withrawer, uint256 amount);
event zExchanged(address indexed from, address indexed to, uint256 amount);
```

- adhere to token contract naming convention: `Exchange()` -> `transfer()`

zAuction/contracts/zAuctionAccountant.sol:L54-L58

```
function Exchange(address from, address to, uint256 amount) external onlyZauction {
    ethbalance[from] = SafeMath.sub(ethbalance[from], amount);
    ethbalance[to] = SafeMath.add(ethbalance[to], amount);
    emit zExchanged(from, to, amount);
}
```

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 zAuction - incomplete / dead code `zWithdraw` and `zDeposit`

`Major` `Fixed`

Resolution
obsolete with changes from zer0-os/zAuction@`135b2aa` removing the `zAccountAccountant`.

Description

The code generally does not appear to be production-ready. The methods `zWithdraw` and `zDeposit` do not appear to be properly implemented. `zWithdraw` rather burns `ETH` balance than withdrawing it for an account (missing transfer) and `zDeposit` manipulates an accounts balance but never receives the `ETH` amount it credits to an account.

Examples

zAuction/contracts/zAuctionAccountant.sol:L44-L52

```
    function zDeposit(address to) external payable onlyZauction {
        ethbalance[to] = SafeMath.add(ethbalance[to], msg.value);
        emit zDeposited(to, msg.value);
    }

    function zWithdraw(address from, uint256 amount) external onlyZauction {
        ethbalance[from] = SafeMath.sub(ethbalance[from], amount);
        emit zWithdrew(from, amount);
    }
```

Recommendation

The methods do not seem to be used by the zAuction contract. It is highly discouraged from shipping incomplete implementations in productive code. Remove dead/unreachable code. Fix the implementations to perform proper accounting before reintroducing them if they are called by zAuction.

5.2 zAuction - Unpredictable behavior for users due to admin front running or general bad timing `Major` `Fixed`

Resolution
obsolete with changes from zer0-os/zAuction@`135b2aa` removing the `zAccountAccountant`. The client provided the following remark:

5.20 accountant deprecated

Description

An administrator of `zAuctionAccountant` contract can update the `zAuction` contract without warning. This has the potential to violate a security goal of the system.

Specifically, privileged roles could use front running to make malicious changes just ahead of incoming transactions, or purely accidental negative effects could occur due to the unfortunate timing of changes.

In general users of the system should have assurances about the behavior of the action they’re about to take.

Examples

- updating the `zAuction` takes effect immediately. This has the potential to fail acceptance of bids by sellers on the now outdated `zAuction` contract as interaction with the accountant contract is now rejected. This forces bidders to reissue their bids in order for the seller to be able to accept them using the Accountant contract. This may also be used by admins to selectively censor the acceptance of accountant based bids by changing the active zAuction address.

zAuction/contracts/zAuctionAccountant.sol:L60-L68

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

    function SetAdmin(address newadmin) external onlyAdmin{
        admin = newadmin;
        emit AdminSet(msg.sender, newadmin);
    }
```

- Upgradeable contracts may introduce the same unpredictability issues where the proxyUpgradeable owner may divert execution to a new zNS registrar implementation selectively for certain transactions or without prior notice to users.

Recommendation

The underlying issue is that users of the system can’t be sure what the behavior of a function call will be, and this is because the behavior can change at any time.

We recommend giving the user advance notice of changes with a time lock. For example, make all system-parameter and upgrades require two steps with a mandatory time window between them. The first step merely broadcasts to users that a particular change is coming, and the second step commits that change after a suitable waiting period. This allows users that do not accept the change to withdraw immediately.

Validate arguments before updating contract addresses (at least != current/0x0). Consider

implementing a 2-step admin ownership transfer (transfer+accept) to avoid losing control of the contract by providing the wrong `ETH` address.

5.3 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

```
function acceptBid(bytes memory signature, uint256 rand, address bidder, uint256 bid, address nftaddre
    address recoveredbidder = recover(toEthSignedMessageHash(keccak256(abi.encode(rand, address(this),
    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, recoveredBidder);
  registrar.setDomainMetadataUri(id, metadata);
  registrar.setDomainRroyaltyAmount(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 explicitly.

5.4 zAuction - pot. initialization fronrunning and unnecessary init function Medium Fixed

Resolution
Addressed with zer0-os/zAuction@`135b2aa` and the following statement: 5.21 init deprecated, constructor added

Description

The `zAuction` initialization method is unprotected and while only being executable once, can be called by anyone. This might allow someone to monitor the mempool for new deployments of this contract and fron-run the initialization to initialize it with different parameters.

A mitigating factor is that this condition can be detected by the deployer as subsequent calls to `init()` will fail.

- Note: this doesn't adhere to common interface naming convention/oz naming convention where this method would be called `initialize`.
- Note: that zNS in contrast relies on ou/initializable pattern with proper naming.
- Note: that this function might not be necessary at all and should be replaced by a constructor instead, as the contract is not used with a proxy pattern.

Examples

zAuction/contracts/zAuction.sol:L22-L26

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

Recommendation

The contract is not used in a proxy pattern, hence, the initialization should be performed in the `constructor` instead.

5.5 zAuction - unclear upgrade path Medium Fixed

Resolution
obsolete with changes from zer0-os/zAuction@`135b2aa` removing the `zAccountAccountant`.

Description

`zAuction` appears to implement an upgrade path for the auction system via `zAuctionAccountant`. `zAuction` itself does not hold any value. The `zAuctionAccountant` can be configured to allow only one `zAution` contract to interact with it. The update of the contract reference takes effect immediately (issue 4.5).

Acceptance of bids via the accountant on the old contract immediately fail after an admin updates the referenced `zAuction` contract while `WETH` bids may still continue. This may create an unfavorable scenario where two contracts may be active in parallel accepting `WETH` bids.

It should also be noted that 2nd layer bids (signed data) using the accountant for the old contract will not be acceptable anymore.

Examples

zAuction/contracts/zAuctionAccountant.sol:L60-L63

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

Recommendation

Consider re-thinking the upgrade path. Avoid keeping multiple versions of the auction contact active.

5.6 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

```
  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, recoveredBidder);
  registrar.setDomainMetadataUri(id, metadata);
  registrar.setDomainRroyaltyAmount(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, uint256 bid, address nftaddre
    address recoveredbidder = recover(toEthSignedMessageHash(keccak256(abi.encode(rand, address(this),
    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.

5.7 zAuction - functionality outlined in specification that is not implemented yet Medium Fixed

Resolution

implemented as `zSale` with changes from zer0-os/zAuction@`135b2aa`.

Description

The specification outlines three main user journeys of which one does not seem to be implemented.

1. Users will be able to do simple transfer of NFTs. - which does not require functionality in the smart contract
2. Users will be able to post NFTs at a sale price, and buy at that price. - does not seem to be implemented
3. Users will be able to post NFTs for auction, bid on auctions, and accept bids - is implemented

Recommendation

User flow (2) is not implemented in the smart contract system. Consider updating the spec or clearly highlighting functionality that is still in development for it to be excluded from security testing.

5.8 zAuction - auctions/offers can be terminated by reusing the auction id Medium Fixed

Resolution

Addressed with zer0-os/zAuction@`8ff0eab` by binding `saleId` to the `seller` in `zSale` and the `auctionId` to the `bidder` in `zAuction`.

In the `zSale` case the `saleId` is chosen by the seller. The offer (signed offer parameters including saleid) is shared on an off-chain channel. The buyer calls `zSale.purchase` to buy the token from the offer. The offer and all offers containing the same seller+saleid are then invalidated.

In `zAuction` there is no seller or someone who initiates an auction. Anyone can bid for nft's held by anyone else. The bidder chooses an auction id. There might be multiple bidders. Since the auctionId is an individual choice and the smart contract does not enforce an auction to be started there may be multiple auctions for the same token but using different auction ids. The current mechanism automatically invalidates all current bids for the token+auctionId combination for the winning bidder. Bids by other holders are not automatically invalidated but they can be invalidated manually via `cancelBidsUnderPrice` for an auctionId. Note that the winning bid is chosen by the nftowner/seller. The new owner of the nft may be able to immediately accept another bid and transfer the token `[seller]--acceptBid-->[newOwner-A]--acceptBid-->[newOwner-B]`.

Description

zer0-os/zAuction@`2f92aa1` introduced a way of tracking auctions/sales by using an `auctionId/saleId`. The id's are unique and the same id cannot be used for multiple auctions/offers.

Two different auctions/offers may pick the same id, the first auction/offer will go through while the latter cannot be fulfilled anymore. This may happen accidentally or intentionally be forced by a malicious actor to terminate active auctions/sales (griefing, front-running).

Examples

Alice puts out an offer for someone to buy nft `X` at a specific price. Bob decides to accept that offer and buy the nft by calling `zSale.purchase(saleid, price, token, ...)`. Mallory monitors the mempool, sees this transaction, front-runs it to fulfill its own sale (for a random nft he owns) reusing the saleid from Bobs transaction. Since Mallories transaction marks the saleid as consumed it terminates Alie's offer and hence Bob cannot buy the token as the transaction will revert.

Recommendation

Consider using `keccak(saleid+nftcontract+nfttokenid)` as the unique sale/auction identifier instead, or alternatively associate the bidder address with the auctionId (require that `consumed[bidder][auctionId]== false`)

5.9 zAuction - hardcoded ropsten token address Minor Fixed

Resolution

Addressed with zer0-os/zAuction@`135b2aa` and the following statement:

> 5.30 weth address in constructor

Note: does not perform input validation as recommended

Description

The auction contract hardcodes the WETH ERC20 token address. this address will not be functional when deploying to mainnet.

Examples

zAuction/contracts/zAuction.sol:L15-L16

```
    IERC20 weth = IERC20(address(0xc778417E063141139Fce010982780140Aa0cD5Ab)); // rinkeby weth
```

Recommendation

Consider taking the used `WETH` token address as a constructor argument. Avoid code changes to facilitate testing! Perform input validation on arguments rejecting `address(0x0)` to facilitate the detection of potential misconfiguration in the deployment pipeline.

5.10 zAuction - accountant allows zero value withdrawals/deposits/exchange Minor Fixed

Resolution
Obsolete. The affected component has been removed from the system with zer0-os/zAuction@`135b2aa`.

Description

Zero value transfers effectively perform a no-operation sometimes followed by calling out to the recipient of the withdrawal.

A transfer where `from==to` or where the value is `0` is ineffective.

Examples

zAuction/contracts/zAuctionAccountant.sol:L38-L42

```
function Withdraw(uint256 amount) external {
    ethbalance[msg.sender] = SafeMath.sub(ethbalance[msg.sender], amount);
    payable(msg.sender).transfer(amount);
    emit Withdrew(msg.sender, amount);
}
```

zAuction/contracts/zAuctionAccountant.sol:L33-L36

```
function Deposit() external payable {
    ethbalance[msg.sender] = SafeMath.add(ethbalance[msg.sender], msg.value);
    emit Deposited(msg.sender, msg.value);
}
```

zAuction/contracts/zAuctionAccountant.sol:L44-L58

```
    function zDeposit(address to) external payable onlyZauction {
        ethbalance[to] = SafeMath.add(ethbalance[to], msg.value);
        emit zDeposited(to, msg.value);
    }

    function zWithdraw(address from, uint256 amount) external onlyZauction {
        ethbalance[from] = SafeMath.sub(ethbalance[from], amount);
        emit zWithdrew(from, amount);
    }

    function Exchange(address from, address to, uint256 amount) external onlyZauction {
        ethbalance[from] = SafeMath.sub(ethbalance[from], amount);
        ethbalance[to] = SafeMath.add(ethbalance[to], amount);
        emit zExchanged(from, to, amount);
    }
```

Recommendation

Consider rejecting ineffective withdrawals (zero value) or at least avoid issuing a zero value `ETH` transfers. Avoid emitting successful events for ineffective calls to not trigger 3rd party components on noop's.

5.11 zAuction - seller should not be able to accept their own bid Minor Fixed

Resolution
Addressed with zer0-os/zAuction@`135b2aa` by disallowing the seller to accept their own bid. The client provided the following note: 5.28 seller != buyer required

Description

A seller can accept their own bid which is an ineffective action that is emitting an event.

Examples

zAuction/contracts/zAuction.sol:L35-L56

```
function acceptBid(bytes memory signature, uint256 rand, address bidder, uint256 bid, address nftaddre
    address recoveredbidder = recover(toEthSignedMessageHash(keccak256(abi.encode(rand, address(this),
    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);
}

/// @dev 'true' in the hash here is the eth/weth switch
function acceptWethBid(bytes memory signature, uint256 rand, address bidder, uint256 bid, address nfta
    address recoveredbidder = recover(toEthSignedMessageHash(keccak256(abi.encode(rand, address(this),
    require(bidder == recoveredbidder, 'zAuction: incorrect bidder');
    require(!randUsed[rand], 'Random nonce already used');
    randUsed[rand] = true;
    IERC721 nftcontract = IERC721(nftaddress);
    weth.transferFrom(bidder, msg.sender, bid);
    nftcontract.transferFrom(msg.sender, bidder, tokenid);
    emit WethBidAccepted(bidder, msg.sender, bid, nftaddress, tokenid);
}
```

Recommendation

Disallow transfers to self.

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 Solidity code and only the Solidity code we note as being within the scope of our review

within this report. 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 Solidity that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty.

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.