$

Competitive Security Assessment

DeekNetwork

Nov 21st, 2024

Secure3

secure3.io

Summary

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

The comprehensive examination and auditing scope includes:

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

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

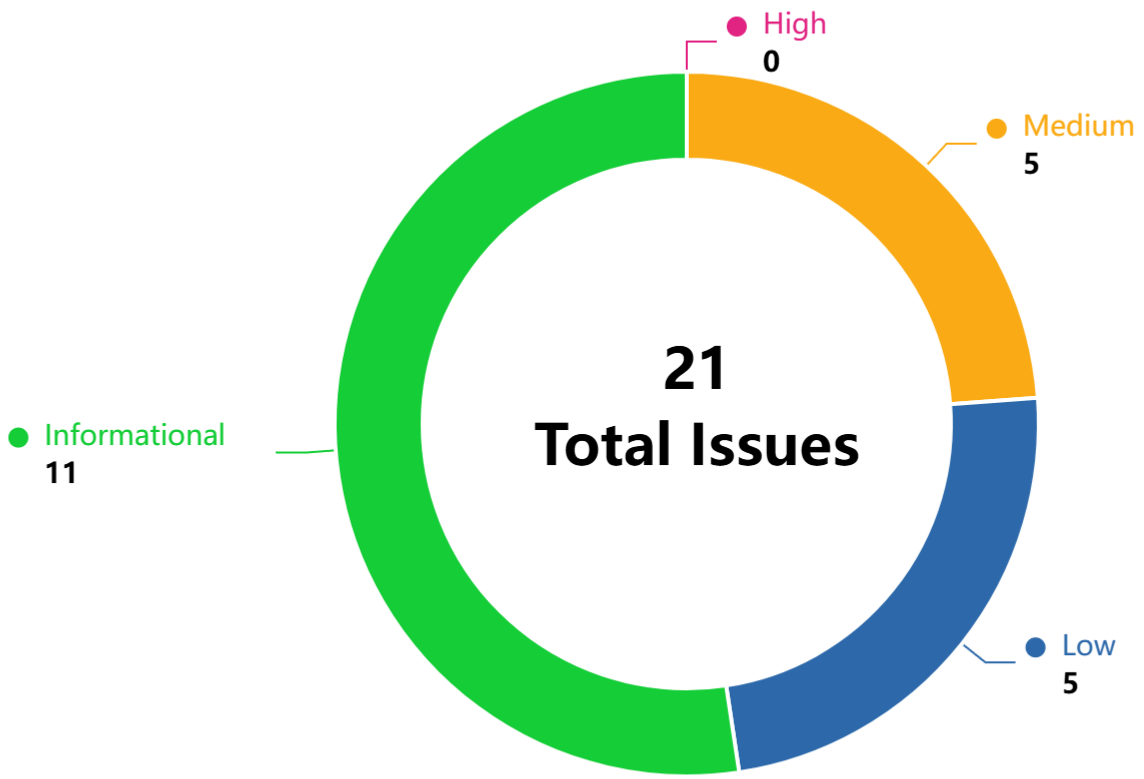
Overview

Project Name	DeekNetwork
Language	solidity
Codebase	- https://gitlab.com/Keccak256-evg/zeek/zeek-contracts/-/tree/zeek_v3_audit - audit version - 03f0a6b965db133dd616947f739626d6678bd978 - final version - be6b0a55745c437c4c318043ff3bf8d9c84c662d

Audit Scope

File	SHA256 Hash
contracts/core/Wish.sol	e7d04385217a564d7f7987776e0252de2f8321c5757e116c797512dd204eb6b8
contracts/base/WishBase.sol	1ada23a5d1b12fc0cf59de762e5354d66b26aece76aebab0a22f63ee4bebd4fe
contracts/core/Governance.sol	ad9539befe1323009877e1651f253c3fd4b0596bb623d7929f3d96853c015a30
contracts/libraries/ZeekDataTypes.sol	c30017254cb0bc7e691158eaff22c95816a15ae60beb44bf9afe3df716a54025
contracts/libraries/ZeekEvents.sol	3231f8812e206c00bd4d9207d68901acf9843065a4b00d9187a34cd933643b3a
contracts/core/Profile.sol	38ffd9c1cdf0829bdb7396481df92b4fc60eb097aad7157466d150b40f99564c
contracts/base/ZeekBase.sol	50e81e1437eef670c0e6cac0d29597ec23719642fe301770fd69f3c6631ed3e3
contracts/libraries/ZeekStorage.sol	9f7c97d46bd5147397f6581857b6797d2296eec0c3f398fb5f539500f95a4ca4
contracts/libraries/ZeekErrors.sol	57f98da0e9c51bf7cdaf905e70f314c634923e634ead449c8056e0628baf77cf
contracts/libraries/Constants.sol	2c81fe07f546d40edeb120d7887d104fcb527eed467e703df8e3f03abbd07593

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
ZEK-1	`tokenVersion == 0` will raise errors	DOS	Medium	Fixed	***
ZEK-2	`bidWish` could be front-run and cause victims to pay more(if they have large allowance)	Race condition	Medium	Fixed	***
ZEK-3	Potential pass the `_validateRecoveredAddress()` in function `offerWish()`	Logical	Medium	Fixed	***
ZEK-4	Potential front-run attack	Logical	Medium	Fixed	***
ZEK-5	An Account could be targeted and prevented from issuing wish by front-running with a same salt	DOS	Medium	Fixed	***
ZEK-6	`_baseTransfer()` always reverts for `token` blacklisted recipients	DOS	Low	Fixed	***

ZEK-7	Users may send ETH alongside a token contribution hence locking up funds	Logical	Low	Fixed	***
ZEK-8	Use disableInitializers to prevent front-running on the initialize function		Low	Fixed	***
ZEK-9	Use `disableInitializers` to prevent front-running on the initialize function	Privilege Related	Low	Fixed	***
ZEK-10	Logical Risk in Profile::padNumber Function	Logical	Low	Acknowledged	***
ZEK-11	in `_bestOffer()` function, it returns by default the 0th index offer and not best `offer`	Logical	Informational	Declined	***
ZEK-12	`setFinance()` does not verify that `newFinance` is not `priorFinance`	Logical	Informational	Fixed	***
ZEK-13	`constants` should be defined rather than using magic numbers	Language Specific	Informational	Fixed	***
ZEK-14	The functionality that has been commented out	Logical	Informational	Fixed	***
ZEK-15	Strange return value `b` of functions	Code Style	Informational	Fixed	***
ZEK-16	Potential array out-of-bounds error	Logical	Informational	Fixed	***
ZEK-17	No Storage Gap for Upgradeable Contract Might Lead to Storage Slot Collision	Logical	Informational	Acknowledged	***
ZEK-18	Missing Zero Address Check	Code Style	Informational	Fixed	***
ZEK-19	Incorrect naming of parameter and struct member can cause incorrect values to be assigned	Logical	Informational	Fixed	***
ZEK-20	Incorrect error thrown in `_validateMsgValue()`	Logical	Informational	Fixed	***

ZEK-21	Inconsistent indexing of events in ZeekEvents.sol can lead to decreased off-chain monitoring efficiency	Language Specific	Informational	Fixed	***
ZEK-22	Gas Optimizations	Gas Optimization	Informational	Fixed	***

ZEK-1:`tokenVersion == 0` will raise errors

Category	Severity	Client Response	Contributor
DOS	Medium	Fixed	***

Code Reference

- code/contracts/base/WishBase.sol#L365-L393

```
365: function _bidAllocate(
366:         uint wishId,
367:         ZeekDataTypes.WishStruct storage wish,
368:         uint256 lastOwner,
369:         uint256 lastValue,
370:         uint256 value
371:     ) internal {
372:         ZeekDataTypes.BidRatio storage rate = _getWishStorage()._bidRatio;
373:
374:         uint256 ownerValue = lastValue + lastValue * rate.owner / 100;
375:         uint256 talentValue = lastValue * rate.talent / 100;
376:
377:         uint256 committeeValue = value - ownerValue - talentValue;
378:         // start to allocate
379:         // only best answer to vault
380:         _vault(_bestOffer(wish).talent, wishId, wish.price.token, wish.price.tokenVersion, talentValue, ZeekDataTypes.WishScene.Bid, ZeekDataTypes.WishParticipant.Talent);
381:
382:         // transfer to owner directly
383:         if (ownerValue > 0) {
384:             _baseTransfer(wish.price.tokenVersion, wish.price.token, ownerValue, payable(_msgSender()), payable(_getProfileStorage()._profileById[lastOwner].owner));
385:         }
386:
387:         if (talentValue > 0) {
388:             _baseCustody(wish.price.tokenVersion, wish.price.token, talentValue, payable(_msgSender()));
389:         }
390:         if (committeeValue > 0) {
391:             _baseTransfer(wish.price.tokenVersion, wish.price.token, committeeValue, payable(_msgSender()), payable(_getGovernanceStorage()._finance));
392:         }
393:     }
```

- code/contracts/core/Wish.sol#L325-L364

```
function _bidWish(ZeekDataTypes.WishBidData calldata data) internal {
        ZeekDataTypes.WishStruct storage wish = _bid(data);

        // emit event
        emit ZeekEvents.WishTransferred(
            data.wishId,
            wish.owner,
            ZeekDataTypes.WishTransferType.Bid,
            wish.price.value,
            wish.price.bidValue,
            wish.modifyTime
        );
    }

    /**
     * Bid Wish Action Function
     * @param data data
     */
    function _bid(
        ZeekDataTypes.WishBidData calldata data
```

```
    ) internal returns (ZeekDataTypes.WishStruct storage) {
        ZeekDataTypes.WishStruct storage wish = _getWish(data.wishId);

        uint256 lastPrice = wish.price.value;
        uint256 lastOwner = wish.owner;
        uint256 nextPrice = wish.price.bidValue;
        uint256 bidder = _bidValidation(wish, nextPrice);

        // switch best answer is not allowed
        // storage change
        wish.price.value = nextPrice;
        wish.price.bidValue = _bidPrice(wish.price.token, nextPrice);
        wish.owner = bidder;
        wish.modifyTime = uint64(block.timestamp);

        // offer Bonus to all
        _bidAllocate(data.wishId, wish, lastOwner, lastPrice, nextPrice);

        return wish;
    }
```

Description

*****: The function `bidWish` is used to bid a wish with a price, it will call `_bid()` function. In `_bid()` function, the last bid value will be returned back to the last owner by calling `_bidAllocate` function:

```
        uint256 lastPrice = wish.price.value;
        uint256 lastOwner = wish.owner;
        uint256 nextPrice = wish.price.bidValue;
        ...
        _bidAllocate(data.wishId, wish, lastOwner, lastPrice, nextPrice);
```

In `_bidAllocate` function, it will call `_baseTransfer` to transfer tokens to the last onwer's account:

```
if (ownerValue > 0) {
            _baseTransfer(wish.price.tokenVersion, wish.price.token, ownerValue, payable(_msgSender()), paya
ble(_getProfileStorage()._profileById[lastOwner].owner));
        }
```

There is a potential DOS attack. The attacker can first create a contract with following code:

```
receive() external payable {
        revert();
}
```

And then the attacker registers the contract as the `_getProfileStorage()._profileById[lastOwner].owner`.
If the `tokenVersion` is 0, when the next bidder tries to call `bidWish` function, the `_baseTransfer` will send ether to the attacker's contract:

```
if (tokenVersion == 0) {
            (bool success, ) = to.call{value: amount}('');
            if (!success) {
                revert ZeekErrors.TransferFailed();
            }
        }
```

Since the attack contract reverts in the `receive()` function, the `bidWish` function will revert too. As a result, no one can bid this wish any more.
The same issue exists in `askWish` function.

Recommendation

*****: Consider converting ETH to `WETH` and transfering `WETH` to the user if it failing to send ether, for example:

```
if (tokenVersion == 0) {
            (bool success, ) = to.call{value: amount}('');
            if (!success) {
                WETH.deposit{value: amount}();
 IERC20(WETH).safeTransfer(to, amount);
            }
        }
```

Client Response

client response : Fixed. This contract has a whitelist token for issue wish.
Only planed USDT and USDC for creating wish.
But it still supports ETH in code part. We'll discuss this case later.
Finally we provide a configuration when open Native Token in the token whitelist.
use WETH for the fallback case.
related commits:
https://gitlab.com/Keccak256-evg/zeek/zeek-contracts/-/commit/8b2db122a190e9563eb43acc687a9991a5c9a

74
https://gitlab.com/Keccak256-evg/zeek/zeek-contracts/-/commit/66d1f3c84ec84ba3fc573504e47ddcdf2ee041f3

ZEK-2: `bidWish` could be front-run and cause victims to pay more(if they have large allowance)

Category	Severity	Client Response	Contributor
Race condition	Medium	Fixed	***

Code Reference

- code/contracts/core/Wish.sol#L343-L364

```
343: function _bid(
344:         ZeekDataTypes.WishBidData calldata data
345:     ) internal returns (ZeekDataTypes.WishStruct storage) {
346:         ZeekDataTypes.WishStruct storage wish = _getWish(data.wishId);
347:
348:         uint256 lastPrice = wish.price.value;
349:         uint256 lastOwner = wish.owner;
350:         uint256 nextPrice = wish.price.bidValue;
351:         uint256 bidder = _bidValidation(wish, nextPrice);
352:
353:         // switch best answer is not allowed
354:         // storage change
355:         wish.price.value = nextPrice;
356:         wish.price.bidValue = _bidPrice(wish.price.token, nextPrice);
357:         wish.owner = bidder;
358:         wish.modifyTime = uint64(block.timestamp);
359:
360:         // offer Bonus to all
361:         _bidAllocate(data.wishId, wish, lastOwner, lastPrice, nextPrice);
362:
363:         return wish;
364:     }
```

Description

***: In the function `_bid`, if a user wants to bid a wish, he can't specify how many he wants to pay as this is recorded as `wish.price.bidValue`.

```
    function _bid(
        ZeekDataTypes.WishBidData calldata data
    ) internal returns (ZeekDataTypes.WishStruct storage) {
        ZeekDataTypes.WishStruct storage wish = _getWish(data.wishId);

        uint256 lastPrice = wish.price.value;
        uint256 lastOwner = wish.owner;
        uint256 nextPrice = wish.price.bidValue;
        uint256 bidder = _bidValidation(wish, nextPrice);

        // switch best answer is not allowed
        // storage change
        wish.price.value = nextPrice;
        wish.price.bidValue = _bidPrice(wish.price.token, nextPrice);
        wish.owner = bidder;
        wish.modifyTime = uint64(block.timestamp);

        // offer Bonus to all
        _bidAllocate(data.wishId, wish, lastOwner, lastPrice, nextPrice);

        return wish;
    }
```

In `_bidAllocate`, the user will have to pay `wish.price.bidValue` to `owner`, `talent` and the `committee(platform)`. For the previous owner, **he will get a profit of** `lastValue * rate.owner / 100`

```
        // transfer to owner directly
        if (ownerValue > 0) {
            _baseTransfer(wish.price.tokenVersion, wish.price.token, ownerValue, payable(_msgSender()), payable(_getProfileStorage()._profileById[lastOwner].owner));
        }

        if (talentValue > 0) {
            _baseCustody(wish.price.tokenVersion, wish.price.token, talentValue, payable(_msgSender()));
        }
        if (committeeValue > 0) {
            _baseTransfer(wish.price.tokenVersion, wish.price.token, committeeValue, payable(_msgSender()), payable(_getGovernanceStorage()._finance));
        }
```

So consider the following scenario:

1. User `A` has approved `$100` of `Token` to `Wish` contract.
2. User `A` tries to call `bid` to buy the wish with `$50` worth of `token`.
3. User `B` front-runs this call and `bid`s first, pushing the `wish` price to `$90`.

4. As a result, User `A` ended up paying `$90`, including `$50 + $50 * rate.owner / 100` to User `B`, causing loss of funds.

Recommendation

***: To mitigate this issue, it is recommended to:

- Allow user specify the amount of token they accept, revert if it doesn't match.
- Add cooldown time of a wish once a user bid.

Client Response

client response : Fixed. Make the change in commit: https://gitlab.com/Keccak256-evg/zeek/zeek-contracts/-/commit/ca5a729de2e90d8d3d1283b05e85b0d90255c741

ZEK-3:Potential pass the `_validateRecoveredAddress()` in function `offerWish()`

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	***

Code Reference

- code/contracts/base/EIP712Base.sol#L21-30
- code/contracts/base/EIP712Base.sol#L21

```
21: function _validateRecoveredAddress(
22:         bytes32 digest,
23:         address signer,
24:         ZeekDataTypes.EIP712Signature calldata sig
25:     ) internal view {
26:         if (sig.deadline < block.timestamp) revert ZeekErrors.SignatureExpired();
27:         if (!signer.isValidSignatureNow(digest, sig.signature)) {
28:             revert ZeekErrors.SignatureInvalid();
29:         }
30:     }
```

```
21: function _validateRecoveredAddress(
```

- code/contracts/core/Wish.sol#L53-76
- code/contracts/core/Wish.sol#L53

```
53: function offerWish(
54:         ZeekDataTypes.WishApplyData calldata vars,
55:         ZeekDataTypes.EIP712Signature calldata applySig
56:     ) external override {
57:         _validateRecoveredAddress(
58:             _calculateDigest(
59:                 keccak256(
60:                     abi.encode(
61:                         ZeekDataTypes.OFFER_WISH_WITH_SIG_TYPEHASH,
62:                         vars.wishId,
63:                         vars.talent,
64:                         vars.linker,
65:                         vars.applyTime,
66:                         vars.applyNonce,
67:                         applySig.deadline
68:                     )
69:                 )
70:             ),
71:             vars.talent,
72:             applySig
```

```
73:         );
74: 
75:         _offerWish(vars);
76:     }
```

```
function offerWish(
```

Description

***: The function `offerWish()` will call the function `_validateRecoveredAddress()` to validate the recovered address.
The `signer.isValidSignatureNow()` will be called in the `_validateRecoveredAddress()` and the `signer` is the paramter in the `offerWish()` given by the user.

```
    function isValidSignatureNow(address signer, bytes32 hash, bytes memory signature) internal view returns (bool) {
        if (signer.code.length == 0) {
            (address recovered, ECDSA.RecoverError err, ) = ECDSA.tryRecover(hash, signature);
            return err == ECDSA.RecoverError.NoError && recovered == signer;
        } else {
            return isValidERC1271SignatureNow(signer, hash, signature);
        }
    }
```

According to the codes in the function `isValidSignatureNow()`, the user can easy to pass the validate if the `signer` is a contract and implement `IERC1271.isValidSignature()` to pass the validate.
As a result, the user can validate the `signer` with a contract address and offer the wish successfully. The `signer(vars.talent)` can be any value rather than the best offer.
***: The function offerWish() will invoke the function _validateRecoveredAddress() for validating the recovered address. In _validateRecoveredAddress(), signer.isValidSignatureNow() will be called. The signer is a parameter provided by the user in offerWish(). According to the code within the function isValidSignatureNow(), if the signer is a contract and implements IERC1271.isValidSignature(), the user can easily pass the verification and there is no need to provide the best offer.

Recommendation

***: Recommend updating the logic to validate the `signer(vars.talent)`.
***: Modify the logic for validating signer(vars.talent).

Client Response

client response : Fixed. Fixed in this commit: https://gitlab.com/Keccak256-evg/zeek/zeek-contracts/-/commit/176ed3f2dae9ca17aa8e2a9abc7ef277525a370a
Checked the signer is var.talent which means signed by var.talent.

ZEK-4:Potential front-run attack

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	***

Code Reference

- code/contracts/core/Wish.sol#L388-L414
- code/contracts/core/Wish.sol#L430-L447

```
388: function _ask(
389:         ZeekDataTypes.WishAskData calldata data
390:     ) internal returns (ZeekDataTypes.WishStruct storage) {
391:         ZeekDataTypes.WishStruct storage wish = _getWish(data.wishId);
392:
393:         // not zero,
394:         uint256 asker = _askValidation(wish);
395:
396:         uint256 lastOwner = wish.owner;
397:         // switch best answer is not allowed
398:         // storage change
399:         wish.price.token = wish.quote.token;
400:         wish.price.tokenVersion = wish.quote.tokenVersion;
401:         wish.price.value = wish.quote.value;
402:         wish.price.bidValue = _bidPrice(
403:             wish.price.token,
404:             wish.price.value
405:         );
406:         wish.quote.value = 0; // clear quote
407:         wish.owner = asker;
```

```
408:         wish.modifyTime = uint64(block.timestamp);
409:
410:         // offer Bonus to all
411:         _askAllocate(lastOwner, wish.price);
412:
413:         return wish;
414:     }
```

```
430: function _cut(
431:         ZeekDataTypes.WishCutData calldata data
432:     ) internal returns (ZeekDataTypes.WishStruct storage) {
433:         ZeekDataTypes.WishStruct storage wish = _getWish(data.wishId);
434:         _cutValidation(data, wish);
435:         // switch best answer is not allowed
436:         // storage change
437:         wish.quote = ZeekDataTypes.TokenValue(
438:             wish.price.token,
439:             wish.price.tokenVersion,
440:             data.quote
441:         );
442:         // wish.quote.tokenVersion = newQuote.tokenVersion;
443:         // wish.quote.tokenVersion = newQuote.tokenVersion;
444:         wish.modifyTime = uint64(block.timestamp);
445:
446:         return wish;
447:     }
```

Description

***: The function `cutWish` is used for the wish owner to create a quote with a lower price. The function `askWish` is used to accpet the quote. if a `cutWish` transaction is executed before a `bidWish` transaction, and a `askWish` transaction is executed after the `bidWish` transaction, the bidder may suffer a loss. Consider following attack vector:

1. Let's say Alice is the wish owner and the bid price is 1 ETH. Bob wants to pay 1 ETH to bid this wish and he sends a transaction to call `bidWish` function.
2. Alice front-run Bob's transaction to call `cutWish` function to set the `wish.quote.value` to 0.01 ETH
3. Bob's transaction is executed successfully. Alice will get back her principal in last bid and the wish owner now is Bob.
4. Alice calls `askWish` to accept the quote created in step 2. Alice only pays 0.01 ETH and get back the wish. Bob lost 0.99 ETH in this attack.

The main problem here is that the quote created by Alice was not cleared after Bob's bid was successful, which led to Bob's loss of funds.

Recommendation

***: Consider clearing the quote after the bid is successful.

Client Response

client response : Fixed. I changed it in this commit: https://gitlab.com/Keccak256-evg/zeek/zeek-contracts/-/commit/fa9433d2098f53d24c0eed9d414947278a7ccc15

ZEK-5:An Account could be targeted and prevented from issuing wish by front-running with a same salt

Category	Severity	Client Response	Contributor
DOS	Medium	Fixed	***

Code Reference

- code/contracts/base/WishBase.sol#L28-L30

```
28: if (_getWishStorage()._wishHistorySalt[data.salt]) {
29:             revert ZeekErrors.WishSaltProcessed();
30:         }
```

- code/contracts/core/Wish.sol#L595

```
595: wishStorage._wishHistorySalt[data.salt] = true;
```

Description

***: In the `issueWish` and `issueWishPlug`, the `_issueValidation` is invoked where `data.salt` is being checked.

```
        if (_getWishStorage()._wishHistorySalt[data.salt]) {
            revert ZeekErrors.WishSaltProcessed();
        }
```

If the salt has been used, the `issueWish` and `issueWishPlug` will revert due to `WishSaltProcessed`.
However, the salt here used is publicly visible and can be used by anyone without any further processing(like `hash it with msg.sender to produce the real salt`).
As a result, an account could be targeted and DoSed from creating wishes by front-running and calling `issueWish` with the same `data.salt` and the `minimum token`. And the `salt` will be considered as used.

```
    function _createWishStorage(
        ZeekDataTypes.WishIssueData calldata data,
        address issuer
    ) internal returns (uint256, ZeekDataTypes.WishStruct storage) {
        WishStorage storage wishStorage = _getWishStorage();

        uint256 profileId = _validateHasProfile(issuer);

        wishStorage._wishHistorySalt[data.salt] = true;
        ...
 }
```

This could lead to the function break-down.
Imagine the following scenario:

1. A celebrity enters `ZEEK`, and he wants to issue a wish.
2. However, his `address` is known by the attacker and his selected salt is visible on-chain.
3. A malicious attacker could front-run `issueWish` using the same `salt`. All he has to pay is the `minimum token` set, and he could later get most of the cost back (by answering it by himself or simply refund).
4. The celebrity's attempt failed due to `WishSaltProcessed`. This could have function failure and bad user experience.

***: In the functions issueWish and issueWishPlug, _issueValidation is called when checking data.salt. If the salt value has been used, then issueWish and issueWishPlug will perform recovery operations due to WishSaltProcessed. However, the salt value used here is publicly visible, and anyone can use it directly without any further processing. Therefore, when running on the front end, one can call issueWish with the same data.salt and the minimum number of tokens to create a wish and locate an account. At this time, the salt value will be regarded as having been used. Such a situation may cause problems with this function.

Recommendation

***: The `salt` should be further processed like `re-hashing together with msg.sender`.
***: Perform further processing on salt.

Client Response

client response : Fixed. commit: https://gitlab.com/Keccak256-evg/zeek/zeek-contracts/-/commit/be6b0a55745c437c4c318043ff3bf8d9c84c662d
check the salt a uint256's first uint160 should be the wish issuer.
client response : Fixed. commit: https://gitlab.com/Keccak256-evg/zeek/zeek-contracts/-/commit/be6b0a55745c437c4c318043ff3bf8d9c84c662d
check the salt a uint256’s first uint160 should be the wish issuer.

ZEK-6: `_baseTransfer()` always reverts for `token` blacklisted recipients

Category	Severity	Client Response	Contributor
DOS	Low	Fixed	***

Code Reference

- code/contracts/base/WishBase.sol#L384

```
384: _baseTransfer(wish.price.tokenVersion, wish.price.token, ownerValue, payable(_msgSender()), payable(_getP
rofileStorage()._profileById[lastOwner].owner));
```

- code/contracts/core/Profile.sol#L65

```
65: _baseTransfer(v.tokenVersion, token, value, address(this), payable(_msgSender()));
```

Description

***: In the `_bid()` function, the protocol calls the `_bidAllocate()` function to offer a bonus to all.

```
  function _bid(
        ZeekDataTypes.WishBidData calldata data
    ) internal returns (ZeekDataTypes.WishStruct storage) {
        ZeekDataTypes.WishStruct storage wish = _getWish(data.wishId);

        uint256 lastPrice = wish.price.value;
        uint256 lastOwner = wish.owner;
        uint256 nextPrice = wish.price.bidValue;
        uint256 bidder = _bidValidation(wish, nextPrice);

        // switch best answer is not allowed
        // storage change
        wish.price.value = nextPrice;
        wish.price.bidValue = _bidPrice(wish.price.token, nextPrice);
        wish.owner = bidder;
        wish.modifyTime = uint64(block.timestamp);

        // offer Bonus to all
        _bidAllocate(data.wishId, wish, lastOwner, lastPrice, nextPrice);

        return wish;
    }
```

In the `_bidAllocate()` function, the protocol calls `_baseTransfer()` to transfer the `wish.price.token` of `ownervalue` to the `owner`.

```
   // transfer to owner directly
        if (ownerValue > 0) {
            _baseTransfer(wish.price.tokenVersion, wish.price.token, ownerValue, payable(_msgSender()), payable(_getProfileStorage()._profileById[lastOwner].owner));
        }
```

The issue here is that if the token is USDT or USDC and the owner is blacklisted, the protocol will be unable to bid.
***: When obtaining the native token reward from the Profile.sol contract, the _baseTransfer() function will be called. This transfers the native token to msg.sender. However, if the token is a native token (such as ETH) and msg.sender is a contract without a receive or fallback function, the call will revert.

Recommendation

***: Allow the administrator to perform privileged actions when the owner address is unable to operate.
***: Consider allowing users to provide a separate recipient address through the claim() function.

Client Response

client response : Fixed. Sounds make sense.
Open address to for claiming
client response : Fixed. Sounds make sense.

ZEK-7:Users may send ETH alongside a token contribution hence locking up funds

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	***

Code Reference

- code/contracts/base/WishBase.sol#L250-L253

```
250: if (bonus.tokenVersion == 20) {
251:            if (msg.value != 0) {
252:                // Ensures users don't accidentally send ETH alongside a token contribution, locking up funds
253:            }
```

- code/contracts/base/ZeekBase.sol#L42-L45

```
42: if (tokenVersion == 20) {
43:            if (msg.value != 0) {
44:                // Ensures users don't accidentally send ETH alongside a token contribution, locking up funds
45:            }
```

Description

***: The `_stakeTokens()` and `_baseCustody()` functions performs a check with the intention to ensure users don't accidentally send ETH alongside a token contribution, locking up funds:

```
        if (bonus.tokenVersion == 20) {
            if (msg.value != 0) {
                // Ensures users don't accidentally send ETH alongside a token contribution, locking up funds
            }
            //...
        }
```

Ideally, when `bonus.tokenVersion == 20`, only `ERC20` tokens should be supplied and hence `msg.value` should be `0`. However, the function does not implement any means to avoid sending `ETH` alongside a token contribution after performing the check.
Therefore, this still leaves a room for `ETH` to be transfered alongside a token contribution thereby locking funds in the contract.

Recommendation

***: Add a `return` statement within the check to halt the token transfer incase some `ETH` value is contained within it.

```
        if (bonus.tokenVersion == 20) {
            if (msg.value != 0) {
                // Ensures users don't accidentally send ETH alongside a token contribution, locking up funds
+               return;
            }
            //...
        }
```

Client Response

client response : Fixed. Fixed in commit:
https://gitlab.com/Keccak256-evg/zeek/zeek-contracts/-/commit/eba38205e2c2070c78212293ab1a76f64b85d5ac
Revert error in this case.

ZEK-8:Use disableInitializers to prevent front-running on the initialize function

Category	Severity	Client Response	Contributor
	Low	Fixed	***

Code Reference

- code/contracts/core/Governance.sol#L24-L37

```
24: function initialize(
25:         string memory name,
26:         string memory symbol
27:     ) external override initializer {
28:         // grant role first
29:         _grantRole(DEFAULT_ADMIN_ROLE, _msgSender());
30:         // save storage
31:         GovernanceStorage storage governanceStorage = _getGovernanceStorage();
32:         governanceStorage._name = name;
33:         governanceStorage._symbol = symbol;
34:         governanceStorage._finance = _msgSender();
35: 
36:         emit ZeekEvents.ZeekInitialized(uint64(block.timestamp));
37:     }
```

Description

***: According to OpenZeppelin's documentation:
An uninitialized contract can be taken over by an attacker. This applies to both a proxy and its implementation contract, which may impact the proxy. To prevent the implementation contract from being used, you should invoke the {_disableInitializers} function in the constructor to automatically lock it when it is deployed.

Recommendation

***: Recommend using disableInitializers to prevent front-running on the initialize function.

Client Response

client response : Fixed. Fixed in this commit: https://gitlab.com/Keccak256-evg/zeek/zeek-contracts/-/commit/1b89cb05831808d6041a0719fdb6a69a446b0545

ZEK-9:Use `disableInitializers` to prevent front-running on the initialize function

Category	Severity	Client Response	Contributor
Privilege Related	Low	Fixed	***

Code Reference

- code/contracts/core/Governance.sol#L19-L37

```
19: contract Governance is IGovernance, ZeekBase, AccessControlUpgradeable {
20:     /*//////////////////////////////////////////////////////////////
21:                         Public functions
22:     //////////////////////////////////////////////////////////////*/
23:     /// @inheritdoc IGovernance
24:     function initialize(
25:         string memory name,
26:         string memory symbol
27:     ) external override initializer {
28:         // grant role first
29:         _grantRole(DEFAULT_ADMIN_ROLE, _msgSender());
30:         // save storage
31:         GovernanceStorage storage governanceStorage = _getGovernanceStorage();
32:         governanceStorage._name = name;
33:         governanceStorage._symbol = symbol;
34:         governanceStorage._finance = _msgSender();
35: 
36:         emit ZeekEvents.ZeekInitialized(uint64(block.timestamp));
37:     }
```

Description

***: The contract `Governance` is an upgradeable contract:

```
contract Governance is IGovernance, ZeekBase, AccessControlUpgradeable
```

The implementation contract behind a proxy can be initialized by any address. This is not a security problem in the sense that it impacts the system directly, as the attacker will not be able to cause any contract to self-destruct or modify any value in the proxy contract. However, taking ownership of implementation contracts can open other attack vectors, like social engineer or phishing attack.
See docs: https://docs.openzeppelin.com/contracts/4.x/api/proxy#Initializable-_disableInitializers--

Recommendation

***: Consider following fix:

```
constructor() {
        _disableInitializers();
}
```

Client Response

client response : Fixed. Fixed in this commit: https://gitlab.com/Keccak256-evg/zeek/zeek-contracts/-/commit/1b89cb05831808d6041a0719fdb6a69a446b0545

ZEK-10:Logical Risk in Profile::padNumber Function

Category	Severity	Client Response	Contributor
Logical	Low	Acknowledged	***

Code Reference

- code/contracts/core/Profile.sol#L131-L140

```
131: function padNumber(uint256 number) internal pure returns (string memory) {
132:         string memory numberString = Strings.toString(number);
133:         uint256 length = bytes(numberString).length;
134:
135:         if (length >= 6) {
136:             return numberString;
137:         } else {
138:             return strConcat(strConcatMultiple("0", 6 - length), numberString);
139:         }
140:     }
```

Description

***: The `padNumber` function in the Profile contract is responsible for generating a string representation of a number, ensuring a minimum length of 6 characters by padding with leading zeros if necessary. This function is used to create `linkCode` values, which are intended to be unique. However, there is a logical risk that padNumber might produce non-unique linkCode values if not properly handled. The padding logic can result in different numbers producing the same string output, leading to potential collisions and compromising the uniqueness of linkCode values.

POC

```
function padNumber(uint256 number) internal pure returns (string memory) {
    string memory numberString = Strings.toString(number);
    uint256 length = bytes(numberString).length;

    if (length >= 6) {
        return numberString;
    } else {
        return strConcat(strConcatMultiple("0", 6 - length), numberString);
    }
}
```

The `padNumber` function generates a linkCode by converting a number to a string and padding it with leading zeros until it reaches a length of 6 characters.
This approach can lead to collisions when different numbers result in the same padded string. For example, `padNumber`(123) produces 000123 and padNumber(12345) produces 012345, but if the function were used with different ranges or overlaps, collisions could occur.
Non-unique `linkCode` values can cause significant issues, such as profile creation failures or incorrect profile associations, especially if linkCode is used as a unique identifier in other parts of the system.

Exploit Scenario

Alice creates a profile with padNumber(123) which results in linkCode 000123. Bob later creates a profile with padNumber(12345) which results in linkCode 012345. While these two examples do not collide directly, the padding logic can lead to other unintended collisions if overlapping ranges or specific edge cases are not handled correctly.

Recommendation

***: #### Ensure Unique linkCode Values:
Implement additional checks to ensure padNumber produces unique linkCode values. This can be done by checking existing linkCode values in the storage and ensuring no duplicates.
Consider using a different method to generate unique identifiers that does not rely solely on padding numbers.
Fix :

```
function _mint(address to, uint256 salt) internal returns (uint256) {
    ProfileStorage storage profileStorage = _getProfileStorage();
    if (profileStorage._profileIdByAddress[to] > 0) {
        revert ZeekErrors.ProfileAlreadyExists();
    }
    uint256 tokenId = ++profileStorage._profileCounter;

    // Ensure uniqueness of linkCode
    string memory linkCode = generateUniqueLinkCode(tokenId);

    _addTokenToAllTokensEnumeration(tokenId);
    profileStorage._profileById[tokenId].owner = to;
    profileStorage._profileById[tokenId].linkCode = linkCode;
    profileStorage._profileById[tokenId].timestamp = uint64(block.timestamp);
    profileStorage._profileIdByAddress[to] = tokenId;

    bytes32 linkCodeHash = keccak256(bytes(linkCode));
    if (profileStorage._profileIdByLinkCodeHash[linkCodeHash] > 0) {
        revert ZeekErrors.LinkCodeAlreadyExists();
    }
    profileStorage._profileIdByLinkCodeHash[linkCodeHash] = tokenId;

    emit ZeekEvents.ProfileCreated(tokenId, salt, to, linkCode, uint64(block.timestamp));

    return tokenId;
}

function generateUniqueLinkCode(uint256 tokenId) internal view returns (string memory) {
    ProfileStorage storage profileStorage = _getProfileStorage();
    string memory linkCode = padNumber(tokenId);
    while (profileStorage._profileIdByLinkCodeHash[keccak256(bytes(linkCode))] > 0) {
        tokenId++;
        linkCode = padNumber(tokenId);
    }
    return linkCode;
}

function padNumber(uint256 number) internal pure returns (string memory) {
    string memory numberString = Strings.toString(number);
    uint256 length = bytes(numberString).length;

    if (length >= 6) {
        return numberString;
    } else {
        return strConcat(strConcatMultiple("0", 6 - length), numberString);
    }
}
```

Client Response

client response : Acknowledged. I'm quite understand your Exploit Scenario. It's cannot be overlap based the linkcode is string.
On the other hand, link code would be overlap when the profile number exceeds 999999. which means 100w user in zeek.
We'll do a contract upgrade when that day comes.

ZEK-11:in `_bestOffer()` function, it returns by default the 0th index offer and not best `offer`

Category	Severity	Client Response	Contributor
Logical	Informational	Declined	***

Code Reference

- code/contracts/base/WishBase.sol#L10

```
10: contract WishBase is ZeekBase {
```

- code/contracts/core/Wish.sol#L18

```
18: contract Wish is IWish, WishBase {
```

Description

*****: the problem lies in `_bestOffer` function which is supposed to return the best offered offer by the issuer now since `offer` is an array of `Offer`, this gives the ability to make multiple offers for multiple talents and then `_bestOffer` returns the best of them
the problem here is

```
File: WishBase.sol
406:     function _bestOffer(ZeekDataTypes.WishStruct storage wish) internal view returns (ZeekDataTypes.Offer memory offer) {
407:         if (wish.offers.length > 0) {
408:             return wish.offers[0];
409:         } else {
410:             return ZeekDataTypes.Offer(0,0,0,0,0);
411:         }
412:     }
```

in line #408 we just return the 0th index of the offers array (the first offer of the issuer)
this is used while unlocking a wish

```
File: Wish.sol
296:     function _unlock(
297:         ZeekDataTypes.WishUnlockData calldata data
298:     ) internal returns (ZeekDataTypes.WishStruct storage, uint256 talent) {
299:         ZeekDataTypes.WishStruct storage wish = _getWish(data.wishId);
300:
301:         uint256 unlocker = _unlockValidation(data, wish);
302:
303:         // storage change
304:         wish.unlocks[unlocker].token = data.token;
305:         wish.unlocks[unlocker].tokenVersion = data.tokenVersion;
306:         wish.unlocks[unlocker].value = data.value;
307:         wish.unlocks[unlocker].timestamp = uint64(block.timestamp);
308:
309:         // token allocation
310:         _unlockAllocate(
311:             data.wishId,
312:             data.token,
313:             data.tokenVersion,
314:             data.value,
315:             _bestOffer(wish).talent <<<@
316:         );
317:
318:         return (wish, unlocker);
319:     }
```

in Line #315 we retreive the `talent` of the best offer by the issuer, so that when we `_unlockAllocate` we transfer the tokens to the talent
now this will always make the first talent offer is the talent receiving the allocate not taking into consideration the `offers` offered after
this will also affect this `_vault` part here in line #380

```
File: code\contracts\base\WishBase.sol
    function _bidAllocate(
        uint wishId,
        ZeekDataTypes.WishStruct storage wish,
        uint256 lastOwner,
        uint256 lastValue,
        uint256 value
    ) internal {
        ZeekDataTypes.BidRatio storage rate = _getWishStorage()._bidRatio;

        uint256 ownerValue = lastValue + lastValue * rate.owner / 100;
        uint256 talentValue = lastValue * rate.talent / 100;

        uint256 committeeValue = value - ownerValue - talentValue;
        // start to allocate
        // only best answer to vault
        _vault(_bestOffer(wish).talent, wishId, wish.price.token, wish.price.tokenVersion, talentValue, ZeekDataTypes.WishScene.Bid, ZeekDataTypes.WishParticipant.Talent);

        // transfer to owner directly
        if (ownerValue > 0) {
            _baseTransfer(wish.price.tokenVersion, wish.price.token, ownerValue, payable(_msgSender()), payable(_getProfileStorage()._profileById[lastOwner].owner));
        }

        if (talentValue > 0) {
            _baseCustody(wish.price.tokenVersion, wish.price.token, talentValue, payable(_msgSender()));
        }
        if (committeeValue > 0) {
            _baseTransfer(wish.price.tokenVersion, wish.price.token, committeeValue, payable(_msgSender()), payable(_getGovernanceStorage()._finance));
        }
    }
```

Recommendation

***: To mititgate this issue**

we need to first make `offers` array to be bounded so that we don't grief `unlocker` or getting the array large enough to have `outOfGas` errors
then we need to implement a loop logic around `_bestOffer` to return the needed result

Client Response

client response : Declined. Actually, currently we only support ONE best offer for ONE wish.
For the future, we planed to support several best offers for ONE wish. So you can treat wish.offers[] as allocation reservation , but it only used index 0 right now.
Hope it helps to understand the design. Thanks~

ZEK-12: `setFinance()` does not verify that `newFinance` is not `priorFinance`

Category	Severity	Client Response	Contributor
Logical	Informational	Fixed	***

Code Reference

- code/contracts/core/Governance.sol#L47-L62

```
47: function setFinance(
48:         address newFinance
49:     ) external override onlyRole(Constants.GOVERANCE_ROLE) {
50:         if (address(0) == newFinance) {
51:             revert ZeekErrors.InvalidAddress();
52:         }
53:         GovernanceStorage storage governanceStorage = _getGovernanceStorage();
54:         address priorFinance = governanceStorage._finance;
55:         governanceStorage._finance = newFinance;
56:         emit ZeekEvents.ZeekFinanceSet(
57:             msg.sender,
58:             priorFinance,
59:             newFinance,
60:             uint64(block.timestamp)
61:         );
62:     }
```

Description

***: The `setFinance()` function is used to set the `newFinance`.

```
    function setFinance(
        address newFinance
    ) external override onlyRole(Constants.GOVERANCE_ROLE) {
        if (address(0) == newFinance) {
            revert ZeekErrors.InvalidAddress();
        }
        GovernanceStorage storage governanceStorage = _getGovernanceStorage();
        address priorFinance = governanceStorage._finance;
        governanceStorage._finance = newFinance;
        //...
    }
```

However, it does not verify that `newFinance` is not `priorFinance`. This means that the same address can be set as the new one which pretty much should not be the case.

Recommendation

***: Just as it is important to ensure that `newFinance` is not `address(0)`, is it just as important to ensure that an old instance of `_finance` is not set as new.

```
    function setFinance(
        address newFinance
    ) external override onlyRole(Constants.GOVERANCE_ROLE) {
        if (address(0) == newFinance) {
            revert ZeekErrors.InvalidAddress();
        }
        GovernanceStorage storage governanceStorage = _getGovernanceStorage();
        address priorFinance = governanceStorage._finance;
+       if (priorFinance == newFinance) {
+           revert ZeekErrors.InvalidParameters();
+       }
        governanceStorage._finance = newFinance;
        //...
    }
```

Client Response

client response : Fixed.
It makes sense for the gas fee.
Commit: https://gitlab.com/Keccak256-evg/zeek/zeek-contracts/-/commit/0cff75e623e5c2b1da5c0d03d3216475ad3af63e

ZEK-13: `constants` should be defined rather than using magic numbers

Category	Severity	Client Response	Contributor
Language Specific	Informational	Fixed	***

Code Reference

- code/contracts/base/ZeekBase.sol#L28
- code/contracts/base/ZeekBase.sol#L31
- code/contracts/base/ZeekBase.sol#L42
- code/contracts/base/ZeekBase.sol#L52
- code/contracts/base/ZeekBase.sol#L67

```
28: if (tokenVersion == 0 && token != address(0)) {
```

```
31: if (tokenVersion == 20 && token == address(0)) {
```

```
42: if (tokenVersion == 20) {
```

```
52: } else if (tokenVersion != 0) {
```

```
67: if (tokenVersion == 0) {
```

Description

***: In `ZeekBase`, the magic number `0` and `20` are used to represent specific tokens. However, for program readability and maintainability, we recommend defining constants rather than using magic numbers.

Recommendation

***: Consider defining constants for number `0` and `20`.

Client Response

client response : Fixed.
Changed it in this commit:
https://gitlab.com/Keccak256-evg/zeek/zeek-contracts/-/commit/6ca382edc0c921fae36849f3d70386c9a8816a55

ZEK-14:The functionality that has been commented out

Category	Severity	Client Response	Contributor
Logical	Informational	Fixed	***

Code Reference

- code/contracts/core/Wish.sol#L86-L97

```
function refundWish(
        ZeekDataTypes.WishRefundData calldata data
    ) external override {
        // _refundWish(data);
    }

    /// @inheritdoc IWish
    function modifyWish(
        ZeekDataTypes.WishModifyData calldata data
    ) external payable override {
        // _modifyWish(data);
    }
```

Description

***: In the current contract, the `refundWish` and `modifyWish` all have their code commented out.

```
    function refundWish(
        ZeekDataTypes.WishRefundData calldata data
    ) external override {
        // _refundWish(data);
    }

    /// @inheritdoc IWish
    function modifyWish(
        ZeekDataTypes.WishModifyData calldata data
    ) external payable override {
        // _modifyWish(data);
    }
```

As a result, this code does not take effect anymore.

Recommendation

***: To mitigate this issue,

- Remove the function + related internal function if the functionality will not be used.

Client Response

client response : Fixed.
Finally, I decided to remove them in this version.
Actually, modify and refund are the future plannings.

commit: https://gitlab.com/Keccak256-evg/zeek/zeek-contracts/-/commit/25be8801bf68a671adfbc11d3aa262ad4df36526

ZEK-15:Strange return value `b` of functions

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	***

Code Reference

- code/contracts/base/WishBase.sol#L72
- code/contracts/base/WishBase.sol#L86
- code/contracts/base/WishBase.sol#L96
- code/contracts/base/WishBase.sol#L111

```
72: ) internal returns (uint256 u) {
```

```
86: ) internal returns (uint256 b) {
```

```
96: ) internal returns (uint256 b) {
```

```
111: ) internal view returns (uint256 b) {
```

Description

*****: The return value is defined in the function header.
For example, in the function `_bidValidation`, `b` will be the returned value.

```
    function _bidValidation(ZeekDataTypes.WishStruct storage wish, uint256 checkValue
    ) internal returns (uint256 b) {
```

However, in the function body, another variable `bidder` is directly returned.

```
        return bidder;
```

This is a strange coding style issue, and will cause issue: if the returned logic doesn't cover all cases, the default value `b` would be returned which will cause further errors.

Recommendation

*****: It is recommended to correct the formatting issue and coding style. For example, change `returns (uint b)` to `returns (uint)`

Client Response

client response : Fixed. commit: https://gitlab.com/Keccak256-evg/zeek/zeek-contracts/-/commit/b4f364814eade2fc2aced6f25522cd4dcc0a8f77

ZEK-16:Potential array out-of-bounds error

Category	Severity	Client Response	Contributor
Logical	Informational	Fixed	***

Code Reference

- code/contracts/core/Governance.sol#L64-L90

```
64: /// @inheritdoc IGovernance
65:     function setOfferRatios(
66:         ZeekDataTypes.WishType[] calldata types,
67:         ZeekDataTypes.OfferRatio[] calldata ratios
68:     ) external override onlyRole(Constants.OPERATION_ROLE) {
69:         _setOfferRatios(types, ratios);
70:
71:         emit ZeekEvents.ZeekWishOfferRatioSet(
72:             ZeekDataTypes.OfferType.Direct,
73:             ratios[0],
74:             ratios[1]
75:         );
76:     }
77:
78:     /// @inheritdoc IGovernance
79:     function setLinkOfferRatios(
80:         ZeekDataTypes.WishType[] calldata types,
81:         ZeekDataTypes.OfferRatio[] calldata ratios
82:     ) external override onlyRole(Constants.OPERATION_ROLE) {
83:         _setLinkOfferRatios(types, ratios);
84:
85:         emit ZeekEvents.ZeekWishOfferRatioSet(
86:             ZeekDataTypes.OfferType.Link,
87:             ratios[0],
88:             ratios[1]
89:         );
90:     }
```

Description

***: The function `setOfferRatios` and `setLinkOfferRatios` will emit `ZeekWishOfferRatioSet` event:

```
emit ZeekEvents.ZeekWishOfferRatioSet(
            ZeekDataTypes.OfferType.Direct,
            ratios[0],
            ratios[1]
        );
```

```
emit ZeekEvents.ZeekWishOfferRatioSet(
            ZeekDataTypes.OfferType.Link,
            ratios[0],
            ratios[1]
        );
```

The issue here is that if the `ratios` array only has one element, the function will revert due to array out-of-bounds error.

Recommendation

*****: Consider adding a check on ratios:

```
function setOfferRatios(
        ZeekDataTypes.WishType[] calldata types,
        ZeekDataTypes.OfferRatio[] calldata ratios
    ) external override onlyRole(Constants.OPERATION_ROLE) {
        require(ratios.length>1,"invalid ratios");
        _setOfferRatios(types, ratios);

        emit ZeekEvents.ZeekWishOfferRatioSet(
            ZeekDataTypes.OfferType.Direct,
            ratios[0],
            ratios[1]
        );
    }
```

```
function setLinkOfferRatios(
        ZeekDataTypes.WishType[] calldata types,
        ZeekDataTypes.OfferRatio[] calldata ratios
    ) external override onlyRole(Constants.OPERATION_ROLE) {
require(ratios.length>1,"invalid ratios");
        _setLinkOfferRatios(types, ratios);

        emit ZeekEvents.ZeekWishOfferRatioSet(
            ZeekDataTypes.OfferType.Link,
            ratios[0],
            ratios[1]
        );
    }
```

Client Response

client response : Fixed. Changed for this issue in commit: https://gitlab.com/Keccak256-evg/zeek/zeek-contracts/-/commit/31ff563e5a63d7d214e1aecb8477d546b6d3d15d
Change the setOfferRatio way:

combined Link and Direct in one function: setOfferRatio
used specified meaning Ratio as questionOfferRatio and referralOfferRatio to define the parameter instead of using arrays.

ZEK-17:No Storage Gap for Upgradeable Contract Might Lead to Storage Slot Collision

Category	Severity	Client Response	Contributor
Logical	Informational	Acknowledged	***

Code Reference

- code/contracts/core/Governance.sol#L19

```
19: contract Governance is IGovernance, ZeekBase, AccessControlUpgradeable {
```

Description

***: For `upgradeable` contracts, there must be storage gap to "allow developers to freely add new state variables in the future without compromising the storage compatibility with existing deployments".
Otherwise it may be very difficult to write new `implementation` code. Without `storage gap`, the variable in child contract might be overwritten by the upgraded base contract if new variables are added to the base contract.

Proof of Concept

`Governance` contract is intended to upgreadable:

```
contract Governance is IGovernance, ZeekBase, AccessControlUpgradeable {
```

However, it does not contain storage `gap`. This could have unintended and very serious consequences to the child contracts, potentially causing loss of user fund or cause the contract to malfunction completely.
Refer to the bottom part of this article: https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable

Recommendation

***: Add storage gap to the contract:

```
contract Governance is IGovernance, ZeekBase, AccessControlUpgradeable {
+    uint256[50] private __gap;
    //...
}
```

Client Response

client response : Acknowledged.
zeek-contract won't use the capacity of upgrade for Governance case.
Leave it as this works for me.

ZEK-18:Missing Zero Address Check

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	***

Code Reference

- code/contracts/core/Governance.sol#L40-L44

```
40: function whitelistApp(address app, bool whitelist)
41:     external override onlyRole(Constants.GOVERANCE_ROLE) {
42:         _getGovernanceStorage()._appWhitelisted[app] = whitelist;
43:         emit ZeekEvents.AppWhitelisted(app, whitelist, block.timestamp);
44:     }
```

Description

***: When setting `newFinance`, the setFinance() checks if the provided address parameter is a `zero` address and reverts with `InvalidAddress();` error if so:

```
    function setFinance(
        address newFinance
    ) external override onlyRole(Constants.GOVERANCE_ROLE) {
>>      if (address(0) == newFinance) {
            revert ZeekErrors.InvalidAddress();
        }
        //...
    }
```

However, this is not performed in some cases such as when whitelisting an `app`:

```
    function whitelistApp(address app, bool whitelist)
    external override onlyRole(Constants.GOVERANCE_ROLE) {
        _getGovernanceStorage()._appWhitelisted[app] = whitelist;
        emit ZeekEvents.AppWhitelisted(app, whitelist, block.timestamp);
    }
```

As seen, the `app` address provided is not sanitized before whitelisting it. This pause no much threat to the protocol but it would be a good practice to enforce `address sanitization` throughout the codebase.

Recommendation

***: Add a zero address check.

```
    function whitelistApp(address app, bool whitelist)
    external override onlyRole(Constants.GOVERANCE_ROLE) {
+       if (address(0) == app) {
+           revert ZeekErrors.InvalidAddress();
+       }
        _getGovernanceStorage()._appWhitelisted[app] = whitelist;
        emit ZeekEvents.AppWhitelisted(app, whitelist, block.timestamp);
    }
```

Client Response

client response : Fixed. Fixed in this commit: https://gitlab.com/Keccak256-evg/zeek/zeek-contracts/-/commit/567bb6441b7664a412b1980c8020872a5d40d84c
This check might be not necessary, it only opened to role governance. But I still fixed for the bottom check.

ZEK-19:Incorrect naming of parameter and struct member can cause incorrect values to be assigned

Category	Severity	Client Response	Contributor
Logical	Informational	Fixed	***

Code Reference

- code/contracts/core/Governance.sol#L93
- code/contracts/core/Governance.sol#L136C2-L155C6

```
93: function setMinimumIssueTokens(
```

```
NaN: function setEarlyUnlockTokens(
NaN:         address token,
NaN:         uint256 tokenVersion,
NaN:         uint256 value,
NaN:         bool valid
NaN:     ) external override onlyRole(Constants.OPERATION_ROLE) {
NaN:         _setEarlyUnlockTokens(token, tokenVersion, value, valid);
NaN:         emit ZeekEvents.ZeekWishUnlockTokenSet(token, tokenVersion, value, valid, true);
NaN:     }
NaN:
NaN:     /// @inheritdoc IGovernance
NaN:     function setUnlockTokens(
NaN:         address token,
NaN:         uint256 tokenVersion,
NaN:         uint256 value,
NaN:         bool valid
NaN:     ) external override onlyRole(Constants.OPERATION_ROLE) {
NaN:         _setUnlockTokens(token, tokenVersion, value, valid);
NaN:         emit ZeekEvents.ZeekWishUnlockTokenSet(token, tokenVersion, value, valid, false);
NaN:     }
```

Description

*****: In the Governance.sol contract, functions _setMinimumIssueTokens(), _setEarlyUnlockTokens() and _setUnlockTokens() take in a `bool valid` parameter. True means valid, False means invalid.
But when the values will be stored in storage as the struct TokenValueSet through the above functions, the fourth member is termed as invalid. This means the values would be interpreted in the opposite way.

```
File: ZeekDataTypes.sol
    struct TokenValueSet {
        address token;
        uint tokenVersion;
        uint256 value;
        bool invalid;
    }
```

This means a value of `bool valid` = true would be interpreter as `bool invalid` = true and the same applies vice versa.

The value of this `bool valid` member is not used anywhere in the codebase as of now. But if used in any integrating contracts or external parties in the future, it could interpret the values incorrectly.

Recommendation

*****: Consider renaming `bool valid` parameter to `bool invalid` or name the fourth struct member of TokenValueSet to valid instead of invalid.

Client Response

client response : Fixed.
Changed it to valid, it should be valid for the correct meaning.

```
struct TokenValueSet {
    address token;
    uint tokenVersion;
    uint256 value;
    bool valid;
}
```

commit: https://gitlab.com/Keccak256-evg/zeek/zeek-contracts/-/commit/5a8be3d2f5ef6dd1d7ee6ec08dac651daba3c214

ZEK-20:Incorrect error thrown in `_validateMsgValue()`

Category	Severity	Client Response	Contributor
Logical	Informational	Fixed	***

Code Reference

- code/contracts/base/WishBase.sol#L167-L179

```
function _validateMsgValue(
        uint256 tokenVersion,
        uint256 tokenValue
    ) internal {
        // check ETH required
        uint256 valueRequired;
        if (tokenVersion == 0) {
            valueRequired = tokenValue;
        }
        if (msg.value != valueRequired) {
            revert ZeekErrors.InsufficientBalance();
        }
    }
```

Description

***:

Take a look at https://github.com/Secure3Audit/code_ZeekNetwork/blob/91522d31c8281348166d19f5c54f0177ece5220a/code/contracts/base/WishBase.sol#L167-L179

```
    function _validateMsgValue(
        uint256 tokenVersion,
        uint256 tokenValue
    ) internal {
        // check ETH required
        uint256 valueRequired;
        if (tokenVersion == 0) {
            valueRequired = tokenValue;
        }
        if (msg.value != valueRequired) {
            revert ZeekErrors.InsufficientBalance();
        }
    }
```

This function is used as the Msg.Value validation, now, evidently the wrong error message is attached, this is because the check is `!=` which would mean that the amount of ETH attached to the call could be `>`/`<`, but the value would only be insufficient if less than.

Recommendation

***:
So either change the error to `EtherDoesNotMatch`, or reimplement `InsufficientBalance` to only be when the attached ether is less than while a new error would be applied when the ether value is gvreater than

Client Response

client response : Fixed.
Fixed in this commit: 229ded1baafef9fe3a727e78c1331e9cae1f6cef
Solution: changed the error from ZeekErrors.InsufficientBalance to ZeekErrors.IncorrectMsgValue.

ZEK-21:Inconsistent indexing of events in ZeekEvents.sol can lead to decreased off-chain monitoring efficiency

Category	Severity	Client Response	Contributor
Language Specific	Informational	Fixed	***

Code Reference

- code/contracts/libraries/ZeekEvents.sol#L63

```
63: event WishApplyAccepted(
```

Description

***: ## Issue & Impact
According to the solidity documentation here, events can index up to 3 members (for non-anonymous events).
These indexed members are used alongside the Keccak hash of the event signature to form the topics of the log entry. This allows applications to efficiently query for values (by setting the hash of the encoded value as the topic).
The current ZeekEvents.sol contract though, inefficiently indexes the members of multiple events declared. Due to this, it decreases the efficiency of querying some of the members present in the events declared.
Here are the inconsistencies:

1. Only 2 members are indexed instead of 3.

```
File: ZeekEvents.sol
64:     event WishApplyAccepted(
65:         uint256 indexed wishId,
66:         uint256 indexed talent,
67:         uint256 linker,
68:         uint256 owner,
69:         uint64 applyTime,
70:         uint256 applyNonce,
71:         uint64 timestamp
72:     );
```

2. Only 2 members are indexed.

```
File: ZeekEvents.sol
    event WishOffered(
        uint256 indexed wishId,
        uint256 indexed talent,
        uint256 linker,
        uint256 owner,
        ZeekDataTypes.OfferRatio values,
        uint64 applyTime,
        uint256 applyNonce,
        uint64 timestamp
    );
```

3. Only 1 member is indexed.

```
File: ZeekEvents.sol
    event WishModified(
        uint256 indexed wishId,
        uint256 balance,
        uint64 deadline,
        uint64 timestamp
    );
```

4. Only 1 member is indexed.

```
File: ZeekEvents.sol
    event WishLinked(
        uint256 indexed wishId,
        uint256 linker,
        uint64 timestamp
    );
```

5. Only 1 member is indexed in the following events.

```
File: ZeekEvents.sol
128:     event WishClosed(uint256 indexed wishId, uint64 timestamp);
129:
130:     /**
131:      * Wish Unlocked Event
132:      */
133:
134:     event WishUnlocked(uint256 indexed wishId, uint256 talent, uint64 timestamp);

File: ZeekEvents.sol
153:     event WishCut(
154:         uint256 indexed wishId,
155:         ZeekDataTypes.TokenValue quote,
156:         uint64 timestamp
157:     );

File: ZeekEvents.sol
200:     event ZeekWishOfferRatioSet (
201:         ZeekDataTypes.OfferType indexed offerType,
202:         ZeekDataTypes.OfferRatio questionOfferRatio,
203:         ZeekDataTypes.OfferRatio referralOfferRatio
204:     );

File: ZeekEvents.sol
257:     event ZeekCutDecimalSet(
258:         address indexed token,
259:         uint256 decimals,
260:         uint64 timestamp
261:     );
```

6. Only 2 members are indexed in the following events.

```
File: ZeekEvents.sol
    event WishTransferred(
        uint256 indexed wishId,
        uint256 indexed owner,
        ZeekDataTypes.WishTransferType transferType,
        uint256 price,
        uint256 bidPrice,
        uint64 timestamp
    );

File: ZeekEvents.sol
    event Claimed (
        uint256 indexed talent,
        address indexed token,
        uint tokenVersion,
        uint256 value,
        uint64 timestamp
    );

File: ZeekEvents.sol
    event AppWhitelisted(address indexed app, bool indexed whitelisted, uint256 timestamp);
```

7. None members are indexed in the following events.

```
File: ZeekEvents.sol
    //@audit none
    event ZeekWishUnlockTokenSet (
        uint issuer,
        uint owner,
        uint talent,
        uint platform,
        bool early
    );

    //@audit none
    event ZeekWishUnlockRatioSet (
        uint issuer,
        uint owner,
        uint talent,
        uint platform,
        bool early
    );

    //@audit none
    event ZeekWishUnlockTokenSet(
        address token,
        uint256 tokenVersion,
        uint256 value,
        bool valid,
        bool early
    );

    //@audit none
    event ZeekWishBidRatioSet(
        uint step,
        uint owner,
        uint talent,
        uint platform
    );

    //@audit none
    event ZeekWishMiniumIssueTokenSet(
        address token,
        uint256 tokenVersion,
        uint256 value,
        bool valid
    );
```

Recommendation

*****: Consider indexing upto the maximum number of members allowed by solidity i.e. 3. This is by adding the indexed keyword to the event members that might be important than the others.

Client Response

client response : Fixed.
Thanks for the suggestion.
I added some indexed modifier in ZeekEvents.
But I still left some event without indexed like

```
event ZeekWishBidRatioSet(
    uint step,
    uint owner,
    uint talent,
    uint platform
);
```

It still needs every indexed field meaningful and indexable.
Changes commit: https://gitlab.com/Keccak256-evg/zeek/zeek-contracts/-/commit/6f49e85664c61ae6a5824b4ae9905aa169b7f372

ZEK-22:Gas Optimizations

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	***

Code Reference

- code/contracts/core/Profile.sol#L77

```
77: function nonces(address singer) external view override returns (uint256) {
```

Description

***: ## Issue & Impact
The _sigNonces() mapping is not used anywhere in the core contracts or even the remaining codebase. Thus, even the nonces() function is redundant since it would return 0 everytime.

```
File: Profile.sol
78:     function nonces(address singer) external view override returns (uint256) {
79:         return _getProfileStorage()._sigNonces[singer];
80:     }
```

Recommendation

***: Consider removing the mapping from the ProfileStorage struct as well as removing the nonces() function.

Client Response

client response : Fixed. Removed useless function `nonces`

Disclaimer

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

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

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

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

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