SALUS SECURITY

FEB 2025

CODE SECURITY ASSESSMENT

D3X EXCHANGE

Overview

Project Summary

- Name: D3x Exchange - D3x contract story
- Platform: Story chain
- Language: Solidity
- Repository:
    - https://github.com/D3X-exchange/d3x-contract-story
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	D3x Exchange - D3x contract story
Version	v2
Type	Solidity
Dates	Feb 10 2025
Logs	Jan 22 2025; Feb 10 2025

Vulnerability Summary

Total High-Severity issues	2
Total Medium-Severity issues	2
Total Low-Severity issues	3
Total informational issues	4
Total	11

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Miscalculation in dynamic spread price impact	High	Numerics	Resolved
2	Centralization risk	Medium	Centralization	Resolved
3	Missing checks on chainlink feed return value	Medium	Data Validation	Resolved
4	Couldn’t receive native token	Medium	Business Logic	Resolved
5	Precision truncation in dynamic spread calculation	Low	Numerics	Acknowledged
6	Strict check	Low	Business Logic	Resolved
7	Lack of data validation	Low	Data Validation	Acknowledged
8	Redundant code	Informational	Redundancy	Resolved
9	Incorrect variable name	Informational	Code Quality	Resolved
10	Hardcoded faucet value for all currencies	Informational	Business Logic	Resolved
11	Unused oracle check in _fetchFeedPrice function	Informational	Business Logic	Resolved

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Miscalculation in dynamic spread price impact

Severity: High	Category: Numerics

Target:
- contracts/d3xManager/D3xManagerLogicCommon.sol

Description

In the D3xManagerLogicCommon contract, the `_calcDynamicSpreadPriceImpact` function is intended to calculate the dynamic spread percentage based on existing open interest and the new trade position size. According to the inline comments, the final opening price should be computed as follows:

contracts/d3xManager/D3xManagerLogicCommon.sol:L58-L102

```
//Dynamic Spread (%) = (Open interest {long/short} + New trade position size / 2) / 1%
depth {above/below}.
function _calcDynamicSpreadPriceImpact(
    ...
)
internal
view
returns (
    ...
)
{
    ...
        //Final opening price = _openPrice +/- _openPrice * Dynamic Spread
        //Final opening price = 3003.19 + (3003.19 * 0.0126 / 100) = 3003.57
        uint64 priceImpact = _toUint64(uint256(priceImpactRateInExtraPoint) * openPrice /
D3xManagerType.EXTEND_POINT);
        priceAfterImpact = long ? openPrice + priceImpact : openPrice - priceImpact;
   }
}
```

However, the actual implementation is missing the division by 100, which leads to the dynamic spread being 100 times larger than intended, for example:

Final opening price = 3003.19 + (3003.19 * 0.0126) = 3041.0319

Recommendation

Modify the priceImpact calculation to include an additional division by 100.

Status

This issue has been resolved by the team with commit 4b4afbe.

2. Centralization risk

Severity: Medium	Category: Centralization

Target:
- contracts/d3xVault/D3xVaultLogic.sol
- contracts/d3xManager/D3xManagerLogic6.sol

Description

The system exhibits centralization risk due to highly privileged functions in key contracts.

In the D3xManagerLogic6 contract, the `transferBack` function allows the owner to withdraw any token with any amount to a specified address.

contracts/d3xManager/D3xManagerLogic6.sol:L246-L248

```
function transferBack(address tokenAddress, address to, uint256 amount) external
onlyOwner {
    FundLibrary._fundFromSelfToSBOrSafetyBox(tokenAddress, to, amount);
}
```

If the private keys for the manager or owner accounts are compromised, an attacker could exploit these functions to drain all assets from the vault or withdraw arbitrary tokens without restriction. This level of control by a single party poses a significant risk to the security and trustworthiness of the system, affecting all users.

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

This issue has been resolved by the team with commit 4b4afbe.

3. Missing checks on chainlink feed return value

Severity: Medium	Category: Data Validation

Target:
- contracts/d3xManager/D3xManagerLogic1.sol

Description

In the `_fetchFeedPrice` function, the implementation retrieves the price using the `IChainlinkFeed(pair.feed1).latestRoundData()` function. However, it does not perform validation to check whether the returned price data is stale.

contracts/d3xManager/D3xManagerLogic1.sol:L508-L577

```
function _fetchFeedPrice(D3xManagerType.Pair memory pair) private view returns (uint64)
{
    if (pair.feed1 == address(0)) {
        return 0;
    }

    uint256 feedPrice = 0;
    int256 feedPriceInt256;
    if (pair.accessType == D3xManagerType.ORACLE_ACCESS_TYPE_CHAINLINK) {
        (, feedPriceInt256,,,) = IChainlinkFeed(pair.feed1).latestRoundData();

        uint256 feedPriceUint256 = _int256ToUint256(feedPriceInt256);
        ...
    }
  ...
}
```

Recommendation

Add checks to ensure the returned price data is fresh, for example, validating the timestamp against a permissible threshold.

Status

This issue has been resolved by the team with commit 4b4afbe.

4. Couldn’t receive native token

Severity: Medium	Category: Business Logic

Target:
- contracts/d3xVault/D3xVaultLogic.sol
- contracts/d3xManager/D3xManagerLogic2.sol

Description

In the D3xVaultLogic and D3xManagerLogic2 contracts, the following functions are intended to handle asset transfers, but they do not include the `payable` modifier, which is required for functions that accept native tokens:

contracts/d3xVault/D3xVaultLogic.sol:L41-L65

```
function receiveAsset(uint256 assetAmount, address who) external {

    address managerAddress = manager();
    address sender = _msgSender();
    require(sender == managerAddress, "only manager");

    FundLibrary._fundFromSBToSelf(_asset, managerAddress, assetAmount);

    _cumulativeGivenAsset += assetAmount;

    emit ReceiveAsset(assetAmount, who);
}

function receiveProfit(uint256 assetAmount, address who) external {

    address managerAddress = manager();
    address sender = _msgSender();
    require(sender == managerAddress, "only manager");

    FundLibrary._fundFromSBToSelf(_asset, managerAddress, assetAmount);

    _cumulativeProfit += assetAmount;

    emit ReceiveProfit(assetAmount, who);
}
```

contracts/d3xManager/D3xManagerLogic2.sol:L29-L58,L92-L110

```
function openTradeMarket(
    D3xManagerType.OpenTradeRequest memory request//do a copy from calldata to memory
)
external
preCheck(true/*isNewTrade*/, true/*isWriteFunction*/)
onlyEOA {

    address who = _msgSender();

    address currency = _openTradeCheck(request, false, who);

    //take fund
    {
        FundLibrary._fundFromSBToSBOrSafetyBox(
            currency,
```

```
            who,
            openTradeAnyMix(),
            request.desiredPositionInCurrency
        );
    }
    ...
}
```

```
function openTradeLimit(D3xManagerType.OpenTradeRequest memory request)
    external
    preCheck(true, /*isNewTrade*/ true /*isWriteFunction*/ )
    onlyEOA
{
    address who = _msgSender();
    address currency = _openTradeCheck(request, true, who);

    //take fund
    {
        FundLibrary._fundFromSBToSBOrSafetyBox(currency, who, openTradeAnyMix(),
request.desiredPositionInCurrency);
    }

    (
        D3xManagerType.Trade storage trade,
        /*uint64 tradeNumber*/
    ) = _addNewTrade(request, D3xManagerType.TRADE_STATE_LIMIT_PENDING, who);
    trade.limitPendingTimestamp = _blockTimestamp();
}
```

These functions invoke the `_fundFromSBToSBOrSafetyBox` and `_fundFromSBToSelf` functions, which can accept native tokens.

However, because the functions do not have the `payable` modifier, they cannot accept native tokens. This can lead to the failure of operations that require native token transfers, malfunctioning contracts, or the inability to execute important operations.

Recommendation

Add the `payable` modifier to the functions that need to accept native tokens.

Status

This issue has been resolved by the team with commit 4b4afbe.

5. Precision truncation in dynamic spread calculation

Severity: Low	Category: Numerics

Target:
- contracts/d3xManager/D3xManagerLogicCommon.sol

Description

In the D3xManagerLogicCommon contract, the `_calcDynamicSpreadPriceImpact` function calculates the dynamic spread based on existing open interest and new trade position size relative to a defined depth.

contracts/d3xManager/D3xManagerLogicCommon.sol:L52-L96

```
//Dynamic Spread (%) = (Open interest {long/short} + New trade position size / 2) / 1%
depth {above/below}.
function _calcDynamicSpreadPriceImpact(
    ...
)
    internal
    view
    returns (
        ...
    )
{
    ...
    //Dynamic Spread (%) = (Open interest {long/short} + New trade position size / 2) /
1% depth {above/below}.
    //0.0126% dynamic spread = (100,000 + 2,480 / 2) / 8,000,000 = 0.012655
    // priceImpactPercentIn10 = ((existOpenInterest + newOpenInterest / 2) *
D3xManagerType.PRECISION) / onePercentDepth / 1e18;
    priceImpactRateInExtraPoint = _toUint24(
        (uint256(existOpenInterest) + newOpenInterest / 2)
            * D3xManagerType.EXTEND_POINT
            / onePercentDepth
    );
    ...
}
```

If we change the value in the comment: existOpenInterest = 10,000 and onePercentDepth = 8,100,000, --> dynamic spread = 0.00138765, --> priceImpactRateInExtraPoint = 1387.

A precision truncation occurs.

Recommendation

Increase precision or implement a rounding mechanism.

Status

This issue has been acknowledged by the team.

6. Strict check

Severity: Low	Category: Business Logic
Target: - contracts/d3xManager/D3xManagerLogic6.sol	

Description

In the `updateTradeTpSlLive` function of the D3xManagerLogic6 contract, there is a validation check for updating the stop-loss (SL) for short trades that uses a strict less-than comparison.

contracts/d3xManager/D3xManagerLogic6.sol:L93-L133

```
function updateTradeTpSlLive(uint64 tradeNumber, uint64 newTp, bool newIsSlSet, uint64
newSl) external {
    address who = _msgSender();
    ...
    if (newIsSlSet != trade.isSlSet || newSl != trade.sl) {
        trade.isSlSet = newIsSlSet;
        if (newIsSlSet) {
            uint256 extremeSl =
                _calcTpSlPrice(trade.openPrice, trade.long, trade.leverage,
D3xManagerType.MAX_SL_P, false);
            if (trade.long) {
                require(extremeSl <= newSl, "612.sl is out of max tolerance");
            } else {
                require(newSl < extremeSl, "613.sl is out of max tolerance");
            }
            trade.sl = newSl;
        } else {
            trade.sl = 0;
        }

        trade.slTimestamp = _blockTimestamp();
    }
}
```

However, based on the context and symmetry with the long trade check, this condition likely should allow the stop-loss to be exactly equal to the extreme stop-loss value. In other words, it should use a less-than-or-equal-to (<=) comparison.

Recommendation

Update the condition in the short trade validation to use less-than-or-equal-to (<=).

Status

This issue has been resolved by the team with commit 4b4afbe.

7. Lack of data validation

Severity: Low	Category: Data Validation

Target:
- contracts/d3xManager/D3xManagerLogic7.sol
- contracts/dependant/proxy/Proxy.sol
- contracts/dependant/proxy/NameServiceProxy.sol

Description

In the project, there are many input values lack corresponding validation, for example:

In the D3xManagerLogic7 contract, the `setGlobalConfig` function allows the contract owner to update the global configuration parameters without any validation of the input data.

contracts/d3xManager/D3xManagerLogic7.sol:L175-L177

```
function setGlobalConfig(D3xManagerType.GlobalConfig calldata input) external onlyOwner
{
    _globalConfig[0] = input;
}
```

Although restricted by the `onlyOwner` modifier, the lack of validation exposes the system to risks of misconfiguration.

It is considered a security best practice to verify addresses against the zero address during initialization or setting. However, this precautionary step is absent for address variables.If set the admin address as `address(0)` by mistake, it's irreversible.

contracts/dependant/proxy/Proxy.sol:L98-L100
contracts/dependant/proxy/NameServiceProxy.sol:L49-L51

```
function sysSetAdmin(address _input) external onlyAdmin {
    adminSlot.sysSaveSlotDataAddress(_input);
}
```

Recommendation

Adding appropriate validations, for example:

```
function setGlobalConfig(D3xManagerType.GlobalConfig calldata input) external onlyOwner
{
    require(input.oiStartTimestamp < block.timestamp, "oiStartTimestamp must be in the
past");
    require(input.oiWindowsDuration != 0, "oiWindowsDuration must be non-zero");
    ...

    _globalConfig[0] = input;
}

function sysSetAdmin(address _input) external onlyAdmin {
    require(_input != 0, "address must be non-zero");
```

```
    adminSlot.sysSaveSlotDataAddress(_input);
}
```

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

8. Redundant code

Severity: Informational	Category: Redundancy

Target:
- contracts/d3xManager/D3xManagerLogic1.sol
- contracts/d3xManager/D3xManagerLogic2.sol
- contracts/d3xManager/D3xManagerLogic3.sol
- contracts/d3xManager/D3xManagerLogic4.sol
- contracts/d3xManager/D3xManagerLogicCommon.sol

Description

Contracts import `hardhat/console.sol` file that is used solely to aid testing during development. This file is not used on the mainnet and should not be imported.

contracts/d3xManager/D3xManagerLogic1.sol:L17
contracts/d3xManager/D3xManagerLogic2.sol:L12
contracts/d3xManager/D3xManagerLogic3.sol:L12
contracts/d3xManager/D3xManagerLogic4.sol:L12
contracts/d3xManager/D3xManagerLogicCommon.sol:12

```
import "hardhat/console.sol";
```

There is no need to assign variables values that are already set in them above.

contracts/d3xManager/D3xManagerLogic4.sol:L172-253

```
function _finishCloseTradeLimit(
            )
    internal {
        bool isHit = false;
        uint64 closePrice = 0;
        uint64 liqPrice = 0;
        ...
        if (tradeSate == D3xManagerType.TRADE_STATE_LIMIT_LIQ_CLOSING) {
            ...
         } else if (tradeSate == D3xManagerType.TRADE_STATE_LIMIT_TP_CLOSING) {
            ...
         } else if (tradeSate == D3xManagerType.TRADE_STATE_LIMIT_SL_CLOSING &&
trade.isSlSet) {
            ...
        } else {
            isHit = false;
            closePrice = 0;
        }
```

Recommendation

Consider removing the redundant code.

Status

This issue has been resolved by the team with commit 4b4afbe.

9. Incorrect variable name

Severity: Informational	Category: Code Quality
Target: - contracts/d3xManager/D3xManagerType.sol	

Description

The Pair struct contains several variables that are currently not being used and are reserved for future use. These variables are named with the prefix reserved followed by the number of bytes they occupy. For example, `reserved16` occupies 16 bytes. The issue is that one of these variables is incorrectly named - `reserved32`, even though it actually occupies 16 bytes.

```
struct Pair {
    ...
    uint16 reserved32;
    address feed1; //online oracle address,  0x0 for disable
    ...
    uint72 reserved72;
}
```

Recommendation

Rename variable `reserved32` to `reserved16`.

Status

This issue has been resolved by the team with commit 4b4afbe.

10. Hardcoded faucet value for all currencies

Severity: Informational	Category: Business Logic
Target: - contracts/d3xManager/D3xManagerLogic6.sol	

Description

The faucet function dispenses different tokens in the same quantity. These tokens have varying values despite having equal quantities. For example, the value of 500 wBTC differs significantly from 500 USDT.

contracts/d3xManager/D3xManagerLogic6.sol:L196-L212

```
function faucet(address currency) external {
    ...
    FundLibrary._fundFromSafetyBoxToSBOrSafetyBox(
              currency,
              faucetAnyDispatch(),
              msg.sender,
              ConstantLibrary.UNIT * 500
    );
}
```

Recommendation

Create a mapping to store the amount of tokens dispensed by this function, depending on the token type.

Status

This issue has been resolved by the team with commit 4b4afbe.

11. Unused oracle check in _fetchFeedPrice function

Severity: Informational	Category: Business Logic

Target:
- contracts/d3xManager/D3xManagerLogic1.sol

Description

In the D3xManagerLogic1 contract, the `_fetchFeedPrice` function contains a specific check for an oracle feed related to the X1 chain:

contracts/d3xManager/D3xManagerLogic1.sol:L494-L548

```
function _fetchFeedPrice(D3xManagerType.Pair memory pair) private view returns (uint64)
{
   if (pair.feed1 == address(0)) {
       return 0;
   }
   ...
   } else if (pair.accessType == D3xManagerType.ORACLE_ACCESS_TYPE_X1) {
       (, feedPriceInt256,,,) =
IExOraclePriceData(address(0x64481ebfFe69d688d754e09918e82C89D8Da2507))
           .latestRoundData(string(abi.encodePacked(pair.from)),
address(0x6CF2a39d1c85aDFB50DA183060DC0d46529F3f9C));

       uint256 feedPriceUint256 = _int256ToUint256(feedPriceInt256);

       if (pair.feedCalculation == D3xManagerType.FEED_CALCULATION_NORMAL) {
           //feedPrice = feedPriceUint256 * D3xManagerType.PRECISION / 1e6;
           feedPrice =
               feedPriceUint256 * (10 ** pair.feedPriceMultiplyDecimal) / (10 **
pair.feedPriceDivideDecimal);
       } else if (pair.feedCalculation == D3xManagerType.FEED_CALCULATION_INVERSE) {
           //feedPrice = D3xManagerType.PRECISION * 1e6 / feedPriceUint256;
           revert("011.unknown feed calculation");
       } else if (pair.feedCalculation == D3xManagerType.FEED_CALCULATION_COMPOSITION) {
           revert("010.unsupported oracle");
       } else {
           revert("011.unknown feed calculation");
       }
   ...
}
```

According to the project team, this segment of the code is specifically related to an oracle for the X1 chain. However, they have confirmed that this oracle will not be configured for the X1 chain. Instead, it is typically set to an empty address, rendering the check and associated logic irrelevant for the current implementation.

Recommendation

Since the code is not used and is unlikely to be utilized in the future (as per the project team), it is recommended to remove or comment out.

Status

This issue has been resolved by the team with commit 4b4afbe.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 28a6604:

File	SHA-1 hash
contracts/d3xManager/D3xManagerInterface.sol	909a5bd2027908a97200d44bc0fc18d7026645f5
contracts/d3xManager/D3xManagerInterface1.sol	3cfb75e45974390efa7738356cf866d7376816b1
contracts/d3xManager/D3xManagerInterface2.sol	707143c270ed54a7b139bb19cde6f29672c2f5a2
contracts/d3xManager/D3xManagerInterface3.sol	e445b52e48b6509a9bd3249682c691b16cf555a8
contracts/d3xManager/D3xManagerInterface4.sol	5cf7c5630c2269372964d5c2bb248b0d3c83bda1
contracts/d3xManager/D3xManagerInterface5.sol	1c9161eba5bd645f7a08e141de7cc4bb1e200b57
contracts/d3xManager/D3xManagerInterface6.sol	b15eb47708153d9a38e54adfa64b43edef11e8c8
contracts/d3xManager/D3xManagerInterface7.sol	7bf0ce95a956bf0614deda4072345896e0c0cd69
contracts/d3xManager/D3xManagerLogicCommon.sol	334ce668c99479734608b00d5888ee8ff0951996
contracts/d3xManager/D3xManagerLayout.sol	247a3df2da96b64c8d85dba7944fcb20d65f1ad0
contracts/d3xManager/D3xManagerLayoutForStruct.sol	5e393aa40040c9e13bb9e8f61ce566417c98dbc2
contracts/d3xManager/D3xManagerType.sol	448ba4e216e00fcb510ea75244fd8a244f10d145
contracts/d3xManager/D3xManagerEvent.sol	7d0a49c1b4c9e451402f7fee2e6b1bf73d2331c5
contracts/d3xManager/D3xManagerStorage.sol	ccf7e5477c95a3b41f4cfeb7303b43a286cbbd27
contracts/d3xManager/D3xManagerLogic1.sol	b719d2acae1e4eb424020ca7c9c3886737ca0526
contracts/d3xManager/D3xManagerLogic2.sol	555ce0b783a09a3706630bb09e90633ab57e2f8f
contracts/d3xManager/D3xManagerLogic3.sol	4471cee27804efa1c903848933ea344d42ebe57a
contracts/d3xManager/D3xManagerLogic4.sol	a8cffa6b5b84c7d83125f16fbd68f3e0374effa7
contracts/d3xManager/D3xManagerLogic5.sol	d625f029973b09b0d96becc395d5781cf14fd692
contracts/d3xManager/D3xManagerLogic6.sol	26c35993edd45ec8e71379ba2825452b066a07e8
contracts/d3xManager/D3xManagerLogic7.sol	ae3c7ac34373dcdbd269ec0d66112a48489bc505
contracts/d3xVault/D3xVaultStorage.sol	5a639f837690defa3ecf06aca15e09e8b411c8ac

contracts/d3xVault/D3xVaultLogic.sol	3afffd069e7f984e8665137e091b758cb7ea6451
contracts/d3xVault/D3xVaultInterface.sol	cb20462168009834230430ea7e56e25d7b1fbdf2
contracts/d3xVault/D3xVaultLayout.sol	7d88c6042dce037385c52142e414b7fc18110352
contracts/d3xVault/D3xVaultEvent.sol	41f9e758ebd7461ba6d9302cf2ad30b1394e0dbe
contracts/d3xVault/D3xVaultType.sol	c7d72cb2aeadf0ad0c6ec03b7e3c75be4d1ca043
contracts/dependant/proxy/EnhancedMap.sol	6d17fedfaeadbca0c1ccac564fcfce9d824fbbf7
contracts/dependant/proxy/SlotData.sol	a216bdecc4d32e65ef30e91e2ef7f2227c0cee09
contracts/dependant/proxy/EnhancedUniqueIndexMap.sol	87676d276ecd7bc4ada258d4302bc31e6801d285
contracts/dependant/proxy/Proxy.sol	720f5e0f29c53234640f5c2885e78195bb77f747
contracts/dependant/proxy/Base.sol	23024962ba7b53475e5242fecfec94e1239f4826
contracts/dependant/proxy/Delegate.sol	e569e2ef2a22de17ae14f4623a903f0d63b7f30f
contracts/dependant/proxy/NameServiceProxy.sol	3fd345183cd7071301d45242306f42a51ac8173d