X28 Security Audit

Report Version 0.1

May 7, 2024

Conducted by **Hunter Security**:

George Hunter, Lead Security Researcher
Windhustler, Senior Security Researcher
Stormy, Associate Security Researcher

Table of Contents

1 About Hunter Security

Hunter Security consists of multiple teams of leading smart contract security researchers. Having conducted over 100 security reviews and reported tens of live smart contract security vulnerabilities protecting over $1B of TVL, our team always strives to deliver top-quality security services to DeFi protocols. For security audit inquiries, you can reach out to us on Telegram or Twitter at *@georgehntr*.

2 Disclaimer

Audits are a time-, resource-, and expertise-bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can reveal the presence of vulnerabilities **but cannot guarantee their absence**.

3 Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - involves a small loss of funds or affects a core functionality of the protocol.
- **Low** - encompasses any unexpected behavior that is non-critical.

3.2 Likelihood

- **High** - a direct attack vector; the cost is relatively low compared to the potential loss of funds.
- **Medium** - only a conditionally incentivized attack vector, with a moderate likelihood.
- **Low** - involves too many or unlikely assumptions; offers little to no incentive.

3.3 Actions required by severity level

- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

4 Executive summary

The Hunter Security team was engaged by X28 to review the X28 contracts during the period from April 26, 2024, to May 1, 2024.

Overview

Project Name	X28
Repository	https://github.com/jakesharpe777/ttx_x28_buy_and_burn
Commit hash	8bcd85f62dabcf377899e1da36ad298729fd657b
Resolution	-
Repository	https://github.com/jakesharpe777/ttx_x28_source_chain
Commit hash	87cd330427f1ca217f08837030476e460468f9f5
Resolution	-
Methods	Manual review & testing

Timeline

-	April 26, 2024	Audit kick-off
v0.1	May 7, 2024	Preliminary report
v1.0	-	Mitigation review

Scope

contracts/AuctionInfo.sol
contracts/BurnInfo.sol
contracts/X28.sol
contracts/BuyAndBurnX28.sol (diffs)

Issues Found

High risk	0
Medium risk	0
Low risk	3

5 Findings

5.1 Low

5.1.1 Composing functionality is not implemented

Severity: Low

Context: X28.sol#L29

Description: *OFT* contract has the option of sending composed messages and having the composer execute them on the receiving chain in separate transactions.

```
function send(
    SendParam calldata _sendParam,
    bytes calldata _extraOptions,
    MessagingFee calldata _fee,
    address _refundAddress,
    bytes calldata _composeMsg,
    bytes calldata /*_oftCmd*/
) external payable virtual returns (MessagingReceipt memory msgReceipt, OFTReceipt
    memory oftReceipt) {
    ...

    (bytes memory message, bytes memory options) = _buildMsgAndOptions(
        _sendParam,
        _extraOptions,
>>>     _composeMsg,
        amountToCreditLD
    );

function _lzReceive(
    Origin calldata _origin,
    bytes32 _guid,
    bytes calldata _message,
    address /*_executor*/,
    bytes calldata /*_extraData*/
) internal virtual override {
      ...

>>>   endpoint.sendCompose(toAddress, _guid, 0, composeMsg);
```

As *OFT* is an abstract contract the contract that extends it should implement the *lzCompose* interface to enable this functionality. Otherwise, the composed messages are non-executable.

See MessagingComposer.sol and EndpointV2.sol logic.

```
function lzCompose(
    address _from,
    address _to,
    bytes32 _guid,
    uint16 _index,
    bytes calldata _message,
    bytes calldata _extraData
) external payable {
    bytes32 expectedHash = composeQueue[_from][_to][_guid][_index];
    bytes32 actualHash = keccak256(_message);
```

```
    if (expectedHash != actualHash) revert Errors.LZ_ComposeNotFound(expectedHash,
        actualHash);

    composeQueue[_from][_to][_guid][_index] = RECEIVED_MESSAGE_HASH;
    ILayerZeroComposer(_to).lzCompose{ value: msg.value }(_from, _guid, _message,
        msg.sender, _extraData);
    emit ComposeDelivered(_from, _to, _guid, _index);
}
```

There is no immediate security risk here, but with the current implementation of *X28* any composed message is non-executable and just a waste of gas and bytes transferred across chains.

Also see the documentation on OApp Composing.

Recommendation: Consider overriding the functions from *OFTCore* to disable sending and receiving composed messages if this functionality is not needed.

Resolution: Pending.

5.1.2 TitanX liquid & stake burning functionalities can't be temporarily disabled once enabled

Severity: Low

Context: X28.sol#L105-L113

Description: The functions *enableLiquidBurning* & *enableStakedBurning* are called by the owner in order to enable the functionality of burning liquid TitanX and burning TitanX stake, once enabled there is no way back which might be problematic in a case the owner wants to temporarily disable some of them for a particular reason:

```
/** @notice enable burning TitanX liquid. Only callable by owner address. */
function enableLiquidBurning() external onlyOwner {
    s_burnLiquid = true;
}

/** @notice enable burning TitanX stake. Only callable by owner address. */
function enableStakedBurning() external onlyOwner {
    s_burnStake = true;
}
```

Recommendation: Consider whether it is worth implementing a functionality pausing the liquid & stake burning.

Resolution: Pending.

5.1.3 OFT decimalConversionRate leads to dust amounts being non-transferable across chains

Severity: Low

Context: X28.sol#L29

Description: LayerZero was built with EVM and non-EVM chains in mind and this influenced the design of the OFT standard.

Each OFT has the concept of shared and local decimals. The default value (if not overridden) is *localDecimals = 18* and *sharedDecimals = 6*. These are used to compute the *decimalsConversionRate = 10**(18-6) = 10**12*: https://github.com/LayerZero-Labs/LayerZero-v2/blob/982c549236622c6bb9eaa6c65afcf1e0e559b624/oapp/contracts/oft/OFTCore.sol#L55

In practice, this means that you can only transfer amounts that are multiples of *1e12* across chains.

An example is trying to transfer *10e12 + 500*. You would only be able to transfer *1e12* and the 500 would be non-transferrable.

This architecture was set due to some non-EVM chains only supporting *uint64* as the maximum value for tokens.

As you can only burn Titanx in liquid/staking/minting mode on Ethereum the transferred amount to another chain would be a multiple of *10e12* so it's transferrable back.

However, there is the ability of burning *X28* tokens on any chain through several functions, e.g.:

```
function userBurnTokens(uint256 amount) public nonReentrant dailyUpdate {
    if (amount == 0) revert X28_InvalidAmount();
    _burn(msg.sender, amount);
    _updateBurnAmount(msg.sender, address(0), amount);
}
```

In this case the remaining amount can become non-transferable.

Recommendation: If there are no future plans to deploy on non-EVM chains, simply overriding the sharedDecimals inside the *X28* and setting its value to 18 will make the decimals conversion rate equal to 1. This makes any amount transferable.

```
function sharedDecimals() public view override returns (uint8) {
    return 18;
}
```

Resolution: Pending.