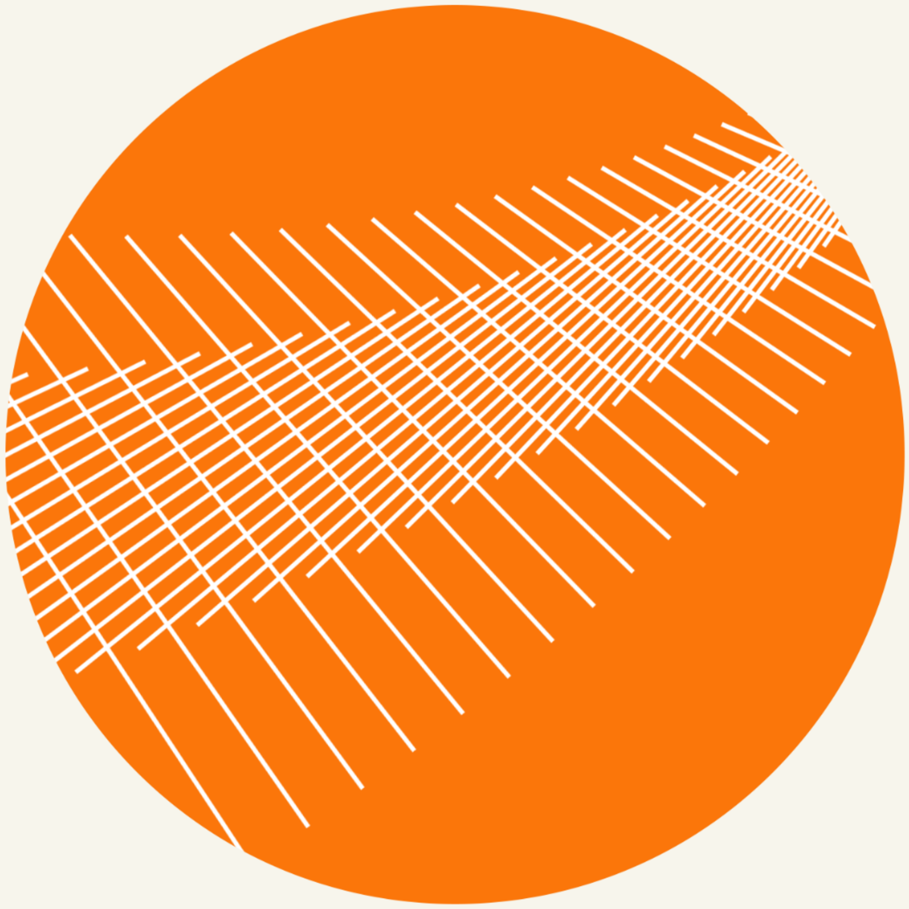
CTDSEC

YOU ARE PROTECTED



Smart contract security audit

Zbyte Token

Table of Contents

1.0 Introduction

1.1 Project engagement

During November of 2023, Zbyte team engaged CTDSec to audit smart contracts that they created. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts. Zbyte provided CTDSec with access to their code repository and whitepaper.

1.2 Disclaimer

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that’s able to withstand the network’s fast-paced and rapidly changing environment, we at CTDSec recommend that Zbyte team put in place a bug bounty program to encourage further and active analysis of the smart contract.

2.0 Coverage

2.1 Target Code and Revision

For this audit, we performed research, investigation, and review of the Zbyte contracts followed by issue reporting, along with mitigation and remediation instructions outlined in this report. The following code files are considered in-scope for the review:

Source file:

zbyte-zbyte.zip: SHA256 -

334d5b40b1383c0a27b53145dd5ac05e4e862736306e07ea6a15d2a0feafb50c

Last version [fix]: https://github.com/Zbyteio/zbyte/tree/7da5f108a3fb7fb6cdc33ffd00d9f7cd2e9dc467

2.2 Attacks made to the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices.

№	Issue description.	Checking status
1	Compiler warnings.	PASSED
2	Race conditions and Reentrancy. Cross-function race conditions.	FIXED
3	Possible delays in data delivery.	PASSED
4	Oracle calls.	PASSED
5	Front running.	PASSED
6	Timestamp dependence.	PASSED
7	Integer Overflow and Underflow.	PASSED
8	DoS with Revert.	PASSED
9	DoS with block gas limit.	PASSED
10	Methods execution permissions.	PASSED
11	Economy model. If application logic is based on an incorrect economic model, the application would not function correctly and participants would incur financial losses. This type of issue is most often found in bonus rewards systems, Staking and Farming contracts, Vault and Vesting contracts, etc.	PASSED
12	The impact of the exchange rate on the logic.	PASSED
13	Private user data leaks.	PASSED
14	Malicious Event log.	PASSED
15	Scoping and Declarations.	PASSED
16	Uninitialized storage pointers.	PASSED

17	Arithmetic accuracy.	PASSED
18	Design Logic.	PASSED
19	Cross-function race conditions.	PASSED
20	Safe Zeppelin module.	PASSED
21	Fallback function security.	PASSED
22	Overpowered functions / Owner privileges	PASSED

3.0 Security Issues

3.1 High severity issues [1 - Fixed]

1. Reentrancy in deposits

Contract: ZbyteEscrow

Functions: deposit()

Issue: The function deposit() is exploitable by re-entrancy attacks when a user executes a deposit.

Fix: Add nonReentrant guard modifier.

Update: Fix added
(https://github.com/Zbyteio/zbyte/tree/7da5f108a3fb7fb6cdc33ffd00d9f7cd2e9dc467)

3.2 Medium severity issues [2 - Inapplicable]

1. Authorization issues

Contract: ZbyteVToken

Functions: transfer(), transferFrom(), mint(), burn(), destroy()

Issue: If setPublicCapability is set on those functions anyone could call them.

Fix: Change requiresAuth modifier or use an alternative way to check if the call is authorized.

Update: Functions are restricted at the moment and in the future they will be managed by DAO.

2. Data slicing and typecasting

Contract: ZbyteForwarderCore

Functions: approveAndDeposit

Issue: The function approveAndDeposit uses data slicing and direct typecasting to extract and validate function signatures and addresses from reqApprove_.data and reqDeposit_.data. This approach is error-prone and relies heavily on the correct formatting of the input data.

Fix: Refactor the way the data is extracted and validated. Utilize more robust methods for decoding and verifying data to reduce the risk of processing incorrect inputs.

Update: This vulnerability is deemed non-applicable. The fields such as .from, .to, and the initial 4 bytes of .data are meticulously defined to align with the expectations of the MinimalForwarder. This precision in data handling assures that the integrity of the data is maintained as per the MinimalForwarder's standards.

3.3 Low severity issues [2 - Inapplicable]

1. Zero address not checked

Contract: ZbyteForwarderDPlat

Functions: zbyteExecute()

Issue: zbyteDPlat address could be zero address when calling preExecute and PostExecute if by a mistake is not set by the owner.

Fix: Add a require to check that is not zero address.

Update: Team will set the state by theirself and transfer ownership to the dao.

2. Chain transfer values are not checked

Contract: Zbytescrow, zbyte

Function: _record (zbyte), deposit (zbytescrow)

Issue: Parameters are not checked, we recommend to add requires to validate the inputs to avoid losing funds.

Fix:

_record function (zbyte contract):

```
function _record(Action action_, uint256 amount_, uint256 chain_) internal
{
 // Validate the action type
 require(action_ == Action.DEPOSIT || action_ == Action.WITHDRAW, "_record:
Invalid action
type");
 // Validate the amount for the DEPOSIT and WITHDRAW actions
 require(amount_ > 0, "_record: Amount must be greater than zero");
 // Validate the chain ID (additional conditions based on the specific use
case)
 require(chain_ > 0, "_record: Invalid chain ID");
 if (action_ == Action.DEPOSIT) {
 _totalSupply += amount_;
 _reserve[chain_] += amount_;
 } else {
 // Solidity 0.8.x automatically checks for underflows, so no need for an
explicit check
 _totalSupply -= amount_;
 _reserve[chain_] -= amount_;
 }
}
```

Zbytescrow (deposit function):

```
function deposit(uint256 relay_,
      uint256 chain_, address receiver_, uint256 amount_) public returns
(bool result) {
 require(relay_ > 0, "ZbyteEscrow: Invalid relay identifier");
require(chain_ > 0, "ZbyteEscrow: Invalid chain identifier");
require(receiver_ != address(0), "ZbyteEscrow: Receiver cannot be zero
address");
require(amount_ > 0, "ZbyteEscrow: Amount must be greater than zero")
return _deposit(relay_, chain_, receiver_, amount_);
}
```

A better implementation could be checking that the chain is valid from a mapping of the current ones that the protocol uses.

Update: All parameters are correctly checked.

3.4 Informational severity issues [1]

1. Testing forwarders in production contracts

Contract: ZbyteForwarderCore

Functions: approveAndDeposit()

Issue: Minimal forwarder contract is used from openzeppelin and it’s only usable for testing purposes.

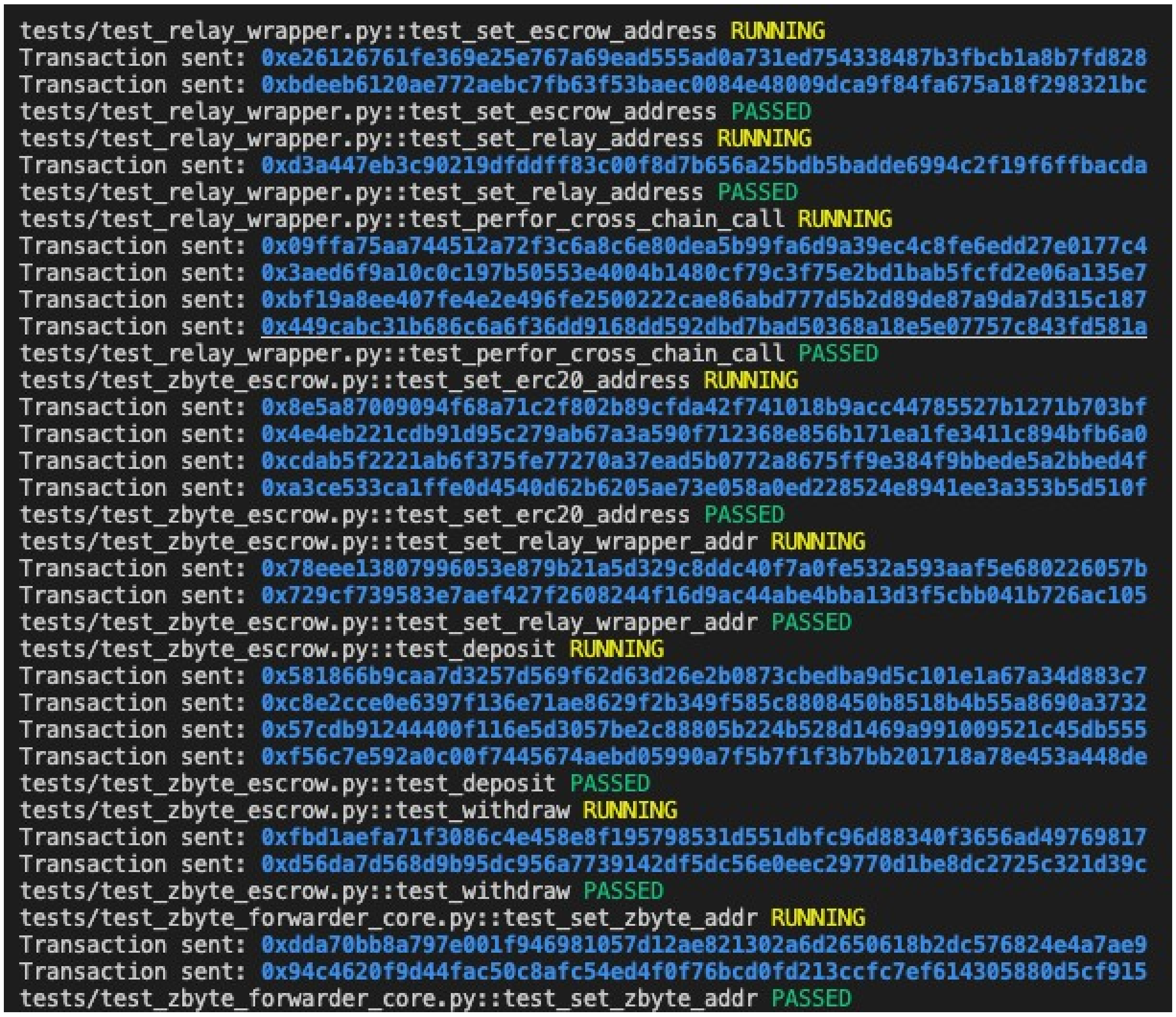
Fix: Use Forwarder.sol from opengsn/gsn github.

Update: As there are no major concerns, teams will keep minimal forwarder light weight.

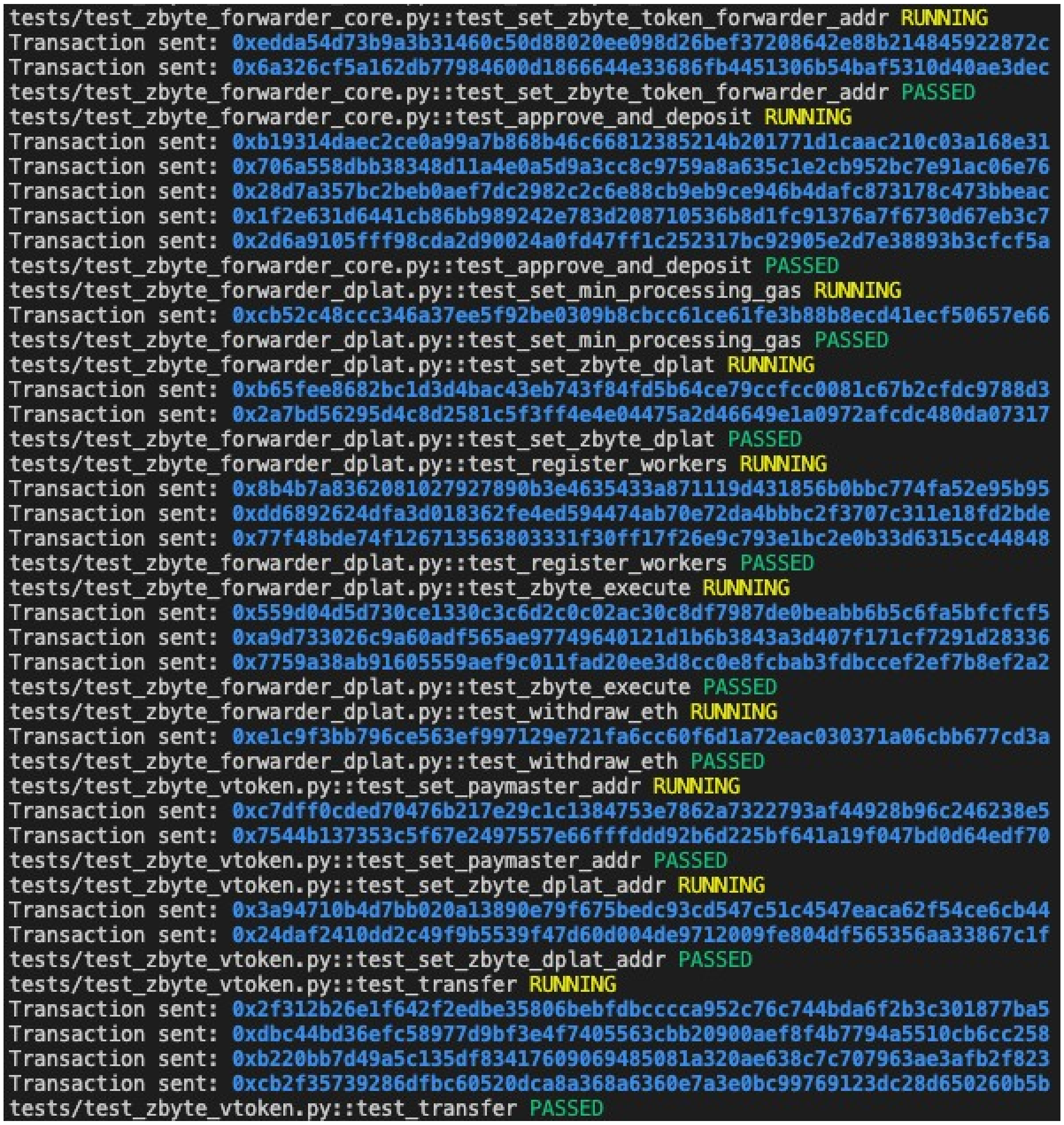
4.0 Testing coverage - python

During the testing phase, custom use cases were written to cover all the logic of contracts in python language. *<u>*Check "5 Annexes" to see the testing code.</u>*

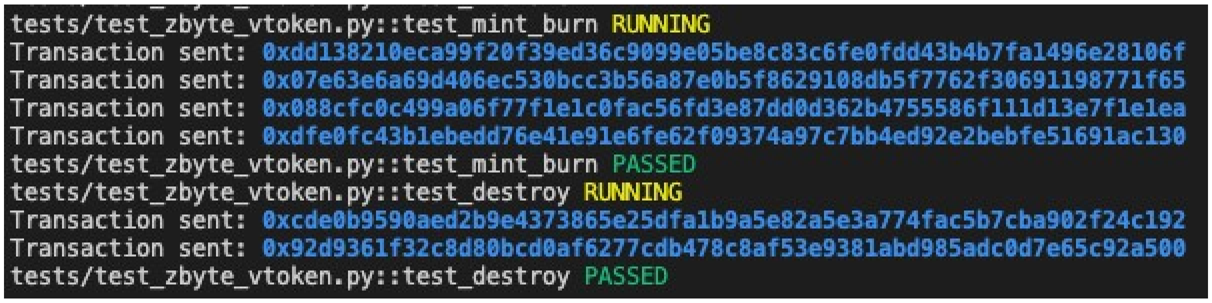
<u>Dplat contracts tests</u>



```
tests/test_relay_wrapper.py::test_set_escrow_address RUNNING
Transaction sent: 0xe26126761fe369e25e767a69ead555ad0a731ed754338487b3fbcb1a8b7fd828
Transaction sent: 0xbdeeb6120ae772aebc7fb63f53baec0084e48009dca9f84fa675a18f298321bc
tests/test_relay_wrapper.py::test_set_escrow_address PASSED
tests/test_relay_wrapper.py::test_set_relay_address RUNNING
Transaction sent: 0xd3a447eb3c90219dfddff83c00f8d7b656a25bdb5badde6994c2f19f6ffbacda
tests/test_relay_wrapper.py::test_set_relay_address PASSED
tests/test_relay_wrapper.py::test_perfor_cross_chain_call RUNNING
Transaction sent: 0x09ffa75aa744512a72f3c6a8c6e80dea5b99fa6d9a39ec4c8fe6edd27e0177c4
Transaction sent: 0x3aed6f9a10c0c197b50553e4004b1480cf79c3f75e2bd1bab5fcfd2e06a135e7
Transaction sent: 0xbf19a8ee407fe4e2e496fe2500222cae86abd777d5b2d89de87a9da7d315c187
Transaction sent: 0x449cabc31b686c6a6f36dd9168dd592dbd7bad50368a18e5e07757c843fd581a
tests/test_relay_wrapper.py::test_perfor_cross_chain_call PASSED
tests/test_zbyte_escrow.py::test_set_erc20_address RUNNING
Transaction sent: 0x8e5a87009094f68a71c2f802b89cfda42f741018b9acc44785527b1271b703bf
Transaction sent: 0x4e4eb221cdb91d95c279ab67a3a590f712368e856b171ea1fe3411c894bfb6a0
Transaction sent: 0xcdab5f2221ab6f375fe77270a37ead5b0772a8675ff9e384f9bbede5a2bbed4f
Transaction sent: 0xa3ce533ca1ffe0d4540d62b6205ae73e058a0ed228524e8941ee3a353b5d510f
tests/test_zbyte_escrow.py::test_set_erc20_address PASSED
tests/test_zbyte_escrow.py::test_set_relay_wrapper_addr RUNNING
Transaction sent: 0x78eee13807996053e879b21a5d329c8ddc40f7a0fe532a593aaf5e680226057b
Transaction sent: 0x729cf739583e7aef427f2608244f16d9ac44abe4bba13d3f5cbb041b726ac105
tests/test_zbyte_escrow.py::test_set_relay_wrapper_addr PASSED
tests/test_zbyte_escrow.py::test_deposit RUNNING
Transaction sent: 0x581866b9caa7d3257d569f62d63d26e2b0873cbedba9d5c101e1a67a34d883c7
Transaction sent: 0xc8e2cce0e6397f136e71ae8629f2b349f585c8808450b8518b4b55a8690a3732
Transaction sent: 0x57cdb91244400f116e5d3057be2c88805b224b528d1469a991009521c45db555
Transaction sent: 0xf56c7e592a0c00f7445674aebd05990a7f5b7f1f3b7bb201718a78e453a448de
tests/test_zbyte_escrow.py::test_deposit PASSED
tests/test_zbyte_escrow.py::test_withdraw RUNNING
Transaction sent: 0xfbd1aefa71f3086c4e458e8f195798531d551dbfc96d88340f3656ad49769817
Transaction sent: 0xd56da7d568d9b95dc956a7739142df5dc56e0eec29770d1be8dc2725c321d39c
tests/test_zbyte_escrow.py::test_withdraw PASSED
tests/test_zbyte_forwarder_core.py::test_set_zbyte_addr RUNNING
Transaction sent: 0xdda70bb8a797e001f946981057d12ae821302a6d2650618b2dc576824e4a7ae9
Transaction sent: 0x94c4620f9d44fac50c8afc54ed4f0f76bcd0fd213ccfc7ef614305880d5cf915
tests/test_zbyte_forwarder_core.py::test_set_zbyte_addr PASSED
```



```
tests/test_zbyte_forwarder_core.py::test_set_zbyte_token_forwarder_addr RUNNING
Transaction sent: 0xedda54d73b9a3b31460c50d88020ee098d26bef37208642e88b214845922872c
Transaction sent: 0x6a326cf5a162db77984600d1866644e33686fb4451306b54baf5310d40ae3dec
tests/test_zbyte_forwarder_core.py::test_set_zbyte_token_forwarder_addr PASSED
tests/test_zbyte_forwarder_core.py::test_approve_and_deposit RUNNING
Transaction sent: 0xb19314daec2ce0a99a7b868b46c66812385214b201771d1caac210c03a168e31
Transaction sent: 0x706a558dbb38348d11a4e0a5d9a3cc8c9759a8a635c1e2cb952bc7e91ac06e76
Transaction sent: 0x28d7a357bc2beb0aef7dc2982c2c6e88cb9eb9ce946b4dafc873178c473bbeac
Transaction sent: 0x1f2e631d6441cb86bb989242e783d208710536b8d1fc91376a7f6730d67eb3c7
Transaction sent: 0x2d6a9105fff98cda2d90024a0fd47ff1c252317bc92905e2d7e38893b3cfcf5a
tests/test_zbyte_forwarder_core.py::test_approve_and_deposit PASSED
tests/test_zbyte_forwarder_dplat.py::test_set_min_processing_gas RUNNING
Transaction sent: 0xcb52c48ccc346a37ee5f92be0309b8cbcc61ce61fe3b88b8ecd41ecf50657e66
tests/test_zbyte_forwarder_dplat.py::test_set_min_processing_gas PASSED
tests/test_zbyte_forwarder_dplat.py::test_set_zbyte_dplat RUNNING
Transaction sent: 0xb65fee8682bc1d3d4bac43eb743f84fd5b64ce79ccfcc0081c67b2cfdc9788d3
Transaction sent: 0x2a7bd56295d4c8d2581c5f3ff4e4e04475a2d46649e1a0972afcdc480da07317
tests/test_zbyte_forwarder_dplat.py::test_set_zbyte_dplat PASSED
tests/test_zbyte_forwarder_dplat.py::test_register_workers RUNNING
Transaction sent: 0x8b4b7a8362081027927890b3e4635433a871119d431856b0bbc774fa52e95b95
Transaction sent: 0xdd6892624dfa3d018362fe4ed594474ab70e72da4bbbc2f3707c311e18fd2bde
Transaction sent: 0x77f48bde74f126713563803331f30ff17f26e9c793e1bc2e0b33d6315cc44848
tests/test_zbyte_forwarder_dplat.py::test_register_workers PASSED
tests/test_zbyte_forwarder_dplat.py::test_zbyte_execute RUNNING
Transaction sent: 0x559d04d5d730ce1330c3c6d2c0c02ac30c8df7987de0beabb6b5c6fa5bfcfcf5
Transaction sent: 0xa9d733026c9a60adf565ae97749640121d1b6b3843a3d407f171cf7291d28336
Transaction sent: 0x7759a38ab91605559aef9c011fad20ee3d8cc0e8fcbab3fdbccef2ef7b8ef2a2
tests/test_zbyte_forwarder_dplat.py::test_zbyte_execute PASSED
tests/test_zbyte_forwarder_dplat.py::test_withdraw_eth RUNNING
Transaction sent: 0xe1c9f3bb796ce563ef997129e721fa6cc60f6d1a72eac030371a06cbb677cd3a
tests/test_zbyte_forwarder_dplat.py::test_withdraw_eth PASSED
tests/test_zbyte_vtoken.py::test_set_paymaster_addr RUNNING
Transaction sent: 0xc7dff0cded70476b217e29c1c1384753e7862a7322793af44928b96c246238e5
Transaction sent: 0x7544b137353c5f67e2497557e66fffddd92b6d225bf641a19f047bd0d64edf70
tests/test_zbyte_vtoken.py::test_set_paymaster_addr PASSED
tests/test_zbyte_vtoken.py::test_set_zbyte_dplat_addr RUNNING
Transaction sent: 0x3a94710b4d7bb020a13890e79f675bedc93cd547c51c4547eaca62f54ce6cb44
Transaction sent: 0x24daf2410dd2c49f9b5539f47d60d004de9712009fe804df565356aa33867c1f
tests/test_zbyte_vtoken.py::test_set_zbyte_dplat_addr PASSED
tests/test_zbyte_vtoken.py::test_transfer RUNNING
Transaction sent: 0x2f312b26e1f642f2edbe35806bebfdbcccca952c76c744bda6f2b3c301877ba5
Transaction sent: 0xdbc44bd36efc58977d9bf3e4f7405563cbb20900aef8f4b7794a5510cb6cc258
Transaction sent: 0xb220bb7d49a5c135df83417609069485081a320ae638c7c707963ae3afb2f823
Transaction sent: 0xcb2f35739286dfbc60520dca8a368a6360e7a3e0bc99769123dc28d650260b5b
tests/test_zbyte_vtoken.py::test_transfer PASSED
```



```
tests/test_zbyte_vtoken.py::test_mint_burn RUNNING
Transaction sent: 0xdd138210eca99f20f39ed36c9099e05be8c83c6fe0fdd43b4b7fa1496e28106f
Transaction sent: 0x07e63e6a69d406ec530bcc3b56a87e0b5f8629108db5f7762f30691198771f65
Transaction sent: 0x088cfc0c499a06f77f1e1c0fac56fd3e87dd0d362b4755586f111d13e7f1e1ea
Transaction sent: 0xdfe0fc43b1ebedd76e41e91e6fe62f09374a97c7bb4ed92e2bebfe51691ac130
tests/test_zbyte_vtoken.py::test_mint_burn PASSED
tests/test_zbyte_vtoken.py::test_destroy RUNNING
Transaction sent: 0xcde0b9590aed2b9e4373865e25dfa1b9a5e82a5e3a774fac5b7cba902f24c192
Transaction sent: 0x92d9361f32c8d80bcd0af6277cdb478c8af53e9381abd985adc0d7e65c92a500
tests/test_zbyte_vtoken.py::test_destroy PASSED
```

5.0 Annexes

Testing code:

Relay_Wrapper:

```
from brownie import (
    reverts
)

from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    get_account,
    get_custom_error_hex
)

from scripts.deploy import (
    deploy_zbyte_vtoken,
    deploy_zbyte_forwarder_dplat,
    deploy_zbyte_escrow,
    deploy_relay_wrapper,
    deploy_zbyte_forwarder_core
)

def test_set_escrow_address(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    burner = get_account(2)
    treasury = get_account(3)
    forwarder = deploy_zbyte_forwarder_core(owner)
    new_zbyte = deploy_zbyte_vtoken(owner, burner)
    forwarder_dplat = deploy_zbyte_forwarder_dplat(owner)
    escrow = deploy_zbyte_escrow(owner, forwarder_dplat.address,
new_zbyte.address, treasury)
    relay = deploy_relay_wrapper(owner, forwarder)

    # assert
    with reverts("Ownable: caller is not the owner"): # no authorized
        relay.setEscrowAddress(escrow.address, {"from": other})
    with reverts('typed error: ' + get_custom_error_hex("ZeroAddress()")):
        relay.setEscrowAddress(ZERO_ADDRESS, {"from": owner})
    assert relay.escrow() == ZERO_ADDRESS
    relay.setEscrowAddress(escrow.address, {"from": owner})
    assert relay.escrow() == escrow.address

def test_set_relay_address(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    forwarder = deploy_zbyte_forwarder_core(owner)
    extra_relay = deploy_relay_wrapper(owner, forwarder)
    relay = deploy_relay_wrapper(owner, forwarder)

    chain_id = 66
    relay_id = 99

    # assert
    with reverts("Ownable: caller is not the owner"): # no authorized
        relay.setRelayAddress(chain_id, relay_id, extra_relay.address,
{"from": other})
    relay.setRelayAddress(chain_id, relay_id, extra_relay.address, {"from":
owner})

def test_perfor_cross_chain_call(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    random_addr = get_account(2)
    callback_contract = get_account(3)
    burner = get_account(4)
    treasury = get_account(5)
    forwarder = deploy_zbyte_forwarder_core(owner)
    new_zbyte = deploy_zbyte_vtoken(owner, burner)
    forwarder_dplat = deploy_zbyte_forwarder_dplat(owner)
    extra_relay = deploy_relay_wrapper(owner, forwarder)
    another_relay = deploy_relay_wrapper(owner, forwarder)
    relay = deploy_relay_wrapper(owner, forwarder)

    chain_id = 66
    relay_id = 99
    # assert
    with reverts(): # caller not scrow
        relay.performCrossChainCall(relay_id, chain_id, chain_id * 2,
random_addr, 0x00, 0, callback_contract, 0x00, {"from": other})
    escrow = deploy_zbyte_escrow(owner, forwarder_dplat.address,
new_zbyte.address, treasury)
    relay.setEscrowAddress(escrow.address, {"from": owner})
    with reverts('typed error: ' +
get_custom_error_hex("InvalidCallBackContract()")):
        relay.performCrossChainCall(relay_id, chain_id, chain_id * 2,
random_addr, 0x00, 1, ZERO_ADDRESS, 0x00, {"from": escrow.address})
    with reverts(): # Relay not set
        relay.performCrossChainCall(relay_id, chain_id, chain_id * 2,
random_addr, 0x00, 1, callback_contract, 0x00, {"from": escrow.address})
    relay.setRelayAddress(chain_id, relay_id, extra_relay.address, {"from":
owner})
    with reverts(): # Relay not set
        relay.performCrossChainCall(relay_id, chain_id, chain_id * 2,
random_addr, 0x00, 1, callback_contract, 0x00, {"from": escrow.address})
    relay.setRelayAddress(chain_id * 2, relay_id, another_relay.address,
{"from": owner})
    relay.performCrossChainCall(relay_id, chain_id, chain_id * 2,
random_addr, 0x00, 1, callback_contract, 0x00, {"from": escrow.address})
```

Zbyte_Escrow:

```
from brownie import (
    reverts
)

from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    get_account,
    get_custom_error_hex
)

from scripts.deploy import (
    deploy_zbyte_vtoken,
    deploy_zbyte_forwarder_dplat,
    deploy_zbyte_escrow,
    deploy_relay_wrapper,
    deploy_zbyte_forwarder_core
)

def test_set_erc20_address(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    burner = get_account(2)
    treasury = get_account(3)
    new_zbyte = deploy_zbyte_vtoken(owner, burner)
    verc20 = deploy_zbyte_vtoken(owner, burner)
    forwarder_dplat = deploy_zbyte_forwarder_dplat(owner)

    # assert
    with reverts('typed error: ' + get_custom_error_hex("ZeroAddress()")):
        deploy_zbyte_escrow(owner, forwarder_dplat.address,
new_zbyte.address, ZERO_ADDRESS)

    escrow = deploy_zbyte_escrow(owner, forwarder_dplat.address,
new_zbyte.address, treasury)
    with reverts("Ownable: caller is not the owner"):
        escrow.setvERC20Address(verc20.address, 1, {"from": other})
    with reverts('typed error: ' + get_custom_error_hex("ZeroAddress()")):
        escrow.setvERC20Address(ZERO_ADDRESS, 1, {"from": owner})
    with reverts('typed error: ' + get_custom_error_hex("ZeroValue()")):
        escrow.setvERC20Address(verc20.address, 0, {"from": owner})
    escrow.setvERC20Address(verc20.address, 1, {"from": owner})

def test_set_relay_wrapper_addr(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    burner = get_account(2)
    treasury = get_account(3)
    forwarder = deploy_zbyte_forwarder_core(owner)
    new_zbyte = deploy_zbyte_vtoken(owner, burner)
    relay = deploy_relay_wrapper(owner, forwarder)
    forwarder_dplat = deploy_zbyte_forwarder_dplat(owner)
    escrow = deploy_zbyte_escrow(owner, forwarder_dplat.address,
new_zbyte.address, treasury)

    # assert
    with reverts("Ownable: caller is not the owner"):
        escrow.setRelayWrapperAddress(relay.address, {"from": other})
    with reverts('typed error: ' + get_custom_error_hex("ZeroAddress()")):
        escrow.setRelayWrapperAddress(ZERO_ADDRESS, {"from": owner})
    escrow.setRelayWrapperAddress(relay.address, {"from": owner})

def test_deposit(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    burner = get_account(2)
    treasury = get_account(3)
    random_addr = get_account(4)
    forwarder = deploy_zbyte_forwarder_core(owner)
    new_zbyte = deploy_zbyte_vtoken(owner, burner)
    relay = deploy_relay_wrapper(owner, forwarder)
    verc20 = deploy_zbyte_vtoken(owner, burner)
    receiver_verc20 = deploy_zbyte_vtoken(owner, burner)
    forwarder_dplat = deploy_zbyte_forwarder_dplat(owner)
    escrow = deploy_zbyte_escrow(owner, forwarder_dplat.address,
new_zbyte.address, treasury)

    transfer_from_func_sig = 0x23b872dd
    mint_func_sig = 0x40c10f19

    # assert
    escrow.setvERC20Address(verc20.address, 1, {"from": owner})
    escrow.setRelayWrapperAddress(relay.address, {"from": owner})
    with reverts(): # no authorized
        escrow.deposit(1, 0, receiver_verc20.address, 1e18, {"from":
owner})
    # set public capabilities
    new_zbyte.setPublicCapability(transfer_from_func_sig, True, {"from":
owner})
    new_zbyte.setPublicCapability(mint_func_sig, True, {"from": owner})
    new_zbyte.setZbyteDPlatAddress(random_addr, {"from": owner})
    with reverts(): # not approved
        escrow.deposit(1, 0, receiver_verc20.address, 1e18, {"from":
other})
    new_zbyte.approve(escrow.address, 1e18, {"from": other})
    with reverts(): # not amount
        escrow.deposit(1, 0, receiver_verc20.address, 1e18, {"from":
```

```
other})
    new_zbyte.mint(other, 2e18, {"from": other})
    with reverts():
        escrow.deposit(1, 0, receiver_verc20.address, 1e18, {"from":
other})

def test_withdraw(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    burner = get_account(2)
    treasury = get_account(3)
    forwarder = deploy_zbyte_forwarder_core(owner)
    new_zbyte = deploy_zbyte_vtoken(owner, burner)
    relay = deploy_relay_wrapper(owner, forwarder)
    verc20 = deploy_zbyte_vtoken(owner, burner)
    forwarder_dplat = deploy_zbyte_forwarder_dplat(owner)
    escrow = deploy_zbyte_escrow(owner, forwarder_dplat.address,
new_zbyte.address, treasury)

    # assert
    escrow.setvERC20Address(verc20.address, 1, {"from": owner})
    escrow.setRelayWrapperAddress(relay.address, {"from": owner})

    with reverts(): # no authorized
        escrow.withdraw(1, 1, other, other, {"from": other})
    with reverts():
        escrow.withdraw(1, 1, other, other, {"from": owner})
```

Zbyte_forwarder_core:

```
from brownie import (
    reverts
)

from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    get_account,
    get_custom_error_hex
)
```

```
from scripts.deploy import (
    deploy_zbyte_vtoken,
    deploy_zbyte_forwarder_dplat,
    deploy_zbyte_forwarder_core,
    deploy_zbyte_escrow
)

def test_set_zbyte_addr(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    burner = get_account(2)
    new_zbyte = deploy_zbyte_vtoken(owner, burner)
    forwarder = deploy_zbyte_forwarder_core(owner)

    # assert
    with reverts("Ownable: caller is not the owner"):
        forwarder.setZbyteAddress(new_zbyte.address, {"from": other})
    with reverts('typed error: ' + get_custom_error_hex("ZeroAddress()")):
        forwarder.setZbyteAddress(ZERO_ADDRESS, {"from": owner})

    assert forwarder.zByteAddress() == ZERO_ADDRESS
    forwarder.setZbyteAddress(new_zbyte.address, {"from": owner})
    assert forwarder.zByteAddress() == new_zbyte.address

def test_set_zbyte_token_forwarder_addr(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    new_forwarder = deploy_zbyte_forwarder_dplat(owner)
    forwarder = deploy_zbyte_forwarder_core(owner)

    # assert
    with reverts("Ownable: caller is not the owner"):
        forwarder.setZbyteTokenForwarderAddress(new_forwarder.address, 
{"from": other})
    with reverts('typed error: ' + get_custom_error_hex("ZeroAddress()")):
        forwarder.setZbyteTokenForwarderAddress(ZERO_ADDRESS, {"from": 
owner})
    forwarder.setZbyteTokenForwarderAddress(new_forwarder.address, {"from": 
owner})
```

```
def test_set_zbyte_token_forwarder_addr(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    burner = get_account(2)
    treasury = get_account(3)
    forwarder_dplat = deploy_zbyte_forwarder_dplat(owner)
    new_zbyte = deploy_zbyte_vtoken(owner, burner)
    escrow = deploy_zbyte_escrow(owner, forwarder_dplat, new_zbyte,
treasury)
    forwarder = deploy_zbyte_forwarder_core(owner)

    # assert
    with reverts("Ownable: caller is not the owner"):
        forwarder.setEscrowAddress(escrow.address, {"from": other})
    with reverts('typed error: ' + get_custom_error_hex("ZeroAddress()")):
        forwarder.setEscrowAddress(ZERO_ADDRESS, {"from": owner})

    assert forwarder.escrowAddress() == ZERO_ADDRESS
    forwarder.setEscrowAddress(escrow.address, {"from": owner})
    assert forwarder.escrowAddress() == escrow.address

def test_approve_and_deposit(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    another = get_account(2)
    extra = get_account(3)
    burner = get_account(4)
    treasury = get_account(5)
    forwarder_dplat = deploy_zbyte_forwarder_dplat(owner)
    new_zbyte = deploy_zbyte_vtoken(owner, burner)
    escrow = deploy_zbyte_escrow(owner, forwarder_dplat.address,
new_zbyte.address, treasury)

    forwarder = deploy_zbyte_forwarder_core(owner)

    '''
    struct ForwardRequest {
        address from;
        address to;
        uint256 value;
        uint256 gas;
        uint256 nonce;
        bytes data;
    }
    '''
    approve_sig = get_custom_error_hex("approve(address,uint256)")
    deposit_sig =
get_custom_error_hex("deposit(uint256,uint256,address,uint256)")
    req_approve = [
        other, another, 1e18, 3000000, 1,

0x015ea7b300000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000de0b6b3a7640000
    ]
    req_deposit = [
        extra, another, 1e18, 3000000, 1, 0x00
    ]

    # assert
    with reverts("approveAndDeposit: Invalid from addresses"):
        forwarder.approveAndDeposit(req_approve, 0x00, req_deposit, 0x00,
{"from": other})
    req_deposit[0] = other
    with reverts("approveAndDeposit: Invalid approve data"):
        forwarder.approveAndDeposit(req_approve, 0x00, req_deposit, 0x00,
{"from": other})

    forwarder.setZbyteAddress(new_zbyte.address, {"from": owner})
    req_approve[1] = new_zbyte.address
    with reverts("approveAndDeposit: Invalid approve data"):
        forwarder.approveAndDeposit(req_approve, 0x00, req_deposit, 0x00,
{"from": other})

    forwarder.setEscrowAddress(escrow.address, {"from": owner})
    req_approve[-1] =
"0x095ea7b30000000000000000000000002c15a315610bfa5248e4cbcbd693320e9d8e03cc
000000000000000000000000000000000000000000000de0b6b3a7640000"
    with reverts():
        forwarder.approveAndDeposit(req_approve, 0x00, req_deposit, 0x00,
{"from": other})
    req_deposit[1] = escrow.address
    req_deposit[-1] = deposit_sig
    forwarder.setZbyteTokenForwarderAddress(forwarder_dplat.address,
{"from": owner})
    with reverts():
        forwarder.approveAndDeposit(req_approve, approve_sig, req_deposit,
deposit_sig, {"from": other})
```

Zbyte_forwarder_dplat:

```
from brownie import (
    reverts
)

from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    get_account,
    get_custom_error_hex
)

from scripts.deploy import (
    deploy_zbyte_forwarder_dplat,
    deploy_zbyte_dplat_payment_facet
)

def test_set_min_processing_gas(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    new_gas = 10000
    forwarder = deploy_zbyte_forwarder_dplat(owner)

    # assert
    with reverts("Ownable: caller is not the owner"):
        forwarder.setMinProcessingGas(new_gas, {"from": other})
    assert forwarder.minProcessingGas() == 0
    forwarder.setMinProcessingGas(new_gas, {"from": owner})
    assert forwarder.minProcessingGas() == new_gas

def test_set_zbyte_dplat(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    new_zbyteDPlat = get_account(2)
    forwarder = deploy_zbyte_forwarder_dplat(owner)

    # assert
    with reverts('typed error: ' + get_custom_error_hex("ZeroAddress()")):
        forwarder.setZbyteDPlat(ZERO_ADDRESS, {"from": owner})
    with reverts("Ownable: caller is not the owner"):
        forwarder.setZbyteDPlat(new_zbyteDPlat, {"from": other})

    assert forwarder.zbyteDPlat() == ZERO_ADDRESS
    forwarder.setZbyteDPlat(new_zbyteDPlat, {"from": owner})
    assert forwarder.zbyteDPlat() == new_zbyteDPlat

def test_register_workers(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    forwarder = deploy_zbyte_forwarder_dplat(owner)

    workers = [
        "0xF22573A2D3BAC95c71e61380ed6747A7FEfCc6aA",
        "0x8BA55781695cC9870D8632d8208f039c5d75eE00",
        "0x48D38f326c7b249764b72B50C05D567CA3b4F9D3"
    ]
    register = [True, False, True]

    # assert
    with reverts():
        forwarder.registerWorkers(workers, register[:1], {"from": owner})
    with reverts("Ownable: caller is not the owner"):
        forwarder.registerWorkers(workers, register, {"from": other})
    workers_fail = [
        "0xF22573A2D3BAC95c71e61380ed6747A7FEfCc6aA",
        ZERO_ADDRESS
    ]
    register_fail = [True, False]
    with reverts('typed error: ' + get_custom_error_hex("ZeroAddress()")):
        forwarder.registerWorkers(workers_fail, register_fail, {"from":
owner})
    forwarder.registerWorkers(workers, register, {"from": owner})

def test_zbyte_execute(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    another = get_account(2)
    forwarder = deploy_zbyte_forwarder_dplat(owner)
    zbyte_dplat = deploy_zbyte_dplat_payment_facet(owner)

    transfer_func_sig = 0xa9059cbb
    '''
    struct ForwardRequest {
        address from;
        address to;
        uint256 value;
        uint256 gas;
        uint256 nonce;
        bytes data;
    }
    '''
    forward_request = [other, another, 1e18, 0, 0, 0x00]

    # assert
    with reverts(): # only worker
        forwarder.zbyteExecute(forward_request, transfer_func_sig, {"from":
owner})
    workers = [other]
    register = [True]
    forwarder.registerWorkers(workers, register, {"from": owner})
    with reverts(): # NotEnoughEtherSent
        forwarder.zbyteExecute(forward_request, transfer_func_sig, {"from":
other})
    with reverts(): # NO zbyteDPlat set
        forwarder.zbyteExecute(forward_request, transfer_func_sig, {"from":
other, "value": 1e18})
    forwarder.setZbyteDPlat(zbyte_dplat.address, {"from": owner})

def test_withdraw_eth(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    another = get_account(2)
    forwarder = deploy_zbyte_forwarder_dplat(owner)
    # assert
    with reverts("Ownable: caller is not the owner"):
        forwarder.withdrawEth(another, {"from": other})
    forwarder.withdrawEth(other, {"from": owner})
```

Zbyte_vtokne:

```
from brownie import (
    reverts
)

from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    get_account,
    get_custom_error_hex
)

from scripts.deploy import (
    deploy_zbyte_vtoken
)

def test_set_paymaster_addr(only_local):
    #arrange
    owner = get_account(0)
    burner = get_account(1)
    other = get_account(2)
    new_paymaster = get_account(3)
    token = deploy_zbyte_vtoken(owner, burner)

    # assert
    with reverts('typed error: ' + get_custom_error_hex("ZeroAddress()")):
        token.setPaymasterAddress(ZERO_ADDRESS, {"from": owner})
    with reverts(""):
        token.setPaymasterAddress(new_paymaster, {"from": other})

    tx = token.setPaymasterAddress(new_paymaster, {"from": owner})
    assert tx.events["PaymasterAddressSet"] is not None
    assert tx.events["PaymasterAddressSet"].items()[0][1] == new_paymaster

def test_set_zbyte_dplat_addr(only_local):
    #arrange
    owner = get_account(0)
    burner = get_account(1)
    other = get_account(2)
    new_zbyte_dplat_addr = get_account(3)
    token = deploy_zbyte_vtoken(owner, burner)

    # assert
    with reverts('typed error: ' + get_custom_error_hex("ZeroAddress()")):
        token.setZbyteDPlatAddress(ZERO_ADDRESS, {"from": owner})
    with reverts(""):
        token.setZbyteDPlatAddress(new_zbyte_dplat_addr, {"from": other})

    tx = token.setZbyteDPlatAddress(new_zbyte_dplat_addr, {"from": owner})
    assert tx.events["ZbyteDPlatAddressSet"] is not None
    assert tx.events["ZbyteDPlatAddressSet"].items()[0][1] ==
new_zbyte_dplat_addr

def test_transfer(only_local):
    #arrange
    owner = get_account(0)
    burner = get_account(1)
    other = get_account(2)
    another = get_account(3)
    new_zbyte_dplat_addr = get_account(3)
    token = deploy_zbyte_vtoken(owner, burner)

    mint_func_sig = 0x40c10f19
    transfer_func_sig = 0xa9059cbb
    transfer_from_func_sig = 0x23b872dd

    with reverts('UNAUTHORIZED'):
        token.transfer(other, 1e18, {"from": owner})

    token.setPublicCapability(mint_func_sig, True, {"from": owner})
    token.setPublicCapability(transfer_func_sig, True, {"from": owner})
    token.setZbyteDPlatAddress(new_zbyte_dplat_addr, {"from": owner})

    # mint some tokens
    token.mint(other, 2e18, {"from": other})
    with reverts('ERC20: transfer amount exceeds balance'):
        token.transfer(another, 3e18, {"from": other})
    tx = token.transfer(another, 1e18, {"from": other})
    assert tx.events["Transfer"] is not None
    assert tx.events["Transfer"][0]['from'] == other
    assert tx.events["Transfer"][0]['to'] == another
    assert tx.events["Transfer"][0]['value'] == 1e18

    with reverts('UNAUTHORIZED'):
        token.transferFrom(other, another, 1e18, {"from": owner})
    token.setPublicCapability(transfer_from_func_sig, True, {"from":
owner})

    with reverts():
        token.transferFrom(other, another, 1e18, {"from": owner})
    token.approve(owner, 1e18, {"from": other})
    tx = token.transferFrom(other, another, 1e18, {"from": owner})
    assert token.balanceOf(another) == 2e18

def test_mint_burn(only_local):
    #arrange
    owner = get_account(0)
    burner = get_account(1)
    other = get_account(2)
    new_zbyte_dplat_addr = get_account(3)
    token = deploy_zbyte_vtoken(owner, burner)

    mint_func_sig = 0x40c10f19
    burn_func_sig = 0x9dc29fac

    # assert
    with reverts('UNAUTHORIZED'):
        token.mint(other, 1e18, {"from": owner})
    token.setPublicCapability(mint_func_sig, True, {"from": owner})
    with reverts('typed error: ' + get_custom_error_hex("ZeroAddress()")):
        token.mint(other, 1e18, {"from": owner})
    token.setZbyteDPlatAddress(new_zbyte_dplat_addr, {"from": owner})

    tx = token.mint(other, 1e18, {"from": other})
    assert tx.events["Transfer"] is not None
    assert tx.events["Transfer"][0]['from'] == ZERO_ADDRESS
    assert tx.events["Transfer"][0]['to'] == other
    assert tx.events["Transfer"][0]['value'] == 1e18

    with reverts('UNAUTHORIZED'):
        token.burn(other, 1e18, {"from": owner})
    token.setPublicCapability(burn_func_sig, True, {"from": owner})
    with reverts():
        token.burn(other, 2e18, {"from": owner})
    tx = token.burn(other, 1e18, {"from": owner})
    assert tx.events["Transfer"] is not None
    assert tx.events["Transfer"][0]['from'] == other
    assert tx.events["Transfer"][0]['to'] == burner
    assert tx.events["Transfer"][0]['value'] == 1e18
    assert token.balanceOf(burner) == 1e18

def test_destroy(only_local):
    #arrange
    owner = get_account(0)
    burner = get_account(1)
    other = get_account(2)
    token = deploy_zbyte_vtoken(owner, burner)

    destroy_func_sig = 0x00f55d9d

    # assert
    with reverts('UNAUTHORIZED'):
        token.destroy(other, {"from": owner})
    token.setPublicCapability(destroy_func_sig, True, {"from": owner})
    with reverts():
        token.destroy(other, {"from": owner})
    tx = token.destroy(burner, {"from": burner})
    assert tx.events["Transfer"] is not None
    assert tx.events["Transfer"][0]['from'] == burner
    assert tx.events["Transfer"][0]['to'] == ZERO_ADDRESS
    assert tx.events["Transfer"][0]['value'] == 0
```

6.0 Summary of the audit

High and medium criticality vulnerabilities were solved by the development team. Contracts can be deployed to production.