

Assure DeFi[®]

THE VERIFICATION **GOLD STANDARD**



Security Assessment

First Crypto Bank



Date: 22/01/2024

Audit Status: PASS

Audit Edition: Advanced



ASSURE DEFI[®]
THE VERIFICATION **GOLD STANDARD**

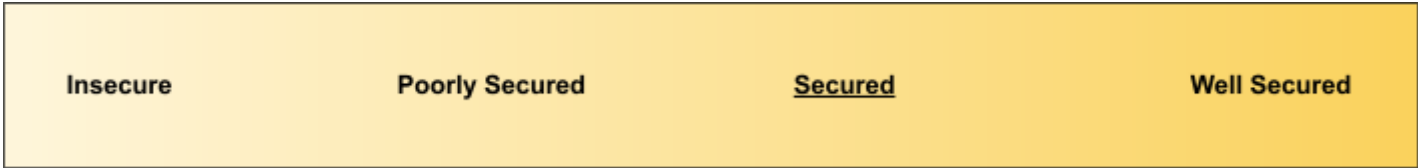
Risk Analysis

Vulnerability summary

Classification	Description
 High	High-level vulnerabilities can result in the loss of assets or manipulation of data.
 Medium	Medium-level vulnerabilities can be challenging to exploit, but they still have a considerable impact on smart contract execution, such as allowing public access to critical functions.
 Low	Low-level vulnerabilities are primarily associated with outdated or unused code snippets that generally do not significantly impact execution, sometimes they can be ignored.
 Informational	Informational vulnerabilities, code style violations, and informational statements do not affect smart contract execution and can typically be disregarded.

Executive Summary

According to the Assure assessment, the Customer's smart contract is **Secured**



Scope

Target Code And Revision

For this audit, we performed research, investigation, and review of the First Crypto Bank contracts followed by issue reporting, along with mitigation and remediation instructions outlined in this report.

Target Code And Revision

Project	Assure
Language	Solidity
Codebase	ERC20ByBankPad.sol [SHA256] - a26af4108f3305efe0ec50a62448e2044cc31c0703636b06a521640931fd1267 ERC20FactoryByBankPad.sol [SHA256] - 0736b1f97d4d15a56eee2257c0c51b2a914af6d0151ca8c03399b0f418ed6f62 ERC20MachineByBankPad.sol [SHA256] - 35777c2b75f758673736aa0eede2ba944d77572314c56620f30ead1cf58638b3
Audit Methodology	Static, Manual

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.

Category	Item
Code review & Functional Review	<ul style="list-style-type: none">• Compiler warnings.• Race conditions and Reentrancy. Cross-function race conditions.• Possible delays in data delivery.• Oracle calls.• Front running.• Timestamp dependence.• Integer Overflow and Underflow.• DoS with Revert.• DoS with block gas limit.• Methods execution permissions.• Economy model.• Private user data leaks.• Malicious Event log.• Scoping and Declarations.• Uninitialized storage pointers.• Arithmetic accuracy.• Design Logic.• Cross-function race conditions.• Safe Zeppelin module.• Fallback function security.• Overpowered functions / Owner privileges

AUDIT OVERVIEW



No high severity issues were found.



1. Unlimited fees in update functions

Contract: ERC20FactoryByBankPadBase.sol

Functions: updateSnipeAutoBurnPercents(), updateTaxLimits()

Issue: The function accepts arguments for burn percentages and taxes that could potentially be set to excessively high values. This can lead to undesirable behavior within the contract.

Fix: Implement a require() statement to validate that the burn percentages and taxes are within acceptable limits, preventing the input of excessively high values that could compromise contract functionality.

2. Contract initialization allows for the setting of unlimited fees

Contract: TaxHelper.sol

Functions: _initializeTax()

Issue: The variables buyTax, sellTax, and treasuryTax are susceptible to being assigned excessively high tax values, potentially leading to adverse contract behavior.

Fix: Incorporate a require() statement to ensure that the values assigned to buyTax, sellTax, and treasuryTax are within reasonable limits, thereby preventing the assignment of disproportionately high taxes that could negatively impact the contract's operations.



No Low severity issues were found.



No informational issues were found.

Testing coverage

During the testing phase, custom use cases were written to cover all the logic of contracts. **Check “Annexes” to see the testing code.*

First Crypto Bank contracts tests

```
contract: ERC20FactoryByBankPad – 70.6%
  ERC20FactoryByBankPadBase.maxTeamAllocForUUID – 100.0%
  ERC20FactoryByBankPadBase.updateBankMachine – 100.0%
  ERC20FactoryByBankPadBase.updateMaxTeamAlloc – 100.0%
  ERC20FactoryByBankPadBase.updateMaxTeamAllocForUUID – 100.0%
  ERC20FactoryByBankPadBase.updateServicePayAccount – 100.0%
  ERC20FactoryByBankPadBase.updateSnipeAutoBurnPercents – 100.0%
  ERC20FactoryByBankPadBase.updateTaxAfterLimits – 100.0%
  ERC20FactoryByBankPadBase.updateTaxLimits – 100.0%
  Ownable._checkOwner – 100.0%
  Address.sendValue – 75.0%
  ERC20FactoryByBankPad.createERC20 – 75.0%
  ERC20FactoryByBankPadBase._buildInheritedConf – 75.0%
  ERC20FactoryByBankPadBase._validateTaxProps – 75.0%
  Pausable._requireNotPaused – 75.0%
  ERC20FactoryByBankPad._deployERC20 – 55.0%
  Address._revert – 0.0%
  Address.functionCallWithValue – 0.0%
  Address.verifyCallResultFromTarget – 0.0%
  ERC20FactoryByBankPad.pauseFactory – 0.0%
  Ownable.transferOwnership – 0.0%
```

```
tests/test_erc20_by_bank_pad.py::test_transfer RUNNING
Transaction sent: 0x8798f460f66b28b7dc6fad19419f0e29520afd68d3f1e694c8ac90c63b1df477
Transaction sent: 0x0d5c4e6271b2cf6b35f96f7ee811120fdfcb432abb9229789481c3c7f102e07b
tests/test_erc20_by_bank_pad.py::test_transfer PASSED
tests/test_erc20_by_bank_pad_base.py::test_constructor RUNNING
Transaction sent: 0xfad60f2cabb2ea0de8e4d0156034e7b3cc42988de8363b2636e083326eb6304f
Transaction sent: 0x4606656ee60f07c0b36b6910cfcf9675efaace1a52c4d1c418d2158767b486eb
Transaction sent: 0xb79112efa74698f26016befae247f20356f11429051bb75aef55f6d91d7f85bb
Transaction sent: 0x803f7087fa0cfb83aed4c1dfa388d6d468da889782f3fad9e0ad4d2d88b4080c
Transaction sent: 0xaa64c3cc9f7ae71f5b30ecb07b7f420185dc992ef02ccc83052028ceda41ac28
tests/test_erc20_by_bank_pad_base.py::test_constructor PASSED
tests/test_erc20_by_bank_pad_base.py::test_launch RUNNING
Transaction sent: 0x431cc16849ce0509beb89aa816bbeb3a08a55e9d15c35924b5b26688b1010c61
Transaction sent: 0x331e704dedd0d1fb7870f01575981c94e48daf8db7d29d441fc6476df329d67d
Transaction sent: 0x6f105787753547cd6c9e233aacec0ed9ae1238cd9595e4c00cf19db2cf42cc3f
tests/test_erc20_by_bank_pad_base.py::test_launch PASSED
tests/test_erc20_factory_by_bank_pad_base.py::test_constructor RUNNING
Transaction sent: 0x163d60e52250e42c12783f2e49308f2b033b0c4772f0c68cfea2726034e4ae0f
tests/test_erc20_factory_by_bank_pad_base.py::test_constructor PASSED
tests/test_erc20_factory_by_bank_pad_base.py::test_create_erc20 PASSED
tests/test_erc20_factory_by_bank_pad_base.py::test_update_bank_machine RUNNING
Transaction sent: 0x3a035b1e1d9d1aac988d4db48bc6c52dfd23471abed7aa6ce87d2599f948c12c
Transaction sent: 0xe62ad435061f145f4cd79472fc26ce310fdd8835e5297118bc244779985dc26
tests/test_erc20_factory_by_bank_pad_base.py::test_update_bank_machine PASSED
```

```
tests/test_erc20_factory_by_bank_pad_base.py::test_max_team_alloc_for_uuid RUNNING
Transaction sent: 0x8b8ebb3ee332d4a59d23870c7358d71a622ab5cd64bfa3f523c665eb16dc6d93
Transaction sent: 0xfce288ee0d735cda0e93d6b421ba49d2531a36cf1cd15edcaf1cce7f15aa5e46
tests/test_erc20_factory_by_bank_pad_base.py::test_max_team_alloc_for_uuid PASSED
tests/test_erc20_factory_by_bank_pad_base.py::test_update_max_team_alloc RUNNING
Transaction sent: 0x4cea83fcc85add89a63dcc00b847b33f016b2e7ad806268aeeecdd118a357da3
Transaction sent: 0x406503f734c99b42f66ed3ffa62a64647f8360851f5b3cb245c4d5a4d0b51d76
tests/test_erc20_factory_by_bank_pad_base.py::test_update_max_team_alloc PASSED
tests/test_erc20_factory_by_bank_pad_base.py::test_update_service_pay_account RUNNING
Transaction sent: 0xe14d43be503963bc3bc74607248934d77d742d11e629f5a62f66f13c0f36f5a3
Transaction sent: 0x2772eb72355b48a047f201ac6b307435e0de8a4330599cbeadef21e97529e3e1
tests/test_erc20_factory_by_bank_pad_base.py::test_update_service_pay_account PASSED
tests/test_erc20_factory_by_bank_pad_base.py::test_update_snipe_auto_burn_percent PASSED
Transaction sent: 0x494d9af196b0bb08d11d207d285d1e5e04565152e431f30ed6df56fd6faca98f
Transaction sent: 0x8e42dcfcca6203230331a3b0b688154d6a445d507948ec10698eef5a6859855c
Transaction sent: 0x6510a536b5c231c5b8cbf3e857502050a091f647c761b2754e22f2cb418a1d7a
Transaction sent: 0x28936cc3d4259adc0bef9b07a5732f707abd96812560d45edb6c267c66a7f5b8
tests/test_erc20_factory_by_bank_pad_base.py::test_update_snipe_auto_burn_percent PASSED
tests/test_erc20_factory_by_bank_pad_base.py::test_update_tax_after_limits RUNNING
Transaction sent: 0x8ae6ce00887ed44a695dd34035c7a96c38bfc39620715606422bc8927c3708c2
Transaction sent: 0xcbf20faaa6b571f28c2ec21e9ab64f0ff954e0e9ae15527764a5349ca6ba4120
Transaction sent: 0xe43d6a0622f15d9597603a63631e89fefeae6e989ed665540b088eed8145a3e0
Transaction sent: 0x01ae08e97f23ed3f5242ac830d272b2aa8009d4e34f0acc3b7caecb93eba1fbf
tests/test_erc20_factory_by_bank_pad_base.py::test_update_tax_after_limits PASSED
tests/test_erc20_factory_by_bank_pad_base.py::test_update_tax_limits RUNNING
Transaction sent: 0xb252edfc194511cbb5f9bddf4a67901cb18c9d7e3e2a28f7622511697143ff74
Transaction sent: 0x4c2666e2f428e61f0e0344c8787237b98c1de7464d9e238c1c033edb69291fbe
Transaction sent: 0xb72764e70e4e5608310d4aac9f0e27254919414a3bd8d70b1984e326f5a600b7
Transaction sent: 0x26bf99673ef39d940943ef4f004db66ce1ce19a116fdf85b8c3df7715608e548
tests/test_erc20_factory_by_bank_pad_base.py::test_update_tax_limits PASSED
```

```
===== 12 passed in 8.00s =====
```

Annexes

Testing code:

ERC20 BY BANK PAD:

```
from brownie import (
    reverts,
)

from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    get_account,
    encode_distrib_param,
    encode_antibot_param,
    encode_tax_param,
    encode_lp_param
)

from scripts.deploy import (
    deploy_factory_bank_pad,
    deploy_bank_pad,
    deploy_router
)

def test_transfer(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)
    pay_account = get_account(3)
    deployer = get_account(4)
    team_account = get_account(5)
    tax_pay_account = get_account(6)
    treasury_account = get_account(7)

    router, pair_addr = deploy_router(owner)
    factory = deploy_factory_bank_pad(
        owner, pay_account, 1, 1, 1, 1
    )
    launch_conf = [
        "e14d0941-b4cc-48c1-9d5d-6d97c6a9173b", "BankPad", "BP",
        encode_distrib_param(100 * (10 ** 18), str(team_account), 1000),
        encode_antibot_param(10000, 10000, 10000, True),
        encode_tax_param(
            router.address, pair_addr,
            str(tax_pay_account), str(treasury_account),
            1000, 1000, 500,
```



```

    ),
    encode_lp_param(False, False, False, 1 * (10 ** 18), 0)
]
inherited_conf = [
    1000, 1000, 1000,
    1000, 1000, 1000,
    1000, 500,
    0, 0, 0, 0
]
]
# test transfer delayed
bank_pad = deploy_bank_pad(
    owner, factory.address, deployer, launch_conf, [
        1000, 1000, 1000,
        1000, 1000, 1000,
        1000, 500,
        0, 0, 60, 0
    ]
)
# Transfer some tokens to others
bank_pad.transfer(other, 5e18, {"from": factory.address})
# launch
trading_delayed = True
trading_disabled = False
bank_pad.launch(trading_delayed, trading_disabled, {"from": factory.address})
with reverts("trading delayed"):
    bank_pad.transfer(extra, 1e18, {"from": other})

# test transfer disabled
bank_pad = deploy_bank_pad(
    owner, factory.address, deployer, launch_conf, [
        1000, 1000, 1000,
        1000, 1000, 1000,
        1000, 500,
        0, 0, 0, 60
    ]
)
# Transfer some tokens to others
bank_pad.transfer(other, 5e18, {"from": factory.address})
# launch
trading_delayed = False
trading_disabled = True
bank_pad.launch(trading_delayed, trading_disabled, {"from": factory.address})
with reverts("trading disabled"):
    bank_pad.transfer(extra, 1e18, {"from": other})

# test transfer
bank_pad = deploy_bank_pad(
    owner, factory.address, deployer, launch_conf, inherited_conf
)

```

```

# Transfer some tokens to others
bank_pad.transfer(other, 10e18, {"from": factory.address})
bank_pad.transfer(extra, 10e18, {"from": factory.address})
# launch
trading_delayed = False
trading_disabled = False
bank_pad.launch(trading_delayed, trading_disabled, {"from": factory.address})

# Set pair to check fees
bank_pad.batchSetAsAmmPair([pair_addr], True, {"from": deployer})

tx = bank_pad.transfer(pair_addr, 10e18, {"from": other})
assert tx.events["Transfer"] is not None

tx = bank_pad.transfer(other, 1e18, {"from": pair_addr})
assert tx.events["Transfer"] is not None

tx = bank_pad.transfer(other, 1e18, {"from": extra})
assert tx.events["Transfer"] is not None

```

Erc20 by bank pad base:

```

from brownie import (
    reverts,
)

from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    get_account,
    encode_distrib_param,
    encode_antibot_param,
    encode_tax_param,
    encode_lp_param
)

from scripts.deploy import (
    deploy_factory_bank_pad,
    deploy_bank_pad_base,
    deploy_router
)

def test_constructor(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    pay_account = get_account(2)
    deployer = get_account(3)
    team_account = get_account(4)

```

```

tax_pay_account = get_account(5)
treasury_account = get_account(6)

router, pair_addr = deploy_router(owner)
factory = deploy_factory_bank_pad(
    owner, pay_account, 1, 1, 1, 1
)
launch_conf = [
    "e14d0941-b4cc-48c1-9d5d-6d97c6a9173b", "BankPad", "BP",
    encode_distrib_param(100 * (10 ** 18), str(team_account), 1000),
    encode_antibot_param(10000, 10000, 10000, False),
    encode_tax_param(
        router.address, pair_addr,
        str(tax_pay_account), str(treasury_account),
        1000, 1000, 500,
    ),
    encode_lp_param(False, False, False, 1 * (10 ** 18), 0)
]
inherited_conf = [
    1000, 1000, 1000,
    1000, 1000, 1000,
    1000, 500,
    0, 0, 0, 0
]
# assert
with reverts("tx limit out of range"):
    deploy_bank_pad_base(
        owner, factory.address, deployer, [
            "e14d0941-b4cc-48c1-9d5d-6d97c6a9173b", "BankPad", "BP",
            encode_distrib_param(100 * (10 ** 18), str(team_account), 1000),
            encode_antibot_param(10000, 0, 10000, False),
            encode_tax_param(
                router.address, pair_addr,
                str(tax_pay_account), str(treasury_account),
                1000, 1000, 500,
            ),
            encode_lp_param(False, False, False, 1 * (10 ** 18), 0)
        ], inherited_conf
    )
with reverts("hold limit out of range"):
    deploy_bank_pad_base(
        owner, factory.address, deployer, [
            "e14d0941-b4cc-48c1-9d5d-6d97c6a9173b", "BankPad", "BP",
            encode_distrib_param(100 * (10 ** 18), str(team_account), 1000),
            encode_antibot_param(0, 10000, 10000, False),
            encode_tax_param(
                router.address, pair_addr,
                str(tax_pay_account), str(treasury_account),
                1000, 1000, 500,

```

```

        ),
        encode_lp_param(False, False, False, 1 * (10 ** 18), 0)
    ], inherited_conf
)
with reverts("tx limit exceeds hold limit"):
    deploy_bank_pad_base(
        owner, factory.address, deployer, [
            "e14d0941-b4cc-48c1-9d5d-6d97c6a9173b", "BankPad", "BP",
            encode_distrib_param(100 * (10 ** 18), str(team_account), 1000),
            encode_antibot_param(9999, 10000, 10000, False),
            encode_tax_param(
                router.address, pair_addr,
                str(tax_pay_account), str(treasury_account),
                1000, 1000, 500,
            ),
            encode_lp_param(False, False, False, 1 * (10 ** 18), 0)
        ], inherited_conf
    )
with reverts("anti-dump limit out of range"):
    deploy_bank_pad_base(
        owner, factory.address, deployer, [
            "e14d0941-b4cc-48c1-9d5d-6d97c6a9173b", "BankPad", "BP",
            encode_distrib_param(100 * (10 ** 18), str(team_account), 1000),
            encode_antibot_param(10000, 10000, 20, False),
            encode_tax_param(
                router.address, pair_addr,
                str(tax_pay_account), str(treasury_account),
                1000, 1000, 500,
            ),
            encode_lp_param(False, False, False, 1 * (10 ** 18), 0)
        ], inherited_conf
    )
with reverts("invalid team account"):
    deploy_bank_pad_base(
        owner, factory.address, deployer, [
            "e14d0941-b4cc-48c1-9d5d-6d97c6a9173b", "BankPad", "BP",
            encode_distrib_param(100 * (10 ** 18), ZERO_ADDRESS, 1000),
            encode_antibot_param(10000, 10000, 25, False),
            encode_tax_param(
                router.address, pair_addr,
                str(tax_pay_account), str(treasury_account),
                1000, 1000, 500,
            ),
            encode_lp_param(False, False, False, 1 * (10 ** 18), 0)
        ], inherited_conf
    )

bank_pad = deploy_bank_pad_base(
    owner, factory.address, deployer, launch_conf, inherited_conf

```



```

    )
    assert bank_pad.balanceOf(factory.address) == 90e18
    assert bank_pad.balanceOf(team_account) == 10e18

def test_launch(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    pay_account = get_account(2)
    deployer = get_account(3)
    team_account = get_account(4)
    tax_pay_account = get_account(5)
    treasury_account = get_account(6)

    router, pair_addr = deploy_router(owner)
    factory = deploy_factory_bank_pad(
        owner, pay_account, 1, 1, 1, 1
    )
    launch_conf = [
        "e14d0941-b4cc-48c1-9d5d-6d97c6a9173b", "BankPad", "BP",
        encode_distrib_param(100 * (10 ** 18), str(team_account), 1000),
        encode_antibot_param(10000, 10000, 10000, False),
        encode_tax_param(
            router.address, pair_addr,
            str(tax_pay_account), str(treasury_account),
            1000, 1000, 500,
        ),
        encode_lp_param(False, False, False, 1 * (10 ** 18), 0)
    ]
    inherited_conf = [
        1000, 1000, 1000,
        1000, 1000, 1000,
        1000, 500,
        0, 0, 0, 0
    ]
    bank_pad = deploy_bank_pad_base(
        owner, factory.address, deployer, launch_conf, inherited_conf
    )

    # assert
    with reverts("caller is not the factory"):
        bank_pad.launch(False, False, {"from": other})
    with reverts("can not delayed and disabled"):
        bank_pad.launch(True, True, {"from": factory.address})

    bank_pad.launch(False, True, {"from": factory.address})
    with reverts("already launched"):
        bank_pad.launch(False, True, {"from": factory.address})

```

Factory by bank pad base:

```
from brownie import (
    reverts,
)

from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    get_account,
    encode_distrib_param,
    encode_antibot_param,
    encode_tax_param,
    encode_lp_param
)

from scripts.deploy import (
    deploy_factory_bank_pad,
    deploy_machine_bank_pad,
    deploy_router
)

def test_constructor(only_local):
    #arrange
    owner = get_account(0)
    pay_account = get_account(1)

    with reverts("invalid pay account"):
        deploy_factory_bank_pad(
            owner, ZERO_ADDRESS, 1, 1, 1, 1
        )
    factory = deploy_factory_bank_pad(
        owner, pay_account, 1, 1, 1, 1
    )

def test_create_erc20(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    pay_account = get_account(2)
    deployer = get_account(3)
    team_account = get_account(4)
    tax_pay_account = get_account(5)
    treasury_account = get_account(6)

    router, pair_addr = deploy_router(owner)

    launch_conf = [
        "53b0147a14414fa2bc9009eebb243555", "BankPad", "BP",
        encode_distrib_param(100 * (10 ** 18), str(team_account), 1000),
```

```

        encode_antibot_param(10000, 10000, 10000, False),
        encode_tax_param(
            router.address, pair_addr,
            str(tax_pay_account), str(treasury_account),
            1000, 1000, 500,
        ),
        encode_lp_param(False, False, False, 1 * (10 ** 18), 0)
    ]
    factory = deploy_factory_bank_pad(
        owner, pay_account, 1, 1, 1, 1
    )
    machine = deploy_machine_bank_pad(owner, factory.address)
    factory.updateBankMachine(machine.address, {"from": owner})
    factory.createERC20(launch_conf, {"from": other})

def test_update_bank_machine(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    pay_account = get_account(2)
    factory = deploy_factory_bank_pad(owner, pay_account, 1, 1, 1, 1)
    with reverts("Ownable: caller is not the owner"):
        factory.updateBankMachine("0x8d7Bb74a772BFC90aFEC092821e61B630f6B552E",
{"from": other})
    with reverts("invalid machine"):
        factory.updateBankMachine(ZERO_ADDRESS, {"from": owner})
    factory.updateBankMachine("0x8d7Bb74a772BFC90aFEC092821e61B630f6B552E", {"from":
owner})

def test_max_team_alloc_for_uuid(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    pay_account = get_account(2)
    factory = deploy_factory_bank_pad(owner, pay_account, 1, 1, 1, 1)
    with reverts("Ownable: caller is not the owner"):
        factory.updateMaxTeamAllocForUUID("a36ccbd8-6a21-4749-8df1-866a26932cb8",
1000, {"from": other})
    with reverts("out of range"):
        factory.updateMaxTeamAllocForUUID("a36ccbd8-6a21-4749-8df1-866a26932cb8",
60000, {"from": owner})

    assert factory.maxTeamAllocForUUID("a36ccbd8-6a21-4749-8df1-866a26932cb8") == 1000
    factory.updateMaxTeamAllocForUUID("a36ccbd8-6a21-4749-8df1-866a26932cb8", 2000,
{"from": owner})
    assert factory.maxTeamAllocForUUID("a36ccbd8-6a21-4749-8df1-866a26932cb8") == 2000

def test_update_max_team_alloc(only_local):
    #arrange

```

```

owner = get_account(0)
other = get_account(1)
pay_account = get_account(2)
factory = deploy_factory_bank_pad(owner, pay_account, 1, 1, 1, 1)
with reverts("Ownable: caller is not the owner"):
    factory.updateMaxTeamAlloc(1000, {"from": other})
with reverts("out of range"):
    factory.updateMaxTeamAlloc(60000, {"from": owner})
factory.updateMaxTeamAlloc(5000, {"from": owner})

def test_update_service_pay_account(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    pay_account = get_account(2)
    factory = deploy_factory_bank_pad(owner, pay_account, 1, 1, 1, 1)
    with reverts("Ownable: caller is not the owner"):
        factory.updateServicePayAccount(other, {"from": other})
    with reverts("invalid pay account"):
        factory.updateServicePayAccount(ZERO_ADDRESS, {"from": owner})
    factory.updateServicePayAccount(other, {"from": owner})

def test_update_snipe_auto_burn_percents(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    pay_account = get_account(2)
    factory = deploy_factory_bank_pad(owner, pay_account, 1, 1, 1, 1)
    with reverts("Ownable: caller is not the owner"):
        factory.updateSnipeAutoBurnPercents(1000,1000,1000, {"from": other})
    with reverts("out of range"):
        factory.updateSnipeAutoBurnPercents(20000,1000,1000, {"from": owner})
    with reverts("out of range"):
        factory.updateSnipeAutoBurnPercents(1000,20000,1000, {"from": owner})
    with reverts("out of range"):
        factory.updateSnipeAutoBurnPercents(1000,1000,20000, {"from": owner})
    factory.updateSnipeAutoBurnPercents(1000,1000,1000, {"from": owner})

def test_update_tax_after_limits(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    pay_account = get_account(2)
    factory = deploy_factory_bank_pad(owner, pay_account, 1, 1, 1, 1)
    with reverts("Ownable: caller is not the owner"):
        factory.updateTaxAfterLimits(1000,1000,1000, {"from": other})
    with reverts("out of range"):
        factory.updateTaxAfterLimits(20000,1000,1000, {"from": owner})
    with reverts("out of range"):

```







```
        factory.updateTaxAfterLimits(1000,20000,1000, {"from": owner})
    with reverts("out of range"):
        factory.updateTaxAfterLimits(1000,1000,20000, {"from": owner})
    factory.updateTaxAfterLimits(1000,1000,1000, {"from": owner})

def test_update_tax_limits(only_local):
    #arrange
    owner = get_account(0)
    other = get_account(1)
    pay_account = get_account(2)
    factory = deploy_factory_bank_pad(owner, pay_account, 1, 1, 1, 1)
    with reverts("Ownable: caller is not the owner"):
        factory.updateTaxLimits(1000,1000,1000, {"from": other})
    with reverts("out of range"):
        factory.updateTaxLimits(20000,1000,1000, {"from": owner})
    with reverts("out of range"):
        factory.updateTaxLimits(1000,20000,1000, {"from": owner})
    with reverts("out of range"):
        factory.updateTaxLimits(1000,1000,20000, {"from": owner})
    factory.updateTaxLimits(1000,1000,1000, {"from": owner})
```

Technical Findings Summary

Findings

Vulnerability Level	Total	Pending	Not Apply	Acknowledged	Resolved
 High	0				
 Medium	2				
 Low	0				
 Informational	0				

Assessment Results

Score Results

Review	Score
Global Score	85/100
Assure KYC	Completed https://assuredefi.com/projects/first-crypto-bank/
Audit Score	80/100

The Following Score System Has been Added to this page to help understand the value of the audit, the maximum score is 100, however to attain that value the project must pass and provide all the data needed for the assessment. Our Passing Score has been changed to 84 Points for a higher standard, if a project does not attain 85% is an automatic failure. Read our notes and final assessment below.

Audit Pass

Following our comprehensive security audit of the Solidity project First Crypto Bank, we inform you that the project has met the necessary security standards. The audit has uncovered 2 medium vulnerabilities that pose significant risks but the team has already completed a thorough KYC process to ensure that the fee percentage is set appropriately and consistently.

Disclaimer

Assure Defi has conducted an independent security assessment to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the reviewed code for the scope of this assessment. This report does not constitute agreement, acceptance, or advocating for the Project, and users relying on this report should not consider this as having any merit for financial advice in any shape, form, or nature. The contracts audited do not account for any economic developments that the Project in question may pursue, and the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude, and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are entirely free of exploits, bugs, vulnerabilities or deprecation of technologies.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence, regardless of the findings presented. Information is provided 'as is, and Assure Defi is under no covenant to audit completeness, accuracy, or solidity of the contracts. In no event will Assure Defi or its partners, employees, agents, or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions or actions with regards to the information provided in this audit report.

The assessment services provided by Assure Defi are subject to dependencies and are under continuing development. You agree that your access or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies with high levels of technical risk and uncertainty. The assessment reports could include false positives, negatives, and unpredictable results. The services may access, and depend upon, multiple layers of third parties.

