// HALBORN

NFTfi - Ethereum

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **March 7th, 2022 - March 25th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	03/23/2022	Ferran.Celades
0.2	Document Edits	03/28/2022	Ferran.Celades
0.3	Draft Review	03/28/2022	Gabi Urrutia
1.0	Remediation Plan	03/29/2022	Ferran Celades
1.1	Remediation Plan Review	03/30/2022	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
Ferran Celades	Halborn	Ferran.Celades@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

NFTfi engaged Halborn to conduct a security audit on their smart contracts beginning on March 7th, 2022 and ending on March 25th, 2022 . 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 a security risk that was acknowledged 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 NFTfi develop-v2.1-audit branch and Pool V5 contract solidity code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of 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 hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Remix IDE)

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.

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

1.4 SCOPE

IN-SCOPE : NFTfi v2.2-07-03-2022-audit branch contracts

The security assessment was scoped to the following smart contract:
Scope-Commit-Id : 77d8476ff40e048dcca108265e863a3ebdfb261a

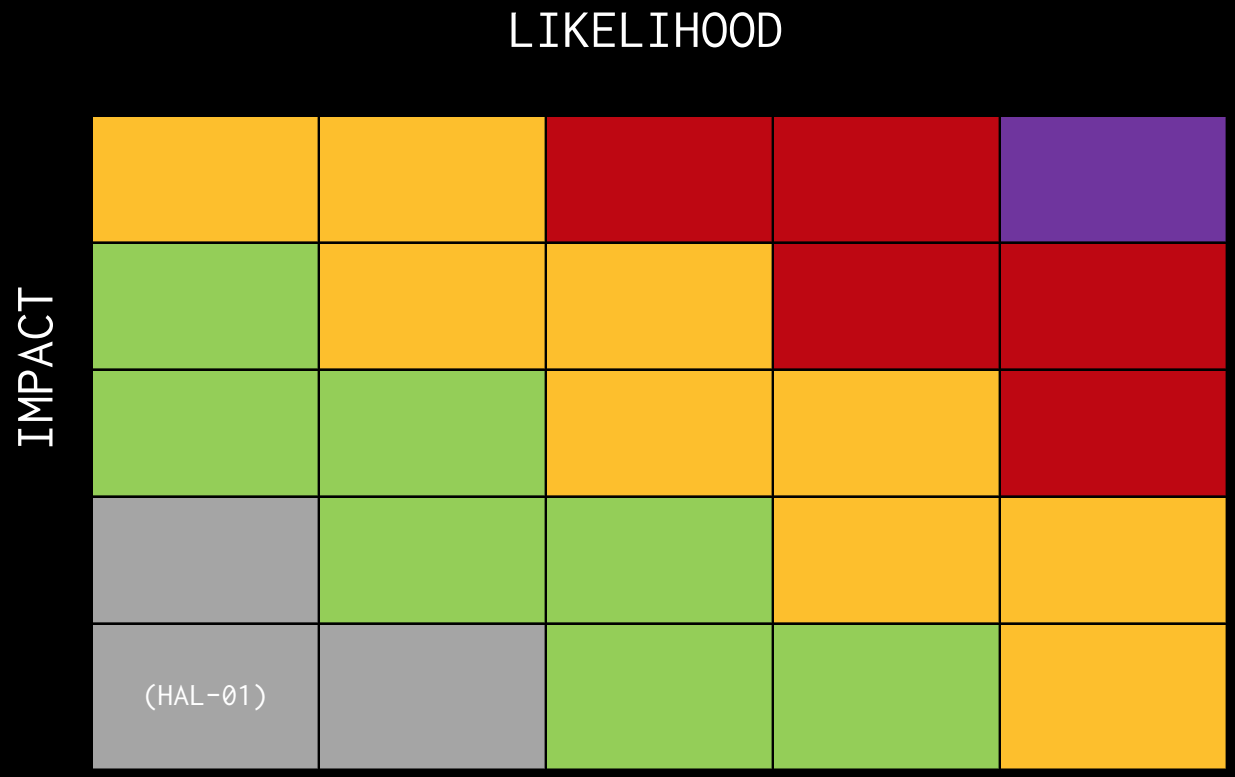
- contracts/NftfiHub.sol
- contracts/airdrop/AirdropFlashLoan.sol
- contracts/airdrop/AirdropReceiver.sol
- contracts/airdrop/AirdropReceiverFactory.sol
- contracts/airdrop/IAirdropReceiverFactory.sol
- contracts/composable/ERC9981155Extension.sol
- contracts/composable/ERC998ERC20Extension.sol
- contracts/composable/ERC998TopDown.sol
- contracts/composable/NftfiBundler.sol
- contracts/interfaces/*.sol
- contracts/loans/BaseLoan.sol
- contracts/loans/direct/DirectLoanCoordinator.sol
- contracts/loans/direct/loanTypes/DirectLoanBaseMinimal.sol
- contracts/loans/direct/loanTypes/DirectLoanFixedOffer.sol
- contracts/loans/direct/loanTypes/IDirectLoanBase.sol
- contracts/loans/direct/loanTypes/LoanAirdropUtils.sol
- contracts/loans/direct/loanTypes/LoanChecksAndCalculations.sol
- contracts/loans/direct/loanTypes/LoanData.sol
- contracts/nftTypeRegistry/nftTypes/CryptoKittiesWrapper.sol
- contracts/nftTypeRegistry/nftTypes/ERC1155Wrapper.sol
- contracts/nftTypeRegistry/nftTypes/ERC721Wrapper.sol
- contracts/permittedLists/PermittedAirdrops.sol
- contracts/permittedLists/PermittedERC20s.sol
- contracts/permittedLists/PermittedNFTsAndTypeRegistry.sol
- contracts/permittedLists/PermittedPartners.sol
- contracts/smartNft/SmartNft.sol
- contracts/utils/ContractKeys.sol
- contracts/utils/NFTfiSigningUtils.sol
- contracts/utils/NFTfiSigningUtilsContract.sol
- contracts/utils/NftReceiver.sol

EXECUTIVE OVERVIEW

- contracts/utils/Ownable.sol
- contracts/utils/TokenTrade.sol

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	1



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
CAUSE-EFFECT CHECKS	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) CAUSE-EFFECT CHECKS - INFORMATIONAL

Description:

The _payBackLoan under DirectLoanBaseMinimal does not pre-check the user's balance by being payoffAmount + adminFee + revenueShare it does so incrementally.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It would be nice if the code cloud pre-check if the user had enough balance for all transfers combined instead of checking and transferring incrementally. It is recommended to always check pre/post balances after a transfer.

Remediation Plan:

ACKNOWLEDGED: The NFTfi team acknowledged this issue.

MANUAL ANALYSIS

The functionDelegateCall is used on:

- airdrop/AirdropFlashLoan.sol
- airdrop/AirdropReceiver.sol
- loans/direct/loanTypes/LoanAirdropUtils.sol

_transferNFT makes a delegate call on the _nftTransferWrapper and _nftWrapper contexts, calling the transferNFT function. This means that all wrappers registered in PermittedNFTsAndTypeRegistry should not register any storage variable; otherwise, storage layout issues and collision could occur. Currently, all contracts present in nftTypeRegistry do not hold any storage variable, which makes them compatible with the delegate calling system.

- loans/direct/loanTypes/DirectLoanBaseMinimal.sol

_transferNFT does delegate to the nftCollateralWrapper loan in the context of the contract that implements DirectLoanBaseMinimal. At this time, there is only one contract that extends DirectLoanBaseMinimal which corresponds to the DirectLoanFixedOffer contract. The DirectLoanFixedOffer contract does not add new storage variables, but will get all the variables present in DirectLoanBaseMinimal which extends BaseLoan, NftReceiver and LoanData. Taking this into account, storage collision could be possible between nftCollateralWrapper and DirectLoanBaseMinimal contract. Although the mappings will be used in the nftCollateralWrappers code, it is still possible that some mapping may collide. When transferring the mapping used in the NFT code are the following:

Listing 1

```
1     mapping(address => uint256) private _balances;
2     mapping(uint256 => address) private _tokenApprovals;
```

The functionCall is used on:

- airdrop/AirdropFlashLoan.sol

- airdrop/AirdropReceiver.sol

Both contracts use a call to a user-specified target with any call data. However, both calldata and target are checked against IPermittedAirdrops for validity, and only allow anything other than the one approved by the owners via setAirdroptPermit or setAirdroptPermits.

4.1 NftfiHub

This contract is a central point for all the contracts registered in the NFTfi system. When a contract needs the address of another contract, it will refer to the hub with a key and the hub will be providing the corresponding address.

- All critical functions like setContract, setContracts are being checked for ownership
- A contract key that does not exist will return a zero address.

4.2 DirectLoanCoordinator

It will have the LOAN_COORDINATOR_ROLE role in the SmartNFT contract

- What if the user burns the promissoryNoteToken and obligationReceiptToken NFTs before the loan period ends?
  - It is not possible to burn tokens, only resolveLoan does it.

4.3 PermittedAirdrop

- Properly manage the ownership
- Critical functions such as setAirdroptPermit and setAirdroptPermits are guarded.

4.4 PermittedNFTsAndTypeRegistry

It is used to register a valid NFT in the NFTfi system. The contract will be used to access NFT wrappers and register new NFT types and permits.

- Critical functions are protected by the owner
- A critical function is protected by using `onlyOwnerOrAirdropFactory` instead of `onlyOwner`. Backtracking the ownership relationship does verify that the check is safe. `AirdropFactory` is cloning new airdrops templates. However, those templates will not have permissions on the `PermittedNFTsAndTypeRegistry` contract.

4.5 nftTypeRegistry/nftTypes

They are registered under the `PermittedNFTsAndTypeRegistry` contract and used with the `AirdropReceiver` contract to wrap the underlying NFT assets. All registered wrappers should not contain any storage variable. They are used by performing a `delegatecall` to the underlying wrapped contract and do not require new storage variables; otherwise, a variable collision could occur. Currently, all contracts present in `nftTypeRegistry` do not hold any new storage variables, which makes them compatible with the delegate calling system.

4.6 SmartNFT

All operations are restricted to the `LOAN_COORDINATOR_ROLE` which is expected to be for the `DirectLoanCoordinator` contract only. However, the `transfer` is allowed to anyone.

As an example, the obligation token can be transferred to the lender:

```
<Transaction '0x5bbd06e6d72e98d7a6a1bf5215b539e667a8f01660e2b11e6b19c2e0a3773440'>
>>> promissoryNoteToken.transferFrom(a[0], a[1], 158474109395590356974, {'from':a[0]})
Transaction sent: 0x0856cd4dbd704f321754376af1878413f2514b64335ca26753895a3b758d317c
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 7
  SmartNft.transferFrom confirmed   Block: 55   Gas used: 50016 (0.42%)

<Transaction '0x0856cd4dbd704f321754376af1878413f2514b64335ca26753895a3b758d317c'>
>>>
```

CALL GRAPH AND INHERITANCE

AirdropFlashLoan:

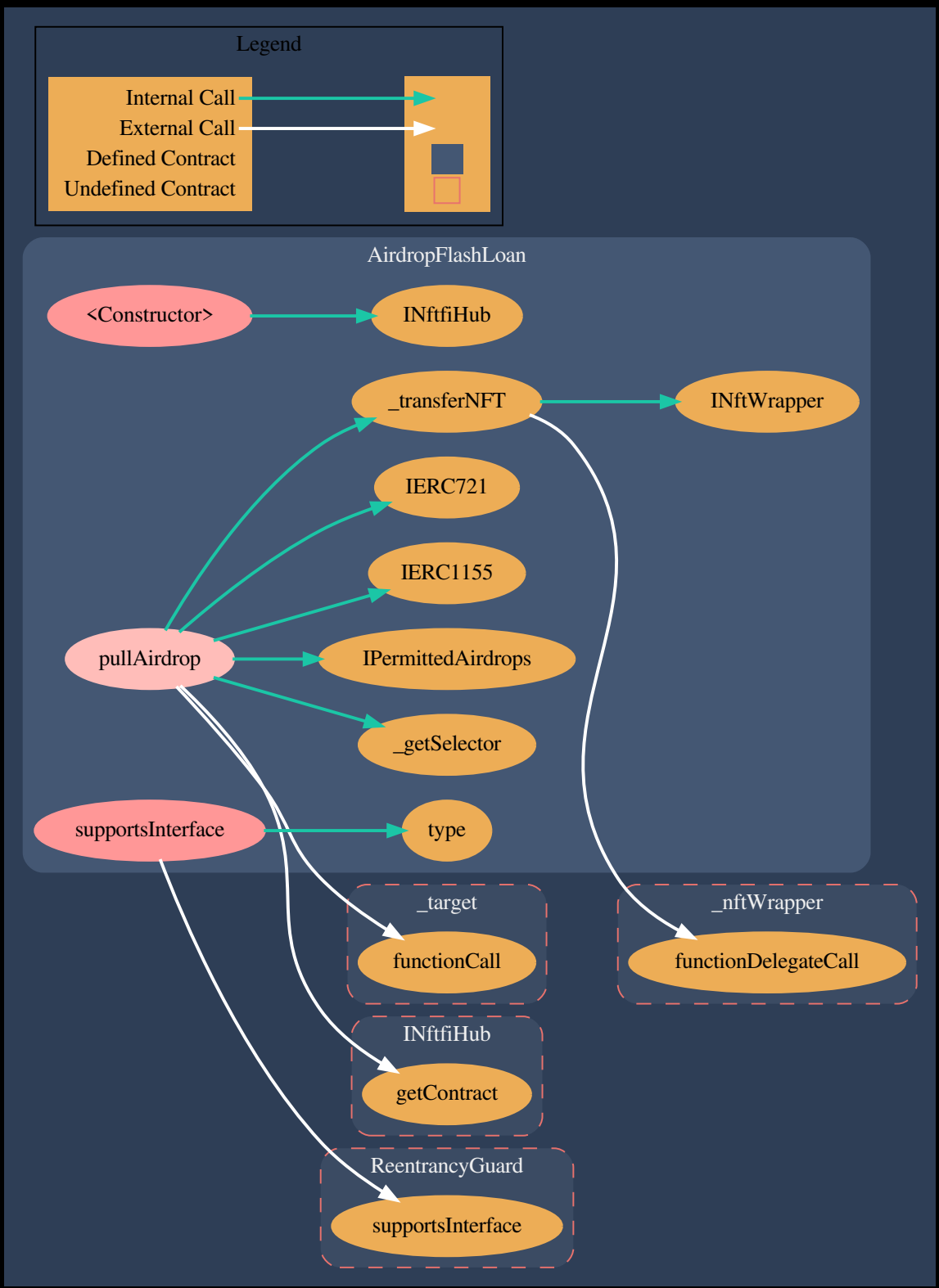


Figure 1: airdrop/AirdropFlashLoan.sol

AirdropReceiver:

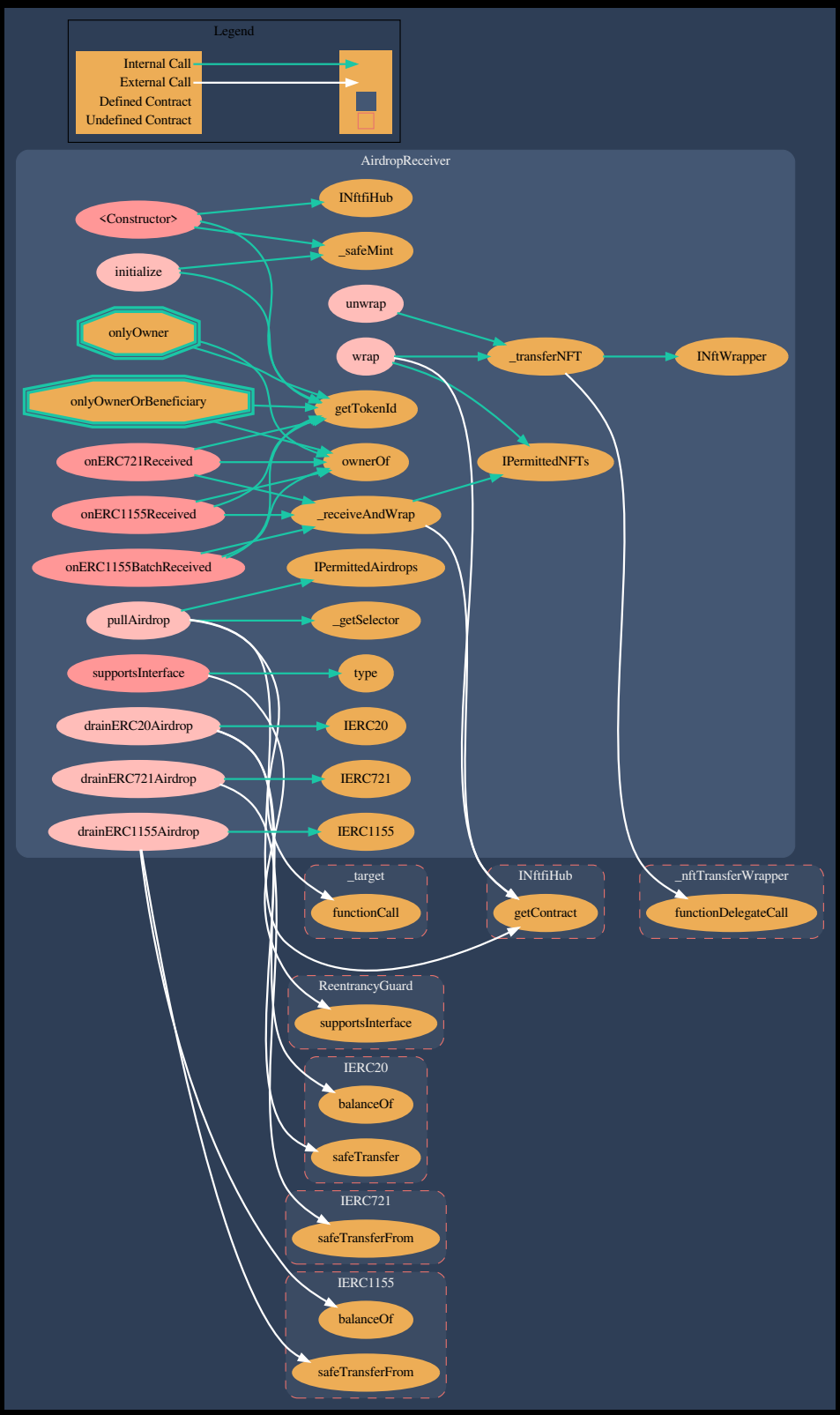


Figure 2: airdrop/AirdropReceiver.sol

DirectLoanBaseMinimal:

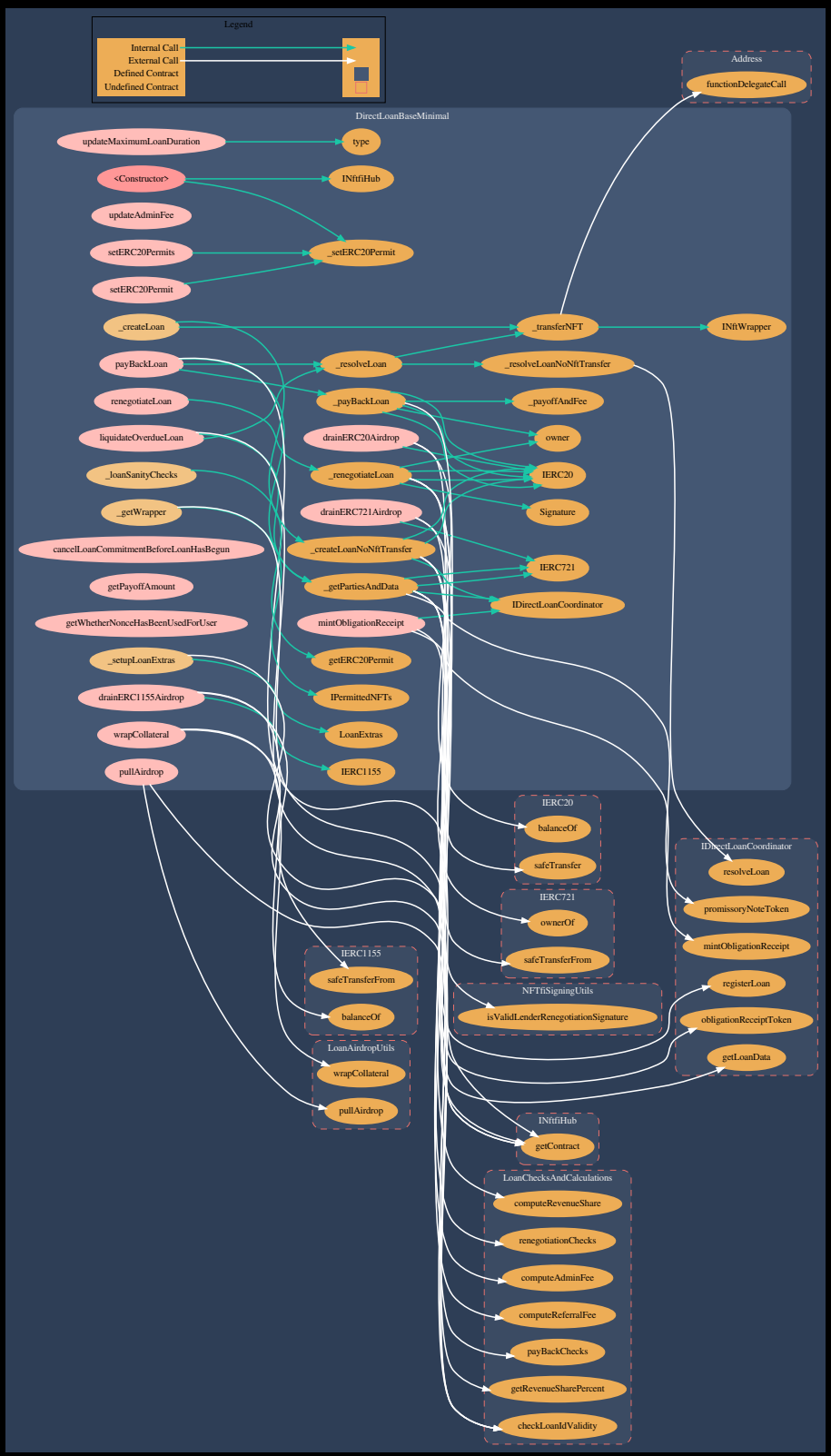


Figure 3: loans/direct/loanTypes/DirectLoanBaseMinimal.sol

DirectLoanFixedOffer:

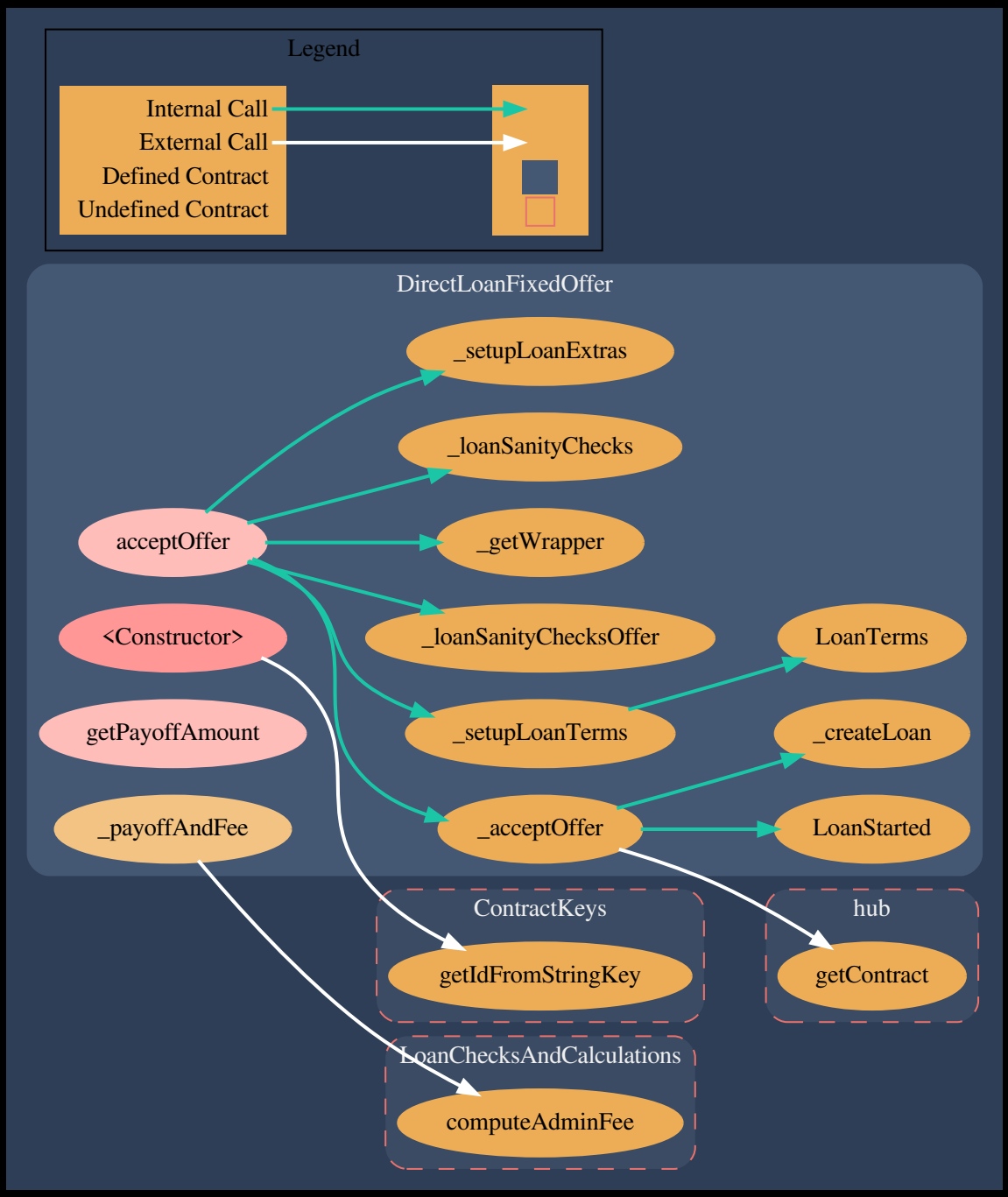


Figure 4: loans/direct/loanTypes/DirectLoanFixedOffer.sol

DirectLoanCoordinator:

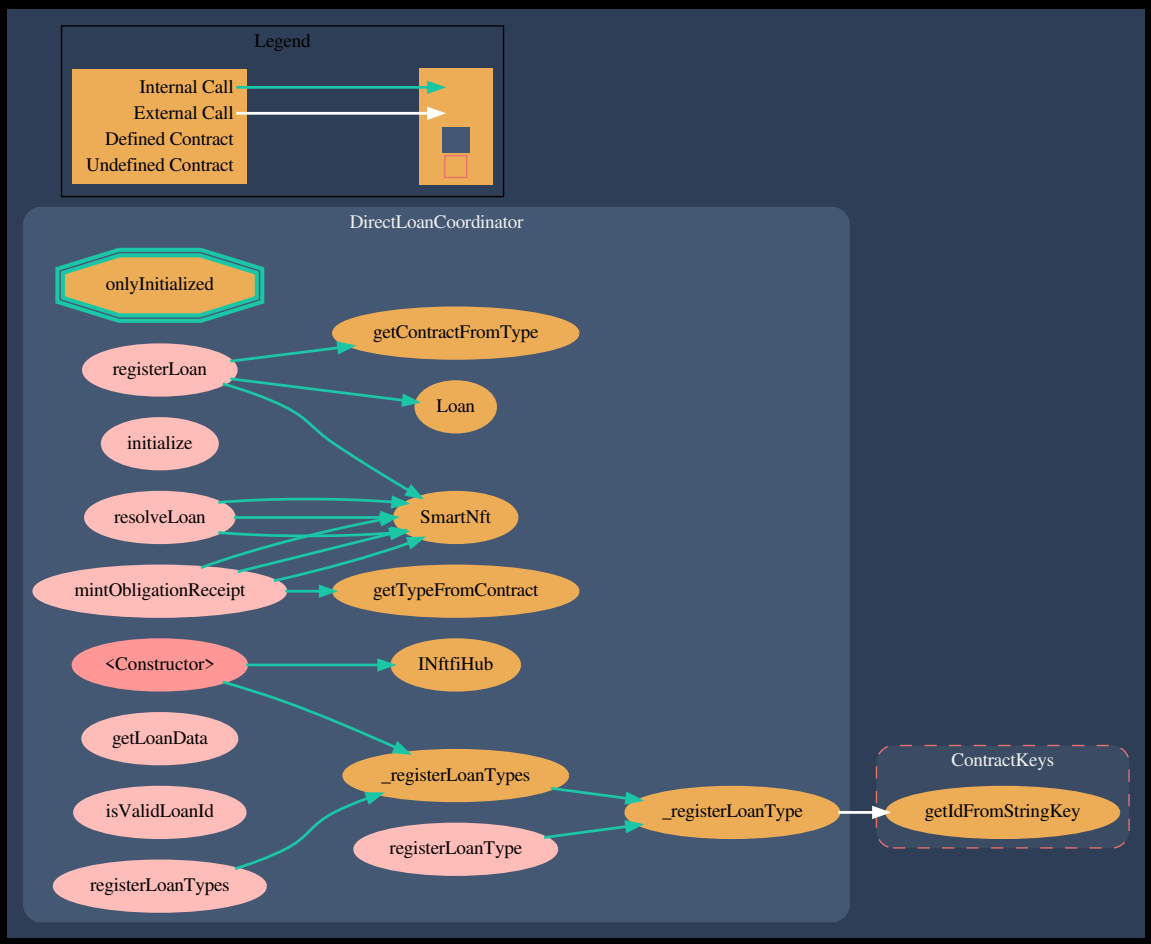


Figure 5: loans/direct/DirectLoanCoordinator.sol

CALL GRAPH AND INHERITANCE

SmartNFT:

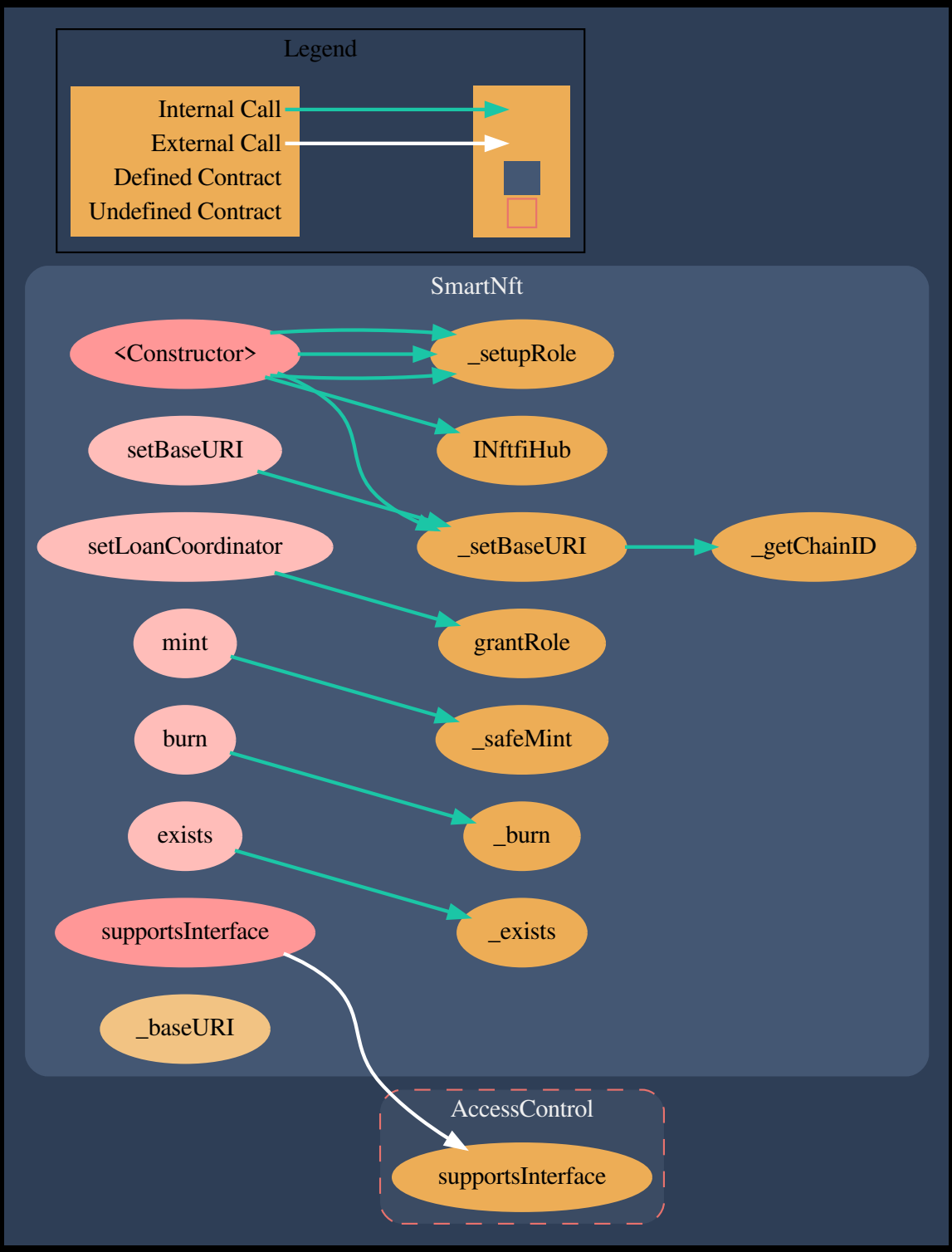


Figure 6: smartNFT/SmartNFT.sol

NFTfiBundler:

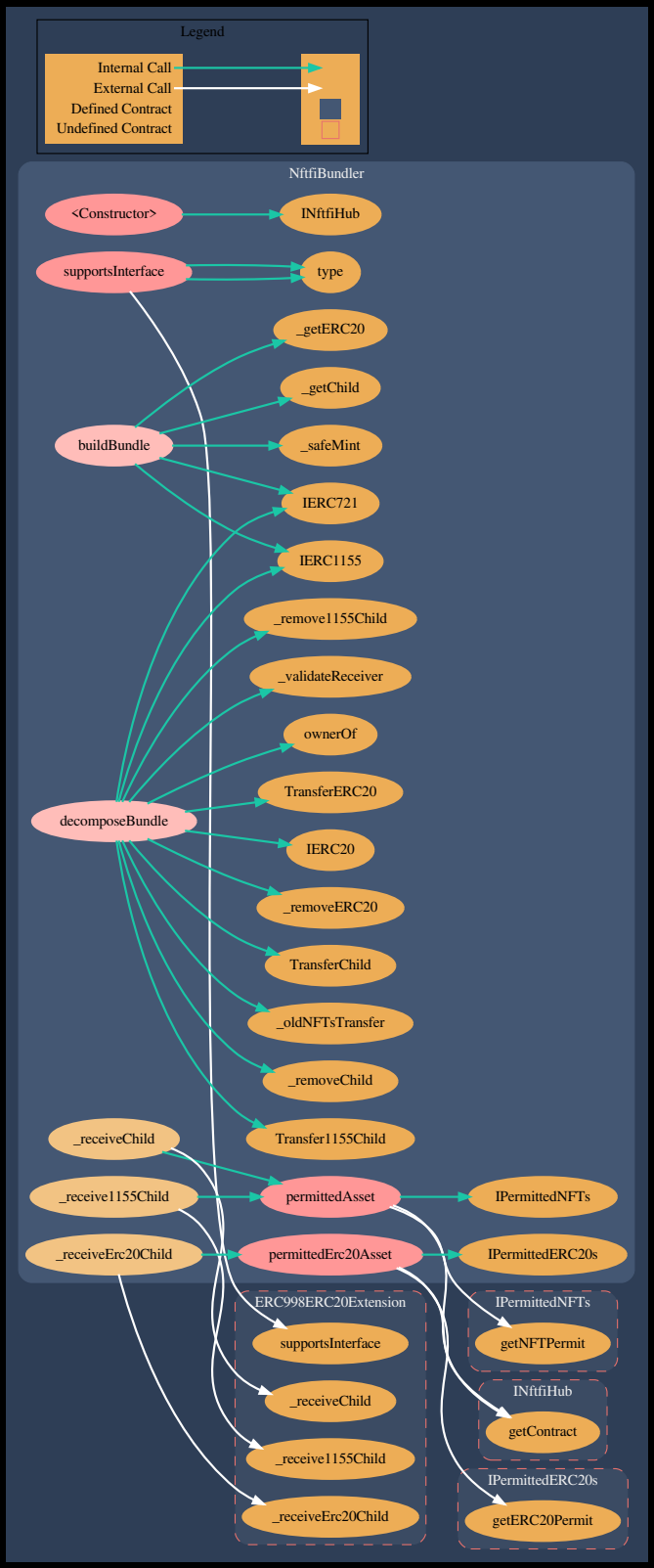


Figure 7: composable/NFTfiBundler.sol

AUTOMATED TESTING

6.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats. 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:

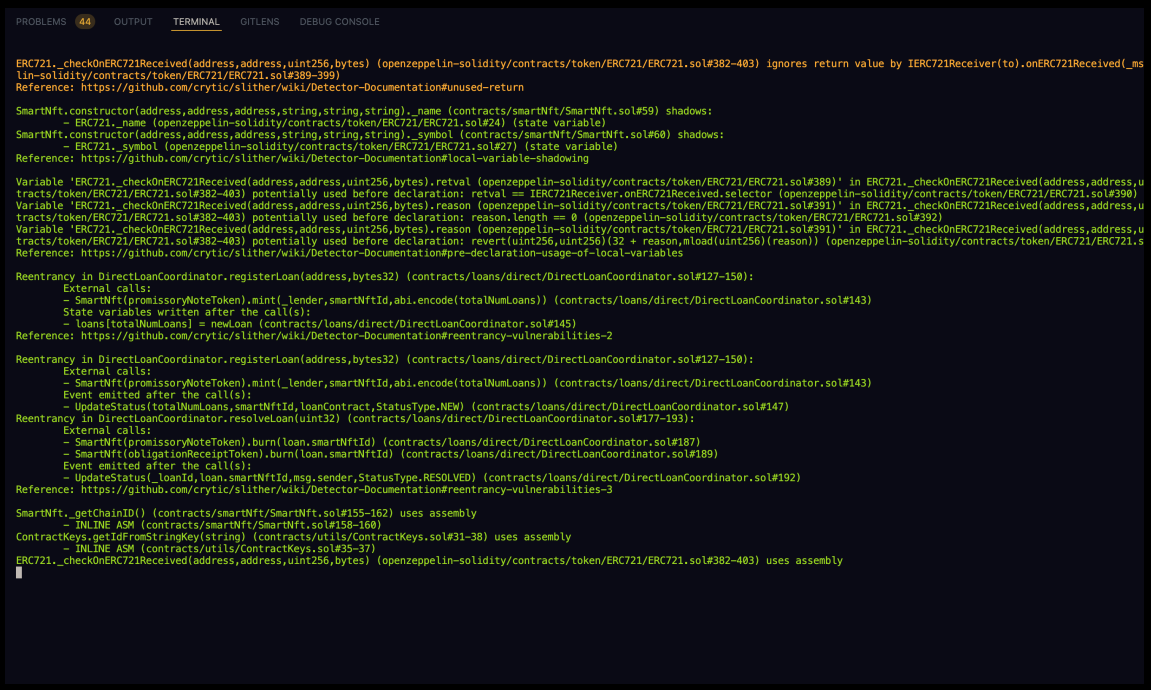


Figure 8: contracts/loans/direct/contracts_loans_direct_-DirectLoanCoordinator_slither

```
PROBLEMS 44  OUTPUT  TERMINAL  GITLENS  DEBUG CONSOLE

Compilation warnings/errors on contracts/loans/direct/loanTypes/DirectLoanFixedOffer.sol:
Warning: Contract code size exceeds 24576 bytes (a limit introduced in Spurious Dragon). This contract may not be deployable on mainnet. Consider enabling the optimizer (with a low "runs" val
ng libraries.
  --> contracts/loans/direct/loanTypes/DirectLoanFixedOffer.sol:57:1:
   |
57 | contract DirectLoanFixedOffer is DirectLoanBaseMinimal {
   | ^ (Relevant source part starts here and spans across multiple lines).



Reentrancy in DirectLoanBaseMinimal.payBackLoan(uint32) (contracts/loans/direct/loanTypes/DirectLoanBaseMinimal.sol#489-507):
        External calls:
        - _payBackLoan(_loanId,borrower,lender,loan) (contracts/loans/direct/loanTypes/DirectLoanBaseMinimal.sol#498)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (openzeppelin-solidity/contracts/token/ERC20/utils/SafeERC20.sol#93)
                - (success,returndata) = target.call{value: value}(data) (openzeppelin-solidity/contracts/utils/Address.sol#132)
                - IERC20(_loan.loanERC20Denomination).safeTransferFrom(msg.sender,_lender,payoffAmount) (contracts/loans/direct/loanTypes/DirectLoanBaseMinimal.sol#921)
                - IERC20(_loan.loanERC20Denomination).safeTransferFrom(msg.sender,loanExtras.revenueSharePartner,revenueShare) (contracts/loans/direct/loanTypes/DirectLoanBaseMinimal.sol#933-
                - IERC20(_loan.loanERC20Denomination).safeTransferFrom(msg.sender,owner(),adminFee) (contracts/loans/direct/loanTypes/DirectLoanBaseMinimal.sol#940)
        - _resolveLoan(_loanId,borrower,loan,loanCoordinator) (contracts/loans/direct/loanTypes/DirectLoanBaseMinimal.sol#500)
                - Address.functionDelegateCall(_loanTerms.nftCollateralWrapper,abi.encodeWithSelector(INftWrapper(_loanTerms.nftCollateralWrapper).transferNFT.selector,_sender,_recipient,_loa
nftCollateralId),NFT not successfully transferred) (contracts/loans/direct/loanTypes/DirectLoanBaseMinimal.sol#885-895)
                - _loanCoordinator.resolveLoan(_loanId) (contracts/loans/direct/loanTypes/DirectLoanBaseMinimal.sol#999)
                - (success,returndata) = target.delegatecall(data) (openzeppelin-solidity/contracts/utils/Address.sol#186)
        External calls sending eth:
        - _payBackLoan(_loanId,borrower,lender,loan) (contracts/loans/direct/loanTypes/DirectLoanBaseMinimal.sol#498)
                - (success,returndata) = target.call{value: value}(data) (openzeppelin-solidity/contracts/utils/Address.sol#132)
        State variables written after the call(s):
        - delete loanIdToLoan[_loanId] (contracts/loans/direct/loanTypes/DirectLoanBaseMinimal.sol#505)
        - delete loanIdToLoanExtras[_loanId] (contracts/loans/direct/loanTypes/DirectLoanBaseMinimal.sol#506)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities

Reentrancy in DirectLoanBaseMinimal._renegotiateLoan(uint32,uint32,uint256,uint256,uint256,uint256,bytes) (contracts/loans/direct/loanTypes/DirectLoanBaseMinimal.sol#728-795):
        External calls:
        - require(bool,string)(NFTfiSigningUtils.isValidLenderRenegotiationSignature(_loanId,_newLoanDuration,_newMaximumRepaymentAmount,_renegotiationFee,Signature(lender,_lenderNonce,_expir
ture is invalid) (contracts/loans/direct/loanTypes/DirectLoanBaseMinimal.sol#750-759)
        - IERC20(loan.loanERC20Denomination).safeTransferFrom(borrower,lender,_renegotiationFee - renegotiationAdminFee) (contracts/loans/direct/loanTypes/DirectLoanBaseMinimal.sol#774-778)
        - IERC20(loan.loanERC20Denomination).safeTransferFrom(borrower,owner(),renegotiationAdminFee) (contracts/loans/direct/loanTypes/DirectLoanBaseMinimal.sol#780)
        State variables written after the call(s):
        - loan.loanDuration = _newLoanDuration (contracts/loans/direct/loanTypes/DirectLoanBaseMinimal.sol#783)
        - loan.maximumRepaymentAmount = _newMaximumRepaymentAmount (contracts/loans/direct/loanTypes/DirectLoanBaseMinimal.sol#784)
Reentrancy in DirectLoanBaseMinimal.liquidateOverdueLoan(uint32) (contracts/loans/direct/loanTypes/DirectLoanBaseMinimal.sol#523-562):
        External calls:
        - _resolveLoan(_loanId,lender,loan,loanCoordinator) (contracts/loans/direct/loanTypes/DirectLoanBaseMinimal.sol#543)
```

Figure 9: contracts/loans/direct/loanTypes/contracts_loans_direct_-loanTypes_DirectLoanFixedOffer_slither

```
PROBLEMS 44  OUTPUT  TERMINAL  GITLENS  DEBUG CONSOLE

LoanAirdropUtils._transferNFT(LoanData.LoanTerms,address,address) (contracts/loans/direct/loanTypes/LoanAirdropUtils.sol#183-199) ignores return value by Address.functionDelegateCall(_loan.nf
r(INftWrapper(_loan.nftCollateralWrapper).transferNFT.selector,_sender,_recipient,_loan.nftCollateralContract,_loan.nftCollateralId),NFT not successfully transferred) (contracts/loans/direct/
LoanAirdropUtils._transferNFTtoAirdropReceiver(LoanData.LoanTerms,address,address) (contracts/loans/direct/loanTypes/LoanAirdropUtils.sol#209-225) ignores return value by Address.functionDele
encodeWithSelector(INftWrapper(_loan.nftCollateralWrapper).wrapAirdropReceiver.selector,_airdropReceiverInstance,_loan.nftCollateralContract,_loan.nftCollateralId,_airdropBeneficiary),NFT was
oans/direct/loanTypes/LoanAirdropUtils.sol#214-224)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Reentrancy in LoanAirdropUtils.pullAirdrop(uint32,LoanData.LoanTerms,address,bytes,address,uint256,bool,uint256,INftfiHub) (contracts/loans/direct/loanTypes/LoanAirdropUtils.sol#65-132):
        External calls:
        - _transferNFT(_loan,address(this),address(airdropFlashLoan)) (contracts/loans/direct/loanTypes/LoanAirdropUtils.sol#98)
                - Address.functionDelegateCall(_loan.nftCollateralWrapper,abi.encodeWithSelector(INftWrapper(_loan.nftCollateralWrapper).transferNFT.selector,_sender,_recipient,_loan.nftColla
 not successfully transferred) (contracts/loans/direct/loanTypes/LoanAirdropUtils.sol#188-198)
                - (success,returndata) = target.delegatecall(data) (openzeppelin-solidity/contracts/utils/Address.sol#186)
        - airdropFlashLoan.pullAirdrop(_loan.nftCollateralContract,_loan.nftCollateralId,_loan.nftCollateralWrapper,_target,_data,_nftAirdrop,_nftAirdropId,_is1155,_nftAirdropAmount,borrower)
irdropUtils.sol#100-111)
        Event emitted after the call(s):
        - AirdropPulledFlashloan(_loanId,borrower,_loan.nftCollateralId,_loan.nftCollateralContract,_target,_data) (contracts/loans/direct/loanTypes/LoanAirdropUtils.sol#124-131)
Reentrancy in LoanAirdropUtils.wrapCollateral(uint32,LoanData.LoanTerms,INftfiHub) (contracts/loans/direct/loanTypes/LoanAirdropUtils.sol#134-174):
        External calls:
        - (instance,receiverId) = factory.createAirdropReceiver(address(this)) (contracts/loans/direct/loanTypes/LoanAirdropUtils.sol#157)
        - _transferNFTtoAirdropReceiver(_loan,instance,borrower) (contracts/loans/direct/loanTypes/LoanAirdropUtils.sol#160)
                - Address.functionDelegateCall(_loan.nftCollateralWrapper,abi.encodeWithSelector(INftWrapper(_loan.nftCollateralWrapper).wrapAirdropReceiver.selector,_airdropReceiverInstance,
lateralId,_airdropBeneficiary),NFT was not successfully migrated) (contracts/loans/direct/loanTypes/LoanAirdropUtils.sol#214-224)
                - (success,returndata) = target.delegatecall(data) (openzeppelin-solidity/contracts/utils/Address.sol#186)
        Event emitted after the call(s):
        - CollateralWrapped(_loanId,borrower,_loan.nftCollateralId,_loan.nftCollateralContract,receiverId,instance) (contracts/loans/direct/loanTypes/LoanAirdropUtils.sol#162-169)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

ContractKeys.getIdFromStringKey(string) (contracts/utils/ContractKeys.sol#31-38) uses assembly
        - INLINE ASM (contracts/utils/ContractKeys.sol#35-37)
Address.isContract(address) (openzeppelin-solidity/contracts/utils/Address.sol#27-37) uses assembly
        - INLINE ASM (openzeppelin-solidity/contracts/utils/Address.sol#33-35)
Address.verifyCallResult(bool,bytes,string) (openzeppelin-solidity/contracts/utils/Address.sol#196-216) uses assembly
        - INLINE ASM (openzeppelin-solidity/contracts/utils/Address.sol#208-211)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity is used:
        - Version used: ['0.8.4', '^0.8.0']
        - 0.8.4 (contracts/airdrop/IAirdropReceiverFactory.sol#3)
        - 0.8.4 (contracts/interfaces/IAirdropFlashLoan.sol#3)
        - 0.8.4 (contracts/interfaces/IDirectLoanCoordinator.sol#3)
        - 0.8.4 (contracts/interfaces/INftWrapper.sol#3)
        - 0.8.4 (contracts/interfaces/INftfiHub.sol#3)
        - 0.8.4 (contracts/interfaces/IPermittedERC20s.sol#3)
        - 0.8.4 (contracts/interfaces/IPermittedPartners.sol#3)
```

Figure 10: contracts/loans/direct/loanTypes/contracts_loans_direct_-loanTypes_LoanAirdropUtils_slither

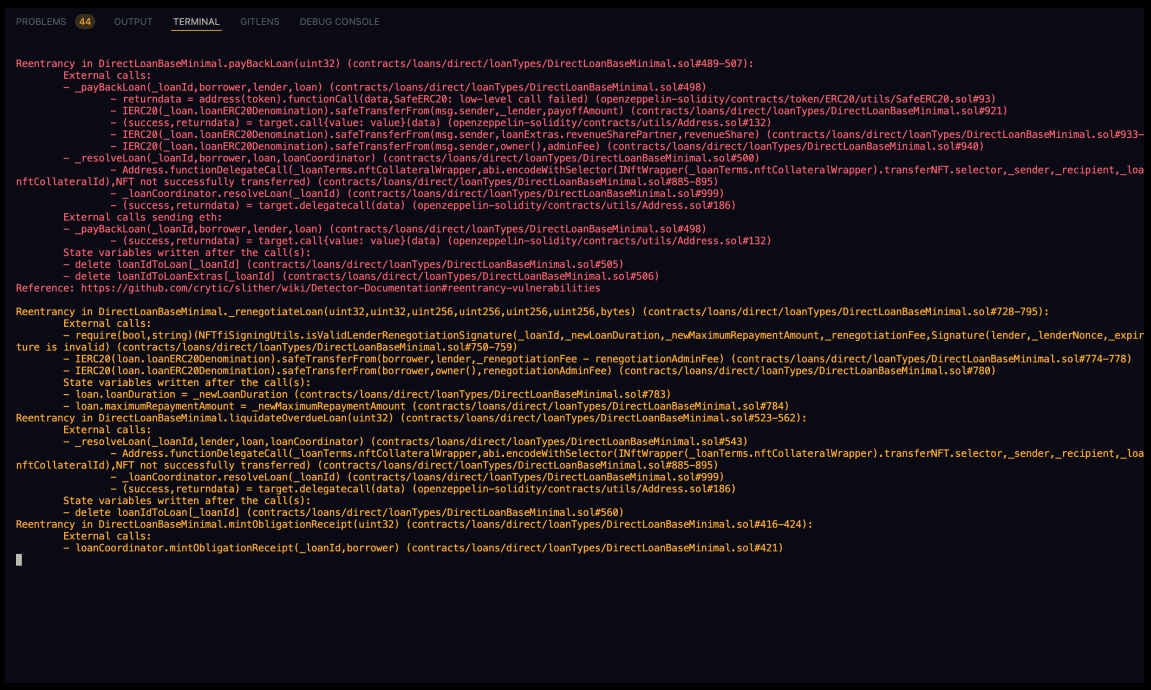


Figure 11: contracts/loans/direct/loanTypes/contracts_loans_direct_-loanTypes_DirectLoanBaseMinimal_slither

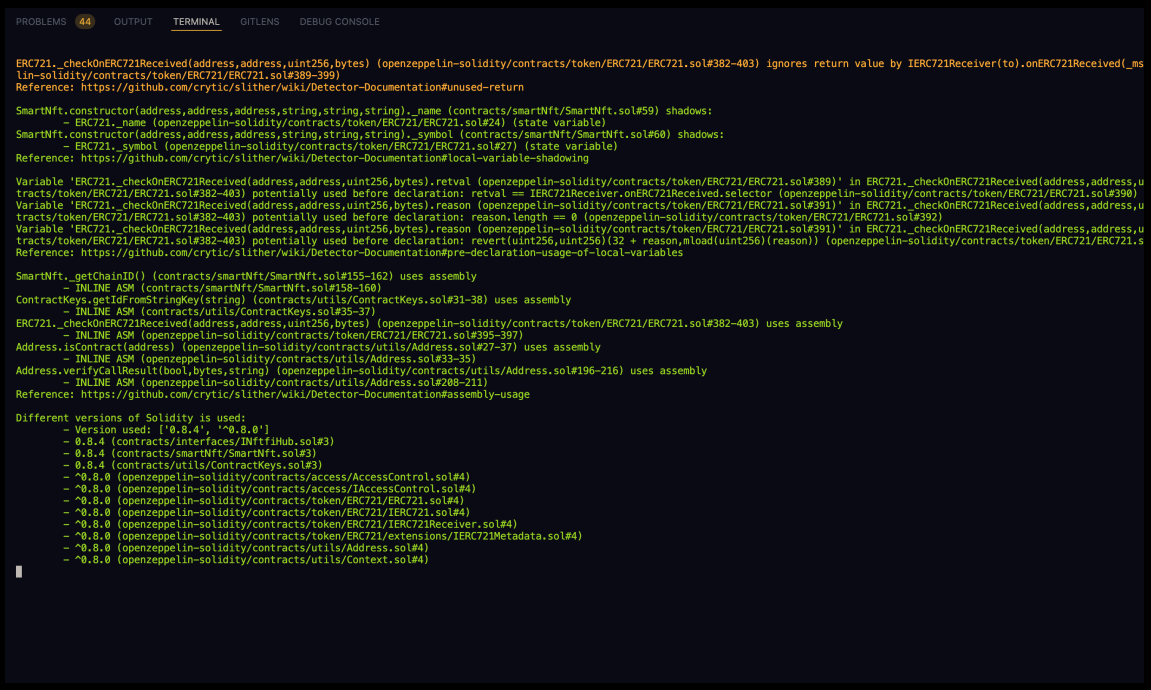


Figure 12: contracts/smartNft/contracts_smartNft_SmartNft_slither

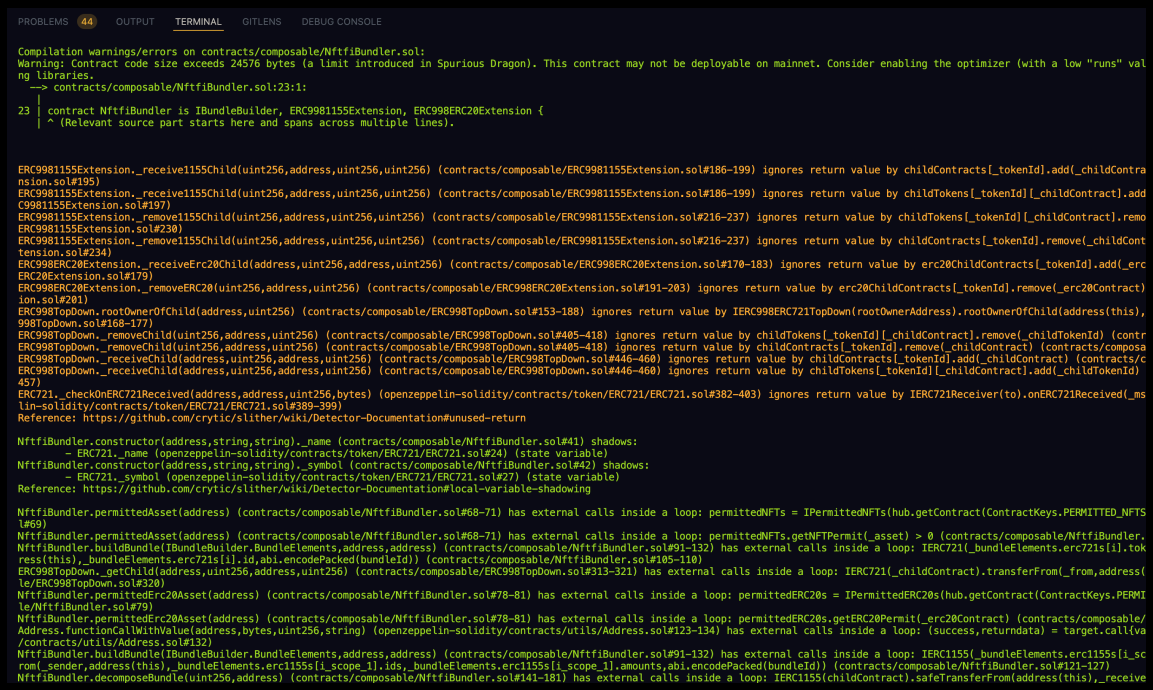


Figure 13: contracts/composable/contracts_composable_NftfiBundler_-slither

Figure 14: contracts/composable/contracts_composable_ERC998TopDown_-slither

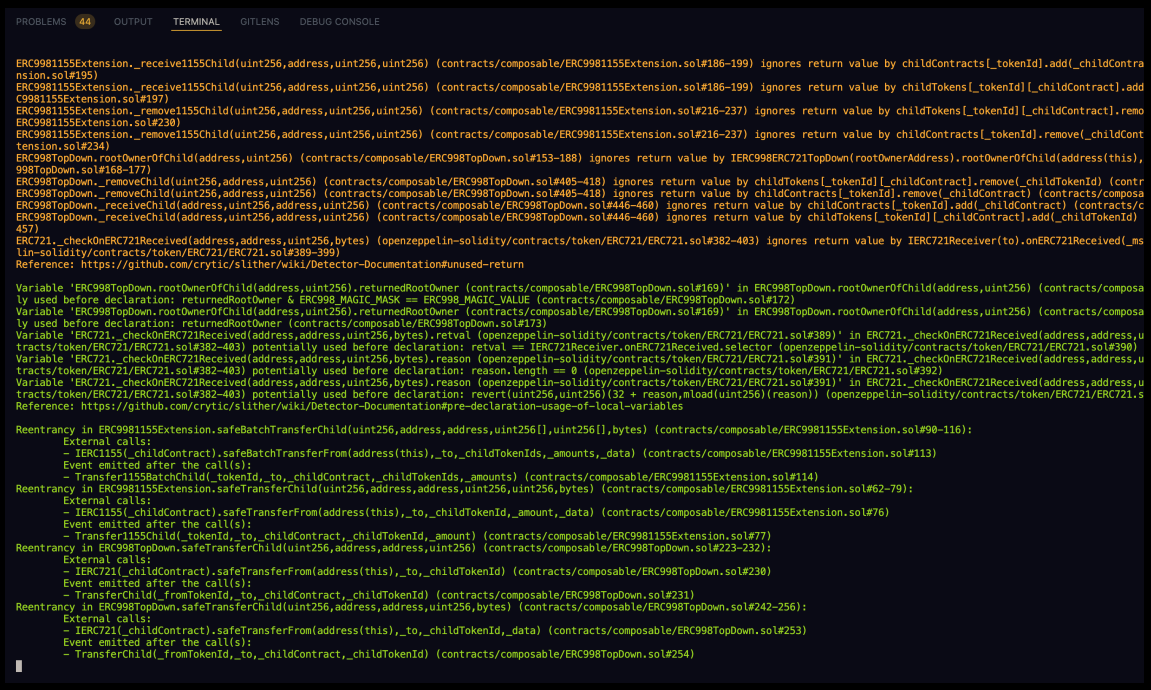


Figure 15: contracts/composable/contracts_composable_-ERC9981155Extension_slither

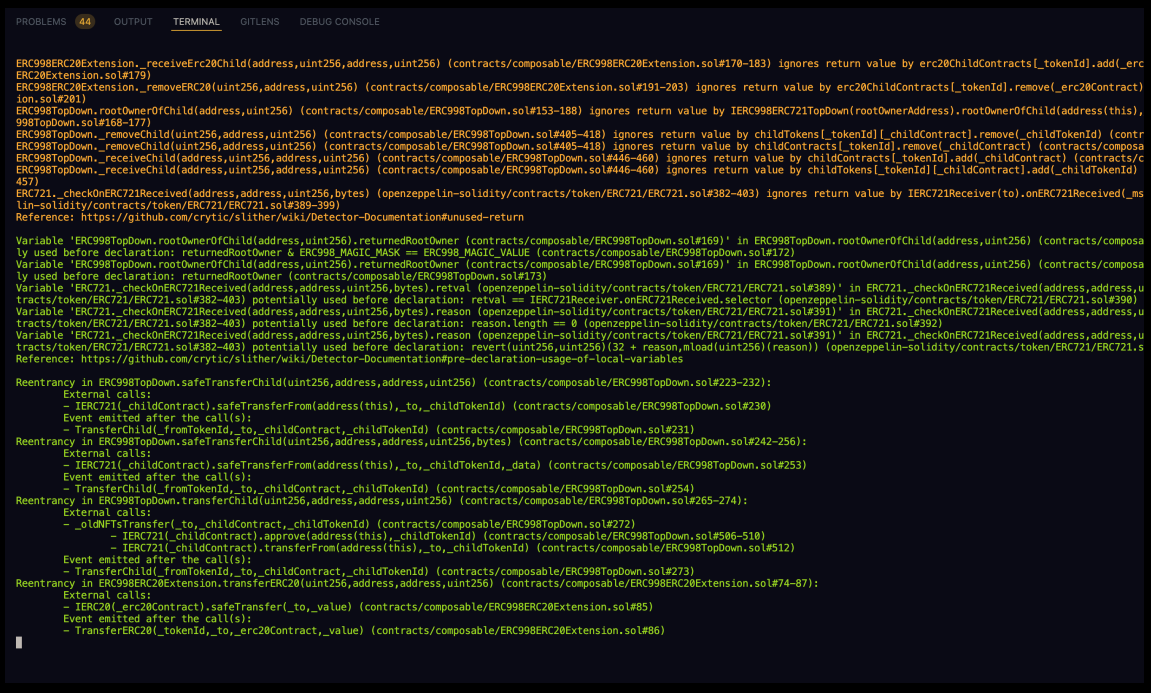


Figure 16: contracts/composable/contracts_composable_-ERC998ERC20Extension_slither

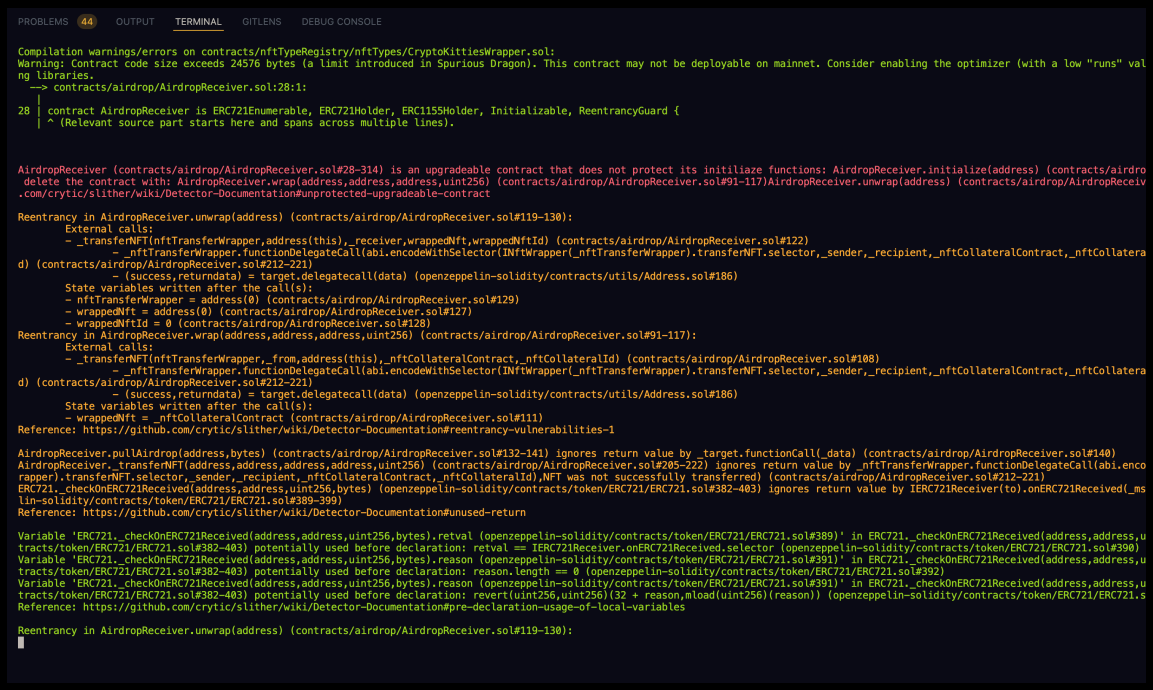


Figure 17: contracts/nftTypeRegistry/nftTypes/contracts_-nftTypeRegistry_nftTypes_CryptoKittiesWrapper_slither

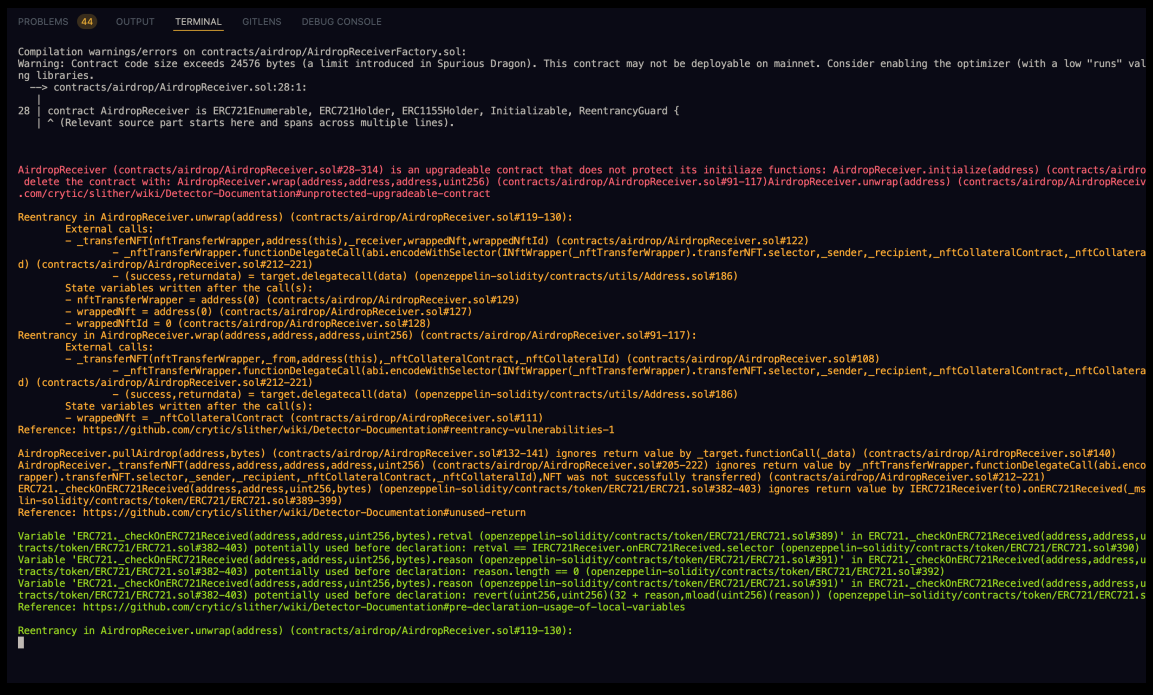


Figure 18: contracts/airdrop/contracts_airdrop_AirdropReceiverFactory_-slither

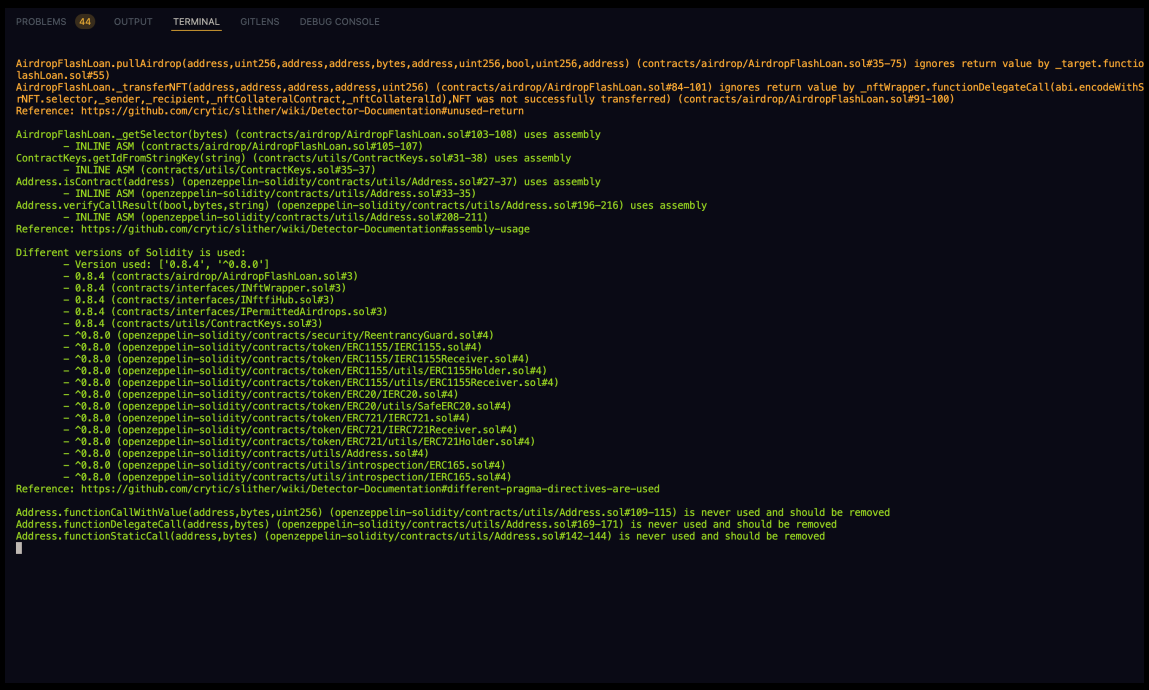


Figure 19: contracts/airdrop/contracts_airdrop_AirdropFlashLoan_slither

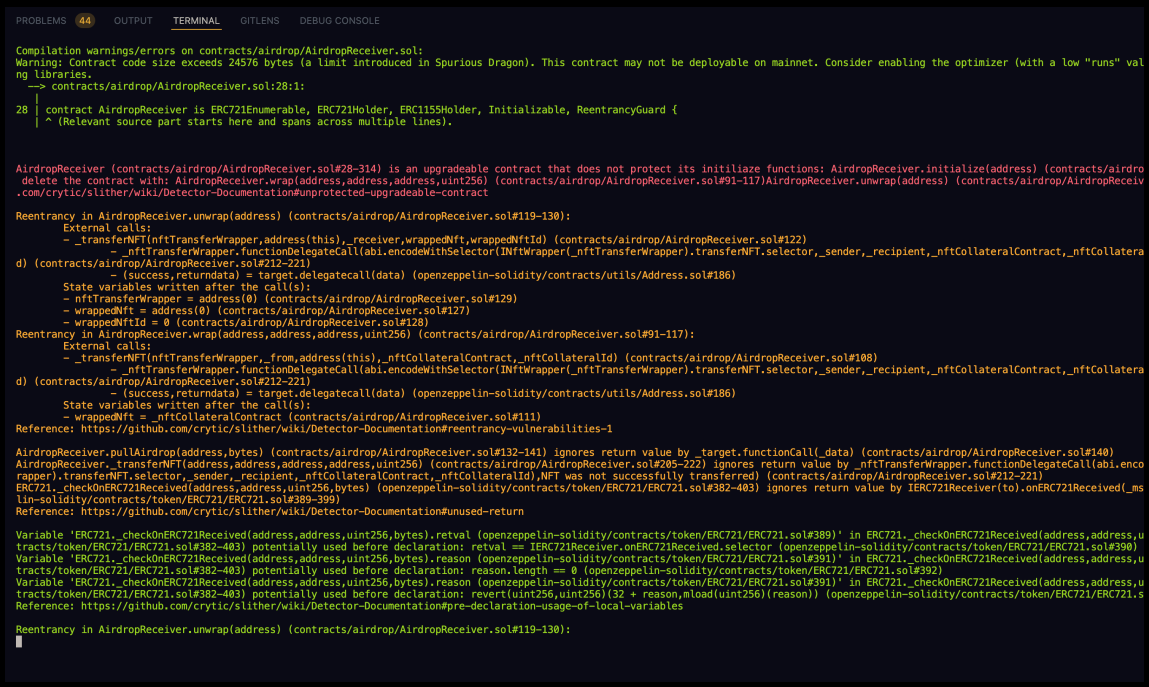


Figure 20: contracts/airdrop/contracts_airdrop_AirdropReceiver_slither

According to the test results, the findings found by these tools were considered false positives. All relevant findings were reviewed by the auditors, and relevant findings were addressed in the report as security

concerns.

THANK YOU FOR CHOOSING

// HALBORN