HALBORN

# Highstreet - 8bit

## Smart Contract Security Audit

Prepared by: **Halborn**
Date of Engagement: **August 30th, 2022 - September 6th, 2022**
Visit: **Halborn.com**

## DOCUMENT REVISION HISTORY

| VERSION | MODIFICATION | DATE | AUTHOR |
|---|---|---|---|
| 0.1 | Document Creation | 09/05/2022 | Francisco González |
| 0.2 | Draft Review | 09/05/2022 | Kubilay Onur Gungor |
| 0.3 | Draft Review | 09/06/2022 | 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 |
| Kubilay Onur Gungor | Halborn | Kubilay.Gungor@halborn.com |
| Francisco González | Halborn | Francisco.Villarejo@halborn.com |

# EXECUTIVE OVERVIEW

## 1.1 INTRODUCTION

Highstreet engaged Halborn to conduct a security audit on their smart contracts beginning on August 30th, 2022 and ending on September 06th, 2022. The security assessment was scoped to the smart contracts provided in the audit/8bit GitHub repository.

## 1.2 AUDIT SUMMARY

The team at Halborn was provided one week for the engagement and assigned one full-time security engineer to audit the security of the smart contract. The security engineers are blockchain and smart-contract security experts with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit to achieve the following:

- Ensure that all functions in the protocol smart contracts are intended.
- Identify potential security issues in Arcade bridge smart contracts.

In summary, Halborn identified many security risks that should be addressed by the HighStreet 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 hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Hardhat, 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.

| | | | | |
|---|---|---|---|---|
| CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL |

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

EXECUTIVE OVERVIEW

## 1.4 SCOPE

IN-SCOPE:

The security assessment was scoped to the following smart contracts included in audit/8bit GitHub repository:

- EightBit.sol
- EightBitMinter.sol
- MinterAccessControl.sol
- TimeLimit.sol

Commit ID:

- fb2eaba6c06703407286127ae1acf17bcabc5897

# 2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

| CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL |
|---|---|---|---|---|
| 0 | 0 | 2 | 0 | 8 |

LIKELIHOOD

IMPACT

| | | | | |
|---|---|---|---|---|
| 🟨 | 🟨 | 🟥 | 🟥 | 🟪 |
| 🟩 | 🟨 (HAL-01) | 🟨 | 🟥 | 🟥 |
| 🟩 | 🟩 | 🟨 (HAL-02) | 🟨 | 🟥 |
| ⬜ | 🟩 | 🟩 | 🟨 | 🟨 |
| ⬜ (HAL-03)<br>(HAL-04)<br>(HAL-05)<br>(HAL-06)<br>(HAL-07)<br>(HAL-08)<br>(HAL-09)<br>(HAL-10) | ⬜ | 🟩 | 🟩 | 🟨 |

| SECURITY ANALYSIS | RISK LEVEL | REMEDIATION DATE |
|---|---|---|
| HAL01 - OWNER CAN RENOUNCE OWNERSHIP | Medium | - |
| HAL02 - LACK OF TRANSFEROWNERSHIP PATTERN | Medium | - |
| HAL03 - REENTRANCY IN EIGHTBIT.SOL | Informational | - |
| HAL04 - CHECKS-EFFECTS-INTERACTIONS PATTERN NOT FOLLOWED ON EIGHTBITMINTER.SOL CONTRACT | Informational | - |
| HAL05 - LACK OF PARAMETER LIMITS | Informational | - |
| HAL06 - END TIME CAN BE SMALLER THAN START TIME | Informational | - |
| HAL07 - USE ++I INSTEAD OF I++ IN LOOPS FOR GAS OPTIMIZATION | Informational | - |
| HAL08 - UNNEEDED INITIALIZATION OF UINT VARIABLES TO 0 | Informational | - |
| HAL09 - SOLC 0.8.9 COMPILER VERSION CONTAINS MULTIPLE BUGS | Informational | - |
| HAL10 - POSSIBLE MISUSE OF PUBLIC FUNCTIONS | Informational | - |

# FINDINGS & TECH DETAILS

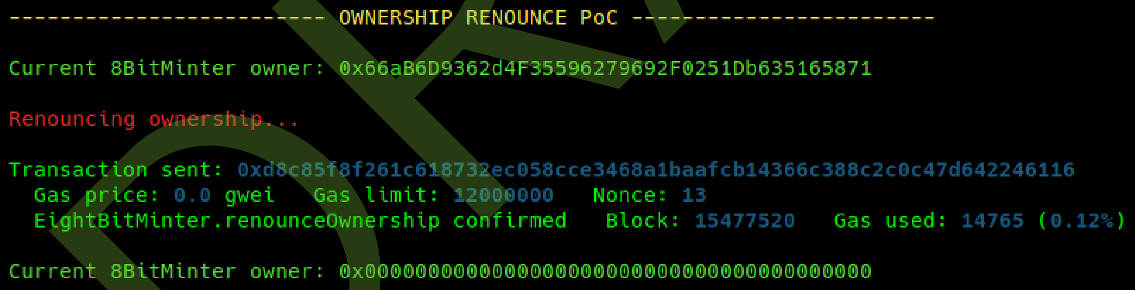
## 3.1 (HAL-01) OWNER CAN RENOUNCE OWNERSHIP - MEDIUM

### Description:

The Owner of the contract is usually the account that deploys the contract. As a result, the Owner can perform some privileged functions (such as increasing MaxSupply or modifying signers). In every smart contract in scope, ownership renounce functions can be used to renounce the Owner role. Renouncing ownership before transferring would result in the contract having no Owner, eliminating the ability to call privileged functions.

### Proof of Concept:

If renounceOwnership() is called in EightBit.sol or EightBitMinter.sol contracts, the ownership of the contract will be transferred to the zero address, rendering the functions containing the onlyOwner modifier unusable:



```
------------------------- OWNERSHIP RENOUNCE PoC ------------------------

Current 8BitMinter owner: 0x66aB6D9362d4F35596279692F0251Db635165871

Renouncing ownership...

Transaction sent: 0xd8c85f8f261c618732ec058cce3468a1baafcb14366c388c2c0c47d642246116
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 13
  EightBitMinter.renounceOwnership confirmed   Block: 15477520   Gas used: 14765 (0.12%)

Current 8BitMinter owner: 0x0000000000000000000000000000000000000000
```

### Recommendation:

It is recommended that the Owner cannot call renounceOwnership() without transferring the Ownership to another address. In addition, if a multi-signature wallet is used, calling the renounceOwnership function should be confirmed for two or more users.

## 3.2 (HAL-02) LACK OF TRANSFEROWNERSHIP PATTERN - MEDIUM

Description:

The current ownership transfer process for all the contracts inheriting from Ownable involves the current operator calling the transferOwnership() function, which will delegate the owner role to the specified address:

Listing 1: Ownable.sol

```solidity
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        _transferOwnership(newOwner);
    }
```

Suppose the nominated operator account is not valid. In that case, it is possible that the owner accidentally transfers ownership to an uncontrolled EOA account, losing access to all functions with the onlyOwner or similar modifiers.

Risk Level:

**Likelihood - 3**
**Impact - 3**

Recommendation:

It is recommended to implement a two-step process where the owner nominates an account, and the nominated account needs to call an acceptOwnerRole() function for the transfer of the ownership to fully succeed. This ensures the nominated EOA account is valid and active.

## 3.3 (HAL-03) REENTRANCY IN EIGHTBIT.SOL - INFORMATIONAL

Description:

Due to the usage of ERC721.sol functions _safeMint() and _safeTransfer() which introduce a callback to msgSender.onERC721Received, EightBit.safeMint() function is vulnerable to reentrancy, allowing any attacker to deploy a malicious contract including an initial call to EightBit.safeMint() and also implementing a function called onERC721Received() which would call again EightBit.safeMint().

However, because EightBit.safeMint() is protected by the onlyMinter modifier, which vastly reduces the attack surface, this finding has been lowered to Informative, since it may introduce unforeseen vulnerabilities if new minting mechanisms are implemented or if the code is reused.

Code Location:

Listing 2: ERC721.sol (Line 401)

```
394     function _checkOnERC721Received(
395         address from,
396         address to,
397         uint256 tokenId,
398         bytes memory data
399     ) private returns (bool) {
400         if (to.isContract()) {
401             try IERC721Receiver(to).onERC721Received(_msgSender(),
  ↳  from, tokenId, data) returns (bytes4 retval) {
402                 return retval == IERC721Receiver.onERC721Received.
  ↳ selector;
403             } catch (bytes memory reason) {
404                 if (reason.length == 0) {
405                     revert("ERC721: transfer to non ERC721Receiver
  ↳  implementer");
406                 } else {
407                     /// @solidity memory-safe-assembly
408                     assembly {
```

```
                        revert(add(32, reason), mload(reason))
                    }
                }
            }
        } else {
            return true;
        }
    }
```

Risk Level:

**Likelihood - 1**
**Impact - 1**

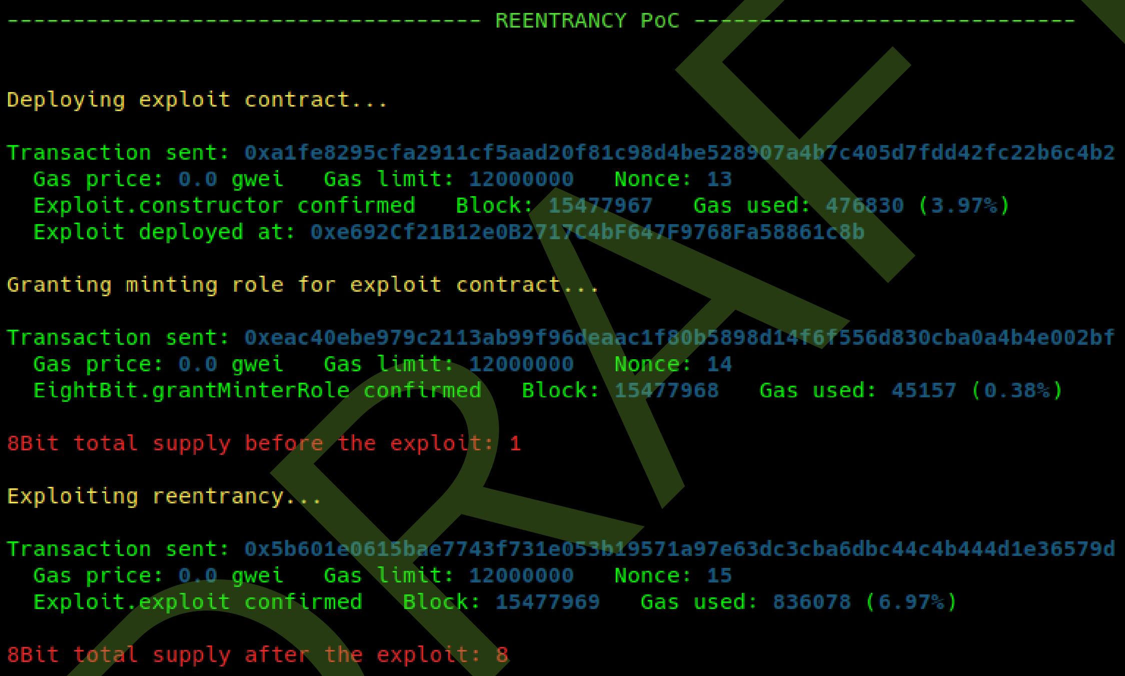
Proof Of Concept:

A malicious contract, reentrancy.sol, can be deployed:

Listing 3: Reentrancy.sol

```
    function _checkOnERC721Received(
        address from,
        address to,
        uint256 tokenId,
        bytes memory data
    ) private returns (bool) {
        if (to.isContract()) {
            try IERC721Receiver(to).onERC721Received(_msgSender(), from, tokenId, data) returns (bytes4 retval) {
                return retval == IERC721Receiver.onERC721Received.selector;
            } catch (bytes memory reason) {
                if (reason.length == 0) {
                    revert("ERC721: transfer to non ERC721Receiver implementer");
                } else {
                    /// @solidity memory-safe-assembly
                    assembly {
                        revert(add(32, reason), mload(reason))
                    }
                }
            }
        } else {
            return true;
        }
    }
```

When calling exploit() function, a new token will be minted calling EightBitContract.safeMint(), and when EightBit contract calls back to reentrancy.onERC721Received(), succesive calls to EightBitContract. safeMint() are performed, creating the reentrancy:



```
------------------------------------ REENTRANCY PoC -----------------------------


Deploying exploit contract...

Transaction sent: 0xa1fe8295cfa2911cf5aad20f81c98d4be528907a4b7c405d7fdd42fc22b6c4b2
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 13
  Exploit.constructor confirmed   Block: 15477967   Gas used: 476830 (3.97%)
  Exploit deployed at: 0xe692Cf21B12e0B2717C4bF647F9768Fa58861c8b

Granting minting role for exploit contract...

Transaction sent: 0xeac40ebe979c2113ab99f96deaac1f80b5898d14f6f556d830cba0a4b4e002bf
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 14
  EightBit.grantMinterRole confirmed   Block: 15477968   Gas used: 45157 (0.38%)

8Bit total supply before the exploit: 1

Exploiting reentrancy...

Transaction sent: 0x5b601e0615bae7743f731e053b19571a97e63dc3cba6dbc44c4b444d1e36579d
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 15
  Exploit.exploit confirmed   Block: 15477969   Gas used: 836078 (6.97%)

8Bit total supply after the exploit: 8
```

Recommendation:

Update the logic of sensitive functions to follow the **Checks-Effects-Interactions** pattern and use OpenZeppelin's ReentrancyGuard via the nonReentrant modifier.

## 3.4 (HAL-04) CHECKS-EFFECTS-INTERACTIONS PATTERN NOT FOLLOWED ON EIGHTBITMINTER.SOL CONTRACT - INFORMATIONAL

Description:

When minting a token using EightBitMinter.mint(), the remaining ETH sent as msg.value after paying for minting fees is returned. This call to _msgSender.transfer() could theoretically render the contract vulnerable to reentrancy if it were not for:

- A gas limit of 2300 is in place for transfer() calls.
- mint() function is protected with nonReentrant modifier.

It has been detected that the Check-Effects-Interactions pattern is not enforced in the mint() function, which is considered a best practice and could prevent the reentrancy at no additional cost if the two security measures commented on above, were not present. However, since actual measures are in place to prevent possible reentrancies, this finding has been lowered to Informative due to the use of the Check-Effects-Interaction pattern.

Code Location:

Listing 4: EightBitMinter.sol (Lines 211,212,237)

```
199   function mint(MintInput memory input_)
200     external
201     payable
202     nonReentrant
203     whenNotPaused
204     afterStartTime
205   {
206     bytes32 inputHash = keccak256(abi.encodePacked(input_.
  ↳ productCode, input_.styleTag));
```

```
    require(msg.value >= mintingFee, "Require payment fee");
    require(block.timestamp <= input_.deadline, "Execution exceed deadline");
    require(!checkOrderStatus(input_), "Minted already");
    _verifyInputSignature(input_);
    _mint(input_);
    orderMinted[inputHash] = true;
  }

  function _verifyInputSignature(MintInput memory input_) internal view {
    uint chainId;
    assembly {
      chainId := chainid()
    }
    require(input_.chainId == chainId, "Invalid network");
    bytes memory encodeData = abi.encode(input_.chainId, input_.user, input_.deadline , input_.productCode, input_.styleTag);
    bytes32 hash_ = keccak256(encodeData);
    bytes32 appendEthSignedMessageHash = ECDSA.toEthSignedMessageHash(hash_);
    address inputSigner = ECDSA.recover(appendEthSignedMessageHash, input_.v, input_.r, input_.s);
    require(signer == inputSigner, "Invalid signer");
  }

  function _mint(MintInput memory input_) internal {
    require(isValidTag(input_.styleTag), "mint exceeed token max");
    Tags memory Tag = styleTable[input_.styleTag];
    uint256 tagId = nextStyleId[input_.styleTag] + Tag.idLevel;
    eightBit.safeMint(input_.user, tagId);

    ++nextStyleId[input_.styleTag];
    uint256 refund = msg.value - mintingFee;
    if (refund > 0) {
      payable(_msgSender()).transfer(refund);
    }
    emit Mint(input_.user, input_.styleTag, input_.productCode, tagId, mintingFee);
  }
```

Risk Level:

**Likelihood - 1**
**Impact - 1**

Recommendation:

Update the logic of sensitive functions to follow the **Checks Effects Interactions** pattern, calling the _mint() function after the inputHash has been marked as minted, not before.

Following this pattern whenever possible is considered a best practice.

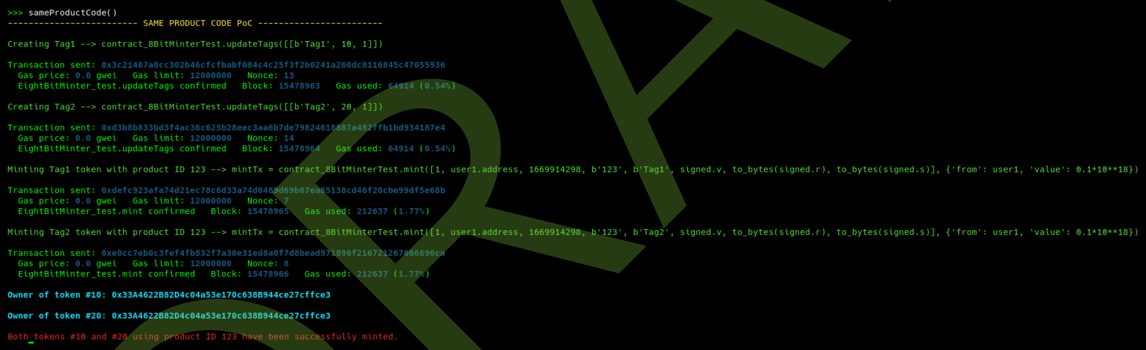
## 3.5 (HAL-05) LACK OF PARAMETER LIMITS - INFORMATIONAL

### Description:

It has been detected that some parameter-modifying functions do not have logical limits. This may cause the contract to function with parameter values that, although allowed, make no sense in the application context or might leave the application in an inconsistent state, which might cause various problems or even render the contract unusable.

### Proof Of Concept - Instance 1:

If signed, it is possible to have more than one token minted using the same productCode value, as long as they do not belong to the same styleTag:

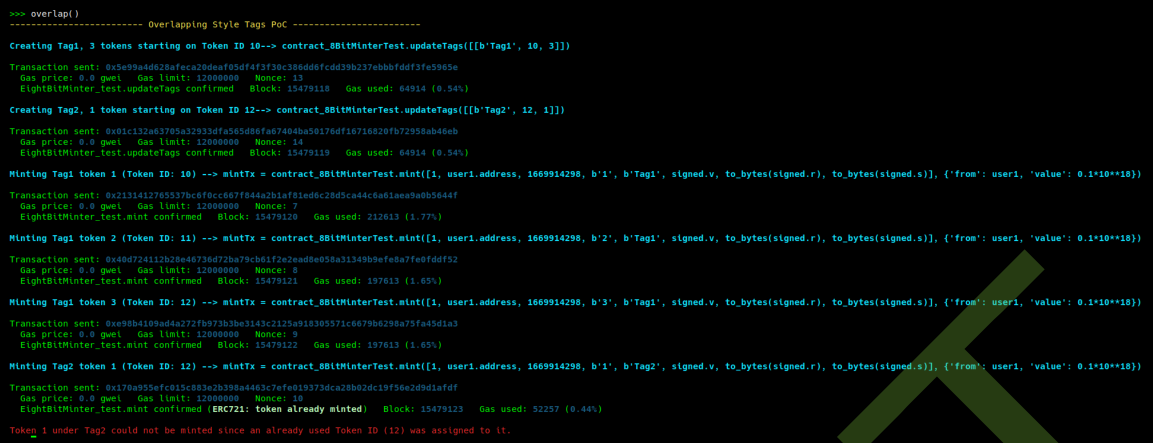


```
>>> sameProductCode()
------------------------- SAME PRODUCT CODE PoC -------------------------

Creating Tag1 --> contract_8BitMinterTest.updateTags([[b'Tag1', 10, 1]])

Transaction sent: 0x3c21487a8cc302b46cfcfbabf084c4c25f3f2b0241a260dc8116845c47055936
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 13
  EightBitMinter_test.updateTags confirmed   Block: 15478963   Gas used: 64914 (0.54%)

Creating Tag2 --> contract_8BitMinterTest.updateTags([[b'Tag2', 20, 1]])

Transaction sent: 0xd3b8b833bd3f4ac36c625b28eec3aa6b7de79824618887a482ffb1bd934187e4
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 14
  EightBitMinter_test.updateTags confirmed   Block: 15478964   Gas used: 64914 (0.54%)

Minting Tag1 token with product ID 123 --> mintTx = contract_8BitMinterTest.mint([1, user1.address, 1669914298, b'123', b'Tag1', signed.v, to_bytes(signed.r), to_bytes(signed.s)], {'from': user1, 'value': 0.1*10**18})

Transaction sent: 0xdefc923afa74d21ec78c6d33a74d0489d69b87ea85138cd46f20cbe99df5e68b
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 7
  EightBitMinter_test.mint confirmed   Block: 15478965   Gas used: 212637 (1.77%)

Minting Tag2 token with product ID 123 --> mintTx = contract_8BitMinterTest.mint([1, user1.address, 1669914298, b'123', b'Tag2', signed.v, to_bytes(signed.r), to_bytes(signed.s)], {'from': user1, 'value': 0.1*10**18})

Transaction sent: 0xe0cc7eb0c3fef4fb832f7a30e31ed8a0f7d8bead971896f216721267866696ca
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 8
  EightBitMinter_test.mint confirmed   Block: 15478966   Gas used: 212637 (1.77%)

Owner of token #10: 0x33A4622B82D4c04a53e170c638B944ce27cffce3

Owner of token #20: 0x33A4622B82D4c04a53e170c638B944ce27cffce3

Both tokens #10 and #20 using product ID 123 have been successfully minted.
```

This could leave the application in an inconsistent state if:
- productCode can take any arbitrary value and is not properly validated when signing the mint request (off-chain).
- Tokens are queried off-chain by their productCode values.

### Proof Of Concept - Instance 2:

If tags included in styleTable are not properly structured, two different tokens under different styleTag could try to be minted under the same tokenID:



```
>>> overlap()
------------------------ Overlapping Style Tags PoC ------------------------

Creating Tag1, 3 tokens starting on Token ID 10--> contract_8BitMinterTest.updateTags([[b'Tag1', 10, 3]])

Transaction sent: 0x5e99a4d628afeca20deaf05df4f3f30c386dd6fcdd39b237ebbbfddf3fe5965e
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 13
  EightBitMinter_test.updateTags confirmed   Block: 15479118   Gas used: 64914 (0.54%)

Creating Tag2, 1 token starting on Token ID 12--> contract_8BitMinterTest.updateTags([[b'Tag2', 12, 1]])

Transaction sent: 0x01c132a63705a32933dfa565d86fa67404ba50176df16716820fb72958ab46eb
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 14
  EightBitMinter_test.updateTags confirmed   Block: 15479119   Gas used: 64914 (0.54%)

Minting Tag1 token 1 (Token ID: 10) --> mintTx = contract_8BitMinterTest.mint([1, user1.address, 1669914298, b'1', b'Tag1', signed.v, to_bytes(signed.r), to_bytes(signed.s)], {'from': user1, 'value': 0.1*10**18})

Transaction sent: 0x2131412765537bc6f0cc667f844a2b1af81ed6c28d5ca44c6a61aea9a0b5644f
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 7
  EightBitMinter_test.mint confirmed   Block: 15479120   Gas used: 212613 (1.77%)

Minting Tag1 token 2 (Token ID: 11) --> mintTx = contract_8BitMinterTest.mint([1, user1.address, 1669914298, b'2', b'Tag1', signed.v, to_bytes(signed.r), to_bytes(signed.s)], {'from': user1, 'value': 0.1*10**18})

Transaction sent: 0x40d724112b28e46736d72ba79cb61f2e2ead8e058a31349b9efe8a7fe0fddf52
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 8
  EightBitMinter_test.mint confirmed   Block: 15479121   Gas used: 197613 (1.65%)

Minting Tag1 token 3 (Token ID: 12) --> mintTx = contract_8BitMinterTest.mint([1, user1.address, 1669914298, b'3', b'Tag1', signed.v, to_bytes(signed.r), to_bytes(signed.s)], {'from': user1, 'value': 0.1*10**18})

Transaction sent: 0xe98b4109ad4a272fb973b3be3143c2125a918305571c6679b6298a75fa45d1a3
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 9
  EightBitMinter_test.mint confirmed   Block: 15479122   Gas used: 197613 (1.65%)

Minting Tag2 token 1 (Token ID: 12) --> mintTx = contract_8BitMinterTest.mint([1, user1.address, 1669914298, b'1', b'Tag2', signed.v, to_bytes(signed.r), to_bytes(signed.s)], {'from': user1, 'value': 0.1*10**18})

Transaction sent: 0x170a955efc015c883e2b398a4463c7efe019373dca28b02dc19f56e2d9d1afdf
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 10
  EightBitMinter_test.mint confirmed (ERC721: token already minted)   Block: 15479123   Gas used: 52257 (0.44%)

Token 1 under Tag2 could not be minted since an already used Token ID (12) was assigned to it.
```

If inconsistencies in styleTags are introduced when loading them into the contract, some tokens would remain unmintable since another token using the same tokenID has already been minted.

### Risk Level:

**Likelihood - 1**
**Impact - 1**

### Recommendation:

It is recommended to enforce logical value limits for critical parameters and check for additional occurrences of this same vulnerability.

However, neither of the instances described above have to be necessarily checked on-chain, as long as productCode values are validated before mint requests are signed for instance 1, and styleTags integrity is checked before they get loaded on-chain using EightBitMinter.updateTags() for instance 2, since performing these checks off-chain will be cheaper, avoiding gas costs.

## 3.6 (HAL-06) ENDTIME CAN BE SMALLER THAN STARTTIME - INFORMATIONAL

Description:

Although the endTime parameter is currently not utilized in the contract, the function _updateEndTime does not check that the endTime is greater than the startTime variable. This could potentially cause unforeseen issues in the future.

Code Location:

Listing 5: TimeLimit.sol (Line 42)

```
41
42      function _updateEndTime(uint256 endTime_, address operator_)
   ↳ internal {
43          endTime = endTime_;
44          emit UpdateEndTime(endTime, operator_);
45      }
```

Risk Level:

**Likelihood - 1**
**Impact - 1**

Recommendation:

Consider using a required statement to enforce the _endTime input parameter to be greater than the startTime parameter.

# 3.7 (HAL-07) USE ++I INSTEAD OF I++ IN LOOPS FOR GAS OPTIMIZATION - INFORMATIONAL

Description:

In the loop within safeBatchTransferFrom() function in EightBit.sol contract, the variable index is incremented using index++. It is known that, in loops, using ++index costs less gas per iteration than index++. This also affects variables incremented inside the loop code block.

The same pattern has been detected in updateTags() function within EightBitMinter.sol contract.

Code Location:

Listing 6: EightBit.sol (Line 124)

```
119   function safeBatchTransferFrom(
120     address from_,
121     address to_,
122     uint256[] memory ids_
123   ) public virtual {
124     for(uint256 index; index < ids_.length; index++) {
125       safeTransferFrom(from_, to_, ids_[index]);
126     }
127   }
```

Listing 7: EightBitMinter.sol (Line 124)

```
123   function updateTags(Tags[] memory tagLists_) external onlyOwner
    ↳ {
124     for(uint i = 0; i < tagLists_.length; i++) {
125       Tags memory Tag = tagLists_[i];
126       styleTable[Tag.name] = Tag;
127     }
128   }
```

Recommendation:

It is recommended to use ++i instead of i++ to increment the value of a uint variable inside a loop. This also applies to the variables declared inside the for loop, not just the iterator. On the other hand, this is not applicable outside of loops.

## 3.8 (HAL-08) UNNEEDED INITIALIZATION OF UINT VARIABLES TO 0 - INFORMATIONAL

Description:

Any variable of type uint is already initialized to 0. uint i = 0 reassigns the 0 to i, which wastes gas.

Code Location:

**Listing 8: EightBitMinter.sol (Line 124)**

```
123   function updateTags(Tags[] memory tagLists_) external onlyOwner
      ↳ {
124     for(uint i = 0; i < tagLists_.length; i++) {
125       Tags memory Tag = tagLists_[i];
126       styleTable[Tag.name] = Tag;
127     }
128   }
```

Risk Level:

**Likelihood - 1**
**Impact - 1**

Recommendation:

It is recommended not to initialize uint variables to 0 to save some gas. For example, use instead:

```
for(uint i; i < tagLists_.length; ++i){
```

## 3.9 (HAL-09) SOLC 0.8.9 COMPILER VERSION CONTAINS MULTIPLE BUGS - INFORMATIONAL

Description:

The scoped contracts have configured the fixed pragma set to 0.8.9. The latest solidity compiler version, 0.8.16, fixed important bugs in the compiler along with new native protections. Version 0.8.9 is missing all these fixes: 0.8.10, 0.8.11, 0.8.12, 0.8.13, 0.8.14, 0.8.15, 0.8.16.

The official Solidity recommendations are: when deploying contracts, the latest released version of Solidity should be used. Apart from exceptional cases, only the latest version receives security fixes.

Risk Level:

**Likelihood - 1**
**Impact - 1**

Recommendation:

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

## 3.10 (HAL-10) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL

Description:

In public functions, array arguments are immediately copied to memory, while external functions can read directly from calldata. Reading calldata is cheaper than memory allocation. Public functions need to write the arguments to memory because public functions may be called internally. Internal calls are passed internally by pointers to memory. Thus, the function expects its arguments to be located in memory when the compiler generates the code for an internal function.

Code Location:

Consider marking the below functions as external instead of the public if they will never be directly called within the same contract or in any of their descendants:

**EightBit.sol**:
burn, safeBatchTransferFrom

**ParameterStore.sol**:
updateStartTime

Risk Level:

**Likelihood - 1**
**Impact - 1**

Recommendation:

Consider as much as possible declaring external variables instead of public variables. As for best practices, you should use external if you expect the function will only ever be called externally, and use public if you need to call the function internally. Mainly, marking both functions as external can save gas.

# AUTOMATED TESTING

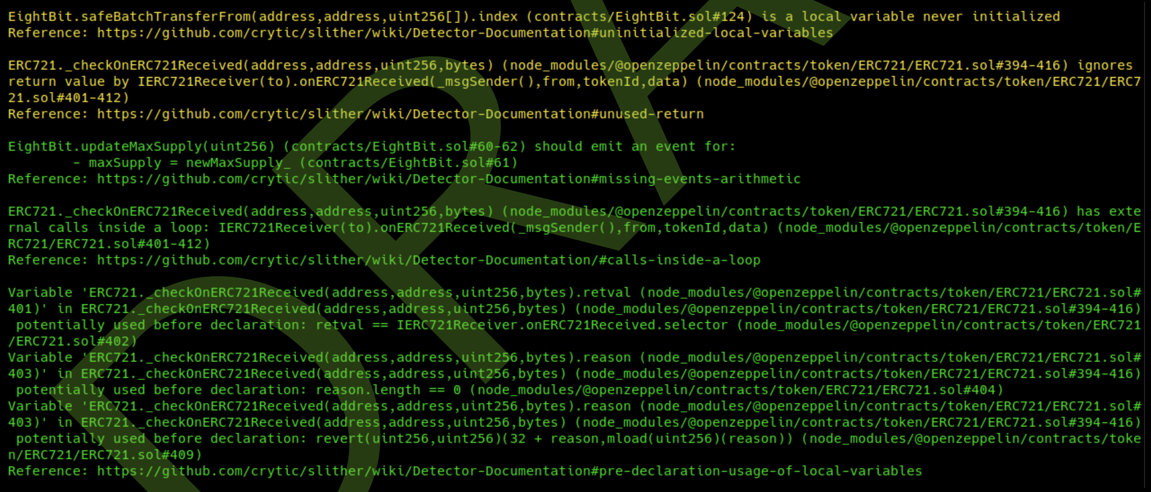
## 4.1 STATIC ANALYSIS REPORT

Description:

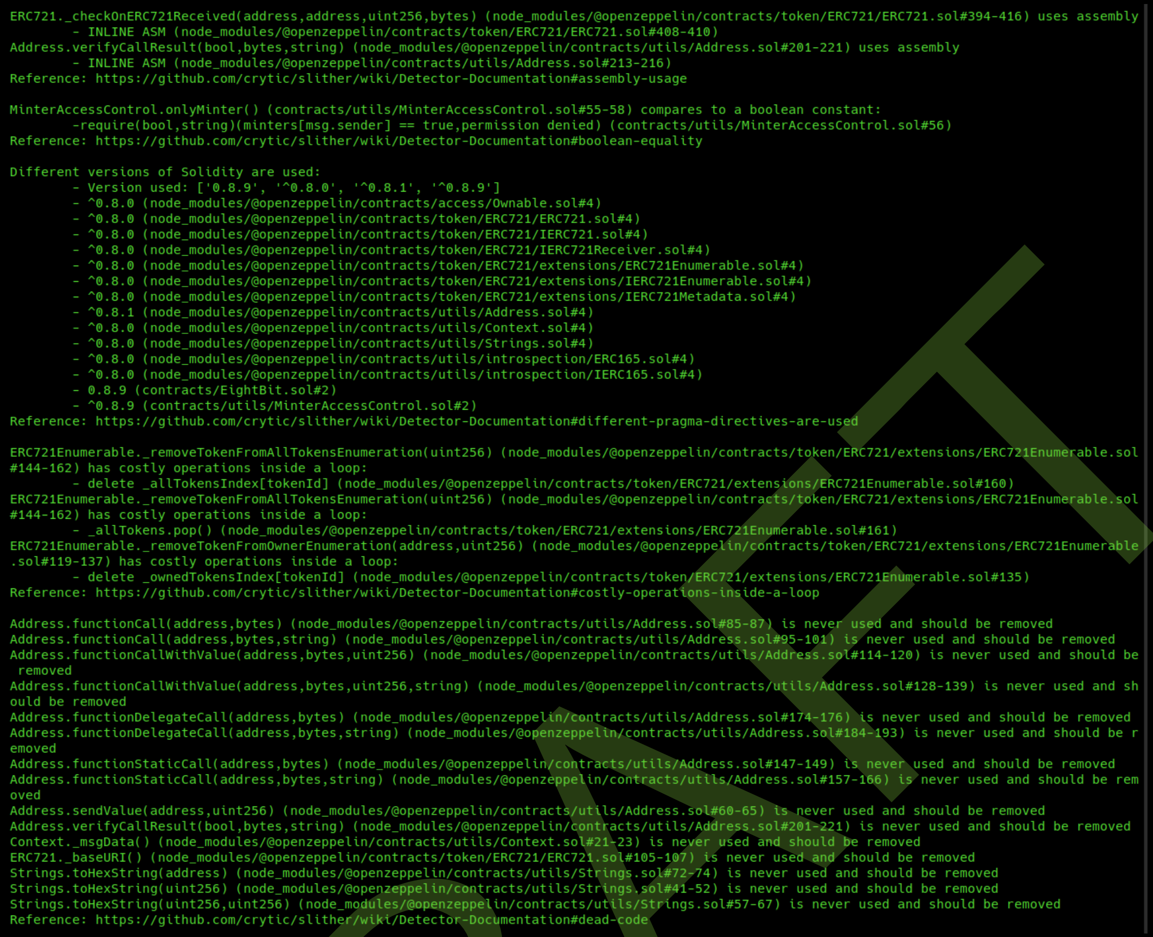
Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their abis 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.

Slither results:

EightBit.sol

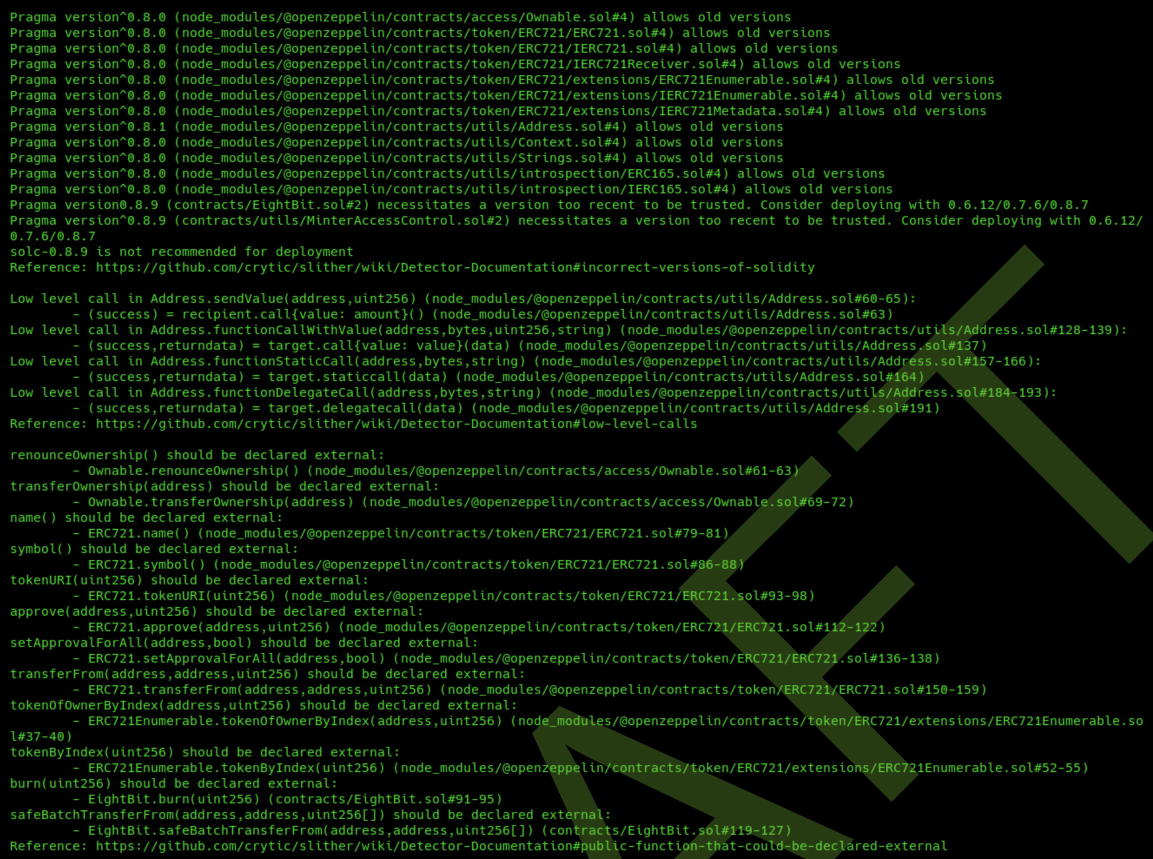


```
EightBit.safeBatchTransferFrom(address,address,uint256[]).index (contracts/EightBit.sol#124) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#394-416) ignores
return value by IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,data) (node_modules/@openzeppelin/contracts/token/ERC721/ERC7
21.sol#401-412)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

EightBit.updateMaxSupply(uint256) (contracts/EightBit.sol#60-62) should emit an event for:
        - maxSupply = newMaxSupply_ (contracts/EightBit.sol#61)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#394-416) has exte
rnal calls inside a loop: IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,data) (node_modules/@openzeppelin/contracts/token/E
RC721/ERC721.sol#401-412)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).retval (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#
401)' in ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#394-416)
 potentially used before declaration: retval == IERC721Receiver.onERC721Received.selector (node_modules/@openzeppelin/contracts/token/ERC721
/ERC721.sol#402)
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).reason (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#
403)' in ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#394-416)
 potentially used before declaration: reason.length == 0 (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#404)
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).reason (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#
403)' in ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#394-416)
 potentially used before declaration: revert(uint256,uint256)(32 + reason,mload(uint256)(reason)) (node_modules/@openzeppelin/contracts/toke
n/ERC721/ERC721.sol#409)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables
```



```
ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#394-416) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#408-410)
Address.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#201-221) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#213-216)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

MinterAccessControl.onlyMinter() (contracts/utils/MinterAccessControl.sol#55-58) compares to a boolean constant:
        -require(bool,string)(minters[msg.sender] == true,permission denied) (contracts/utils/MinterAccessControl.sol#56)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality

Different versions of Solidity are used:
        - Version used: ['0.8.9', '^0.8.0', '^0.8.1', '^0.8.9']
        - ^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Enumerable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#4)
        - ^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/ERC165.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/IERC165.sol#4)
        - 0.8.9 (contracts/EightBit.sol#2)
        - ^0.8.9 (contracts/utils/MinterAccessControl.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

ERC721Enumerable._removeTokenFromAllTokensEnumeration(uint256) (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol
#144-162) has costly operations inside a loop:
        - delete _allTokensIndex[tokenId] (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol#160)
ERC721Enumerable._removeTokenFromAllTokensEnumeration(uint256) (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol
#144-162) has costly operations inside a loop:
        - _allTokens.pop() (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol#161)
ERC721Enumerable._removeTokenFromOwnerEnumeration(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable
.sol#119-137) has costly operations inside a loop:
        - delete _ownedTokensIndex[tokenId] (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol#135)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop

Address.functionCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#85-87) is never used and should be removed
Address.functionCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#95-101) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#114-120) is never used and should be
 removed
Address.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#128-139) is never used and sh
ould be removed
Address.functionDelegateCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#174-176) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#184-193) is never used and should be r
emoved
Address.functionStaticCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#147-149) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#157-166) is never used and should be rem
oved
Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#60-65) is never used and should be removed
Address.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#201-221) is never used and should be removed
Context._msgData() (node_modules/@openzeppelin/contracts/utils/Context.sol#21-23) is never used and should be removed
ERC721._baseURI() (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#105-107) is never used and should be removed
Strings.toHexString(address) (node_modules/@openzeppelin/contracts/utils/Strings.sol#72-74) is never used and should be removed
Strings.toHexString(uint256) (node_modules/@openzeppelin/contracts/utils/Strings.sol#41-52) is never used and should be removed
Strings.toHexString(uint256,uint256) (node_modules/@openzeppelin/contracts/utils/Strings.sol#57-67) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

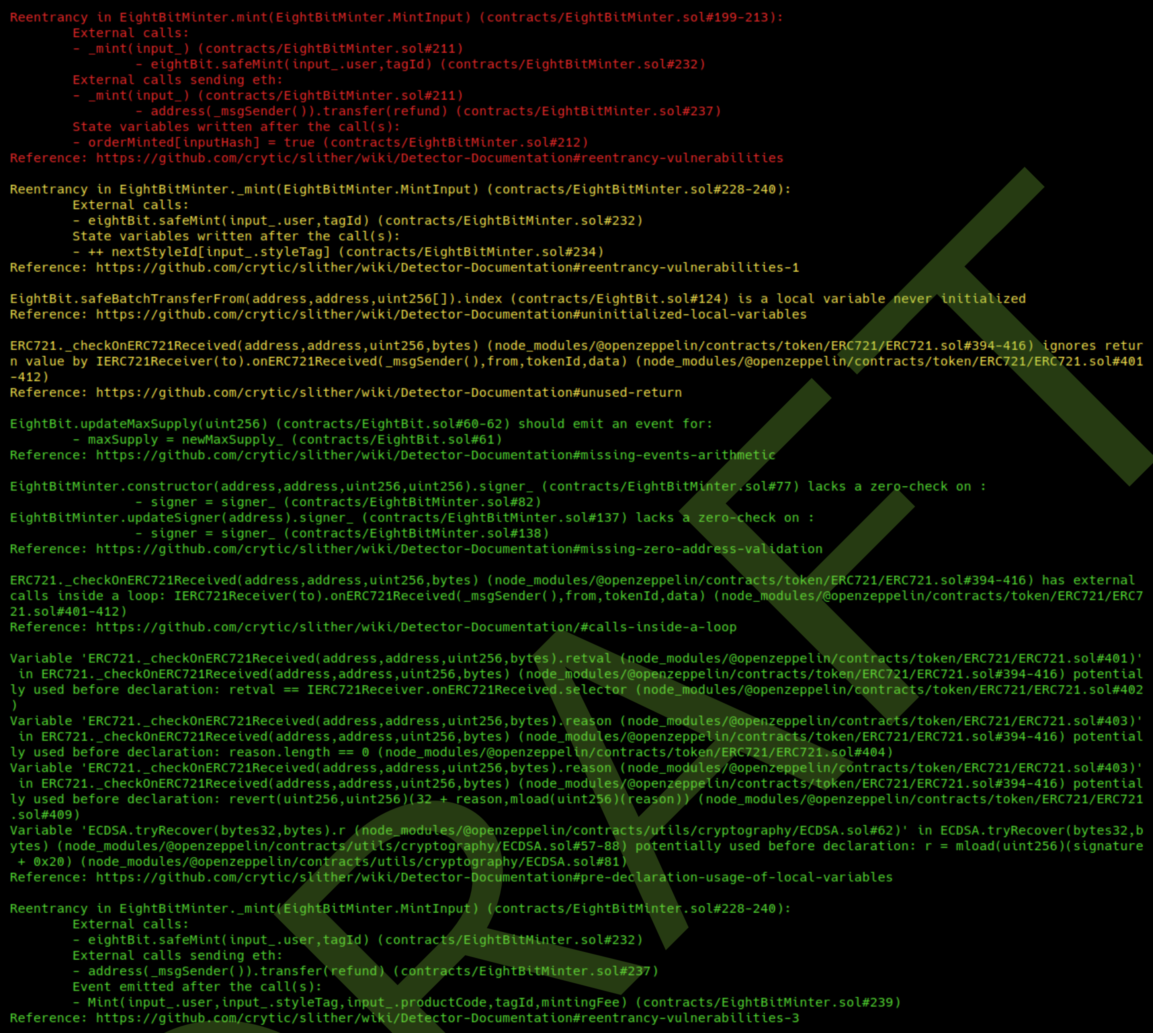
AUTOMATED TESTING



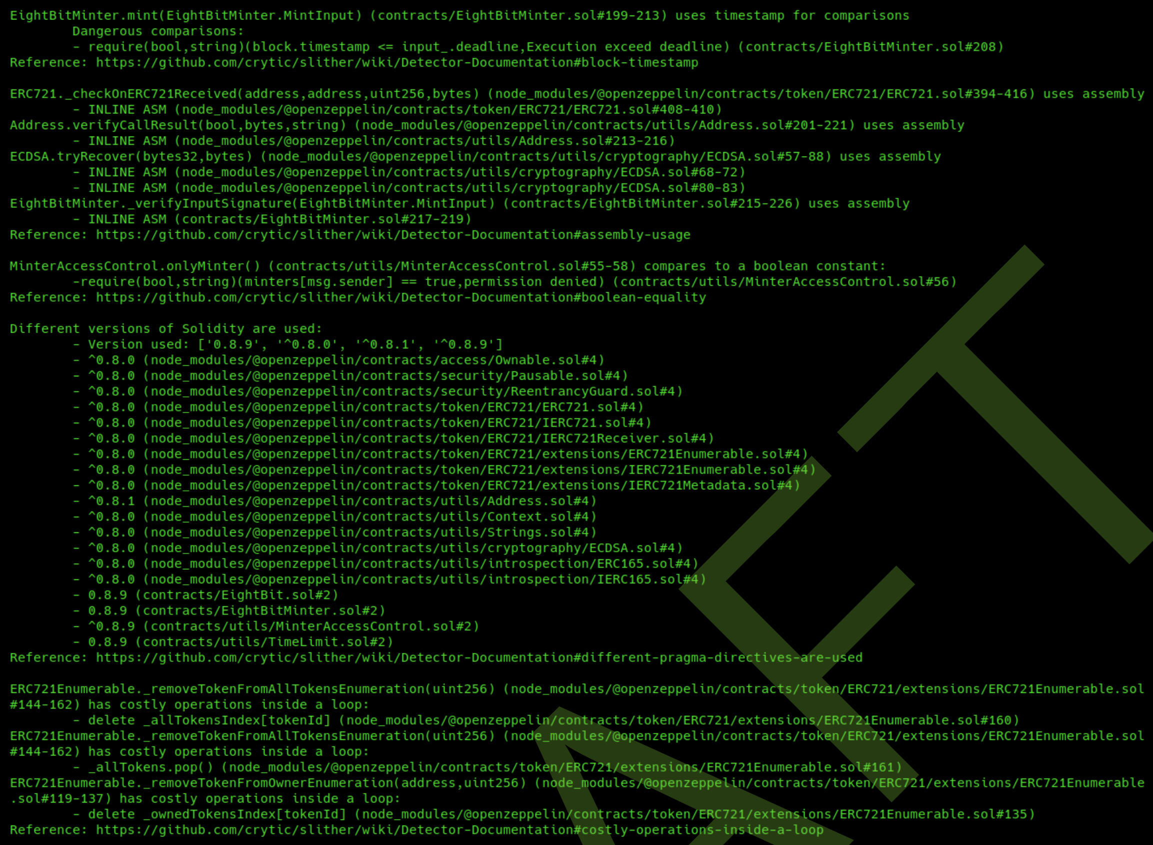
```
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Enumerable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#4) allows old versions
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/ERC165.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/IERC165.sol#4) allows old versions
Pragma version0.8.9 (contracts/EightBit.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version^0.8.9 (contracts/utils/MinterAccessControl.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/
0.7.6/0.8.7
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#60-65):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts/utils/Address.sol#63)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#128-139):
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
Low level call in Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#157-166):
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#164)
Low level call in Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#184-193):
        - (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#191)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (node_modules/@openzeppelin/contracts/access/Ownable.sol#61-63)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (node_modules/@openzeppelin/contracts/access/Ownable.sol#69-72)
name() should be declared external:
        - ERC721.name() (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#79-81)
symbol() should be declared external:
        - ERC721.symbol() (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#86-88)
tokenURI(uint256) should be declared external:
        - ERC721.tokenURI(uint256) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#93-98)
approve(address,uint256) should be declared external:
        - ERC721.approve(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#112-122)
setApprovalForAll(address,bool) should be declared external:
        - ERC721.setApprovalForAll(address,bool) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#136-138)
transferFrom(address,address,uint256) should be declared external:
        - ERC721.transferFrom(address,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#150-159)
tokenOfOwnerByIndex(address,uint256) should be declared external:
        - ERC721Enumerable.tokenOfOwnerByIndex(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.so
l#37-40)
tokenByIndex(uint256) should be declared external:
        - ERC721Enumerable.tokenByIndex(uint256) (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol#52-55)
burn(uint256) should be declared external:
        - EightBit.burn(uint256) (contracts/EightBit.sol#91-95)
safeBatchTransferFrom(address,address,uint256[]) should be declared external:
        - EightBit.safeBatchTransferFrom(address,address,uint256[]) (contracts/EightBit.sol#119-127)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

AUTOMATED TESTING

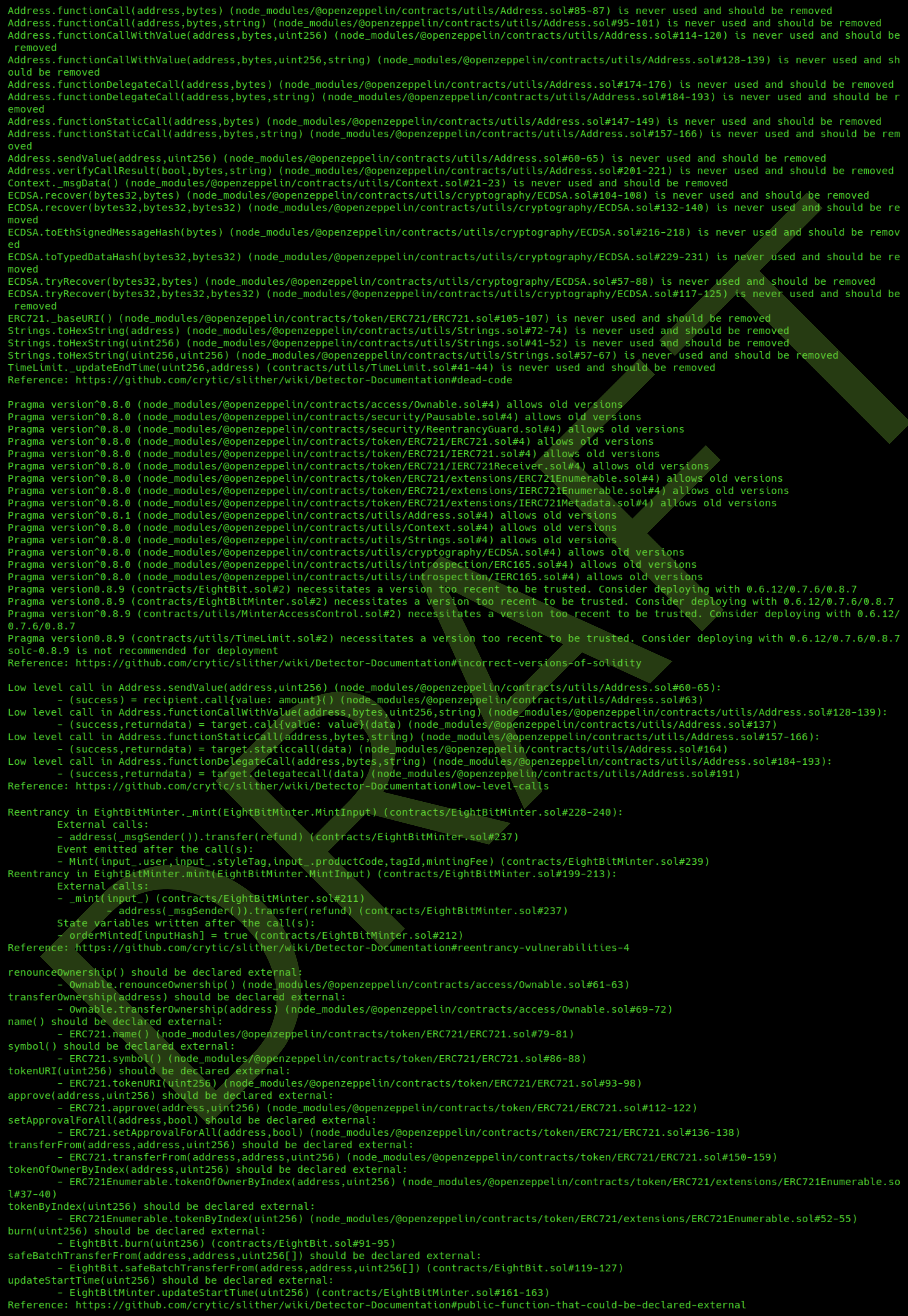
## EightBitMinter



```
Reentrancy in EightBitMinter.mint(EightBitMinter.MintInput) (contracts/EightBitMinter.sol#199-213):
        External calls:
        - _mint(input_) (contracts/EightBitMinter.sol#211)
                - eightBit.safeMint(input_.user,tagId) (contracts/EightBitMinter.sol#232)
        External calls sending eth:
        - _mint(input_) (contracts/EightBitMinter.sol#211)
                - address(_msgSender()).transfer(refund) (contracts/EightBitMinter.sol#237)
        State variables written after the call(s):
        - orderMinted[inputHash] = true (contracts/EightBitMinter.sol#212)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities

Reentrancy in EightBitMinter._mint(EightBitMinter.MintInput) (contracts/EightBitMinter.sol#228-240):
        External calls:
        - eightBit.safeMint(input_.user,tagId) (contracts/EightBitMinter.sol#232)
        State variables written after the call(s):
        - ++ nextStyleId[input_.styleTag] (contracts/EightBitMinter.sol#234)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

EightBit.safeBatchTransferFrom(address,address,uint256[]).index (contracts/EightBit.sol#124) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#394-416) ignores retur
n value by IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,data) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#401
-412)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

EightBit.updateMaxSupply(uint256) (contracts/EightBit.sol#60-62) should emit an event for:
        - maxSupply = newMaxSupply_ (contracts/EightBit.sol#61)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

EightBitMinter.constructor(address,address,uint256,uint256).signer_ (contracts/EightBitMinter.sol#77) lacks a zero-check on :
                - signer = signer_ (contracts/EightBitMinter.sol#82)
EightBitMinter.updateSigner(address).signer_ (contracts/EightBitMinter.sol#137) lacks a zero-check on :
                - signer = signer_ (contracts/EightBitMinter.sol#138)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#394-416) has external
calls inside a loop: IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,data) (node_modules/@openzeppelin/contracts/token/ERC721/ERC7
21.sol#401-412)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).retval (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#401)'
 in ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#394-416) potential
ly used before declaration: retval == IERC721Receiver.onERC721Received.selector (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#402
)
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).reason (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#403)'
 in ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#394-416) potential
ly used before declaration: reason.length == 0 (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#404)
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).reason (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#403)'
 in ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#394-416) potential
ly used before declaration: revert(uint256,uint256)(32 + reason,mload(uint256)(reason)) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721
.sol#409)
Variable 'ECDSA.tryRecover(bytes32,bytes).r (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#62)' in ECDSA.tryRecover(bytes32,b
ytes) (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#57-88) potentially used before declaration: r = mload(uint256)(signature
 + 0x20) (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#81)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables

Reentrancy in EightBitMinter._mint(EightBitMinter.MintInput) (contracts/EightBitMinter.sol#228-240):
        External calls:
        - eightBit.safeMint(input_.user,tagId) (contracts/EightBitMinter.sol#232)
        External calls sending eth:
        - address(_msgSender()).transfer(refund) (contracts/EightBitMinter.sol#237)
        Event emitted after the call(s):
        - Mint(input_.user,input_.styleTag,input_.productCode,tagId,mintingFee) (contracts/EightBitMinter.sol#239)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```



```
EightBitMinter.mint(EightBitMinter.MintInput) (contracts/EightBitMinter.sol#199-213) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= input_.deadline,Execution exceed deadline) (contracts/EightBitMinter.sol#208)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#394-416) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#408-410)
Address.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#201-221) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#213-216)
ECDSA.tryRecover(bytes32,bytes) (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#57-88) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#68-72)
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#80-83)
EightBitMinter._verifyInputSignature(EightBitMinter.MintInput) (contracts/EightBitMinter.sol#215-226) uses assembly
        - INLINE ASM (contracts/EightBitMinter.sol#217-219)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

MinterAccessControl.onlyMinter() (contracts/utils/MinterAccessControl.sol#55-58) compares to a boolean constant:
        -require(bool,string)(minters[msg.sender] == true,permission denied) (contracts/utils/MinterAccessControl.sol#56)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality

Different versions of Solidity are used:
        - Version used: ['0.8.9', '^0.8.0', '^0.8.1', '^0.8.9']
        - ^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/security/Pausable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/security/ReentrancyGuard.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Enumerable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#4)
        - ^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/ERC165.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/IERC165.sol#4)
        - 0.8.9 (contracts/EightBit.sol#2)
        - 0.8.9 (contracts/EightBitMinter.sol#2)
        - ^0.8.9 (contracts/utils/MinterAccessControl.sol#2)
        - 0.8.9 (contracts/utils/TimeLimit.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

ERC721Enumerable._removeTokenFromAllTokensEnumeration(uint256) (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol
#144-162) has costly operations inside a loop:
        - delete _allTokensIndex[tokenId] (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol#160)
ERC721Enumerable._removeTokenFromAllTokensEnumeration(uint256) (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol
#144-162) has costly operations inside a loop:
        - _allTokens.pop() (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol#161)
ERC721Enumerable._removeTokenFromOwnerEnumeration(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable
.sol#119-137) has costly operations inside a loop:
        - delete _ownedTokensIndex[tokenId] (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol#135)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
```

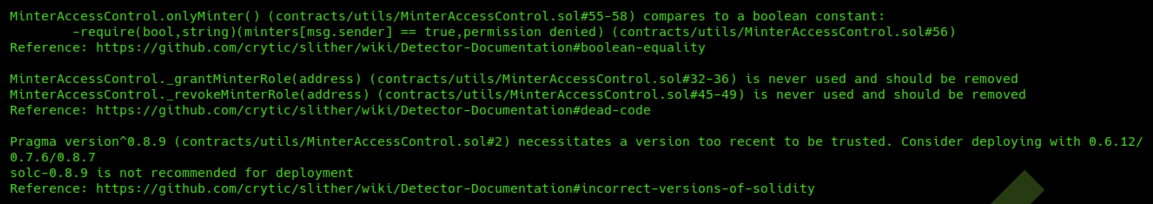


```
Address.functionCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#85-87) is never used and should be removed
Address.functionCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#95-101) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#114-120) is never used and should be
 removed
Address.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#128-139) is never used and sh
ould be removed
Address.functionDelegateCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#174-176) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#184-193) is never used and should be r
emoved
Address.functionStaticCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#147-149) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#157-166) is never used and should be rem
oved
Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#60-65) is never used and should be removed
Address.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#201-221) is never used and should be removed
Context._msgData() (node_modules/@openzeppelin/contracts/utils/Context.sol#21-23) is never used and should be removed
ECDSA.recover(bytes32,bytes) (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#104-108) is never used and should be removed
ECDSA.recover(bytes32,bytes32,bytes32) (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#132-140) is never used and should be re
moved
ECDSA.toEthSignedMessageHash(bytes) (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#216-218) is never used and should be remov
ed
ECDSA.toTypedDataHash(bytes32,bytes32) (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#229-231) is never used and should be re
moved
ECDSA.tryRecover(bytes32,bytes) (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#57-88) is never used and should be removed
ECDSA.tryRecover(bytes32,bytes32,bytes32) (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#117-125) is never used and should be
 removed
ERC721._baseURI() (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#105-107) is never used and should be removed
Strings.toHexString(address) (node_modules/@openzeppelin/contracts/utils/Strings.sol#72-74) is never used and should be removed
Strings.toHexString(uint256) (node_modules/@openzeppelin/contracts/utils/Strings.sol#41-52) is never used and should be removed
Strings.toHexString(uint256,uint256) (node_modules/@openzeppelin/contracts/utils/Strings.sol#57-67) is never used and should be removed
TimeLimit._updateEndTime(uint256,address) (contracts/utils/TimeLimit.sol#41-44) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/security/Pausable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/security/ReentrancyGuard.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Enumerable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#4) allows old versions
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/ERC165.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/IERC165.sol#4) allows old versions
Pragma version0.8.9 (contracts/EightBit.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version0.8.9 (contracts/EightBitMinter.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version^0.8.9 (contracts/utils/MinterAccessControl.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/
0.7.6/0.8.7
Pragma version0.8.9 (contracts/utils/TimeLimit.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#60-65):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts/utils/Address.sol#63)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#128-139):
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
Low level call in Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#157-166):
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#164)
Low level call in Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#184-193):
        - (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#191)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Reentrancy in EightBitMinter._mint(EightBitMinter.MintInput) (contracts/EightBitMinter.sol#228-240):
        External calls:
        - address(_msgSender()).transfer(refund) (contracts/EightBitMinter.sol#237)
        Event emitted after the call(s):
        - Mint(input_.user,input_.styleTag,input_.productCode,tagId,mintingFee) (contracts/EightBitMinter.sol#239)
Reentrancy in EightBitMinter.mint(EightBitMinter.MintInput) (contracts/EightBitMinter.sol#199-213):
        External calls:
        - _mint(input_) (contracts/EightBitMinter.sol#211)
                - address(_msgSender()).transfer(refund) (contracts/EightBitMinter.sol#237)
        State variables written after the call(s):
        - orderMinted[inputHash] = true (contracts/EightBitMinter.sol#212)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4

renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (node_modules/@openzeppelin/contracts/access/Ownable.sol#61-63)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (node_modules/@openzeppelin/contracts/access/Ownable.sol#69-72)
name() should be declared external:
        - ERC721.name() (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#79-81)
symbol() should be declared external:
        - ERC721.symbol() (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#86-88)
tokenURI(uint256) should be declared external:
        - ERC721.tokenURI(uint256) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#93-98)
approve(address,uint256) should be declared external:
        - ERC721.approve(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#112-122)
setApprovalForAll(address,bool) should be declared external:
        - ERC721.setApprovalForAll(address,bool) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#136-138)
transferFrom(address,address,uint256) should be declared external:
        - ERC721.transferFrom(address,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#150-159)
tokenOfOwnerByIndex(address,uint256) should be declared external:
        - ERC721Enumerable.tokenOfOwnerByIndex(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.so
l#37-40)
tokenByIndex(uint256) should be declared external:
        - ERC721Enumerable.tokenByIndex(uint256) (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol#52-55)
burn(uint256) should be declared external:
        - EightBit.burn(uint256) (contracts/EightBit.sol#91-95)
safeBatchTransferFrom(address,address,uint256[]) should be declared external:
        - EightBit.safeBatchTransferFrom(address,address,uint256[]) (contracts/EightBit.sol#119-127)
updateStartTime(uint256) should be declared external:
        - EightBitMinter.updateStartTime(uint256) (contracts/EightBitMinter.sol#161-163)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

## TimeLimit.sol

```
TimeLimit._updateEndTime(uint256,address) (contracts/utils/TimeLimit.sol#41-44) is never used and should be removed
TimeLimit._updateStartTime(uint256,address) (contracts/utils/TimeLimit.sol#36-39) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version0.8.9 (contracts/utils/TimeLimit.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

## MinterAccessControl.sol



```
MinterAccessControl.onlyMinter() (contracts/utils/MinterAccessControl.sol#55-58) compares to a boolean constant:
        -require(bool,string)(minters[msg.sender] == true,permission denied) (contracts/utils/MinterAccessControl.sol#56)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality

MinterAccessControl._grantMinterRole(address) (contracts/utils/MinterAccessControl.sol#32-36) is never used and should be removed
MinterAccessControl._revokeMinterRole(address) (contracts/utils/MinterAccessControl.sol#45-49) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.9 (contracts/utils/MinterAccessControl.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/
0.7.6/0.8.7
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

- Issues found by slither are either reported above or false positives.

## 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 in order to locate any vulnerabilities.

MythX results:

EightBit.sol

TimeLimit.sol

MinterAccessControl.sol

- No major issues were found by MythX.

THANK YOU FOR CHOOSING

// HALBORN