Uranium3o8 - ERC20 Token

Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **September 19th, 2023 - September 21st, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	09/22/2023
0.2	Document Update	09/25/2023
0.3	Draft Version	09/25/2023
0.4	Draft Review	09/25/2023
0.5	Draft Review	09/25/2023
1.0	Remediation Plan	10/05/2023
1.1	Remediation Plan Review	10/06/2023
1.2	Remediation Plan Review	10/06/2023

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
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Uranium3o8 (codenamed Pizza) is the team's ERC20 token, intended to act as an asset-backed token.

Uranium3o8 engaged Halborn to conduct a security assessment on their smart contracts beginning on September 19th, 2023 and ending on September 21st, 2023. The security assessment was scoped to the smart contracts provided in the u308/pizza GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

1.2 ASSESSMENT SUMMARY

Halborn was provided three days for the engagement and assigned one full-time security engineer to review the security of the smart contracts in scope. The engineer is a blockchain and smart contract security expert with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the smart contracts.
- Ensure that smart contract functionality operates as intended.

In summary, Halborn identified some security risks, which were mostly addressed by Uranium3o8. The main ones were the following:
- Restrict Minters to burn tokens only up to their used allowance.
- Prevent adding the same Minter multiple times to the ERC20 contract.
- Review the current implementation of the mintByMinter() function and ensure that it is in line with business requirements.

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 assessment. 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 assessment:

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

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium: (Y:M) High: (Y:H) Critical (Y:H)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N)	1
	Partial (R:P)	0.5
	Full (R:F)	0.25
Scope (s)	Changed (S:C)	1.25
	Unchanged (S:U)	1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

2.4 SCOPE

Code repositories:

1. Uranium3o8 Smart Contracts

- Repository: u308/pizza
- Commit ID: 98ca6331844497ecb98719184a657ff9c7933103
- Smart contracts in scope:
  1. Blacklistable (src/Blacklistable.sol)
  2. Migrator.sol (src/Migrator.sol)
  3. Pausable.sol (src/Pausable.sol)
  4. Pizza.sol (src/Pizza.sol)
  5. Rescuable.sol (src/Rescuable.sol)
- Fix commit ID: 40300a0f04f026fe9b52ce3361a76c8ebd56db91

Note that Pizza is the codename of the Uranium3o8 ERC20 token.

Out-of-scope

- Third-party libraries and dependencies.
- Economic attacks.

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	3	2	12

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) INVALID ALLOWANCE CHECK IN BURNBYMINTER	Medium (5.0)	SOLVED - 09/27/2023
(HAL-02) DUPLCATE MINTERS ALLOWED	Medium (5.0)	SOLVED - 09/30/2023
(HAL-03) CENTRALIZATION RISK: MINTERS CAN BURN FROM ANYONE	Medium (5.0)	SOLVED - 09/27/2023
(HAL-04) BURNEDBYMINTER EVENT MIGHT EMITTED WITH INACCURATE AMOUNT	Low (2.5)	SOLVED - 09/29/2023
(HAL-05) MISSING ZERO ADDRESS CHECKS	Low (2.5)	RISK ACCEPTED
(HAL-06) SINGLE STEP OWNERSHIP TRANSFER PROCESS	Informational (1.0)	ACKNOWLEDGED
(HAL-07) OWNER CAN RENOUNCE OWNERSHIP	Informational (1.0)	ACKNOWLEDGED
(HAL-08) LACK OF EMERGENCY STOP PATTERN IMPLEMENTATION	Informational (1.0)	ACKNOWLEDGED
(HAL-09) MISSING EVENTS FOR CONTRACT OPERATIONS	Informational (0.8)	ACKNOWLEDGED
(HAL-10) ITERATING OVER A DYNAMIC ARRAY	Informational (0.5)	SOLVED - 09/30/2023
(HAL-11) MINTER ALLOWANCE CANNOT BE FULLY USED	Informational (0.0)	SOLVED - 09/27/2023
(HAL-12) REDUNDANT REENTRANCY PROTECTION	Informational (0.0)	ACKNOWLEDGED
(HAL-13) REDUNDANT OVERFLOW CHECKS	Informational (0.0)	SOLVED - 09/27/2023
(HAL-14) REDUNDANT TOTAL SUPPLY AND BALANCE CHECKS	Informational (0.0)	SOLVED - 09/27/2023
(HAL-15) USING REVERT STRINGS INSTEAD OF CUSTOM ERRORS	Informational (0.0)	ACKNOWLEDGED
(HAL-16) FOR LOOPS CAN BE GAS OPTIMIZED	Informational (0.0)	ACKNOWLEDGED

(HAL-17) INEFFICIENT ISMINTER CHECK	Informational (0.0)	SOLVED - 09/30/2023

FINDINGS & TECH DETAILS

4.1 (HAL-01) INVALID ALLOWANCE CHECK IN BURNBYMINTER - MEDIUM (5.0)

Description:

Minters can mint tokens up to their minterAllowance balance. Minting tokens increases their minterUsedAllowance amount. They can also burn tokens up to their minterUsedAllowance limit, which decreases this number by the number of tokens burned.

However, it was identified that the lack of validation in the burnByMinter () function allowed Minters to burn tokens over their minterUsedAllowance limit.

Code Location:

The amount is burned from the user regardless of the usedAllowance:

Listing 1: Pizza.sol (Lines 242-248)

```
229 function burnByMinter(
230     address from,
231     uint256 amount
232 )
233     external
234     nonReentrant
235     onlyMinters
236     whenNotPaused
237     notBlacklisted(msg.sender)
238     notBlacklisted(from)
239 {
240     require(totalSupply() >= amount);
241     require(balanceOf(from) >= amount);
242     _burn(from, amount);
243     uint256 usedAllowance = minterUsedAllowance[msg.sender];
244     if (usedAllowance < amount) {
245         delete minterUsedAllowance[msg.sender];
246     } else {
247         minterUsedAllowance[msg.sender] -= amount;
248     }
```

```
    emit BurnedByMinter(amount, from);
}
```

Proof of Concept:

In the following proof of concept, the Minter user burns more tokens than their minterUsedAllowance:

```
>>> pizza.configureMinter(minter1, 1000 * 10**18, {'from': masterMinter})
Transaction sent: 0x200196cefaad51fadd2d782d0bba3faf82f542667d2721e5ecb21f7170c5c775
>>> pizza.minterUsedAllowance(minter1)
10000000000000000000
>>> pizza.balanceOf(alice)
1000000000000000000000
>>> pizza.burnByMinter(alice, 1000000000000000000000, {'from': minter1})
Transaction sent: 0xf973808bdc6cbecb315fe919c01cff3a220b9e7a3e07629d1bc4a656a8972653
  Gas price: 0.0 gwei   Gas limit: 30000000   Nonce: 1
  Pizza.burnByMinter confirmed   Block: 18211339   Gas used: 27547 (0.09%)

<Transaction '0xf973808bdc6cbecb315fe919c01cff3a220b9e7a3e07629d1bc4a656a8972653'>
>>> pizza.balanceOf(alice)
0
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:M/Y:N/R:N/S:U (5.0)

Recommendation:

It is recommended to restrict the Minters to burn tokens only up to their usedAllowance.

Remediation Plan:

SOLVED: The Uranium3o8 team solved the issue in commit 36cf4c4 by only allowing Minters to burn tokens up to their usedAllowance.

4.2 (HAL-02) DUPLCATE MINTERS ALLOWED - MEDIUM (5.0)

Description:

It was identified that adding the same Minter multiple times to the Pizza contract is possible. Adding the same user again updates their minterAllowance, but it does not change their minterUsedAllowance to zero. This might result in invalid internal accounting and unexpected results.

If an account is added twice as a Minter, calling the removeMinter() function only removes their first occurrence from the minterAddresses array, and the account would still retain their Minter rights until all their occurrences are removed from the minterAddresses array.

Note that, in the current contract implementation, Minters with zero minterAllowance can burn unlimited tokens.

Code Location:

The same Minter can be added multiple times to the Pizza contract:

Listing 2: Pizza.sol

```
function configureMinter(
    address minter,
    uint256 minterAllowanceAmount
) external nonReentrant whenNotPaused onlyMasterMinter returns (
 ↳ bool) {
    require(minter != masterMinter, "trying to add masterMinter");
    minters[minter] = true;
    minterAllowance[minter] = minterAllowanceAmount;
    minterAddresses.push(minter);
    emit MinterConfigured(minter, minterAllowanceAmount);
    return true;
}
```

The removeMinter function only removes the first occurrence:

Listing 3: Pizza.sol (Lines 322,323)

```
303 function removeMinter(
304     address minter
305 ) external nonReentrant whenNotPaused onlyMasterMinter returns (
 ↳ bool) {
306     require(minter != address(0), "Zero address!");
307     require(minter != owner(), "You cannot remove the owner!");
308     require(isMinter(minter), "Address not on list of minter
 ↳ addresses!");
309     // deleting value from mapping
310     delete minters[minter];
311     delete minterAllowance[minter];
312     delete minterUsedAllowance[minter];
313
314     // Find and remove the minter from the list of minter
 ↳ addresses
315     for (uint256 i = 0; i < minterAddresses.length; i++) {
316         // if the programme finds a slot in the array whose entry
 ↳ is the minter
317         // it replaces it with the last element in the array
318         if (minterAddresses[i] == minter) {
319             minterAddresses[i] = minterAddresses[
320                 minterAddresses.length - 1
321             ];
322             minterAddresses.pop();
323             break;
324         }
325     }
326     emit MinterRemoved(minter);
327     return true;
328 }
```

Proof of Concept:

In the following proof of concept, the same user added as a Minter twice, and after calling the removeMinter() function, they still retain their Minter role:

```
>>> pizza.configureMinter(minter1, 1000 * 10**18, {'from': masterMinter})
Transaction sent: 0x200196cefaad51fadd2d782d0bba3faf82f542667d2721e5ecb21f7170c5c775
  Gas price: 0.0 gwei   Gas limit: 30000000   Nonce: 0
  Pizza.configureMinter confirmed   Block: 18211335   Gas used: 111293 (0.37%)

<Transaction '0x200196cefaad51fadd2d782d0bba3faf82f542667d2721e5ecb21f7170c5c775'>
>>> pizza.isMinter(minter1)
True
>>> pizza.minterAddresses(0)
'0x74cCC9494993422E25E06ae067b2c01D82ccD082'
>>> pizza.configureMinter(minter1, 5000 * 10**18, {'from': masterMinter})
Transaction sent: 0xd73efb15d3698dcc1717932d883196490b0273e74628171db81c7f3031cafbd3
  Gas price: 0.0 gwei   Gas limit: 30000000   Nonce: 1
  Pizza.configureMinter confirmed   Block: 18211336   Gas used: 62105 (0.21%)

<Transaction '0xd73efb15d3698dcc1717932d883196490b0273e74628171db81c7f3031cafbd3'>
>>> pizza.minterAddresses(0)
'0x74cCC9494993422E25E06ae067b2c01D82ccD082'
>>> pizza.minterAddresses(1)
'0x74cCC9494993422E25E06ae067b2c01D82ccD082'
>>> pizza.removeMinter(minter1, {'from': masterMinter})
Transaction sent: 0xc393e9ad0b1ded9330e03c6dffbcd675a13550ccd29ad4c06746fea2b5ce8f89
  Gas price: 0.0 gwei   Gas limit: 30000000   Nonce: 2
  Pizza.removeMinter confirmed   Block: 18211337   Gas used: 33107 (0.11%)

<Transaction '0xc393e9ad0b1ded9330e03c6dffbcd675a13550ccd29ad4c06746fea2b5ce8f89'>
>>> pizza.isMinter(minter1)
True
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:M/A:M/D:M/Y:N/R:N/S:U (5.0)

Recommendation:

It is recommended to prevent adding the same Minter multiple times to the Pizza contract.

It is also recommended to review the current implementation of the Minter configuration upgrade logic and ensure that it is in line with business requirements.

Remediation Plan:

SOLVED: The Uranium3o8 team solved the issue in commits 8f56830 and 36cf4c4 by preventing adding the same Minter multiple times to the Pizza contract and updating the configureMinter logic to match business requirements better.

4.3 (HAL-03) CENTRALIZATION RISK: MINTERS CAN BURN FROM ANYONE - MEDIUM (5.0)

Description:

Minters can mint tokens up to their minterAllowance balance. Minting tokens increases their minterUsedAllowance amount. They can also burn tokens up to their minterUsedAllowance limit, which decreases this number by the number of tokens they burned.

However, it was identified that Minters can burn tokens from any arbitrary address, not just the tokens that they minted. This implementation also allows Minters to transfer arbitrary amount of tokens between non-blacklisted users by burning tokens from the source address and minting tokens to the destination address.

Note that the MasterMinter account can configure the Minters.

Code Location:

Minters can mint tokens up to their minterAllowance limit:

Listing 4: Pizza.sol (Lines 203-205,214)

```
192 function mintByMinter(
193     address to,
194     uint256 amount
195 )
196     external
197     nonReentrant
198     onlyMinters
199     whenNotPaused
200     notBlacklisted(msg.sender)
201     notBlacklisted(to)
202 {
203     uint256 allowance = minterAllowance[msg.sender];
204     uint256 usedAllowance = minterUsedAllowance[msg.sender];
```

```
    require(usedAllowance + amount < allowance, "Insufficient allowance");
    require(
        totalSupply() + amount > totalSupply(),
        "issuing negative amount to total supply"
    );
    require(
        balanceOf(to) + amount > balanceOf(to),
        "issuing negative amount to owner"
    );
    _mint(to, amount);
    minterUsedAllowance[msg.sender] += amount;
    emit MintedByMinter(amount, to);
}
```

Minters can burn tokens up to their usedAllowance limit:

Listing 5: Pizza.sol (Lines 242-248)

```
229 function burnByMinter(
230     address from,
231     uint256 amount
232 )
233     external
234     nonReentrant
235     onlyMinters
236     whenNotPaused
237     notBlacklisted(msg.sender)
238     notBlacklisted(from)
239 {
240     require(totalSupply() >= amount);
241     require(balanceOf(from) >= amount);
242     _burn(from, amount);
243     uint256 usedAllowance = minterUsedAllowance[msg.sender];
244     if (usedAllowance < amount) {
245         delete minterUsedAllowance[msg.sender];
246     } else {
247         minterUsedAllowance[msg.sender] -= amount;
248     }
249     emit BurnedByMinter(amount, from);
250 }
```

Proof of Concept:

In the following proof of concept, the Minter user burns tokens from a different user:

```
>>> pizza.transfer(alice, 100 * 10**18, {'from': masterMinter})
Transaction sent: 0xceb71df6c9b637270604ea90e078230e045ba062a5a052a2c437bd94377587b2
  Gas price: 0.0 gwei   Gas limit: 30000000   Nonce: 2
  Pizza.transfer confirmed   Block: 18211338   Gas used: 53744 (0.18%)

<Transaction '0xceb71df6c9b637270604ea90e078230e045ba062a5a052a2c437bd94377587b2'>
>>> pizza.balanceOf(alice)
100000000000000000000
>>> pizza.burnByMinter(alice, 100 * 10**18, {'from': minter1})
Transaction sent: 0x5432ed6b8097b0c45ebac9b9334a79dd02954d5e49f422248656f4a6fa0c6189
  Gas price: 0.0 gwei   Gas limit: 30000000   Nonce: 1
  Pizza.burnByMinter confirmed   Block: 18211339   Gas used: 27547 (0.09%)

<Transaction '0x5432ed6b8097b0c45ebac9b9334a79dd02954d5e49f422248656f4a6fa0c6189'>
>>> pizza.balanceOf(alice)
0
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:M/Y:N/R:N/S:U (5.0)

Recommendation:

It is recommended to review the current implementation and ensure that it is in line with business requirements.

It is also recommended to employ multi-signature access for every high-privileged accounts.

Remediation Plan:

SOLVED: The Uranium3o8 team solved the issue in commit 36cf4c4 by only allowing the Minters to burn their own tokens.

4.4 (HAL-04) BURNEDBYMINTER EVENT MIGHT EMITTED WITH INACCURATE AMOUNT - LOW (2.5)

Description:

It was identified in the Pizza contract that the burnByMinter() function emits the original amount parameter in the BurnedByMinter event, even if the amount of tokens actually burned is less. As a result, blockchain monitoring systems might log invalid burnt amounts related to the burnByMinter() function.

Code Location:

Listing 6: Pizza.sol (Lines 244,245,249)

```
229 function burnByMinter(
230     address from,
231     uint256 amount
232 )
233     external
234     nonReentrant
235     onlyMinters
236     whenNotPaused
237     notBlacklisted(msg.sender)
238     notBlacklisted(from)
239 {
240     require(totalSupply() >= amount);
241     require(balanceOf(from) >= amount);
242     _burn(from, amount);
243     uint256 usedAllowance = minterUsedAllowance[msg.sender];
244     if (usedAllowance < amount) {
245         delete minterUsedAllowance[msg.sender];
246     } else {
247         minterUsedAllowance[msg.sender] -= amount;
248     }
249     emit BurnedByMinter(amount, from);
250 }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:U (2.5)

Recommendation:

It is recommended to emit the BurnedByMinter event with the actual number of tokens burned instead of the amount parameter.

Remediation Plan:

SOLVED: The Uranium3o8 team solved the issue in commit 8f56830 by emitting the actual number of tokens burned instead of the amount parameter.

4.5 (HAL-05) MISSING ZERO ADDRESS CHECKS - LOW (2.5)

Description:

It was identified that the _oldToken and _newToken constructor parameters in the Migrator contract lack zero address validation.

Note that the old and new token addresses are configured in the constructor and cannot be changed later.

Code Location:

Listing 7: Migrator.sol

```
    constructor(address _oldToken, address _newToken) {
        oldTokenAddress = _oldToken;
        newTokenAddress = _newToken;
        oldToken = IPizza(oldTokenAddress);
        newToken = IPizza(newTokenAddress);
    }
```

Code Location:

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:L/D:N/Y:N/R:N/S:U (2.5)

Recommendation:

It is recommended to add zero address validation for the _oldToken and _newToken parameters.

Remediation Plan:

RISK ACCEPTED: The Uranium3o8 team made a business decision to accept the risk of this finding and not alter the Migrator contract.

FINDINGS & TECH DETAILS

4.6 (HAL-06) SINGLE STEP OWNERSHIP TRANSFER PROCESS - INFORMATIONAL (1.0)

Description:

Ownership of the contracts can be lost as the Pizza and Migrator contracts are inherited from the Ownable contract and their ownership can be transferred in a single-step process. The address the ownership is changed to should be verified to be active or willing to act as the owner.

Code Location:

Listing 8: openzeppelin-contracts/contracts/access/Ownable.sol

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

Listing 9: openzeppelin-contracts/contracts/access/Ownable.sol

```
function _transferOwnership(address newOwner) internal virtual {
    address oldOwner = _owner;
    _owner = newOwner;
    emit OwnershipTransferred(oldOwner, newOwner);
}
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:M/D:N/Y:N/R:N/S:U (1.0)

Recommendation:

Consider using the Ownable2Step library over the Ownable library.

Remediation Plan:

ACKNOWLEDGED: The Uranium3o8 team acknowledged this finding in the Migrator contract. The issue was solved in the Pizza contract in commit 36cf4c4 by using the Ownable2Step library.

4.7 (HAL-07) OWNER CAN RENOUNCE OWNERSHIP - INFORMATIONAL (1.0)

Description:

The Owner of the contract is usually the account that deploys the contract. As a result, the Owner can perform some privileged functions. In the Pizza and Migrator contracts, the renounceOwnership() function is used to renounce the Owner permission. Renouncing ownership before transferring would result in the contract having no Owner, eliminating the ability to call privileged functions.

Code Location:

Listing 10: openzeppelin-contracts/contracts/access/Ownable.sol

```
function renounceOwnership() public virtual onlyOwner {
    _transferOwnership(address(0));
}
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:M/D:N/Y:N/R:N/S:U (1.0)

Recommendation:

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

Remediation Plan:

ACKNOWLEDGED: The Uranium3o8 team acknowledged this finding in the Migrator contract. The issue was solved in the Pizza contract in commit

36cf4c4 by preventing the renounceOwnership() function from being used by overriding it.

4.8 (HAL-08) LACK OF EMERGENCY STOP PATTERN IMPLEMENTATION - INFORMATIONAL (1.0)

Description:

It was identified that the Pausable module is not used in the Migrator contract. Emergency stop patterns allow the project team to pause crucial functionalities, while being in the state of emergency, e.g., being under adversary attack. In the case the emergency stop pattern is not used, critical functions cannot be temporarily disabled.

Code Location:

For example, it is not possible to pause the Migrator function in the contract:

Listing 11: Migrator.sol

```
function migrate(uint256 oldTokenAmount) public nonReentrant {
    oldToken.transferFrom(msg.sender, address(this),
↳ oldTokenAmount);
    newToken.transfer(msg.sender, oldTokenAmount);
    emit Migrated(oldTokenAmount);
}
```

BVSS:

AO:A/AC:L/AX:H/C:N/I:L/A:N/D:L/Y:N/R:N/S:U (1.0)

Recommendation:

Consider using the emergency stop pattern in the Migrator contract.

Remediation Plan:

ACKNOWLEDGED: The Uranium3o8 team acknowledged this finding.

4.9 (HAL-09) MISSING EVENTS FOR CONTRACT OPERATIONS - INFORMATIONAL (0.8)

Description:

It was identified that the depositOldTokenReserves(), depositNewTokenReserves(), withdrawOldTokenReserves() and withdrawNewTokenReserves() functions from the Migrator contract do not emit any events. As a result, blockchain monitoring systems might not be able to timely detect suspicious behaviors related to these functions.

Note that the oldTokensWithdrawn, Deposited and TokenAddressesSet events are declared but not used in the Migrator contract.

BVSS:

AO:A/AC:L/AX:H/C:N/I:N/A:N/D:L/Y:N/R:N/S:U (0.8)

Recommendation:

Adding events for all important operations is recommended to help monitor the contracts and detect suspicious behavior. A monitoring system that tracks relevant events would allow the timely detection of compromised system components.

Remediation Plan:

ACKNOWLEDGED: The Uranium3o8 team acknowledged this finding. Events were only added to the withdrawOldTokenReserves() and withdrawNewTokenReserves() functions in commit 36cf4c4.

4.10 (HAL-10) ITERATING OVER A DYNAMIC ARRAY - INFORMATIONAL (0.5)

Description:

It was identified that the Pizza contract iterates through dynamic arrays in its isMinter() and removeMinter() functions. These functions may revert because they run out of gas, causing denial of service.

Code Location:

Listing 12: Pizza.sol (Lines 91-95)

```
91 function isMinter(address minter) public view returns (bool) {
92     for (uint i = 0; i < minterAddresses.length; i++) {
93         if (minterAddresses[i] == minter) {
94             return true;
95         }
96     }
97     return false;
98 }
```

Listing 13: Pizza.sol (Lines 308,315-325)

```
303 function removeMinter(
304     address minter
305 ) external nonReentrant whenNotPaused onlyMasterMinter returns (
    ↳ bool) {
306     require(minter != address(0), "Zero address!");
307     require(minter != owner(), "You cannot remove the owner!");
308     require(isMinter(minter), "Address not on list of minter
    ↳ addresses!");
309     // deleting value from mapping
310     delete minters[minter];
311     delete minterAllowance[minter];
312     delete minterUsedAllowance[minter];
313
314     // Find and remove the minter from the list of minter
    ↳ addresses
315     for (uint256 i = 0; i < minterAddresses.length; i++) {
```

```
        // if the programme finds a slot in the array whose entry is the minter
        // it replaces it with the last element in the array
        if (minterAddresses[i] == minter) {
            minterAddresses[i] = minterAddresses[
                minterAddresses.length - 1
            ];
            minterAddresses.pop();
            break;
        }
    }
...
}
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:L/D:N/Y:N/R:N/S:U (0.5)

Recommendation:

It is recommended to review the functions involving dynamic arrays and assess their potential gas usage, and limit their sizes when necessary.

Remediation Plan:

SOLVED: The Uranium3o8 team solved the issue in commits 36cf4c4 and 40300a0 by limiting the number of Minters and optimizing the isMinter() function.

4.11 (HAL-11) MINTER ALLOWANCE CANNOT BE FULLY USED - INFORMATIONAL (0.0)

Description:

It was identified that the mintByMinter() function in the Pizza contract requires the used allowance of the Minters to be strictly lesser than their allowance. This might not be the intended behavior, as the Minters could expect to be able to use their full allowance.

Code Location:

Listing 14: Pizza.sol (Line 205)

```
192 function mintByMinter(
193     address to,
194     uint256 amount
195 )
196     external
197     nonReentrant
198     onlyMinters
199     whenNotPaused
200     notBlacklisted(msg.sender)
201     notBlacklisted(to)
202 {
203     uint256 allowance = minterAllowance[msg.sender];
204     uint256 usedAllowance = minterUsedAllowance[msg.sender];
205     require(usedAllowance + amount < allowance, "Insufficient
  ↳ allowance");
206     require(
207         totalSupply() + amount > totalSupply(),
208         "issuing negative amount to total supply"
209     );
210     require(
211         balanceOf(to) + amount > balanceOf(to),
212         "issuing negative amount to owner"
213     );
214     _mint(to, amount);
```

```
    minterUsedAllowance[msg.sender] += amount;
    emit MintedByMinter(amount, to);
}
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

It is recommended to allow the Minters to use their full allowance.

Remediation Plan:

SOLVED: The Uranium3o8 team solved the issue in commit 36cf4c4 by allowing the Minters to use their full allowance.

4.12 (HAL-12) REDUNDANT REENTRANCY PROTECTION - INFORMATIONAL (0.0)

Description:

It was identified that the Pizza contract uses the reentrancy guards to protect its functions. This is unnecessary, as there are no external calls to other contracts in these functions.

It was also identified that the migrate() function in the Migrator contract uses a reentrancy guard. This is unnecessary, as both oldToken and newToken are trusted contracts and managed by the Uranium3o8 team.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Consider removing the unnecessary reentrancy guards from the Pizza and Migrator contracts to save gas and reduce complexity.

Remediation Plan:

ACKNOWLEDGED: The Uranium3o8 team acknowledged this finding.

4.13 (HAL-13) REDUNDANT OVERFLOW CHECKS - INFORMATIONAL (0.0)

Description:

It was identified that the Pizza contract uses unnecessary overflow checks in the issue() and mintByMinter() functions, since these checks have been made the default in Solidity since v0.8.0.

Code Location:

Listing 15: Pizza.sol (Lines 147-155)

```
138 function issue(
139     uint256 amount
140 )
141     external
142     nonReentrant
143     onlyMasterMinter
144     whenNotPaused
145     notBlacklisted(msg.sender)
146 {
147     require(
148         totalSupply() + amount > totalSupply(),
149         "issuing negative amount to total supply"
150     );
151
152     require(
153         balanceOf(masterMinter) + amount > balanceOf(masterMinter)
    ↳ ,
154         "issuing negative amount to owner"
155     );
156
157     _mint(masterMinter, amount);
158     emit Issued(amount);
159 }
```

Listing 16: Pizza.sol (Lines 206-213)

```
192 function mintByMinter(
193     address to,
194     uint256 amount
195 )
196     external
197     nonReentrant
198     onlyMinters
199     whenNotPaused
200     notBlacklisted(msg.sender)
201     notBlacklisted(to)
202 {
203     uint256 allowance = minterAllowance[msg.sender];
204     uint256 usedAllowance = minterUsedAllowance[msg.sender];
205     require(usedAllowance + amount < allowance, "Insufficient
    ↳ allowance");
206     require(
207         totalSupply() + amount > totalSupply(),
208         "issuing negative amount to total supply"
209     );
210     require(
211         balanceOf(to) + amount > balanceOf(to),
212         "issuing negative amount to owner"
213     );
214     _mint(to, amount);
215     minterUsedAllowance[msg.sender] += amount;
216     emit MintedByMinter(amount, to);
217 }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Consider removing the unnecessary overflow checks from the Pizza contract to save gas and reduce complexity.

Remediation Plan:

SOLVED: The Uranium3o8 team solved the issue in commit 36cf4c4 by removing the unnecessary overflow checks.

4.14 (HAL-14) REDUNDANT TOTAL SUPPLY AND BALANCE CHECKS - INFORMATIONAL (0.0)

Description:

It was identified that the Pizza contract uses redundant total supply and balance checks in the redeem() and burnByMinter() functions, since these values are validated in the ERC20 contract, from which the Pizza contract is inherited.

Code Location:

Unnecessary total supply and balance checks in the redeem() function:

Listing 17: Pizza.sol (Lines 178-179)

```
169 function redeem(
170     uint256 amount
171 )
172     external
173     nonReentrant
174     onlyMasterMinter
175     whenNotPaused
176     notBlacklisted(msg.sender)
177 {
178     require(totalSupply() >= amount);
179     require(balanceOf(masterMinter) >= amount);
180     _burn(masterMinter, amount);
181     emit Redeemed(amount);
182 }
```

Unnecessary total supply and balance checks in the burnByMinter() function:

Listing 18: Pizza.sol (Lines 240,241)

```
229 function burnByMinter(
230     address from,
231     uint256 amount
232 )
233     external
234     nonReentrant
235     onlyMinters
236     whenNotPaused
237     notBlacklisted(msg.sender)
238     notBlacklisted(from)
239 {
240     require(totalSupply() >= amount);
241     require(balanceOf(from) >= amount);
242     _burn(from, amount);
243     uint256 usedAllowance = minterUsedAllowance[msg.sender];
244     if (usedAllowance < amount) {
245         delete minterUsedAllowance[msg.sender];
246     } else {
247         minterUsedAllowance[msg.sender] -= amount;
248     }
249     emit BurnedByMinter(amount, from);
250 }
```

These checks are unnecessary, as these values are validated in the _burn() function of the ERC20 contract:

Listing 19: openzeppelin-contracts/contracts/token/ERC20/ERC20.sol (Lines 282,283)

```
277 function _burn(address account, uint256 amount) internal virtual {
278     require(account != address(0), "ERC20: burn from the zero
    ↳ address");
279
280     _beforeTokenTransfer(account, address(0), amount);
281
282     uint256 accountBalance = _balances[account];
283     require(accountBalance >= amount, "ERC20: burn amount exceeds
    ↳ balance");
284     unchecked {
285         _balances[account] = accountBalance - amount;
286         // Overflow not possible: amount <= accountBalance <=
```

```
 ↳ totalSupply.
            _totalSupply -= amount;
        }

        emit Transfer(account, address(0), amount);

        _afterTokenTransfer(account, address(0), amount);
}
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Consider removing the unnecessary total supply and balance checks from the Pizza contract to save gas and reduce complexity.

Remediation Plan:

SOLVED: The Uranium3o8 team solved the issue in commit 36cf4c4 by removing the unnecessary total supply and balance checks.

4.15 (HAL-15) USING REVERT STRINGS INSTEAD OF CUSTOM ERRORS - INFORMATIONAL (0.0)

Description:

Starting from Solidity v0.8.4, there is a convenient and gas-efficient way to explain to users why an operation failed through the use of custom errors. If the revert string uses strings to provide additional information about failures (e.g. `require(minters[msg.sender], "Pizza: caller is not a minter");`), but they are rather expensive, especially when it comes to deploying cost, and it is difficult to use dynamic information in them.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

It is recommended to implement custom errors instead of reverting strings.

An example implementation of the initialization checks using custom errors:

Listing 20: Using Custom Errors

```
error NotMinter();

modifier onlyMinters() {
    // require(minters[msg.sender], "Pizza: caller is not a minter");
    if(minters[msg.sender] == false) revert NotMinter();
    _;
}
```

Remediation Plan:

ACKNOWLEDGED: The Uranium3o8 team acknowledged this finding. Custom errors are not implemented to maintain readability.

4.16 (HAL-16) FOR LOOPS CAN BE GAS OPTIMIZED - INFORMATIONAL (0.0)

Description:

It was identified that the for loops employed in the Pizza contract can be gas optimized by the following principles:

- Unnecessary reading of the array length on each iteration wastes gas.
- A postfix (e.g. i++) operator was used to increment the i variables. It is known that, in loops, using prefix operators (e.g. ++i) costs less gas per iteration than postfix operators.
- It is also possible to further optimize loops by using unchecked loop index incrementing and decrementing.

Code Location:

Listing 21: Pizza.sol (Line 91)

```
91 for (uint i = 0; i < minterAddresses.length; i++) {
92     if (minterAddresses[i] == minter) {
93         return true;
94     }
95 }
```

Listing 22: Pizza.sol (Line 315)

```
315 for (uint256 i = 0; i < minterAddresses.length; i++) {
316     // if the programme finds a slot in the array whose entry is
    ↳ the minter
317     // it replaces it with the last element in the array
318     if (minterAddresses[i] == minter) {
319         minterAddresses[i] = minterAddresses[
320             minterAddresses.length - 1
321         ];
322         minterAddresses.pop();
323         break;
```

```
    }
}
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Consider caching array lengths outside of loops, as long the size is not changed during the loop.

Consider using the unchecked `++i` operation instead of `i++` to increment the values of the `uint` variable inside the loop. It is noted that using unchecked operations requires particular caution to avoid overflows, and their use may impair code readability.

The following code is an example of the above recommendations:

Listing 23: For Loop Optimization

```
uint256 length = minterAddresses.length;
for (uint i = 0; i < length;) {
    if (minterAddresses[i] == minter) {
        return true;
    }
    unchecked { ++i; }
}
```

FINDINGS & TECH DETAILS

Remediation Plan:

ACKNOWLEDGED: The Uranium3o8 team acknowledged this finding. Gas optimizations are not implemented to maintain readability.

4.17 (HAL-17) INEFFICIENT ISMINTER CHECK - INFORMATIONAL (0.0)

Description:

It was identified that the isMinter() function of the Pizza contract is inefficient because instead of using the minters hashmap state variable, it loops through the minterAddresses array to determine if the address is a Minter. Using the minterAddresses array costs more gas and increases complexity.

Code Location:

The isMinter() function loops through the minterAddresses array:

Listing 24: Pizza.sol

```
function isMinter(address minter) public view returns (bool) {
    for (uint i = 0; i < minterAddresses.length; i++) {
        if (minterAddresses[i] == minter) {
            return true;
        }
    }
    return false;
}
```

The onlyMinters() modifier uses the minters hashmap for the same check:

Listing 25: Pizza.sol

```
modifier onlyMinters() {
    require(minters[msg.sender], "Pizza: caller is not a minter");
    _;
}
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Consider using the `minters` hashmap in the isMinter function instead of the `minterAddresses` array to determine if a user is a `Minter`.

Remediation Plan:

SOLVED: The Uranium3o8 team solved the issue in commit 40300a0 by using the `minters` hashmap.

AUTOMATED TESTING

5.1 STATIC ANALYSIS REPORT

Description:

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

Results:

contracts/Pizza.sol
Slither did not identify any vulnerabilities in the contract.

contracts/Migrator.sol

Slither results for Migrator.sol	
Finding	**Impact**
Migrator.withdrawOldTokenReserves(uint256) (contracts/Migrator.sol#48-50) ignores return value by oldToken.transfer(owner(),amount) (contracts/Migrator.sol#49)	High
Migrator.depositOldTokenReserves(uint256) (contracts/Migrator.sol#32-34) ignores return value by oldToken.transferFrom(owner(),address(this),amount) (contracts/Migrator.sol#33)	High
Migrator.migrate(uint256) (contracts/Migrator.sol#82-86) ignores return value by newToken.transfer(msg.sender,oldTokenAmount) (contracts/Migrator.sol#84)	High
Migrator.withdrawNewTokenReserves(uint256) (contracts/Migrator.sol#56-58) ignores return value by newToken.transfer(owner(),amount) (contracts/Migrator.sol#57)	High

Finding	Impact
Migrator.depositNewTokenReserves(uint256) (contracts/Migrator.sol#40-42) ignores return value by newToken.transferFrom(owner(),address(this),amount) (contracts/Migrator.sol#41)	High
Migrator.migrate(uint256) (contracts/Migrator.sol#82-86) ignores return value by oldToken.transferFrom(msg.sender,address(this),oldTokenAmount) (contracts/Migrator.sol#83)	High
Migrator.constructor(address,address)._oldToken (contracts/Migrator.sol#21) lacks a zero-check on : - oldTokenAddress = _oldToken (contracts/Migrator.sol#22)	Low
Migrator.constructor(address,address)._newToken (contracts/Migrator.sol#21) lacks a zero-check on : - newTokenAddress = _newToken (contracts/Migrator.sol#23)	Low
End of table for Migrator.sol	

contracts/Pausable.sol

Slither did not identify any vulnerabilities in the contract.

contracts/Rescuable.sol

Slither results for Pizza.sol	
Finding	**Impact**
Rescuable.rescueERC20(address,address,uint256) (contracts/Rescuable.sol#33-39) ignores return value by IERC20(tokenContract).transfer(to,amount) (contracts/Rescuable.sol#38)	High
End of table for Pizza.sol	

contracts/Blacklistable.sol

Slither did not identify any vulnerabilities in the contract.

The findings obtained as a result of the Slither scan were reviewed. The lack of checking the return value of the token transfer high-risk vulnerabilities in the Migrator function were determined as false positives as the token contracts will be developed and managed by the Uranium3o8 team and are expected to be reverted in case of failure, and the Rescuable contracts as the rescueERC20 function can be called only by authorized users to rescue accidentally sent tokens from the Pizza contract.

5.2 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.

Results:

contracts/Migrator.sol
MythX did not identify any vulnerabilities in the contract.

contracts/Pizza.sol
MythX did not identify any vulnerabilities in the contract.

contracts/Pausable.sol
MythX did not identify any vulnerabilities in the contract.

contracts/Rescuable.sol
MythX did not identify any vulnerabilities in the contract.

contracts/Blacklistable.sol
MythX did not identify any vulnerabilities in the contract.

THANK YOU FOR CHOOSING

HALBORN