BLOCKHAT
SECURITY

FlexHype

Smart Contract Security Audit

Prepared by BlockHat

December 12th, 2024 - December 15th, 2024

BlockHat.io

contact@blockhat.io

Document Properties

Client	FlexHype
Version	1.0
Classification	Public

Scope

The FlexHype Contract in the FlexHype Repository

Link	Address
https://bscscan.com/address/0x186ec0EAd35148d83ADA01A550379fAf433874B6code	0x186ec0EAd35148d83ADA01A550379fAf433874B6

Contacts

COMPANY	CONTACT
BlockHat	contact@blockhat.io

Contents

1 Introduction

FlexHype engaged BlockHat to conduct a security assessment on the FlexHype beginning on December 12th, 2024 and ending December 15th, 2024. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About FlexHype

Issuer	FlexHype
Website url	`https://flexhype.org/`
Type	Solidity Smart Contract
Audit Method	Whitebox

1.2 Approach & Methodology

BlockHat used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

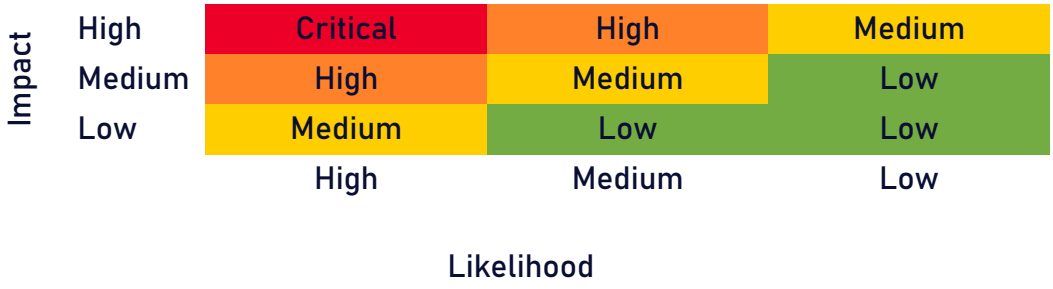
1.2.1 Risk Methodology

Vulnerabilities or bugs identified by BlockHat are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.



2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the FlexHype implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include , 7 low-severity, 1 informational-severity vulnerabilities.

Vulnerabilities	Severity	Status
Insufficient Validation of Parameters in Constructor	LOW	Unresolved
Unnecessary Use of uint256[1] memory amountParams in Constructor	LOW	Unresolved
No Safeguards for Reflection Fee Overwriting	LOW	Unresolved
Insufficient Validation for decimalsToSet	LOW	Unresolved
Code Repetition in transfer and transferFrom Functions	LOW	Unresolved
Missing Validation for Zero Address in excludeFromFeesAndLimits	LOW	Unresolved
Unnecessary Function Overrides Without Modifications	LOW	Unresolved
Non-Adherence to Solidity Naming Conventions	INFORMATIONAL	Unresolved

3 Finding Details

A FlexHype.sol

A.1 Insufficient Validation of Parameters in Constructor [LOW]

Description:

The constructor does not perform thorough validation of its input parameters, which could lead to misconfiguration or unexpected behavior. For instance: - amountParams[0] (Maximum token amount per address) is only checked for being non-zero when _isMaxAmountOfTokensSet is true, but there is no upper limit enforced. - bpsParams values (basis points for tax, deflation, etc.) are validated collectively, but individual parameters are not checked for reasonable ranges.

Listing 1: FlexHype.sol

```
91   constructor(
92     string memory name_,
93     string memory symbol_,
94     uint256 initialSupplyToSet,
95     uint8 decimalsToSet,
96     address tokenOwner,
97     ERC20ConfigProps memory customConfigProps,
98     string memory newDocumentUri,
99     address _taxAddress,
100    uint256[3] memory bpsParams,
101    uint256[1] memory amountParams
102  )
103    ReflectiveV3ERC20(
104      name_,
105      symbol_,
106      tokenOwner,
107      initialSupplyToSet,
```

```
108       decimalsToSet,
109       initialSupplyToSet != 0 ? bpsParams[2] : 0,
110       customConfigProps._isReflective
111     )
112   {
113     // reflection feature can't be used in combination with burning/
          ↪ deflation
114     // or reflection config is invalid if no reflection BPS amount is
          ↪ provided
115     if (
116       (customConfigProps._isReflective &&
117         (customConfigProps._isBurnable
118           customConfigProps._isDeflationary))
119       (!customConfigProps._isReflective && bpsParams[2] != 0)
120     ) {
121       revert InvalidReflectiveConfig();
122     }

124     if (customConfigProps._isMaxAmountOfTokensSet) {
125       if (amountParams[0] == 0) {
126         revert InvalidMaxTokenAmount(amountParams[0]);
127       }
128     }
129     if (decimalsToSet > 18) {
130       revert InvalidDecimals(decimalsToSet);
131     }

133     bpsInitChecks(customConfigProps, bpsParams, _taxAddress);

135     LibCommon.validateAddress(tokenOwner);

137     taxAddress = _taxAddress;

139     taxBPS = bpsParams[0];
```

```
140     deflationBPS = bpsParams[1];
141     initialSupply = initialSupplyToSet;
142     initialMaxTokenAmountPerAddress = amountParams[0];
143     initialDocumentUri = newDocumentUri;
144     initialTokenOwner = tokenOwner;
145     _decimals = decimalsToSet;
146     configProps = customConfigProps;
147     documentUri = newDocumentUri;
148     maxTokenAmountPerAddress = amountParams[0];

150     if (tokenOwner != msg.sender) {
151       transferOwnership(tokenOwner);
152     }
153   }
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

- Validate amountParams[0] for both lower and upper bounds to ensure reasonable token limits per address.
- Validate individual bpsParams values (tax, deflation, reflection fees) against logical thresholds before aggregating.

Status - Unresolved

A.2 Unnecessary Use of uint256[1] memory amountParams in Constructor [LOW]

Description:

The constructor of the FlexHype contract uses uint256[1] memory amountParams to store a single parameter (maxTokenAmount). This approach is redundant and introduces unnecessary complexity, as the parameter can be passed directly as a single variable instead of an array.

Listing 2: FlexHype.sol

```
91  constructor(
92      string memory name_,
93      string memory symbol_,
94      uint256 initialSupplyToSet,
95      uint8 decimalsToSet,
96      address tokenOwner,
97      ERC20ConfigProps memory customConfigProps,
98      string memory newDocumentUri,
99      address _taxAddress,
100     uint256[3] memory bpsParams,
101     uint256[1] memory amountParams // Only one value (maxTokenAmount) is
            ↪ stored here
102 )
```

Risk Level:

Likelihood – 1

Impact - 1

Recommendation:

Replace uint256[1] memory amountParams with a single uint256 maxTokenAmount variable for better readability and simplicity.

Status - Unresolved

A.3 No Safeguards for Reflection Fee Overwriting [LOW]

Description:

The setReflectionConfig function does not validate _feeBPS for realistic bounds. Without proper safeguards, the fee can be set to excessively high value or zero, disrupting token rewards.

Listing 3: FlexHype.sol

```
307 function setReflectionConfig(uint256 _feeBPS) external onlyOwner {
308   if (!isReflective()) {
309     revert TokenIsNotReflective();
310   }
311   super._setReflectionFee(_feeBPS);
312   emit ReflectionConfigSet(_feeBPS);
313 }
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Validate _feeBPS to ensure it falls within a reasonable range and prevent disruptions to tokenomics by adding the check in this smart contract or ReflectiveV3ERC20.sol

Status - Unresolved

A.4 Insufficient Validation for decimalsToSet [LOW]

Description:

The constructor only checks if decimalsToSet > 18 but does not enforce a minimum value, such as 1. Setting decimals to 0 or another non-standard value could cause compatibility issues with token standards.

Listing 4: FlexHype.sol

```
129 if (decimalsToSet > 18) {
130   revert InvalidDecimals(decimalsToSet);
131 }
```

Risk Level:

Likelihood – 1
Impact - 1

Recommendation:

Enforce a minimum decimal value of 1 and ensure compliance with ERC-20 standards for decimals.

Status - Unresolved

A.5 Code Repetition in transfer and transferFrom Functions [LOW]

Description:

The FlexHype contract contains repetitive code segments within the transfer and transferFrom functions. This repetition increases the complexity, size, and maintenance overhead of the contract. Additionally, repeated code can lead to potential bugs or inconsistencies when updating the logic, especially in critical functions like transfer and transferFrom.

Listing 5: FlexHype.sol

```
311  function transfer(
312    address to,
313    uint256 amount
314   ) public virtual override returns (bool) {
315    uint256 taxAmount = _taxAmount(msg.sender, to, amount);
316    uint256 deflationAmount = _deflationAmount(msg.sender, to, amount);
317    uint256 amountToTransfer = amount - taxAmount - deflationAmount;

319    if (isMaxAmountOfTokensSet() && !isFeesAndLimitsExcluded[to]) {
320      if (balanceOf(to) + amountToTransfer > maxTokenAmountPerAddress) {
321        revert DestBalanceExceedsMaxAllowed(to);
322      }
323    }

325    if (taxAmount != 0) {
326      _transferNonReflectedTax(msg.sender, taxAddress, taxAmount);
327    }
328    if (deflationAmount != 0) {
329      _burn(msg.sender, deflationAmount);
330    }
331    return super.transfer(to, amountToTransfer);
332   }

334   /// @notice Transfers tokens from one address to another
335   /// @dev Overrides the ERC20 transferFrom function with added tax and
        ↪ deflation logic
336   /// @param from The address which you want to send tokens from
337   /// @param to The address which you want to transfer to
338   /// @param amount The amount of tokens to be transferred
339   /// @return True if the transfer was successful
340   function transferFrom(
341    address from,
342    address to,
```

```
343       uint256 amount
344     ) public virtual override returns (bool) {
345       uint256 taxAmount = _taxAmount(from, to, amount);
346       uint256 deflationAmount = _deflationAmount(from, to, amount);
347       uint256 amountToTransfer = amount - taxAmount - deflationAmount;

349       if (isMaxAmountOfTokensSet() && !isFeesAndLimitsExcluded[to]) {
350         if (balanceOf(to) + amountToTransfer > maxTokenAmountPerAddress) {
351           revert DestBalanceExceedsMaxAllowed(to);
352         }
353       }

355       if (taxAmount != 0) {
356         _transferNonReflectedTax(from, taxAddress, taxAmount);
357       }
358       if (deflationAmount != 0) {
359         _burn(from, deflationAmount);
360       }

362       return super.transferFrom(from, to, amountToTransfer);
363     }
```

Risk Level:

Likelihood – 2

Impact - 1

Recommendation:

To improve readability and maintainability, refactor the common code segments within the transfer and transferFrom functions into a shared internal function.

Status - Unresolved

A.6 Missing Validation for Zero Address in excludeFromFeesAndLimits [LOW]

Description:

The excludeFromFeesAndLimits function does not validate the account parameter to ensure it is a non-zero address. If address(0) is passed as an argument, it could inadvertently or maliciously be added to the exclusion list, potentially leading to undefined or undesirable behavior.

Listing 6: FlexHype.sol

```
391 function excludeFromFeesAndLimits(
392     address account
393 ) external onlyOwner {
394     if (isFeesAndLimitsExcluded[account]) {
395         revert AlreadyExcludedFromFeesAndLimits();
396     }
397     if (feesAndLimitsExcluded.length >= MAX_EXCLUSION_LIMIT) {
398         revert ReachedMaxExclusionLimit();
399     }
401     isFeesAndLimitsExcluded[account] = true;
402     feesAndLimitsExcluded.push(account);
404     emit ExcludeFromFeesAndLimits(account);
405 }
```

Risk Level:

Likelihood – 2

Impact - 1

Recommendation:

Add a validation check to ensure the account parameter is not a zero address. This can be done using a require statement: solidity require(account != address(0), "Invalid account address");

Status - Unresolved

A.7 Unnecessary Function Overrides Without Modifications [LOW]

Description:

The contract overrides certain functions from its parent contracts without making any meaningful modifications to their behavior. This introduces unnecessary complexity and increases the gas cost of deploying the contract. For instance: 1. renounceOwnership: Simply calls the parent function without additional logic. 2. transferOwnership: Also calls the parent function without any changes.

Listing 7: FlexHype.sol

```
347 function renounceOwnership() public override onlyOwner {
348     super.renounceOwnership();
349 }

351 function transferOwnership(address newOwner) public override onlyOwner {
352     super.transferOwnership(newOwner);
353 }
```

Risk Level:

Likelihood – 1
Impact - 1

Recommendation:

Remove unnecessary overrides when the base implementation is sufficient. If specific behavior is required in the future, the functions can be overridden at that time. For example:

Status - Unresolved

A.8 Non-Adherence to Solidity Naming Conventions [INFORMATIONAL]

Description:

The FlexHype contract does not fully conform to the Solidity naming conventions as outlined in the Solidity style guide. Some variable names and function names deviate from the recommended guidelines, which can impact the readability, maintainability, and consistency of the codebase.

Key points from the Solidity style guide include:

- Use camelCase for function and variable names, starting with a lowercase letter (e.g., myVariable, updateCounter).
- Use PascalCase for contract, struct, and enum names, starting with an uppercase letter (e.g., MyContract, UserStruct, ErrorEnum).
- Use UPPERCASE for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).

Violations in the FlexHype contract include: - Variables such as _taxAddress, _feeBPS, _taxBPS, and _deflationBPS do not follow camelCase naming conventions.

Recommendation:

Adopt the Solidity naming conventions across the codebase to improve readability and maintainability. Update the variables mentioned above to conform to camelCase or UPPERCASE standards where applicable. For instance:

Listing 8: FlexHype.sol

```
48 address taxAddress; // Replace \_taxAddress
49 uint256 feeBPS; // Replace \_feeBPS
50 uint256 taxBPS; // Replace \_taxBPS
51 uint256 deflationBPS; // Replace \_deflationBPS
```

Status - Unresolved

4 Conclusion

We examined the design and implementation of FlexHype in this audit and found several issues of various severities. We advise FlexHype team to implement the recommendations contained in all 8 of our findings to further enhance the code's security. It is of utmost priority to start by addressing the most severe exploit discovered by the auditors then followed by the remaining exploits, and finally we will be conducting a re-audit following the implementation of the remediation plan contained in this report.

We would much appreciate any constructive feedback or suggestions regarding our methodology, audit findings, or potential scope gaps in this report.

BLOCKHAT
SECURITY

For a Smart Contract Audit, contact us at contact@blockhat.io