Veridise

# Auditing Report

## Hardening Blockchain Security with Formal Methods

FOR

STICKR

Veridise Inc.

Sept. 17, 2025

► **Prepared For:**

GumBall Labs
https://gum-ball.gitbook.io/gumball-protocol/

► **Prepared By:**

Ajinkya Rajput
Tyler Daimond

► **Contact Us:**

contact@veridise.com

► **Version History:**

October 22, 2025 V2
October 10, 2025 V1

# Contents



 

# 1 Executive Summary

From Sept. 17, 2025 to Sept. 26, 2025, GumBall Labs engaged Veridise to conduct a security assessment of their project, STICKR. The security assessment covered their factory contracts that deploy contracts for trading NFTs and an ERC20 contract that acts as an AMM that allows people to stake their tokens to earn rewards based on their staking. Veridise conducted the assessment over 16 person-days, with 2 security analysts reviewing the project over 8 days on commit `baf8251`. The review strategy involved a tool-assisted analysis of the program source code performed by Veridise security analysts as well as a thorough code review.

**Project Summary.** The security assessment covered STICKR which provides the implementations and factories for deploying ERC-20 and ERC-721 tokens. These tokens are intertwined within the STICKR ecosystem. The two main concepts these tokens represent are Boards and Content. A Board is deployed by means of an `ERC20` with a corresponding `ERC721` collection. The `ERC20` token represents shares in this board. The token includes its own bonding-curve style AMM in which the entire initial supply is owned by said AMM. A virtual reserve value is provided to set the initial price that tokens are sold at in terms of a quote token, with purchases following the standard Uniswap-V2 style swapping. The virtual reserves eliminate the requirement of upfront liquidity.

The Board is a place for users to post Stickers(Content), which are individual tokens of the corresponding ERC721 collection. This collection is unique in that someone can always purchase a user's Content for a roughly 10% premium over the price the user originally paid, and users do not themselves have the right to transfer their ownership. Part of this premium is sent back to the underlying ERC20 Token's bonding curve in order to boost the price of the shares of the Board. Additionally, a portion of every purchase's premium is sent to a `Rewarder` contract. This will distribute rewards in a Synthetix-style staking contract to holders of the Content, based on the value of the Content at the time of purchase. The remaining portion of this premium is sent to original creator of the content (along with the original purchase price).

Users can borrow Quote tokens against the ERC20 tokens that they hold. This value is based upon the floor price, which is the price that the last token of the circulating supply can be sold back to the AMM for. This floor value will only ever increase, therefore the AMM protects against bad debt being held.

Lastly, `Core`, `Router`, and `Multicall` contracts are provided for deploying tokens, interacting with the Board and Contents, and obtaining swapping and token info, respectively.

**Code Assessment.** The STICKR developers provided the source code of the STICKR contracts for the code review. The source code appears to be mostly original code written by the STICKR developers. To facilitate the Veridise security analysts' understanding of the code, the STICKR developers provided an article that describes the philosophy behind the design of the protocol, the designed tokenomics and mechanics of pricing. No documentation was provided on the code itself to the Veridise analysts for the assessment.

The source code contained a test suite, which the Veridise security analysts noted contains the unit tests for external calls. The test suite also contains integration tests that check for multiple

user interactions. The analysts noted that the values of calls were not checked for correctness, and negative tests were lacking.

**Summary of Issues Detected.** The security assessment uncovered 12 issues, 1 of which is assessed to be of high or critical severity by the Veridise analysts. Specifically, the issue V-STKR-VUL-001 identifies a denial of service attack vector where an attacker can render the boards usable by frontrunning the first transaction on the share tokens AMM. The Veridise analysts also identified 1 medium-severity issue, V-STKR-VUL-002, that points to missing slippage protection while collecting content. An attacker can make a user spend the entire approved allowance by frontrunning collect requests. Additionally, 4 low-severity issues and 6 warnings were identified. The STICKR developers have acknowledged 3 issues, have provided fixes for 8 issues and 2 issues are still open.

**Recommendations.** After conducting the assessment of the protocol, the security analysts had a few suggestions to improve the STICKR.

*Use Updated Dependencies.* The project heavily relies on OpenZeppelin contracts. The version currently used in the project is over two years out of date. The analysts recommend using a more recent version to ensure efficient and secure implementations of the token contracts. Note that this may require some refactoring of the code.

*Project Organization.* As mentioned, the Veridise analysts noted that the project contained no documentation. Additionally, interfaces that are used between components of the project are redefined in every file. This severely hinders the readability of the code, and increases the chance of interfaces becoming mismatched. The analysts highly recommend documenting all functionality of the code and standardizing the interfaces into a single file. Also, camelCase or snake_case should consistently be used, instead of both.

*Revamp Test Suite.* The test suite contains some positive tests to ensure functions do not revert. However, there is a lack of negative testing. Additionally, values are not checked to be mathematically correct, instead they are simply checked if they are equal or not to 0. For instance, testing a `buy()` on the Token AMM should check the output value, instead of just checking that the output value is greater than zero.

*Greater precision.* As mentioned in V-STKR-VUL-004, we recommend always using the greatest amount of precision possible. We extend this recommendation to the debt of each user in the Token AMM, as calculations are mostly done in wad precision anyways.

*Miscellaneous Recommendations.* The security analysis also recommend the following miscellaneous changes:

- Implement a `sync()` function similar to Uniswaps-V2 so that any Quote or Token sent to the AMM can be applied to the reserves.
- Some places use the `SafeMath` functions such as `divWadDown`, whereas others will manually perform a multiplication and divide by `PRECISION`. We recommend replacing all of these manual implementations with the `SafeMath` calls.
- Store frequently used variables inside of contracts. For example, the `Router` grabs the `quote` token from `core` on every `buy()` invocation.

**Disclaimer.** We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

# 2 Project Dashboard

**Table 2.1:** Application Summary.

| Name | Version | Type | Platform |
|---|---|---|---|
| STICKR | `baf8251` | Solidity | Ethereum |

**Table 2.2:** Engagement Summary.

| Dates | Method | Consultants Engaged | Level of Effort |
|---|---|---|---|
| Sept. 17–Sept. 26, 2025 | Manual & Tools | 2 | 16 person-days |

**Table 2.3:** Vulnerability Summary.

| Name | Number | Acknowledged | Fixed |
|---|---|---|---|
| Critical-Severity Issues | 1 | 0 | 0 |
| High-Severity Issues | 0 | 0 | 0 |
| Medium-Severity Issues | 1 | 1 | 1 |
| Low-Severity Issues | 4 | 4 | 3 |
| Warning-Severity Issues | 6 | 5 | 2 |
| Informational-Severity Issues | 0 | 0 | 0 |
| TOTAL | 12 | 10 | 6 |

**Table 2.4:** Category Breakdown.

| Name | Number |
|---|---|
| Logic Error | 6 |
| Data Validation | 5 |
| Maintainability | 1 |

# 3 Security Assessment Goals and Scope

## 3.1 Security Assessment Goals

The engagement was scoped to provide a security assessment of STICKR's source code. During the assessment, the security analysts aimed to answer questions such as:

- Does the protocol allow the all the user redeem all their shares?
- Does the invariant `maxSupply>=reserveTokenAmount` always hold for all the deployed ERC20 Tokens?
- Does the protocol distribute the pool in proportion to the shares held by users?
- Does the protocol implement the bonding curve math faithfully?
- Does the protocol allow an user to borrow more value than invested?
- Does the integration contract report correct value?
- Does the rewarder distribute the reward fairly?
- Does the protocol protect other users form malicious board owners?
- Can funds be locked in the protocol?
- Can a content owner evade collection by other users?
- Can a content owner transfer stickers outside of collect mechanism?
- Can an attacker exploit rounding errors to withdraw funds from shares of other users?

## 3.2 Security Assessment Methodology & Scope

**Security Assessment Methodology.** To address the questions above, the security assessment involved a combination of human experts and automated program analysis & testing tools. In particular, the security assessment was conducted with the aid of the following techniques:

- *Static analysis.* To identify potential common vulnerabilities, security analysts leveraged Veridise's custom smart contract analysis tool Vanguard. Vanguard tools are designed to find instances of common smart contract vulnerabilities, such as reentrancy, use-before-def and uninitialized variables.
- *Fuzzing/Property-based Testing.* Security analysts leveraged fuzz testing to determine if the protocol may deviate from the expected behavior. To do this, the desired behavior of the protocol was formulated as [V] specifications and then tested using Veridise's fuzzing framework OrCa to determine if a violation of the specification can be found.

*Scope*. The scope of this security assessment is limited to the `src/` folder of the source code provided by the STICKR developers, which contains the smart contract implementation of the STICKR.

*Methodology*. Veridise security analysts read the STICKR documentation and then began a review of the code assisted by both static analyzers and automated testing.

During the security assessment, the Veridise security analysts regularly met with the STICKR developers to ask questions about the code. Veridise security analysts also referenced tests in test folder to understand intended behavior of parts of code

## 3.3 Classification of Vulnerabilities

When Veridise security analysts discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise.

The severity of a vulnerability is evaluated according to the Table 3.1.

**Table 3.1:** Severity Breakdown.

| | Somewhat Bad | Bad | Very Bad | Protocol Breaking |
|---|---|---|---|---|
| Not Likely | Info | Warning | Low | Medium |
| Likely | Warning | Low | Medium | High |
| Very Likely | Low | Medium | High | Critical |

The likelihood of a vulnerability is evaluated according to the Table 3.2.

**Table 3.2:** Likelihood Breakdown

| | |
|---|---|
| Not Likely | A small set of users must make a specific mistake |
| Likely | Requires a complex series of steps by almost any user(s)<br>- OR -<br>Requires a small set of users to perform an action |
| Very Likely | Can be easily performed by almost anyone |

The impact of a vulnerability is evaluated according to the Table 3.3:

**Table 3.3:** Impact Breakdown

| | |
|---|---|
| Somewhat Bad | Inconveniences a small number of users and can be fixed by the user |
| Bad | Affects a large number of people and can be fixed by the user<br>- OR -<br>Affects a very small number of people and requires aid to fix |
| Very Bad | Affects a large number of people and requires aid to fix<br>- OR -<br>Disrupts the intended behavior of the protocol for a small group of users through no fault of their own |
| Protocol Breaking | Disrupts the intended behavior of the protocol for a large group of users through no fault of their own |

# 4 Trust Model

## 4.1 Operational Assumptions

In addition to assuming that any out-of-scope components behave correctly, Veridise analysts assumed that users will confirm the `Core` values, such as the `treasury` or `tokenFactory`, are correctly set.

## 4.2 Privileged Roles

**Roles.** This section describes in detail the specific roles present in the system, and the actions each role is trusted to perform. The roles are grouped based on two characteristics: privilege-level and time-sensitivity. *limited-authority* roles have a negative, but manageable impact if compromised. *Non-emergency* roles perform actions like deployments and configurations which can be planned several hours or days in advance.

During the review, Veridise analysts assumed that the role operators perform their responsibilities as intended. Protocol exploits relying on the below roles acting outside of their privileged scope are considered outside of scope.

- Limited-authority, non-emergency roles:
  - `ContentFactory.owner` can set if content submission is moderated.
  - `ContentFactory.owner` can set the moderators.
  - `ContentFactory.owner` can set the rewards available to holders.
  - `ContentFactory.owner` can set the uri of the ERC721.
  - `Core.owner` can set the treasury, tokenFactory, contentFactory and rewarderFactory.

**Operational Recommendations.** Highly-privileged, emergency operations should be tested in example scenarios to ensure the role operators are available and ready to respond when necessary.

Full validation of operational security practices is beyond the scope of this review. Users of the protocol should ensure they are confident that the operators of privileged keys are following best practices such as:

- Never storing a protocol key in plaintext, on a regularly used phone, laptop, or device, or relying on a custom solution for key management.
- Using separate keys for each separate function.
- Storing multi-sig keys in a diverse set of key management software/hardware services and geographic locations.
- Enabling 2FA for key management accounts. SMS should *not* be used for 2FA, nor should any account which uses SMS for 2FA. Authentication apps or hardware are preferred.
- Validating that no party has control over multiple multi-sig keys.
- Performing regularly scheduled key rotations for high-frequency operations.
- Securely storing physical, non-digital backups for critical keys.

- Actively monitoring for unexpected invocation of critical operations and/or deployed attack contracts.
- Regularly drilling responses to situations requiring emergency response such as pausing/unpausing.

# 5 Vulnerability Report

This section presents the vulnerabilities found during the security assessment. For each issue found, the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.) is specified. Table 5.1 summarizes the issues discovered:

**Table 5.1:** Summary of Discovered Vulnerabilities.

| ID | Description | Severity | Status |
|---|---|---|---|
| V-STKR-VUL-001 | Burning tokens can lead to Denial-of-Service | Critical | Open |
| V-STKR-VUL-002 | No slippage protection during collections | Medium | Fixed |
| V-STKR-VUL-003 | Zero values reported when totalSupply is zero | Low | Fixed |
| V-STKR-VUL-004 | Dust becomes stuck in Rewarder | Low | Fixed |
| V-STKR-VUL-005 | Wrong values calculated in buyTokenOut() | Low | Confirming Fix |
| V-STKR-VUL-006 | Content assumes decimal value | Low | Fixed |
| V-STKR-VUL-007 | Maintainability Issues | Warning | Partially Fixed |
| V-STKR-VUL-008 | Missing Validations of immutable variables | Warning | Confirming Fix |
| V-STKR-VUL-009 | Incorrect comparison leads to spurious reverts | Warning | Fixed |
| V-STKR-VUL-010 | Unconventional tokens may break accounting | Warning | Acknowledged |
| V-STKR-VUL-011 | Mismatched fee calculations | Warning | Fixed |
| V-STKR-VUL-012 | Asymptotic behavior of protocol | Warning | Open |

## 5.1 Detailed Description of Issues

### 5.1.1 V-STKR-VUL-001: Burning tokens can lead to Denial-of-Service

| | | | |
|---|---|---|---|
| **Severity** | Critical | **Commit** | baf8251 |
| **Type** | Logic Error | **Status** | Open |
| **Location(s)** | `src/TokenFactory.sol:320-331` | | |
| **Confirmed Fix At** | N/A | | |

The `_burnTokenReserves()` function is used to adjust the `maxSupply` and `reserveTokenAmt` when a user tokens are burnt. The goal of this function is to preserve the ratio of circulating Tokens to Tokens in the AMM before and after the burn.

```
function _burnTokenReserves(uint256 tokenAmt) internal {
    uint256 m = maxSupply;
    uint256 y = reserveTokenAmt;
    if (m <= y) revert Token__InvalidShift();

    uint256 reserveBurn = y.mulWadDown(tokenAmt).divWadDown(m - y);

    reserveTokenAmt -= reserveBurn;
    maxSupply -= (tokenAmt + reserveBurn);

    emit Token__BurnReserves(tokenAmt, reserveBurn);
}
```

**Snippet 5.1:** Snippet from `TokenFactory.sol:_burnTokenReserves()`

As can be seen, the `reserveBurn` variable is assigned the value `y*tokenAmt/(m-y)`. Therefore, the larger proportion a user owns of the circulating supply, the greater effect they have on reducing the `maxSupply` and `reserveTokenAmt` permanently. In the worst case scenario, a user that owns and burns the full circulating supply will essentially break the AMM and prevent any purchases of Token in the future. If an attacker immediately purchases and burns Token after deployment, they can immediately break the AMM.

For example, assume a Token is deployed with a `maxSupply` of 100 (ignoring decimals). After an attacker purchases 1 Token, the `reserveTokenAmt` will become 99. The call to `burn()` with 1 Token will lead to the `reserveBurn` variable being calculated as `99*1/(100-99) = 99` and therefore `reserveTokenAmt` and `maxSupply` will become 0.

**Impact** A user that burns the full circulating supply will lead to a irrecoverable DoS of the deployed Token. Additionally, early holders have a disproportional amount of power in regards to manipulating the supply of the Token.

**Recommendation** Rework or remove the burning mechanism.

**Developer Response** Developers enforce the board creators to buy and lock in `CORE_TOKEN_REQUIRED` number of tokens.

**Updated Veridise Response** The auditors derived the following minimum safe value for `CORE_TOKEN_REQUIRED` parameterized over

- maximum reduction in `sharesWithContract` by a factor `f`
- the amount the attacker has to spend to execute this attack `V`

```
CORE_TOKEN_REQUIRED >=  V.10^4/(f-1)
```

This value is derived as follows

First, Sticker buys L shares and locks them.

```
Y_max = InitMax
sharesWithContract = InitMax - L
Y_circulation = L
```

Next, the attacker buys x shares as the second buyer.

```
Y_max = InitMax
sharesWithContract = InitMax - (L + x)
Y_circulation = x + L
```

If the attacker burns all x shares, the burn amount is:

```
burn amount = x * sharesWithContract / (Y_max - sharesWithContract)
substitute Y_max - sharesWithContract = x + L
burn amount = sharesWithContract * x / (x + L)
```

After the burn, we can compute the new `sharesWithContract`

```
sharesWithContract_new = sharesWithContract - burn amount
                       = sharesWithContract - sharesWithContract * x / (x + L)
                       = sharesWithContract * (1 - x/(x + L))
                       = sharesWithContract * L / (x + L)
```

The initial price per share is

```
price_per_share = 0.0001$
```

The defense should require the attacker can reduce `sharesWithContract` by at most a factor `f`. So,

```
sharesWithContract_new >= sharesWithContract / f
sharesWithContract * L / (x + L) >= sharesWithContract / f
# divide both sides by sharesWithContract (>0):
L / (x + L) >= 1 / f
# multiply both sides by (x + L):
L >= (x + L) / f
# rearrange:
L * f >= x + L
L * f - L >= x
L * (f - 1) >= x
L >= x / (f - 1)
```

The attacker is assumed to spend `V` to purchase `x` shares, so:

```
x = V.10^4 // approximately
```

therefore:

```
L >= (V.10^4) / (f - 1)
```

From this, one may conclude that the final safety bound for `CORE_TOKEN_REQUIRED` must satisfy

```
CORE_TOKEN_REQUIRED >= V.10^4 / (f - 1)
```

The developers should choose a reasonable `V` (attacker cost) and `f` (acceptable reduction factor), then set `CORE_TOKEN_REQUIRED` accordingly.

### 5.1.2 V-STKR-VUL-002: No slippage protection during collections

| | | | |
|---|---|---|---|
| **Severity** | Medium | **Commit** | baf8251 |
| **Type** | Logic Error | **Status** | Fixed |
| **Location(s)** | `src/ContentFactory.sol:97-126` | | |
| **Confirmed Fix At** | 7c14fa3 | | |

The `collect()` function is used to forcibly buy the `tokenId` ERC721 NFT from the current holder. This is done by calculating the `nextPrice` from the `getNextPrice()` function, which takes into account the current price. If the user has given infinite approval to the contract (as is common to avoid the gas cost of calling `approve()` for every invocation) then he will pay whatever value `nextPrice` is in order to purchase the NFT. Therefore this function is susceptible to frontrunning.

**Impact** An attacker that observes a call to `collect()` may frontrun the purchase in order to instantly profit from the increase in price. The attacker may purchase the NFT from themselves many times in order to inflate the `nextPrice` in order to drain as much as possible from the approved funds to this contract, although this may not be profitable to the attacker.

**Recommendation** Include a function parameter to `collect()` that enforces the maximum value a user is willing to pay.

**Developer Response** The developers now include a `maxPrice` parameter in which the `nextPrice` cannot be greater than `maxPrice`.

### 5.1.3 V-STKR-VUL-003: Zero values reported when totalSupply is zero

| | | | |
|---|---|---|---|
| **Severity** | Low | **Commit** | baf8251 |
| **Type** | Logic Error | **Status** | Fixed |
| **Location(s)** | `src/Multicall.sol:165-167, 202-206` | | |
| **Confirmed Fix At** | aa580fc | | |

Various fields of the `getTokenData()` and `getContentData()` are assigned based on the condition of whether the `totalSupply` is 0 or not.

```
uint256 contentQuoteRewardForDuration =
    totalContentStaked == 0 ? 0 : IToken(token).rawToWad(IRewarder(rewarder).
    getRewardForDuration(quote));
uint256 contentTokenRewardForDuration =
    totalContentStaked == 0 ? 0 : IRewarder(rewarder).getRewardForDuration(token);
```

**Snippet 5.2:** Snippet from `Multicall.sol:getTokenData()`

This is not correct, as rewards can be provided to the Rewarder contract even if there are no deposits.

**Impact** Incorrect values will be returned from these functions when the `totalSupply` is 0.

**Recommendation** Remove the ternary operators for fields that still have values even when the `totalSupply` is 0.

**Developer Response** The developers now calculate rewards regardless if the `totalContentStaked` is 0.

### 5.1.4 V-STKR-VUL-004: Dust becomes stuck in Rewarder

| | | | |
|---|---|---|---|
| **Severity** | Low | **Commit** | baf8251 |
| **Type** | Logic Error | **Status** | Fixed |
| **Location(s)** | `src/RewarderFactory.sol:87-103` | | |
| **Confirmed Fix At** | 2ed6912 | | |

When an `amount` of `token` is added to the Rewader. its `rewardRate` is calculated as `amount / DURATION`. Although `amount` is guaranteed to be greater than `DURATION` in order to prevent rounding down to 0, this calculation will still always round down. Therefore, any non-multiple of `DURATION` will lead to up to `DURATION-1` dust being unclaimed by stakers. If the given `token` has a low amount of decimals, this may be a significant amount of money.

**Impact** Up to `DURATION-1` dust may become stuck in the `Rewarder`, unclaimable by users. Additionally, tokens with a low amount of decimals may be prevented from topping up rewards due to the `DURATION`-sized barrier.

**Recommendation** Store the reward information for tokens and accounts in WAD for higher precision. This should also allow the `amount > DURATION` check to be removed.

**Developer Response** The developers now store the `rewardRate` in terms of WAD, and have modified the other functions to take this into account.

### 5.1.5 V-STKR-VUL-005: Wrong values calculated in buyTokenOut()

| | | | |
|---|---|---|---|
| **Severity** | Low | **Commit** | baf8251 |
| **Type** | Logic Error | **Status** | Confirming Fix |
| **Location(s)** | `src/Multicall.sol:269` | | |
| **Confirmed Fix At** | 0eb979f, 0eb979f | | |

The `buyTokenOut()` function calculates the amount of `quoteWadIn` required for the given `tokenAmtOut` as seen below

```
uint256 xv = IToken(token).reserveVirtQuoteWad();
uint256 xr = IToken(token).reserveRealQuoteWad();
uint256 x0 = xv + xr;
uint256 y0 = IToken(token).reserveTokenAmt();

if (tokenAmtOut > y0) return (0, 0, 0, 0);

uint256 quoteWadIn = DIVISOR.mulDivDown(x0.mulDivDown(y0, y0 - tokenAmtOut) - x0,
    DIVISOR - FEE);
/// Veridise elided
```

**Snippet 5.3:** Snippet from `Multicall.sol:buyTokenOut()`:

This calculation can be viewed as taking the floor of `t=(x0*y0)/(y0-tokenAmtOut)` and then, again, the floor of `(DIVISOR*t)/(DIVISOR-FEE)`. However, the calculation in the `Token` AMM uses the ceiling function instead:

```
feeRaw = (quoteRawIn * FEE) / DIVISOR;
uint256 netRaw = quoteRawIn - feeRaw;
uint256 netWad = rawToWad(netRaw);

uint256 y0 = reserveTokenAmt;
uint256 x0 = reserveVirtQuoteWad + reserveRealQuoteWad;
uint256 x1 = x0 + netWad;
if (x1 == 0) revert Token__DivideByZero();

uint256 y1 = x0.mulWadUp(y0).divWadUp(x1);
tokenAmtOut = y0 - y1;
```

**Snippet 5.4:** Snippet from `TokenFactory.sol:_processBuy()`:

A similar issue exists in `sellTokenIn()`.

**Impact** The incorrect value may be calculated in `buyTokenOut()` and `sellTokenIn()`. This may lead to overpayment or reverting swaps.

**Recommendation** Match the Multicall and Token AMM implementations.

**Developer Response** Developers have removed the `buyTokenOut()` function but `sellTokenIn()` is still present

### 5.1.6 V-STKR-VUL-006: Content assumes decimal value

| | | | |
|---|---|---|---|
| **Severity** | Low | **Commit** | baf8251 |
| **Type** | Data Validation | **Status** | Fixed |
| **Location(s)** | `src/TokenFactory.sol:217` | | |
| **Confirmed Fix At** | 6c61f8d | | |

Whenever a content is created, the owner deploys it for free. Then following deployment, each subsequent buy is more expensive than the last.The `getNextPrice()` function calculates the value that the next purchaser must pay. It does so by increasing the previous price by 10% and adding in 1e6.

```
function getNextPrice(uint256 tokenId) public view returns (uint256) {
    return (id_Price[tokenId] * 11) / 10 + 1e6;
}
```

**Snippet 5.5:** Snippet from `TokenFactory.sol:getNextPrice()`

The value of 1e6 is hardcoded, as the developers intend for the underlying `quote` token to be USDC (which has 6 decimals). However, there is no requirement that is the case. Therefore, the constant used for increasing the price may be inaccurate in regards to the underlying quote.

**Impact** The value may increase much slower or quicker than anticipated.

**Recommendation** Assign an immutable variable in the constructor to determine what the constant in this function should be. This could either be based on the underlying decimals of the quote token, or a value directly provided to the constructor.

**Developer Response** The developers have replaced the `1e6` hardcode with an `initialPrice` that is set during construction.

### 5.1.7 V-STKR-VUL-007: Maintainability Issues

| | | | |
|---|---|---|---|
| **Severity** | Warning | **Commit** | baf8251 |
| **Type** | Maintainability | **Status** | Partially Fixed |
| **Location(s)** | `src/RewarderFactory.sol:96` | | |
| **Confirmed Fix At** | 7be082f | | |

The security analysts identified the following places that may lead to issues in maintaining and understanding the code

#### Duplicate code and definitions

1. Many definitions are redundant. Each file defines interfaces for interacting with the other contracts. Put the interfaces in their own file, and import those instead.
2. Global variables are also redundantly defined, such as `FEE` or `DIVISOR`. These should also be put into a file that is used by all contracts.
3. `Rewarder`: The `getRewardTokens()` is unnecessary, as the array is public and therefore the contract will have two getter methods for the array. If this specific function name is desired, then one can rename the array to the desired name.
4. `Rewarder:notifyRewardAmount()`: The else block of the if statement mirrors the functionality defined in `left()`. Instead of reimplementing this, one could get the value from `left()` and simply set the reward rate as `(amount + left())/DURATION`.

#### Unneeded implementations

1. Many contracts, such as the `ContentFactory` and `TokenFactory` implement override methods that do nothing but call the `super` version of the method. These are unnecessary should be removed.
2. `ContentFactory:collect()`: The `(surplus*3)/9` computations should just be `surplus/3`

    a) Additionally, the amount of tokens sent to the `heal()` method should instead be `surplus-surplus*2/3` to avoid unused dust.

3. `Token`: The `getMarketPrice()` and `getFloorPrice()` methods include an unnecessary computation that multiplies a number by `1e18` while simultaneously dividing by `1e18`. Remove this computation.

#### Other

1. `Multicall:getTokenData()`: Use the `DURATION` variable instead of hardcoding the `7` value
2. `Content` disables the public transfer methods, but does not disable approvals. Disabling approvals will provide more defense-in-depth.
3. `Router` only allows setting the affiliate of a user once. This means that a compromised affiliate cannot be changed. The user should be able to change their affiliate whenever they wish.

**Impact** The maintainability and the ability to understand the code becomes more difficult, and bugs may be introduced in the future.

**Recommendation** Implement the recommendations for each issue.

**Developers Response** Developers have partially fixed this issue

### 5.1.8 V-STKR-VUL-008: Missing Validations of immutable variables

| | | | |
|---|---|---|---|
| **Severity** | Warning | **Commit** | baf8251 |
| **Type** | Data Validation | **Status** | Confirming Fix |
| **Location(s)** | `src/`<br>▶ `ContentFactory.sol` | | |
| **Confirmed Fix At** | 8f2aa8d | | |

**Description** Validations are missing for many important variables in the following locations

1. TokenFactory.sol
   :131: Missing non zero validation for destination address argument `to` of function `buy()`
2. TokenFactory.sol
   :151: Missing non zero validation for destination address argument `to` of function `sell()`
3. TokenFactory.sol
   :169: Missing non zero validation for destination address argument `to` of function `borrow()`
4. TokenFactory.sol
   :180: Missing non zero validation for borrower address for which the repayment is done argument `to` of function `repay()`
5. ContentFactory.sol: 75: The state variable `uri` is set without performing validations on the argument `_uri`.
6. ContentFactory.sol: 76:
   The state variable `token` is set without performing validations on the argument `_token`.
7. ContentFactory.sol: 77:
   The state variable `quote` is set
   without performing validations on the argument `_quote`.
8. ContentFactory.sol: 168:
   The state variable `uri` is set without performing validations on the argument `_uri`.
9. Core.sol:63:
   The state variable `quote` is set without performing validations on the argument `_quote`.
10. Core.sol:64, 103:
    The state variable `tokenFactory` is set without performing validations on the argument `_tokenFactory`.
11. Core.sol:65, 108:
    The state variable `contentFactory` is set without performing validations on the argument `_contentFactory`.
12. Core.sol:66, 113 :
    The state variable `rewarderFactory` is set without performing validations on the argument `_rewarderFactory`.
13. Core.sol:98:
    The state variable `treasury` is set without performing validations on the argument `_treasury`.
14. TokenFactory.sol
    :114:
    The state variable `core` is set without performing validations on the argument `_treasury`.

**Impact** Not having proper validations may allow misconfigurations and may require additional actions from developers.

**Recommendations** Implement the suitable validations for the locations mentioned above.

**Developers Response** Developers have implemented fixes for 1-4

### 5.1.9 V-STKR-VUL-009: Incorrect comparison leads to spurious reverts

| | | | |
|---|---|---|---|
| **Severity** | Warning | **Commit** | baf8251 |
| **Type** | Data Validation | **Status** | Fixed |
| **Location(s)** | `src/Multicall.sol:267` | | |
| **Confirmed Fix At** | 0eb979f | | |

The `buyTokenOut()` method calculates the required `quoteRawIn` in order to receive the requested `tokenAmtOut` from the `token`'s AMM. The calculation of this can be seen below:

```
/// Veridise elided
if (tokenAmtOut > y0) return (0, 0, 0, 0);
uint256 quoteWadIn = DIVISOR.mulDivDown(x0.mulDivDown(y0, y0 - tokenAmtOut) - x0,
    DIVISOR - FEE);
/// Veridise elided
```

**Snippet 5.6:** Snippet from `Multicall.sol:buyTokenOut()`

If the requested `tokenAmtOut` is equal to the entire reserve of the AMM, then a divide by zero will occur due to the `y0 - tokenAmtOut` and the call will revert.

**Impact** Attempts to buy the residing Token supply in the AMM will revert.

**Recommendation** Change the if statement's comparison to `tokenAmtOut >= y0`.

**Developer Response** The `buyTokenOut()` function has been removed.

### 5.1.10 V-STKR-VUL-010: Unconventional tokens may break accounting

| | | | |
|---|---|---|---|
| **Severity** | Warning | **Commit** | baf8251 |
| **Type** | Data Validation | **Status** | Acknowledged |
| **Location(s)** | `src/RewarderFactory.sol:92` | | |
| **Confirmed Fix At** | N/A | | |

The `Rewarder` contract allows the `Content` add any token as a reward token. The `nofityRewardAmount()` function in the `Rewarder` contract transfers an `amount` of tokens to the contract for distribution. The `rewardRate` of the token is therefore dependent on this value. The contract assumes that the `safeTransferFrom()` call of the `token` will transfer the exact amount of tokens to this contract. However, non-standard tokens such as those with a fee-on-transfer or rebasing will break the accounting that this contract relies on.

**Impact** Users may be prevented from claiming their rewards, requiring manual intervention in the form of sending tokens directly to the contract.

**Recommendation** If fee-on-transfer tokens may be used, then calculate rewards based on the change of the contract's balance before and after transferring. Additionally, document that rebasing tokens (or additionally, fee-on-transfer tokens) are strictly not supported.

**Developers Response** Developers have acknowledged this issue

### 5.1.11 V-STKR-VUL-011: Mismatched fee calculations

| | | | |
|---|---|---|---|
| **Severity** | Warning | **Commit** | baf8251 |
| **Type** | Logic Error | **Status** | Fixed |
| **Location(s)** | `src/`<br>▶ `Multicall.sol:227-353` | | |
| **Confirmed Fix At** | fb84f1a | | |

In `Token`, the `_processBuy()` function calculates the charged fee Quote fee in terms of raw. The `_processSell()` function calculates the charged Token fee in terms of wad (as Token is wad). Conversely, all functions in `Multicall` calculate the fee in terms of wad. This mismatch for buys may lead to slightly incorrect values being calculated.

**Impact** The slightly incorrect fee may be used during Multicall calculations, and therefore an overpayment may occur.

**Recommendation** In `_processBuy()` calculate the fee in wad before converting to raw.

**Developer Response** The developers changed the `Multicall` fee calculation to use raw.

### 5.1.12 V-STKR-VUL-012: Asymptotic behavior of protocol

| | | | |
|---|---|---|---|
| **Severity** | Warning | **Commit** | baf8251 |
| **Type** | Data Validation | **Status** | Open |
| **Location(s)** | `src/`<br>▶ `ContentFactory.sol` | | |
| **Confirmed Fix At** | N/A | | |

**Description** **Bonding curve burns and heals**

The `TokenFactory` contract deploys AMMs on bonding curve. The bonding curve has the shares owned by contract on y-axis and the amount of backing token on x-axis. The initial point is chosen by a fixed amount of virtual backing token and all shares owned by the contract. The curve shifts as a result of two operations: burn and heal. The heal shift occurs whenever backing tokens are transferred to the contract, while burn shift occurs shares are burnt by a user. Both these shifts increase the value of shares.

Whenever the shares are burnt the total number of shares in circulation are decreased by

`(shares_held_by_contract/shares_in_circulation) . burn_amount` Since the total shares always decreases and never increases, asymptotically the total shares will reach a very low numerical value. This will result in rounding errors when any further operations are performed.

**Monotonic increase in price of content**

The price for collecting the content for the first time is 1$ and the price for every subsequent collection in `previous_price * 1.1 + 1`. Therefore, the price of the content always increases. Asymptotically, this price may become so high that the content may not be collected further.

**Impact**

- Over a sufficient long period of time, rounding errors may become large
  Over a sufficient long period of time, t
  he content may become prohibitively expensive

**Developers Response** The developers are planning to update the content collection logic.

# 6 Fuzz Testing

## 6.1 Methodology

One of the goals of the security assessment was to fuzz test STICKR to evaluate correctness of ERC20 `Token` tokenomics and operations.

The fuzzing goals included checking that

- The bonding curve shifts (heals and burnTokenReserves) do not violate identified invariants
- ERC20 transfer, burn and approve mechanisms work as intended

The Veridise security analysts used the OrCa tool to fuzz the project. The analysts captured the intended behavior of the system by writing invariants as logical formulae which should hold after each transaction. The invariants were encoded as statements in the [V] specification language and passed to the OrCa tool, together with the project program code.

## 6.2 Properties Fuzzed

Table 6.1 describes the fuzz-tested invariants. The second column describes the invariant informally in English, and the third shows the total amount of compute time spent fuzzing this property. The last column indicates the number of bugs identified when fuzzing the invariant.

The Veridise team devoted a total of 1 compute-hours to fuzzing this protocol, identifying a total of 0 bugs.

**Table 6.1:** Invariants Fuzzed.

| Specification | Invariant | Minutes Fuzzed | Bugs Found |
|---|---|---|---|
| V-STKR-SPEC-001 | Stickr maxSupply less than reserveTokens | 60 | 0 |

## 6.3 Detailed Description of Fuzzed Specifications

### 6.3.1 V-STKR-SPEC-001: Stickr maxSupply less than reserveTokens

| Minutes Fuzzed | 60 | Bugs Found | 0 |
|---|---|---|---|

#### Specification

**Scope** The ERC20 tokens that represent shares of a board of STICKR.

**Natural Language** The ERC20 tokens is initialized with a `maxSupply` amount of tokens. The tokens owned by ERC20 contract are tracked by storage variable `reserveTokenAmt`. Initially, the ERC20 contract owns all of these tokens. As the tokens are bought the `reserveTokenAmt` is decreased and when the tokens are sold back to the protocol, the `reserveTokenAmt` is increased and also, the curve shifts by decreasing `maxSupply`

**Formal**

```
vars: Token token
spec: []!finished(token.*, token.reserveTokenAmt > token.maxSupply || token.
    totalSupply() > token.maxSupply)
```

# A Appendix

## A.1 Intended Behavior: Non-Issues of Note

### A.1.1 V-STKR-APP-VUL-001: minTradeSize uses the same value for both buys and sells

| | | | |
|---|---|---|---|
| **Severity** | Warning | **Commit** | baf8251 |
| **Type** | Data Validation | **Status** | Intended Behavior |
| **Location(s)** | `src/TokenFactory.sol:96-99` | | |

When determining if a swap is valid, the `MIN_TRADE_SIZE` determines the minimum amount that can be exchanged. However, this value is applied to the input number of QUOTE for buys and the input number of TOKEN for sells. Given that these tokens have different values and decimals, the trade size allows buys or sellers smaller than intended. Additionally, this constant assumes a certain decimal value which may not correspond to either token.

**Impact** Buys or sells that are below the intended threshold are allowed.

**Recommendation** Have a separate variable for buys and sells, and base it off of each input's decimals.

**Developer Response** This value is irrelevant to decimals that the token has. It's just set so that when fees get processed there is enough precision so that the amounts don't come out to 0.