shieldify

GlueX

SECURITY REVIEW

Date: 3 July 2025

CONTENTS

1. About Shieldify

Positioned as the first hybrid Web3 Security company, Shieldify shakes things up with a unique subscription-based auditing model that entitles the customer to unlimited audits within its duration, as well as top-notch service quality thanks to a disruptive 6-layered security approach. The company works with very well-established researchers in the space and has secured multiple millions in TVL across protocols, also can audit codebases written in Solidity, Rust, Go, Vyper, Move and Cairo.

Learn more about us at shieldify.org.

2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

3. About GlueX

GlueX provides a unified suite of on-chain and off-chain services designed to abstract the complexities of decentralized finance (DeFi). It standardizes the integration of different blockchains, exchange venues, and liquidity providers into a common framework to simplify, optimize and expedite your access to DeFi.

Want to learn more about the inner workings of GlueX Protocol? Dive into
[GlueX Protocol documentation](https://docs.gluex.xyz/gluex-protocol).

4. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4.1 Impact

- **High** – results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** – results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** – losses will be limited but bearable – and covers vectors similar to griefing attacks that can be easily repaired

4.2 Likelihood

- **High** – almost certain to happen and highly lucrative for execution by malicious actors

- **Medium** – still relatively likely, although only conditionally possible
- **Low** – requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

5. Security Review Summary

The security review lasted 2 days, with a total of 16 hours dedicated to the audit by one researcher from the Shieldify team.

The audit report identified three High, two Medium and one Low severity findings. Mainly related to insufficient surpluses, flawed deadline setting and missing centralization validation checks.

The protocol's team has done a great job providing support and responses to all of the questions that the Shieldify researchers had.

5.1 Protocol Summary

Project Name	**GlueX Protocol**
Repository	gluex-router-contract
Type of Project	DeFi, Router
Audit Timeline	2 days
Review Commit Hash	9c754a3985fa32b72d847c88c83575f29a86bc01
Fixes Review Commit Hash	288c29f528abba1755b1581be402cfd8fa242b70

5.2 Scope

The following smart contracts were in the scope of the security review:

File	**nSLOC**
router_v1/GluexRouter.sol	232
Total	**232**

6. Findings Summary

The following number of issues have been identified, sorted by their severity:

- **High** issues: **3**
- **Medium** issues: **2**
- **Low** issues: **1**

ID	Title	Severity	Status
[H-01]	Protocol Can Receive Zero Shares Despite Surplus	High	Fixed
[H-02]	Partner Can Receive Zero Share Despite Surplus	High	Acknowledged
[H-03]	Swap Can Execute After Deadline Expires	High	Acknowledged
[M-01]	`_MAX_FEE` Can be Greater Than `100` Percentage	Medium	Fixed
[M-02]	`_MIN_FEE` Can be Greater Than `_MAX_FEE`	Medium	Fixed
[L-01]	Missing Zero Address Validation	Low	Acknowledged

7. Findings

[H-01] Protocol Can Receive Zero Shares Despite Surplus

Severity

High Risk

Description

The contract computes `protocolShare` by subtracting `partnerShare` from `surplus` and adding a portion of `slippage` based on `protocolSlippageShare`. However, there is no enforcement of a minimum threshold for `protocolSlippageShare`, allowing it to be zero or arbitrarily small. This may result in the protocol receiving negligible or no compensation from a transaction that generates a valid surplus and slippage, undermining protocol sustainability and revenue assumptions.

Location of Affected Code

File: router_v1/GluexRouter.sol#L230-L232

```
function swap(
    IExecutor executor,
    RouteDescription calldata desc,
    Interaction[] calldata interactions
) external payable returns (uint256 finalOutputAmount, uint256 surplus,
   uint256 slippage) {
// code
  uint256 protocolSurplus = surplus - partnerShare;
  uint256 protocolSlippage = slippage * desc.protocolSlippageShare) /
     10000;
  uint256 protocolShare = protocolSurplus + protocolSlippage;
// code
}
```

Impact

The protocol may be deprived of its expected revenue from transaction surplus and slippage, weakening its economic model and ability to fund operations or rewards.

Recommendation

Enforce a minimum threshold for `protocolSlippageShare` to ensure consistent protocol compensation.

```
+ require(desc.protocolSlippageShare >= MIN_SHARE, "protocol share too
    low");
```

Team Response

Fixed.

[H-02] Partner Can Receive Zero Share Despite Surplus

Severity

High Risk

Description

The contract calculates `partnerShare` based on the `partnerSurplusShare` and `partnerSlippageShare` percentages provided in the `desc` struct. However, it does not enforce a minimum threshold for these shares, allowing them to be set to zero or an insignificantly low value. This undermines the expected incentive mechanism for partners who facilitate transactions. Without enforcing a floor, a misconfigured or malicious `desc` input can completely exclude a partner from their entitled earnings even when surplus or slippage exists.

Location of Affected Code

File: router_v1/GluexRouter.sol#L223-L227

```
function swap(
    IExecutor executor,
    RouteDescription calldata desc,
    Interaction[] calldata interactions
) external payable returns (uint256 finalOutputAmount, uint256 surplus,
   uint256 slippage) {
// code
  if (surplus > 0 || slippage > 0) {
    uint256 partnerSurplus = (surplus * desc.partnerSurplusShare) /
       10000;
    uint256 partnerSlippage = (slippage * desc.partnerSlippageShare) /
       10000;
    uint256 partnerShare = partnerSurplus + partnerSlippage;
  }
// code
}
```

Impact

Partners may receive no compensation for their participation, reducing incentives for integration and potentially breaching contractual agreements with third-party stakeholders.

Recommendation

Enforce a minimum threshold for `partnerSurplusShare` and `partnerSlippageShare` to guarantee non-zero rewards.

```
+ require(desc.partnerSurplusShare >= MIN_SHARE && desc.
    partnerSlippageShare >= MIN_SHARE, "partner share too low");
```

Team Response

Acknowledged.

[H-03] Swap Can Execute After Deadline Expires

Severity

High Risk

Description

The `swap()` function lacks a deadline enforcement mechanism, despite the official documentation explicitly requiring that swaps revert if the current block timestamp exceeds a specified deadline.

> Reverts: Deadline passed: If the block timestamp exceeds the deadline.

This missing check allows transactions to be mined at any time in the future, potentially after market conditions have changed or slippage assumptions are no longer valid. Such behavior undermines user protection against stale or manipulated execution.

Location of Affected Code

File: router_v1/GluexRouter.sol#L167-L171

Impact

Attackers or miners can intentionally delay transaction inclusion and front-run or back-run users, leading to unfair execution prices or failed economic assumptions. This introduces both financial and reputational risk.

Recommendation

Add a deadline check at the start of the swap function to revert if the current timestamp exceeds the allowed window.

```
+ require(block.timestamp <= desc.deadline, "deadline passed");
```

Team Response

Acknowledged.

[M-01] `_MAX_FEE` Can be Greater Than `100` Percentage

Severity

Medium Risk

Description

The `setMaxFee()` functions allow the `treasury` to update `_MAX_FEE`. The `setMaxFee()` permits values exceeding 100% (i.e., >10000 basis points), which can lead to economically irrational fees.

Location of Affected Code

File: router_v1/GluexRouter.sol#L392

```
function setMaxFee(uint256 maxFee) external onlyTreasury {
    _MAX_FEE = maxFee;
}
```

Impact

The protocol may charge excessive or invalid fees, leading to transaction reverts, economic failure, or user loss of trust.

Recommendation

Enforce that `maxFee` does not exceed `10000`

```
function setMaxFee(uint256 maxFee) external onlyTreasury {
+   require(maxFee <= 10000, "max fee cannot exceed 100%");
    _MAX_FEE = maxFee;
}
```

Team Response

Fixed.

[M-02] `_MIN_FEE` Can be Greater Than `_MAX_FEE`

Severity

Medium Risk

Description

The `setMinFee()` function does not ensure that the new minimum is less than or equal to `_MAX_FEE`, allowing a state where `_MIN_FEE > _MAX_FEE`. This misconfiguration can render the fee logic unusable, result in user lockout, or violate economic assumptions.

Location of Affected Code

File: router_v1/GluexRouter.sol#L401

```
function setMinFee(uint256 minFee) external onlyTreasury {
    _MIN_FEE = minFee;
}
```

Impact

Incorrect fee bounds also break any logic that assumes valid fee ranges.

Recommendation

Ensure that `minFee` does not exceed `_MAX_FEE`.

```
function setMinFee(uint256 minFee) external onlyTreasury {
+   require(minFee <= _MAX_FEE, "min fee cannot exceed max fee");
    _MIN_FEE = minFee;
}
```

Team Response

Fixed.

[L-01] Missing Zero Address Validation

Severity

Low Risk

Description

The contracts were found to be setting immutable addresses without proper validations for zero addresses.

Location of Affected Code

File: router_v1/GluexRouter.sol#L106

```
constructor(address gluexTreasury, address nativeToken) {
// Ensure the addresses are not zero
    checkZeroAddress(gluexTreasury);

    _gluexTreasury = gluexTreasury;
@>> _nativeToken = nativeToken;
}
```

Impact

If address type parameters do not include a zero-address check, contract functionality may become unavailable.

Recommendation

Add a zero address validation to all the functions where addresses are being set.

Team Response

Acknowledged.

shieldify

Thank you!