shieldify

GlueX Protocol

SECURITY REVIEW

Date: 27 December 2024

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 Protocol

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.

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 4 days with a total of 128 hours dedicated to the audit by four researchers from the Shieldify team.

The audit report identified several medium-severity issues, including potential fee evasion, stuck funds in contracts, and incorrect imports, as well as a low-severity issue with transactions failing due to low gas limits.

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**
Repository	gluex-router gluex-executor
Type of Project	DeFi, Router
Audit Timeline	4 days
Review Commit Hash - gluex router	1df4eaef9c3063a3171961e1f8bba3eb83c6b7e1
Review Commit Hash - gluex executor	84709cb973df0426fd59062ec872138bf6a7f53b
Fixes Review Commit Hash - gluex router	b9bec3c9a421f88efd252728633df952ca4e2150
Fixes Review Commit Hash - gluex executor	89b08ced5155015b07c9f8f8d9911d61ac3f6f5c

5.2 Scope

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

File	nSLOC
gluex_router_executor/Executor.sol	25
gluex_router_executor/interfaces/IExecutor.sol	4
gluex_router_executor/utils/EthReceiver.sol	10
gluex_router/contracts/GluexRouter.sol	129
gluex_router/contracts/interfaces/IDaiLikePermit.sol	7
gluex_router/contracts/interfaces/IERC20.sol	36
gluex_router/contracts/interfaces/IERC20Permit.sol	76
gluex_router/contracts/interfaces/IExecutor.sol	9
gluex_router/contracts/interfaces/IPermit2.sol	36
Total	**332**

6. Findings Summary

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

- **Medium** issues: **5**
- **Low** issues: **2**
- **Info** issues: **1**

ID	Title	Severity	Status
[M-01]	RoutingFee Can Be Set to `1 Wei` to Evade Fees	Medium	Acknowledged
[M-02]	Excess `msg.value` Can Be Stuck In The `Executor.sol` Contract	Medium	Fixed
[M-03]	`desc.effectiveOutputAmount` Can Be Set At a Lower Amount to Pay a Lesser Routing Fee	Medium	Acknowledged
[M-04]	Imports Cannot Be Read Because of Incorrect File Location	Medium	Fixed
[M-05]	Tokens Will Get Stuck In The Executor In Certain Instances Of Swaps	Medium	Acknowledged
[L-01]	Low Gas Limits Could Stuck Transactions for Certain Recipients	Low	Fixed
[L-02]	Missing Deadline Check in `GluexRouter.swap()` Function	Low	Fixed
[I-01]	Nothing Happens in the `uniTransfer()` Function When the Amount Parameter Is Equal to Zero	Info	Fixed

7. Findings

[M-01] RoutingFee Can Be Set to `1 Wei` to Evade Fees

Severity

Medium Risk

Description

When using the protocol, the user will call `GlueXRouter.swap()` and set the `RouteDescription` as the parameter. The user can control the routing fee and can set it to 1 wei to evade fees.

```
struct RouteDescription {
    IERC20 inputToken;
    IERC20 outputToken;
    address payable inputReceiver;
    address payable outputReceiver;
    uint256 inputAmount;
    uint256 outputAmount;
@>  uint256 routingFee;
    uint256 effectiveOutputAmount;
    uint256 minOutputAmount;
    bool isPermit2;
}
```

The `routingFee` cannot be 0, but it can be 1 wei.

```
function swap(
    Executor executor,
@>  RouteDescription calldata desc,
    Interaction[] calldata interactions
) external payable returns (uint256 finalOutputAmount) {
    // code

    require(desc.routingFee > 0, "Negative routing fee");

    // code
}
```

Location of Affected Code

File: contracts/GluexRouter.sol#L35

File: contracts/GluexRouter.sol#L90

Impact

Users can evade fees.

Recommendation

Consider setting the fee at a base percent \codex{(maybe 0.1 - 1%)} and every time a `swap()` happens, a percentage of the fees will go directly to the fee treasury.

Team Response

Acknowledged, the risk will be mitigated off-chain, with access granted only to a select group of users.

[M-02] Excess `msg.value` Can Be Stuck In The `Executor.sol` Contract

Severity

Medium Risk

Description

The `swap()` function is payable, which means it can allow native tokens.

```
function swap(
    IExecutor executor,
    RouteDescription calldata desc,
    Interaction[] calldata interactions
) external payable returns (uint256 finalOutputAmount) {
```

Native tokens will be sent to the executor through the low-level call:

```
IExecutor(executor).executeRoute{value: msg.value}(interactions);
```

If users intend to use the `swap()` functionality without native tokens and accidentally add native tokens, it will be stuck in the Executor contract.

Since the Executor contract has no access control, anyone can withdraw funds from that contract through the `executeRoute()` function.

Location of Affected Code

File: contracts/GluexRouter.sol#L141

Impact

Native tokens will be stuck in the contract.

Recommendation

If the input token is not a native token, make sure `msg.value` is not used.

```
function swap(
    IExecutor executor,
    RouteDescription calldata desc,
    Interaction[] calldata interactions
) external payable returns (uint256 finalOutputAmount) {
    // code

    if (!isNativeTokenInput) {
+       require(msg.value == 0, "Msg.value stuck in contract");
        inputToken.safeTransferFromUniversal(
            msg.sender,
            desc.inputReceiver,
            desc.inputAmount,
            desc.isPermit2
        );`
    }

    // code
}
```

Team Response

Fixed.

[M-03] `desc.effectiveOutputAmount` Can Be Set At a Lower Amount to Pay a Lesser Routing Fee

Severity

Medium Risk

Description

When calculating fees, fees are only taken into account after the swap if the output amount is greater than the `effectiveOutputAmount`.

```
function swap(
    IExecutor executor,
    RouteDescription calldata desc,
    Interaction[] calldata interactions
) external payable returns (uint256 finalOutputAmount) {
    // code

    // Get output amount including positive slippage
    uint256 routeOutputAmount = outputBalanceAfter - outputBalanceBefore;

    if (routeOutputAmount > desc.effectiveOutputAmount + desc.routingFee)
        {
        // User get total amount incl positive slippage minus routing fee
        finalOutputAmount = routeOutputAmount - desc.routingFee;
    } else {
        // Routing fee should only be above the effective output expected
            by the user
        if (routeOutputAmount > desc.effectiveOutputAmount) {
            finalOutputAmount = desc.effectiveOutputAmount;
        } else {
            // If execution is lower than expected (negative slippage),
               then no routing fee
            finalOutputAmount = routeOutputAmount;
        }
    }

    // code
}
```

In cases of swap where the liquidity is extremely high and the swap is small, slippage can be set to the lowest (1) without much issues, which can bypass the routingFee since `routeOutputAmount < desc.effective`
.

Location of Affected Code

File: contracts/GluexRouter.sol#L152-L165

Impact

Users can evade fees.

Recommendation

Consider setting the fee as a percentage of the `routeOutputAmount`, or a flat fee depending on how big the swap is.

Team Response

Acknowledged, the risk will be mitigated off-chain, with access granted only to a select group of users.

[M-04] Imports Cannot Be Read Because of Incorrect File Location

Severity

Medium Risk

Description

When testing the deployment of the contracts in `Remix`, there is an error in imports, the file location is incorrect in the import statement. `./` should be used instead of directly calling it from `utils/EthReceive.sol`

```
// SPDX-License-Identifier: GPL-2.0-or-later
pragma solidity ^0.8.0;

@> import {EthReceiver} from "utils/EthReceiver.sol";
@> import {Interaction} from "base/RouterStructs.sol";
@> import {IExecutor} from "interfaces/IExecutor.sol";

/**
 * @title GenericExecutor
 * @notice Executor that executes an array of interactions
 */
```

Tested in Remix, imports cannot be read because of incorrect file location.

Location of Affected Code

File: contracts/GluexRouter.sol#L4-L6

Impact

The contract cannot be deployed.

Recommendation

Add `./` to the imports.

```
// SPDX-License-Identifier: GPL-2.0-or-later
pragma solidity ^0.8.0;

import {EthReceiver} from "./utils/EthReceiver.sol";
import {Interaction} from "./base/RouterStructs.sol";
import {IExecutor} from "./interfaces/IExecutor.sol";

/**
 * @title GenericExecutor
 * @notice Executor that executes an array of interactions
 */
```

Team Response

Fixed.

[M-05] Tokens Will Get Stuck In The Executor In Certain Instances Of Swaps

Severity

Medium Risk

Description

Most swaps uses the ExactToken -> Token structure, which means that the input amount is exactly what is sent and the output amount varies on the liquidity. The Token -> ExactToken structure also exists, whereby the output amount is fixed but the input amount varies.

In UniswapV2, `swapTokensForExactTokens()` is a function that swaps an input token for the exact output token amount.

In this protocol, when `swap()` is called, the whole input amount will be sent to the Executor contract (desc.inputReceiver is the executor contract).

```
  if (!isNativeTokenInput) {
      inputToken.safeTransferFromUniversal(
          msg.sender,
          desc.inputReceiver,
          desc.inputAmount,
          desc.isPermit2
      );
}
```

If such `swapTokensForExactTokens()` functions are used, the remaining input tokens will be stuck in the Executor contract. If the caller does not know about this, then he may lose out on some tokens.

Foundry test:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "forge-std/Test.sol";
import "forge-std/StdUtils.sol"; // Import StdUtils.sol for dealing with
    tokens
import {GluexRouter} from "../contracts/GluexRouter.sol";
import {GluexExecutor} from "../contracts/Executor.sol";
import {IExecutor} from "../contracts/interfaces/IExecutor.sol";
import {IERC20} from "../contracts/interfaces/IERC20.sol";
import {Interaction} from "../contracts/base/RouterStructs.sol";
import "forge-std/console.sol";

contract GluexRouterTest is Test {
    GluexRouter router;
    GluexExecutor executor;
    IERC20 inputToken;
    IERC20 outputToken;
    address payable user = payable(address(0x456));

    // ERC20 contracts for actual tests
    address USDC = address(0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48);
    address WETH = address(0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2);
    address ETH = address(0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE);
    address Alice = address(1);
    address UniSwapV2Router = address(0
       x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);
    address UniSwapV2Pair = address(0
       xB4e16d0168e52d35CaCD2c6185b44281Ec28C9Dc);

    function setUp() public {
        router = new GluexRouter(address(this), ETH);
        executor = new GluexExecutor();
        inputToken = IERC20(WETH);
        outputToken = IERC20(USDC);
        deal(address(WETH), address(user), 1 ether);
        console.log("Setup DONE");
    }

    function testStuckTokens() public {
        console.log("EXECUTOR WETH BALANCE Before: ", inputToken.
           balanceOf(address(executor)));
        console.log("EXECUTOR USDC BALANCE Before: ", outputToken.
           balanceOf(address(executor)));
        vm.startPrank(user);
        inputToken.approve(address(router), 10e18);
        uint256 inputAmount = 1 ether;
        uint256 minOutputAmount = 100;
        uint256 expectedOutputAmount = 100;
        uint256 routingFee = 1;

        console.log("WETH BALANCE BEFORE: ", inputToken.balanceOf(address
           (user)));
        console.log("USDC BALANCE BEFORE: ", outputToken.balanceOf(
           address(user)));
        // Define RouteDescription
        GluexRouter.RouteDescription memory desc = GluexRouter.
           RouteDescription({
            inputToken: inputToken,
            outputToken: outputToken,
            inputReceiver: payable(address(executor)),
            outputReceiver: payable(user),
            inputAmount: inputAmount,
            outputAmount: expectedOutputAmount,
            routingFee: routingFee,
            effectiveOutputAmount: expectedOutputAmount - routingFee,
            minOutputAmount: 1,
            isPermit2: false
        });

        // Create Interaction array
        Interaction[] memory interactions = new Interaction[](2);

        // Interaction 1: Approve
        interactions[0] = Interaction({
            target: WETH,
            value: 0,
            callData: abi.encodeWithSignature(
                "approve(address,uint256)",
                address(UniSwapV2Router),
                1e18
            )
        });

        // Interaction 2: Swap
        address[] memory path = new address[](2);
        path[0] = WETH;
        path[1] = USDC;

        interactions[1] = Interaction({
            target: address(UniSwapV2Router),
            value: 0,
            callData: abi.encodeWithSignature(
                "swapTokensForExactTokens(uint256,uint256,address[],
                   address,uint256)",
                3900e6,
                1e18,
                path,
                address(router),
                block.timestamp + 60
            )
        });
        // Call swap and assert the result
        uint256 initialUSDCBalance = outputToken.balanceOf(user);
        router.swap{value: 0}(executor, desc, interactions);

        uint256 finalUSDCBalance = outputToken.balanceOf(user);

        console.log("WETH BALANCE After: ", inputToken.balanceOf(address(
           user)));
        console.log("USDC BALANCE After: ", outputToken.balanceOf(address
           (user)));

        console.log("EXECUTOR WETH BALANCE After: ", inputToken.balanceOf
           (address(executor)));
        console.log("EXECUTOR USDC BALANCE After: ", outputToken.
           balanceOf(address(executor)));
        console.log(address(user).balance);
        vm.stopPrank();
    }
}
```

Run with `forge test --fork-url https://rpc.ankr.com/eth -vv --mt testStuckTokens\codex`, output will be:

```
Ran 1 test for test/Route.t.sol:GluexRouterTest
[PASS] testStuckTokens() (gas: 276552)
Logs:
  Setup DONE
  EXECUTOR WETH BALANCE Before:  0
  EXECUTOR USDC BALANCE Before:  0
  WETH BALANCE BEFORE:  1000000000000000000
  USDC BALANCE BEFORE:  0
  WETH BALANCE After:  0
  USDC BALANCE After:  3899999999
  EXECUTOR WETH BALANCE After:  21364774417361964
  EXECUTOR USDC BALANCE After:  0
  0
```

Note that the executor starts with 0 WETH and 0 USDC, and the user starts with 1e18 WETH.

- User calls swapTokensForExactTokens and sets output amount to be 3900 USDC and input as 1e18 WETH
- Since WETH is currently above >3900, the executor has ~21364774417361964 (0.02136 ETH) left over inside.

Impact

Tokens will be stuck in the executor (and eventually can be stolen by directly calling the executor to approve funds to the malicious caller).

Recommendation

Sweep funds from the executor to the router after all the interactions have been completed. Otherwise, ensure that user's understand the risks of using the `swap()` function and ensure that users have an interaction that sweeps the funds from the executor before `swap()` is completed.

Team Response

Acknowledged, Version 1 of the GlueX Router is designed for "exact input" routes only, and the public documentation will be updated to reflect this.

[L-01] Low Gas Limits Could Stuck Transactions for Certain Recipients

Severity

Low Risk

Description

The hardcoded `5_000` gas limit is not high enough for the protocol to be able to receive and distribute rewards.

It is not good to have the gas limit fixed. This will cause the function to revert if insufficient gas is used. Even in the solidity documentation, we can see this warning for the gas option:

> It is best to avoid relying on hardcoded gas values in your smart contract code, regardless of whether the state is read from or written to, as this can have many pitfalls. Also, access to gas might change in the future.

Location of Affected Code

File: contracts/GluexRouter.sol#L195

```
function uniTransfer(
    IERC20 token,
    address payable to,
    uint256 amount
) internal {
    if (amount > 0) {
        if (address(token) == _nativeToken) {
            if (address(this).balance < amount) revert
               InsufficientBalance();

            // solhint-disable-next-line avoid-low-level-calls
@>          (bool success, ) = to.call{ value: amount, gas:
   _RAW_CALL_GAS_LIMIT }("");

            // code
        }
    }
}
```

Recommendation

Consider removing or increasing the hardcoded `5_000` gas limit in [GluexRouter::uniTransfer()]. A possible approach is to implement a setter function for the `_RAW_CALL_GAS_LIMIT`.

Team Response

Fixed.

[L-02] Missing Deadline Check in the `GluexRouter.swap()` Function

Severity

Low Risk

Description

The `swap()` function in the `GluexRouter.sol` contract executes a route, delegating all calls encoded interactions to the `executor`.

However, this function does not include a **deadline check** to ensure that the swap is executed within a specific time frame.

This omission could expose the contract to certain risks, such as stale transactions that are executed later than intended.

Although the slippage check helps mitigate the impact of price changes, swaps executed without a time limit could still be exposed to high market volatility over a longer period. If market conditions change drastically, the user's transaction could be negatively impacted.

Impact

Users who send more ETH than required will not receive a refund for the excess amount.

Location of Affected Code

File: contracts/GluexRouter.sol#L88

Recommendation

We recommend adding a **deadline parameter** to the `swap()` function to ensure that transactions are executed within a specified time frame. This will give users additional protection against market volatility and stale transactions.

```
function swap(
        IExecutor executor,
        RouteDescription calldata desc,
        Interaction[] calldata interactions
+       uint256 deadline
)
    external
    payable
    returns (
        uint256 finalOutputAmount
) {
+   if (deadline < block.timestamp) revert DeadlineExpired();
}
```

Team Response

Fixed.

[I-01] Nothing Happens in the `uniTransfer()` Function When the Amount Parameter Is Equal to Zero

Severity

Info

Description

In the uniTransfer function, if the amount parameter is equal to zero, nothing happens and the transaction is executed successfully.

Location of Affected Code

File: contracts/GluexRouter.sol#L191

```
function uniTransfer(
    IERC20 token,
    address payable to,
    uint256 amount
) internal {
```

```
@>   if (amount > 0) {
         if (address(token) == _nativeToken) {
             if (address(this).balance < amount)
                 revert InsufficientBalance();

             // solhint-disable-next-line avoid-low-level-calls
             (bool success, ) = to.call{
                 value: amount,
                 gas: _RAW_CALL_GAS_LIMIT
             }("");

             if (!success) revert NativeTransferFailed();
         } else {
             token.safeTransfer(to, amount);
         }
@>   }
}
```

Recommendation

We recommend reverting the transaction when the amount is zero.

Team Response

Fixed.

our shielding · Your smart contracts, our shielding · Your smart c

shieldify

Thank you!