



Pingu Exchange

v2

Smart Contract Security Audit

Prepared by ShellBoxes

November 6th, 2025 - November 11th, 2025

[Shellboxes.com](https://shellboxes.com)

contact@shellboxes.com

Document Properties

Client	Pingu
Version	1.0
Classification	Public

Scope

Repository	Commit Hash
https://github.com/OxSuperPingu/pingu-protocol	55b60b9213bb4d147e36d8b403dbbba7e42c74e9

Re-Audit

Repository	Commit Hash
https://github.com/OxSuperPingu/pingu-protocol	32cd097bf78bcb4272125e2d6d33d6c87bec6d4

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1	Introduction	4
1.1	About Pingu	4
1.2	Approach & Methodology	4
1.2.1	Risk Methodology	5
2	Findings Overview	6
2.1	Summary	6
2.2	Key Findings	6
3	Finding Details	8
SHB.1	Liquidations Revert When Execution Fee Exceeds Margin	8
SHB.2	ADL Pays Fees Out of Pool Reserves Instead of Trader Funds	9
SHB.3	Buybacks swap with <code>amountOutMin = 0</code> (no slippage guard)	10
SHB.4	Unchecked ERC20 return values allow silent failures	12
SHB.5	Missing bounds on fee share and buyback reward (bps) can revert core flows	13
SHB.6	Keeper reward is paid even if swap output is dust	15
4	Best Practices	17
BP.1	Use custom errors instead of revert strings	17
BP.2	Emit events for governance parameter changes	18
BP.3	Make <code>Signable.verifyExternal</code> and use <code>callData</code>	19
BP.4	Deduplicate and validate route hints on write	20
BP.5	Cache factory and Pingu in <code>_computePath</code> to avoid repeated SLOADs	22
5	Conclusion	24
6	Scope Files	25
6.1	Audit	25
6.2	Re-Audit	27
7	Disclaimer	29

1 Introduction

Pingu engaged [ShellBoxes](#) to conduct a security assessment on the Pingu Exchange v2 beginning on November 6th, 2025 and ending November 11th, 2025. 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 Pingu

Pingu is a decentralized perpetuals platform where you can trade crypto and forex from your Web3 wallet with up to 100x leverage using ETH or USDC as collateral, provide liquidity to earn real yield as pools capture traders losses plus 45% of fees, and stake PINGU to receive 35% of fees with rewards paid in ETH or USDC.

Issuer	Pingu
Website	https://pingu.exchange
Type	Solidity Smart Contract
Documentation	Pingu Exchange Docs
Audit Method	Whitebox

1.2 Approach & Methodology

ShellBoxes 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 ShellBoxes 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.

Impact	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low
		Likelihood		

2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the Pingu Exchange v2 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 **2** critical-severity, **1** high-severity, **3** medium-severity vulnerabilities.

Vulnerabilities	Severity	Status
SHB.1. Liquidations Revert When Execution Fee Exceeds Margin	CRITICAL	Fixed
SHB.3. Buybacks swap with <code>amountOutMin = 0</code> (no slippage guard)	CRITICAL	Fixed
SHB.5. Missing bounds on fee share and buyback reward (bps) can revert core flows	HIGH	Fixed
SHB.2. ADL Pays Fees Out of Pool Reserves Instead of Trader Funds	MEDIUM	Fixed
SHB.4. Unchecked ERC20 return values allow silent failures	MEDIUM	Fixed
SHB.6. Keeper reward is paid even if swap output is dust	MEDIUM	Acknowledged

3 Finding Details

SHB.1 Liquidations Revert When Execution Fee Exceeds Margin

- Severity: **CRITICAL**
- Likelihood: 3
- Status: Fixed
- Impact: 3

Description:

`_liquidatePosition` subtracts the execution fee from `position.margin` without clamping. When markets are configured with high fees relative to allowed leverage, `fee > margin` and the subtraction underflows, reverting the liquidation transaction. The insolvent position remains open indefinitely, allowing attackers to accumulate unbounded negative PnL against LPs.

Files Affected:

SHB.1.1: Fee subtraction can underflow

```
431 /* Processor.sol */
432 uint256 fee = (position.size * marketInfo.fee) / BPS_DIVIDER;
433 // ...
434 pool.creditTraderLoss(user, asset, market, position.margin - fee); //
    ↪ underflow if fee > margin
435 positions.creditFee(0, user, asset, market, fee, true, keeper);
```

Recommendation:

Before subtracting, either (a) clamp the fee to `position.margin` and handle the zero-margin case, or (b) reject configurations where `maxLeverage * feeBps > 10_000`. Ensuring `pool.creditTraderLoss` never receives a negative value keeps liquidation always executable.

Updates

The team has fixed the issue by adding the marketStore condition and checking if feeBps is less than 10_000 to prevent the liquidation underflow.

SHB.2 ADL Pays Fees Out of Pool Reserves Instead of Trader Funds

- Severity: **MEDIUM**
- Likelihood: 1
- Status: Fixed
- Impact: 3

Description:

In the ADL path, when `pnl >= 0` the trader first receives the entire margin (`fundStore.transferOut(asset, user, position.margin)`). The protocol then calls `positions.creditFee` with the calculated fee, but no funds were retained to cover it. As a result, the fee distribution (keeper/treasury/stakers/pool) is sourced from whatever assets remain in FundStore—i.e., LP capital—compounding pool losses during ADL events.

Files Affected:

SHB.2.1: Margin returned before fees are carved out

```
583 /* Processor.sol */
584 if (pnl < 0) {
585     return (false, "!loss");
586 } else {
587     pool.debitTraderProfit(user, asset, market, uint256(pnl));
588     fundStore.transferOut(asset, user, position.margin);
589 }
590 positions.creditFee(0, user, asset, market, fee, true, keeper);
```

Recommendation:

Deduct the execution fee from the trader's margin before transferring it out (clamping if the fee exceeds the margin) so that `positions.creditFee` is always funded by collected fees rather than LP reserves. After fixing, the pool only pays the trader's profit, not additional fee leakage.

Updates

The team has fixed the issue by changing the logic to return `position.margin - fee`, therefore, the fees are no longer from the pool reserves.

SHB.3 Buybacks swap with `amountOutMin = 0` (no slippage guard)

- Severity: **CRITICAL**
- Likelihood: 3
- Status: Fixed
- Impact: 3

Description:

`Swapper.swapTokensForBaseAuto` hardcodes `amountOutMin = 0` when calling the Uniswap V2 router, so any keeper/user-triggered buyback can be sandwiched with arbitrary slippage. Because `BuyBack.swapAssetsForPingu` is publicly callable when unpaused, a public caller can be frontrun/backrun by MEV and the protocol will accept any price, leading to potentially catastrophic value loss.

Files Affected:

SHB.3.1: Unprotected swap accepts any output; `Swapper.sol:204-210`

```
204 /* Swapper.sol */
205 router.swapExactTokensForTokens(
206 amountIn,
```

```

207 0,
208 path,
209 address(this),
210 block.timestamp + 5 minutes
211 )

```

SHB.3.2: Public buyback entrypoint is callable by anyone when unpaused; Buy-Back.sol:53-64

```

53 /* BuyBack.sol */
54 /// @notice Swap assets for Pingu
55 function swapAssetsForPingu() external {
56   if (buyBackStore.isPaused()) {
57     require(msg.sender == gov, "!paused");
58   }
59
60   address[] memory assets = assetStore.getAssetList();
61   uint256 pinguBought = swapper.swapTokensForBaseAuto(assets, msg.sender);
62   emit PinguBought(pinguBought);
63
64
65 }

```

Recommendation:

Add a configurable onchain slippage guard. For example, compute `expectedOut` with `router.getAmountsOut(amountIn, path)` and `enforce` `amountOutMin = expectedOut * (BPS_DIVIDER - maxSlippageBps) / BPS_DIVIDER`. Optionally validate price against a TWAP/oracle and split large swaps. Keep `maxSlippageBps` under governance with sane bounds and emit it in events for monitoring.

Updates

The team has fixed the issue by introducing a `maxSlippageBps` setter and enforcing `amountOutMin` to prevent zero minimum swaps.

SHB.4 Unchecked ERC20 return values allow silent failures

- Severity: **MEDIUM**
- Likelihood: 2
- Status: Fixed
- Impact: 2

Description:

`_approveTokenIfNeeded` calls `approve` twice and ignores the boolean return, and `withdrawERC20` calls `transfer` without checking the result. Non standard ERC20s that return `false` instead of reverting can cause allowances or withdrawals to silently fail, leaving balances stuck without a revert signal.

Files Affected:

SHB.4.1: Ignored boolean returns on approve; Swapper.sol:266-273

```
266 /* Swapper.sol */
267 IERC20 erc20 = IERC20(token);
268 uint256 currentAllowance = erc20.allowance(address(this), spender);
269 if (currentAllowance < amount) {
270   if (currentAllowance > 0) {
271     erc20.approve(spender, 0);
272   }
273   erc20.approve(spender, type(uint256).max);
274 }
```

SHB.4.2: Ignored boolean return on transfer; Swapper.sol:406-412

```
406 /* Swapper.sol */
407 function withdrawERC20(address token, uint256 amount) external onlyGov {
408   require(IEC20(token).balanceOf(address(this)) >= amount, "Swapper:
     ↪ insufficient balance");
409   IERC20(token).transfer(msg.sender, amount);
410 }
```

Recommendation:

Use `OpenZeppelin SafeERC20` for `safeApprove` and `safeTransfer` (or `require(erc20.transfer(...), "TRANSFER_FAILED")`). Avoid setting unlimited allowances when possible; otherwise reset to zero before reapproving and check results.

Updates

The team has fixed the issue by replacing all the ERC20 calls with the SafeERC20 to avoid silent failures.

SHB.5 Missing bounds on fee share and buyback reward (bps) can revert core flows

- Severity: **HIGH**
- Likelihood: 3
- Status: Fixed
- Impact: 2

Description:

`BuyBackStore.setFeeShare` and `setBuyBackReward` accept arbitrary values without clamping to `<= BPS_DIVIDER`. If governance sets values `> > 10,000` bps, downstream arithmetic underflows and reverts: in the swapper, `amountIn = candidate - buyBackRewardAmount` can underflow; in positions fee split, `feeToTreasury = netFee - ...` can underflow when the buyback share exceeds net fees.

Files Affected:

SHB.5.1: Unbounded bps setters; BuyBackStore.sol:39–47

```
39 /* BuyBackStore.sol */
40 function setFeeShare(uint256 _feeShare) external onlyGov {
41     feeShare = _feeShare;
42 }
```

```

43 function setBuyBackReward(uint256 _buyBackReward) external onlyGov {
44     buyBackReward = _buyBackReward;
45 }

```

SHB.5.2: Reward deducted from candidate without bounds; Swapper.sol:188–191

```

188 /* Swapper.sol */
189 uint256 buyBackRewardAmount = (buyBackReward * candidate) / BPS_DIVIDER;
190 uint256 amountIn = candidate - buyBackRewardAmount;
191 if (amountIn == 0) continue;

```

SHB.5.3: Fee split uses buyBackShare; underflow if misconfigured; Positions.sol:699–716

```

699 /* Positions.sol */
700 uint256 netFee = fee - keeperFee - referrerRebate;
701 uint256 feeToBuyBack = buyBackShare == 0 ? 0 : (netFee * buyBackShare) /
    ↪ BPS_DIVIDER;
702 uint256 feeToPool = poolShare == 0 ? 0 : (netFee * poolShare) /
    ↪ BPS_DIVIDER;
703 uint256 feeToTreasury = netFee - feeToBuyBack - feeToPool;
704 poolStore.incrementBufferBalance(asset, feeToPool / UNIT);
705 buyBackStore.incrementAssetBalance(asset, feeToBuyBack / UNIT);

```

Recommendation:

Clamp inputs in setters: `require(_feeShare <= BPS_DIVIDER, "!feeShare")` and `require(_buyBackReward <= BPS_DIVIDER, "!reward")`. Consider asserting that all fee partitions sum to `<< BPS_DIVIDER` at configuration time to prevent underflow/DoS in fee paths.

Updates

The team has fixed the issue by adding the missing `require` to ensure the fee share and reward bps stay within valid limits.

SHB.6 Keeper reward is paid even if swap output is dust

- Severity: **MEDIUM**
- Likelihood: 2
- Status: Acknowledged
- Impact: 2

Description:

The keeper reward (`buyBackRewardAmount`) is paid solely based on the swap call not reverting; it does not depend on the realized output amount. During periods of extreme slippage (or after intentional price manipulation), the swap can return negligible output while the caller still receives the reward, causing economic leakage.

Files Affected:

SHB.6.1: Reward carved out from inventory regardless of realized output; Swapper.sol:188–191

```
188 /* Swapper.sol */
189 uint256 buyBackRewardAmount = (buyBackReward * candidate) / BPS_DIVIDER;
190 uint256 amountIn = candidate - buyBackRewardAmount;
```

SHB.6.2: Reward paid on any non-reverting swap; Swapper.sol:217–231

```
217 /* Swapper.sol */
218 if (swapSucceeded) {
219     buyBackStore.decrementAssetBalance(tokenIn, amountIn);
220     if (buyBackRewardAmount > 0) {
221         fundStore.transferOut(tokenIn, userAddress, buyBackRewardAmount);
222         emit RewardPaid(userAddress, tokenIn, buyBackRewardAmount);
223         buyBackStore.decrementAssetBalance(tokenIn, buyBackRewardAmount);
224     }
225     emit TokensSwapped(userAddress, tokenIn, amountIn, received);
226 }
```

Recommendation:

Tie reward payment to efficacy. For example, require `received >= minEffectiveOut` (configurable bps vs. `getAmountsOut`) before paying the reward, or scale the reward by `received`. Alternatively, pay the reward from the realized output, ensuring zero or dust outputs yield zero reward.

Updates

The team acknowledged the issue and stated that, although the reward is paid whenever the swap succeeds (even if the output is dust), they consider the risk acceptable because: (i) each swap uses the full available balance and is gated by a per-asset `minAmount`, limiting how often it can be triggered; (ii) they will only use highly liquid assets (e.g., MON/USDC) and carefully configure `maxSlippageBps`; and (iii) they can pause the buy-back and will actively monitor buy-back events via an on-chain listener script.

4 Best Practices

BP.1 Use custom errors instead of revert strings

Description:

The codebase still uses string-based reverts in multiple places (e.g., `BuyBack.sol` (lines 55--64), `Swapper.sol` (lines 115--125), `Pingu.sol` (lines 25--26)). Strings allocate memory and increase bytecode/runtime gas; custom errors are cheaper and provide typed failures that are easier to monitor on-chain.

Files Affected:

BP.1.1: Representative string reverts; `BuyBack.sol:55-64`, `Swapper.sol:115-125`, `Pingu.sol:25-26`

```
55 /* BuyBack.sol */
56 if (buyBackStore.isPaused()) {
57     require(msg.sender == gov, "!paused");
58 }
59 /* Swapper.sol */
60 require(newRouter != address(0), "Swapper: router is zero");
61 require(Address.isContract(newRouter), "Swapper: router must be a
    ↪ contract");
62 /* Pingu.sol */
63 require(amount > 0, "!amount");
```

Recommendation:

Define custom errors and replace string messages to reduce gas and standardize failure modes.

```
1 error Paused();
2 error NotGov();
3 error ZeroAddress();
4 error NotContract();
```

```

5 error InvalidAmount();
6
7 if (buyBackStore.isPaused()) {
8     if (msg.sender != gov) revert Paused();
9 }
10 if (newRouter == address(0)) revert ZeroAddress();
11 if (!Address.isContract(newRouter)) revert NotContract();
12 if (amount == 0) revert InvalidAmount();

```

BP.2 Emit events for governance parameter changes

Description:

BuyBackStore.setFeeShare, setBuyBackReward, and setPaused mutate critical configuration without emitting events, complicating off-chain monitoring and audits (BuyBackStore.sol (lines 37--53)).

Files Affected:

BP.2.1: Setters without events; BuyBackStore.sol:37-53

```

37 function setFeeShare(uint256 _feeShare) external onlyGov { feeShare =
    ↪ _feeShare; }
38 function setBuyBackReward(uint256 _buyBackReward) external onlyGov {
    ↪ buyBackReward = _buyBackReward; }
39 function setPaused(bool _paused) external onlyGov { paused = _paused; }

```

Recommendation:

Emit structured events with old/new values to enable alerting and forensics.

```

1 event FeeShareUpdated(uint256 previous, uint256 next);
2 event BuyBackRewardUpdated(uint256 previous, uint256 next);
3 event PausedSet(bool previous, bool next);

```

```

4
5 function setFeeShare(uint256 _feeShare) external onlyGov {
6     emit FeeShareUpdated(feeShare, _feeShare);
7     feeShare = _feeShare;
8 }
9 function setBuyBackReward(uint256 _buyBackReward) external onlyGov {
10    emit BuyBackRewardUpdated(buyBackReward, _buyBackReward);
11    buyBackReward = _buyBackReward;
12 }
13 function setPaused(bool _paused) external onlyGov {
14    emit PausedSet(paused, _paused);
15    paused = _paused;
16 }

```

BP.3 Make Signable.verify external and use calldata

Description:

Signable.verify is invoked from other contracts (e.g., Referral.sol (lines 50--80)), but its signature uses public with string/bytes in memory (Signable.sol (lines 26--38)), incurring unnecessary memory copies.

Files Affected:

BP.3.1: Public + memory parameters; Signable.sol:26-38

```

26 function verify(
27     string memory refcode,
28     address user,
29     bytes memory signature
30 ) public view returns (bool) {
31     bytes32 structHash = keccak256(abi.encode(TYPEHASH, keccak256(bytes(
        ↪ refcode)), user));

```

```

32     bytes32 digest = _hashTypedDataV4(structHash);
33     // ...
34 }

```

Recommendation:

Switch to `external view` and `calldata` parameters to avoid copies and reduce gas.

```

1 function verify(
2     string calldata refcode,
3     address user,
4     bytes calldata signature
5 ) external view returns (bool) {
6     bytes32 structHash = keccak256(abi.encode(TYPEHASH, keccak256(bytes(
7         ↪ refcode)), user));
8     bytes32 digest = _hashTypedDataV4(structHash);
9     // ...
10 }

```

BP.4 Deduplicate and validate route hints on write

Description:

`setRouteHints` blindly stores the provided list (`Swapper.sol` (lines 156--165)), while `_computePath` re-deduplicates/filters on every swap (`Swapper.sol` (lines 276--334)). Normalizing once at write time reduces per-swap overhead and enforces cleanliness of governance inputs.

Files Affected:

BP.4.1: Hints set without normalization; `Swapper.sol:156-165`

```

156 function setRouteHints(address[] calldata tokens) external onlyGov {
157     delete routeHints;
158     for (uint256 i = 0; i < tokens.length; i++) {

```

```

159     routeHints.push(tokens[i]); // accepts duplicates/zeros until
        ↪ later filtered
160 }
161 }

```

Recommendation:

Filter zeros, remove duplicates, and (optionally) pre-validate pairs when setting.

```

1  event RouteHintsUpdated(address[] hints);
2
3  function setRouteHints(address[] calldata tokens) external onlyGov {
4      delete routeHints;
5      address[] memory tmp = new address[](tokens.length);
6      uint256 n = 0;
7
8      for (uint256 i; i < tokens.length; ) {
9          address t = tokens[i];
10         if (t != address(0)) {
11             bool seen;
12             for (uint256 j; j < n; ) {
13                 if (tmp[j] == t) { seen = true; break; }
14                 unchecked { ++j; }
15             }
16             if (!seen) { tmp[n++] = t; }
17         }
18         unchecked { ++i; }
19     }
20
21     for (uint256 k; k < n; ) {
22         routeHints.push(tmp[k]);
23         unchecked { ++k; }
24     }
25
26     // shrink for event

```

```

27     assembly { mstore(tmp, n) }
28     emit RouteHintsUpdated(tmp);
29 }

```

BP.5 Cache factory and Pingu in `_computePath` to avoid repeated SLOADs

Description:

`_computePath` and helpers repeatedly read `factory` and `Pingu` inside loops (`Swapper.sol` (lines 276--367)). Caching these to local variables reduces SLOADs and external calls within hot paths.

Files Affected:

BP.5.1: Repeated state reads in path computation; `Swapper.sol:276-334, 352-367`

```

276 if (_pairExists(tokenIn, address(Pingu))) { /* ... */ }
277 address pair = factory.getPair(tokenA, tokenB);
278 for (uint256 i = 0; i < hops.length; i++) {
279     address mid = hops[i];
280     if (_pairExists(tokenIn, mid) && _pairExists(mid, address(Pingu))) {
281         ↔ /* ... */ }
281 }

```

Recommendation:

Cache state to locals and reuse across checks; also fold `_pairExists` to a single `getPair` call where possible.

```

1 IUniswapV2Factory f = factory;
2 address base = address(Pingu);
3 uint256 len = hops.length;
4
5 for (uint256 i; i < len; ) {

```

```
6     address mid = hops[i];
7     if (f.getPair(tokenIn, mid) != address(0) && f.getPair(mid, base) !=
8         ↪ address(0)) {
9         // build path ...
10    }
11    unchecked { ++i; }
12 }
```

5 Conclusion

In this audit, we examined the design and implementation of Pingu Exchange v2 contract and discovered several issues of varying severity. Pingu team addressed 5 issues raised in the initial report and implemented the necessary fixes, while classifying the rest as a risk with low-probability of occurrence. Shellboxes' auditors advised Pingu Team to maintain a high level of vigilance and to keep those findings in mind in order to avoid any future complications.

6 Scope Files

6.1 Audit

Files	MD5 Hash
pingu-protocol/Roles.sol	86c38d19b5639dd2c6fe2089dc9a3689
pingu-protocol/Rebate.sol	5346d18709d07ea055dc34aef7376e86
pingu-protocol/ReferralStore.sol	87fc161050c3b78ee4e380771502f9c8
pingu-protocol/IERC20.sol	f42c16e907c13674a1ec9611e6da3a20
pingu-protocol/AssetStore.sol	b50c210e15f7d9974d8aeed89ec1a465
pingu-protocol/FundingStore.sol	c8a39323d6d491ba5102c995e3a42514
pingu-protocol/ERC20.sol	ac61f64b139cf5fa08b22258397d6692
pingu-protocol/RiskStore.sol	6f0718ea3508ff88b95ac08ea0a11d48
pingu-protocol/PythPriceUtils.sol	bee69315bafd041aa87602d3217467d2
pingu-protocol/Pool.sol	78a65ea7ff2008b9d6dd988564203fa4
pingu-protocol/Ownable.sol	0af4517bed5561349ebd746f52793232
pingu-protocol/Positions.sol	c2cd3c99e0de22e5d471704230722558
pingu-protocol/Signable.sol	61207632cfa2db2af859f14be7a5d9eb
pingu-protocol/RoleStore.sol	c674cc6f02ccc0bc936e2e5c01f11b15
pingu-protocol/BuyBack.sol	c1feb139a929786600510dbfbbb8b2f1
pingu-protocol/Referral.sol	6ac353837ff62d09ec15763aebec7790
pingu-protocol/DataStore.sol	d47c7878fca57a4e974f3643b72a0eca

pingu-protocol/RebateStore.sol	e06a1d9e5358847620fbb4f23c1f0629
pingu-protocol/Governable.sol	fb72c03c59da618e54b66e487c54453
pingu-protocol/Proxy.sol	c92cc104d6fb75b653d410392d018dfb
pingu-protocol/FundStore.sol	eb0561766095a5b496a3636531124e5c
pingu-protocol/EnumerableSet.sol	bd1bcd216a9c5cb3c0a17a09e4c9c7ae
pingu-protocol/Swapper.sol	c5e792d74f106e49225040adbd80473c
pingu-protocol/IERC20Metadata.sol	a2186a2141c6123b6d6dfdb4580944e2
pingu-protocol/PythUpdater.sol	2e27aeacead1408d099704c774e864a2
pingu-protocol/MarketStore.sol	158e3f3d37fe53a47affd5f0465b37e7
pingu-protocol/Processor.sol	93028f395483a5049d3ba3a633512eee
pingu-protocol/Context.sol	313fcc99f1338da53d498e8ad7228ece
pingu-protocol/StakingStore.sol	595176b905f2dcec24e9a0afce65bdb6
pingu-protocol/Pingu.sol	c71382fdc34b7fcd09f7ebd23bbe41f6
pingu-protocol/Orders.sol	eb7c2027cfde7eed96891ecf1649eab5
pingu-protocol/Funding.sol	05bcc5d1e19cf32dccf2282123eba0d6
pingu-protocol/PositionStore.sol	5a7f4fc31c10e1234c01eb1345149860
pingu-protocol/OrderStore.sol	570fb50be34c79e754e205560ba9ae8e
pingu-protocol/PoolStore.sol	f2b5a3c2102698960f725c7e08bc9386
pingu-protocol/FundingUpdater.sol	e713bb691eeaae0f1ce9674642bd20b1
pingu-protocol/BuyBackStore.sol	fa6f5524d2163c920f8be354842012ca
pingu-protocol/Staking.sol	efdafb638f58c77787ac5310c118c97a

6.2 Re-Audit

Files	MD5 Hash
./Roles.sol	86c38d19b5639dd2c6fe2089dc9a3689
./Rebate.sol	5346d18709d07ea055dc34aef7376e86
./ReferralStore.sol	87fc161050c3b78ee4e380771502f9c8
./IERC20.sol	f42c16e907c13674a1ec9611e6da3a20
./AssetStore.sol	b50c210e15f7d9974d8aeed89ec1a465
./FundingStore.sol	c8a39323d6d491ba5102c995e3a42514
./ERC20.sol	ac61f64b139cf5fa08b22258397d6692
./RiskStore.sol	6f0718ea3508ff88b95ac08ea0a11d48
./PythPriceUtils.sol	bee69315bafd041aa87602d3217467d2
./Pool.sol	78a65ea7ff2008b9d6dd988564203fa4
./Ownable.sol	0af4517bed5561349ebd746f52793232
./Positions.sol	c2cd3c99e0de22e5d471704230722558
./Signable.sol	9e4c3eb251a5a7ad1b9cd4b7406f4bf1
./RoleStore.sol	c674cc6f02ccc0bc936e2e5c01f11b15
./BuyBack.sol	c1feb139a929786600510dbfbbb8b2f1
./Referral.sol	6ac353837ff62d09ec15763aebec7790
./DataStore.sol	d47c7878fca57a4e974f3643b72a0eca
./RebateStore.sol	e06a1d9e5358847620fbb4f23c1f0629
./Governable.sol	fbb72c03c59da618e54b66e487c54453

./Proxy.sol	c92cc104d6fb75b653d410392d018dfb
./FundStore.sol	eb0561766095a5b496a3636531124e5c
./EnumerableSet.sol	bd1bcd216a9c5cb3c0a17a09e4c9c7ae
./Swapper.sol	cc92bd06c4efef698da7faef45d7a1ee
./IERC20Metadata.sol	a2186a2141c6123b6d6dfdb4580944e2
./PythUpdater.sol	2e27aeacead1408d099704c774e864a2
./MarketStore.sol	4b76377ec7cec51f185498ae407b30b0
./Processor.sol	b3e2f56110305326530346f7e1d56e75
./Context.sol	313fcc99f1338da53d498e8ad7228ece
./StakingStore.sol	595176b905f2dcec24e9a0afce65bdb6
./Pingu.sol	c71382fdc34b7fcd09f7ebd23bbe41f6
./Orders.sol	eb7c2027cfde7eed96891ecf1649eab5
./Funding.sol	05bcc5d1e19cf32dccf2282123eba0d6
./PositionStore.sol	5a7f4fc31c10e1234c01eb1345149860
./OrderStore.sol	570fb50be34c79e754e205560ba9ae8e
./PoolStore.sol	f2b5a3c2102698960f725c7e08bc9386
./FundingUpdater.sol	e713bb691eeaae0f1ce9674642bd20b1
./BuyBackStore.sol	abca38a83a247b9a2dc3ae76ba8aafe4
./Staking.sol	efdafb638f58c77787ac5310c118c97a

7 Disclaimer

Shellboxes reports should not be construed as "endorsements" or "disapprovals" of particular teams or projects. These reports do not reflect the economics or value of any "product" or "asset" produced by any team or project that engages Shellboxes to do a security evaluation, nor should they be regarded as such. Shellboxes Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the examined technology, nor do they provide any indication of the technology's proprietors, business model, business or legal compliance. Shellboxes Reports should not be used in any way to decide whether to invest in or take part in a certain project. These reports don't offer any kind of investing advice and shouldn't be used that way. Shellboxes Reports are the result of a thorough auditing process designed to assist our clients in improving the quality of their code while lowering the significant risk posed by blockchain technology. According to Shellboxes, each business and person is in charge of their own due diligence and ongoing security. Shellboxes does not guarantee the security or functionality of the technology we agree to research; instead, our purpose is to assist in limiting the attack vectors and the high degree of variation associated with using new and evolving technologies.



For a Contract Audit, contact us at contact@shellboxes.com