Veridise. Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Davos

Stable Asset Protocol and Bridge

Veridise Inc.
August 31, 2023

► **Prepared For:**

Davos

► **Prepared By:**

Jon Stephens
Xiangan He

► **Contact Us:** contact@veridise.com

► **Version History:**

Jun. 28, 2023   Initial Draft

Contents

From May 26, 2023 to June 22, 2023, Davos engaged Veridise to review the security of their Stable Asset Protocol and Bridge. The review covered the Davos bridge, Davos's MakerDAO fork, Ceros router and vaults, and liquidity reward mechanisms. Veridise conducted the assessment over 8 person-weeks, with 2 engineers reviewing code over 4 weeks over several commits listed in the audit scope. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual auditing.

Code assessment. The Davos developers provided the source code of the Stable Asset Protocol and Bridge contracts for review. To facilitate the Veridise auditors' understanding of the code, the Davos developers held calls with the auditing team to answer various questions and to confirm findings. The source code also contained some documentation in the form of READMEs and documentation comments on functions and storage variables. The source code contained a test suite, which the Veridise auditors noted covers key protocol user-flows and some edge cases.

Summary of issues detected. The audit uncovered 25 issues, 3 of which are assessed to be of high or critical severity by the Veridise auditors. Specifically, the protocol may overpay users upon withdrawing from strategies that require time to unstake or gather the requested funds. Additionally, a lack of decimal checks on withdrawals from the Davos bridge may also cause potential overpaying to users. The Veridise auditors also identified several medium-severity issues, including potential denial of service attacks to the bridge and users being sent the incorrect amount of funds from strategies.

Recommendations. After auditing the protocol, the auditors had a few suggestions to improve the security of Davos's Stable Asset Protocol and Bridge. Most notably, the off-chain component of the bridge was not in scope of the audit. While the auditors did point out some issues that could occur if validation was not performed in the off-chain component, the list should not be considered exhaustive, nor should this component be considered secure. As the on-chain bridge contracts require trust in the off-chain logic, we highly recommend that the Davos seek a security review of this component.

Additionally, the MasterVault contract allows the Davos developers to add investment strategies in the future. Since the MasterVault contract makes some assumptions about the strategies, we would recommend that the developers document the properties that should hold for a strategy to properly interact with the MasterVault.

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
Stable Asset Protocol and Bridge	`8cc37a7, 7869f69, 89cdc07`	Solidity	Ethereum

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
May 26 - June 22, 2023	Manual & Tools	2	8 person-weeks

Table 2.3: Vulnerability Summary.

Name	Number	Resolved
Critical-Severity Issues	0	0
High-Severity Issues	3	3
Medium-Severity Issues	2	2
Low-Severity Issues	10	10
Warning-Severity Issues	10	10
Informational-Severity Issues	0	0
TOTAL	25	25

Table 2.4: Category Breakdown.

Name	Number
Data Validation	7
Maintainability	7
Logic Error	4
Denial of Service	2
Locked Funds	2
Best Practices	2
Access Control	1

3.1 Audit Goals

The engagement was scoped to provide a security assessment of Stable Asset Protocol and Bridge's smart contracts. In our audit, we sought to answer the following questions:

- ▶ Can a user DoS the bridge?
- ▶ Are users able to withdraw from the bridge if and only if they have valid bridging requests?
- ▶ Will messages from the off-chain component be processed correctly?
- ▶ Can a user bridge funds from an un-approved source?
- ▶ Does the DeFi application handle user debt accounting correctly? Can users be overpaid?
- ▶ Do oracles provide correct prices?
- ▶ Are emergency shutdown measures implemented?
- ▶ Does the protocol follow Solidity best practices?
- ▶ Are there protections against mis-behaving strategies in the MasterVault?
- ▶ Can a user prevent others from receiving payment?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved a combination of human experts and automated program analysis & testing tools. In particular, we conducted our audit with the aid of the following techniques:

- ▶ *Static analysis.* To identify potential common vulnerabilities, we leveraged our custom smart contract analysis tool Vanguard, as well as the open-source tool Slither. These tools are designed to find instances of common smart contract vulnerabilities, such as reentrancy and uninitialized variables.
- ▶ *Fuzzing/Property-based Testing.* We also leverage fuzz testing to determine if the protocol may deviate from the expected behavior. To do this, we formalize the desired behavior of the protocol as [V] specifications and then use our fuzzing framework OrCa to determine if a violation of the specification can be found.

Scope. The scope of this audit is limited to the on-chain contracts of the davos-contracts and davos-bridge repositories provided by the Davos developers. The scope for the individual repositories is given individually below:

davos-contracts at commit `8cc37a7`

- ▶ `contracts/interaction.sol`
- ▶ `contracts/masterVault/`
- ▶ `contracts/strategies/`
- ▶ `contracts/libraries/`
- ▶ `contracts/ceros/`

    ► `contracts/oracle/MaticOracle.sol`

davos-bridge at commit `7869f69`

    ► `contracts/`

Additionally, the Davos developers requested to extend the davos-contracts scope after the audit began to include the following contracts. As such, the Veridise auditors had less time to conduct the review of these contracts.

davos-contracts at commit `89cdc07`

    ► `contracts/masterVault/MasterVault_V2.sol`
    ► `contracts/oracle/WstETHOracle.sol`

Methodology. Veridise auditors reviewed the reports of previous audits for the Stable Asset Protocol and Bridge, inspected the provided tests, and read the Stable Asset Protocol and Bridge documentation. They then began a manual audit of the code assisted by both static analyzers and automated testing. During the audit, the Veridise auditors regularly met with the Davos developers to ask questions about the code.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

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

In this case, we judge the likelihood of a vulnerability as follows in 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) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

In addition, we judge the impact of a vulnerability as follows in 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 - OR - 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 - OR - 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

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-DAV-VUL-001	No Decimal Check on Withdraw	High	Fixed
V-DAV-VUL-002	Protocol may overpay users	High	Fixed
V-DAV-VUL-003	Invalid Bridge Withdrawals may be Accepted	High	Fixed
V-DAV-VUL-004	Withdraw DoS	Medium	Acknowledged
V-DAV-VUL-005	Too Many Funds may be Withdrawn	Medium	Fixed
V-DAV-VUL-006	WithdrawUnsettled will always revert	Low	Fixed
V-DAV-VUL-007	Payable functions do not credit users	Low	Fixed
V-DAV-VUL-008	No Check that Strategy Exists when Migrated	Low	Fixed
V-DAV-VUL-009	Lack of Inclusion of End module	Low	Intended Behavior
V-DAV-VUL-010	Return value should be marked as incomplete	Low	Fixed
V-DAV-VUL-011	Potential for DoS in WaitingPool	Low	Acknowledged
V-DAV-VUL-012	Approve Lazily rather than Eagerly	Low	Acknowledged
V-DAV-VUL-013	Strategy can Manipulate Debt	Low	Fixed
V-DAV-VUL-014	Debt may be Canceled with Fees	Low	Fixed
V-DAV-VUL-015	Return Value doesn't Match Documentation	Low	Fixed
V-DAV-VUL-016	Value not Explicitly Returned	Warning	Fixed
V-DAV-VUL-017	Value can be a Boolean	Warning	Intended Behavior
V-DAV-VUL-018	State changes should follow CEI pattern	Warning	Fixed
V-DAV-VUL-019	Get Token Decimals instead of Hardcoding	Warning	Fixed
V-DAV-VUL-020	Validate Yield Margin on Initialization	Warning	Fixed
V-DAV-VUL-021	No constructor calling _disableInitializers() i. . .	Warning	Fixed
V-DAV-VUL-022	Validate or Directly use Oracle Decimals	Warning	Fixed
V-DAV-VUL-023	Perform Item Validation	Warning	Acknowledged
V-DAV-VUL-024	Funds can potentially get stuck in bridge	Warning	Fixed
V-DAV-VUL-025	Incomplete Withdraw Marked as Complete	Warning	Fixed

4.1 Detailed Description of Issues

4.1.1 V-DAV-VUL-001: No Decimal Check on Withdraw

Severity	High	Commit	7869f69
Type	Data Validation	Status	Fixed
File(s)			DavosBridge.sol
Location(s)			withdraw

The ERC20 token standard allows developers to set a `decimals` field which is used to determine what should be considered to be a single unit of the token (and therefore allow users to trade partial tokens). While 18 is commonly used, tokens such as USDC use different values for these decimals. The bridge allows such tokens to be sent over the bridge, but does so by converting the tokens to their equivalent 18 decimal representation. It only does this on the deposit side of the bridge, however using the function shown below.

```
1  function _amountErc20Token(address fromToken, uint256 totalAmount) internal returns (
       uint256) {
2      /* scale amount to 18 decimals */
3      require(IERC20Extra(fromToken).decimals() <= 18, "DavosBridge/decimals-overflow")
       ;
4      totalAmount *= (10**(18 - IERC20Extra(fromToken).decimals()));
5      return totalAmount;
6  }
```

Snippet 4.1: Function that scales an input amount to 18 decimals

Impact   Most developers deploy the same contract with the same initialization on all ethereum-based chains. It is therefore likely that if the conversion to 18 decimals is required on one side of the bridge, it will need to be converted back on the other side. Without logic validating the decimals on the receiving (withdrawal) side of the bridge, we believe it is likely that this conversion will cause too many funds to be minted on the receiving side of the bridge.

Recommendation   Either validate that all tokens on the withdrawal side of the chain have 18 decimals, or include logic to convert to the appropriate decimals on the receiving side of the chain as well.

4.1.2 V-DAV-VUL-002: Protocol may overpay users

Severity	High	**Commit**	8cc37a7
Type	Logic Error	**Status**	Fixed
File(s)		MasterVault.sol	
Location(s)		_withdrawFromActiveStrategies	

The `_withdrawFromActiveStrategies` function allows Davos to retrieve funds that have been allocated to strategies. It does so by looping over all active strategies and requesting the given amount of funds from them. One thing to note though, is strategies may have differ in how they retrieve funds based on their `class`. For example, `IMMEDIATE` strategies can send funds back to the pool immediately without any delay. `DELAYED` strategies, however, require time to unstake or gather the requested funds. As a result, when making a request to a `DELAYED` strategy, Davos will send funds directly to the requested recipient. It should be noted, however, that this function loops over all strategies and in every iteration, the return values are overwritten if a withdrawal attempt is made.

```
1  function _withdrawFromActiveStrategies(address _recipient, uint256 _amount, Type
       class) private returns(uint256 withdrawn, bool incomplete, bool delayed) {
2      for(uint8 i = 0; i < strategies.length; i++) {
3          if(strategyParams[strategies[i]].active && (strategyParams[strategies[i]].
       class == class || class == Type.ABSTRACT) && strategyParams[strategies[i]].debt
       >= _amount) {
4              _recipient = strategyParams[strategies[i]].class == Type.DELAYED ?
       _recipient : address(this);
5              delayed = strategyParams[strategies[i]].class == Type.DELAYED ? true :
       false;
6              (withdrawn, incomplete) = _withdrawFromStrategy(_recipient, strategies[i],
       _amount);
7          }
8      }
9  }
```

Snippet 4.2: Location where future iterations of the loop will overwrite data from previous iterations

Impact   For `IMMEDIATE` and (presumably) `ABSTRACT` strategies, this can cause `MasterVault` to withdraw too many funds from the strategies. As in some locations, such as `cancelDebt` calls to `_withdrawFromActiveStrategies` are then followed by forwarding the entire pool balance, this can also cause too many funds to be forwarded to other locations. For `DELAYED` strategies, there is a similar effect where this can request too many funds to be forwarded to the given recipient and therefore for that recipient to be overpaid.

Recommendation   Track the amount of funds removed and only remove the requested amount of funds.

4.1.3 V-DAV-VUL-003: Invalid Bridge Withdrawals may be Accepted

Severity	High	**Commit**	7869f69	
Type	Data Validation	**Status**	Fixed	
File(s)		DavosBridge.sol		
Location(s)		_withdrawWarped		

To prevent invalid bridging requests, the DavosBridge contract performs some additional validation to ensure withdrawals come from known sources. As such, they validate that the depositing contract comes from a known source and that the source token is allowed to be bridged to the destination token by storing approved bridge destinations as mappings. On a withdraw, the validation does not check that the accessed index has been defined (i.e. is not address(0)), which could allow the bridge to accept unapproved bridging requests as shown below.

```
1  function _withdrawWarped(EthereumVerifier.State memory state, ProofParser.Proof
       memory proof) internal {
2      require(warpDestination(state.toToken, proof.chainId) == state.fromToken, "
       DavosBridge/bridge-from-unknown-destination");
3      IERC20Mintable(state.toToken).mint(state.toAddress, state.totalAmount);
4
5      emit WithdrawMinted(state.receiptHash, state.fromAddress, state.toAddress, state.
       fromToken, state.toToken, state.totalAmount);
6  }
7
8  function warpDestination(address fromToken, uint256 toChain) public view returns(
       address) {
9      return _warpDestinations[keccak256(abi.encodePacked(fromToken, block.chainid,
       _bridgeAddressByChainId[toChain]))];
10 }
```

Snippet 4.3: Location where an unapproved destination token can be minted if state.fromToken is the 0 address

Impact   If the off-chain consensus mechanism does not perform proper validation, this could allow invalid transaction receipts to be accepted. For example, if consensus approves and signs a receipt from an unsupported chain, where fromToken is address(0) an attacker could steal funds from the bridge.

Recommendation   Ensure that the accessed mappings do not return address(0)

```
1 function withdraw(bytes calldata, /* encodedProof */ bytes calldata rawReceipt, bytes
      memory proofSignature) external override nonReentrant whenNotPaused {
2     ...
3
4     ProofParser.Proof memory proof = ProofParser.parseProof(proofOffset);
5     require(_bridgeAddressByChainId[proof.chainId] == state.contractAddress, "
      DavosBridge/event-from-unknown-bridge");
6
7     ...
8 }
```

Snippet 4.4: Location where an invalid chain may be accepted if `state.contractAddress` is the 0 address

4.1.4 V-DAV-VUL-004: Withdraw DoS

Severity	Medium	Commit	7869f69
Type	Denial of Service	Status	Acknowledged
File(s)		EthereumVerifier.sol	
Location(s)		parseTransactionReceipt	

From the provided tests, it appears that the off-chain component of the bridge will be responsible for processing transaction receipts to gather the emitted events, sign the receipt and then forward it to the requested bridge contract. In turn, the `DavosBridge` contract will process the emitted events to determine if any relevant bridging events were emitted. In the on-chain logic, it is then assumed that a single bridging request is present in the logs, as shown below, which could be used to perform a Denial of Service attack.

```
1  function parseTransactionReceipt(uint256 receiptOffset)
2      internal
3      pure
4      returns (State memory state, PegInType pegInType)
5  {
6      ...
7
8      uint256 logsIter = CallDataRLPReader.beginIteration(logs);
9      for (; logsIter < iter; ) {
10         uint256 log = logsIter;
11         logsIter = CallDataRLPReader.next(logsIter);
12         /* make sure there is only one peg-in event in logs */
13         PegInType logType = _decodeReceiptLogs(state, log);
14         if (logType != PegInType.None) {
15             require(
16                 pegInType == PegInType.None,
17                 "EthereumVerifier: multiple logs"
18             );
19             pegInType = logType;
20         }
21     }
22     /* don't allow to process if peg-in type is unknown */
23     require(pegInType != PegInType.None, "EthereumVerifier: missing logs");
24     return (state, pegInType);
25 }
```

Snippet 4.5: Location where receipts are processed and only a single bridge request is accepted

Impact   As we do not have access to the off-chain computation performed by the bridge, we do not know if there is any post-processing done on the transaction receipt. If, however, there is not then any transaction that bridges funds twice will be rejected when performing a withdraw. Additionally, malicious developers can DOS the bridge by emitting events with the same signature as the `DepositWarped` event to inject multiple logs.

4.1.5 V-DAV-VUL-005: Too Many Funds may be Withdrawn

Severity	Medium		**Commit**	8cc37a7
Type	Data Validation		**Status**	Fixed
File(s)		MasterVault.sol		
Location(s)		withdrawUnderlying		

When performing a withdraw, it might be the case that the MasterVault does not have sufficient funds to pay back the user. If this occurs, it will attempt to retrieve any additional funds necessary from a strategy. As withdrawing from strategies may incur additional fees, the contract adds the amount withdrawn to the current balance of the pool. However, there is currently no validation if the return value is non-zero.

Impact A buggy strategy may cause users to be sent too few or too many funds.

Recommendation Validate that the amount withdrawn is within expected bounds.

Developer Response We will have authorized strategies only. For incomplete withdrawals, it is not possible to find bounds. However, tolerance() method has been added for a complete full withdrawal where bounds are checked.

```
 1 │ function withdrawUnderlying(address _account, uint256 _amount) external payable
   │     override nonReentrant whenNotPaused onlyOwnerOrProvider returns (uint256 assets)
   │     {
 2 │     ...
 3 │
 4 │     if(underlyingBalance < _amount) {
 5 │       uint256 debt = waitingPool.getUnbackedDebt();
 6 │       Type class = debt == 0 ? Type.ABSTRACT : Type.IMMEDIATE;
 7 │
 8 │       (uint256 withdrawn, bool incomplete, bool delayed) =
   │     _withdrawFromActiveStrategies(_account, _amount + debt - underlyingBalance, class
   │     );
 9 │
10 │       if(withdrawn == 0 || debt != 0 || incomplete) {
11 │         assets = _assessFee(assets, withdrawalFee);
12 │         waitingPool.addToQueue(_account, assets);
13 │         if(totalAssetInVault() > 0)
14 │           SafeERC20Upgradeable.safeTransfer(IERC20Upgradeable(asset()), address(
   │     waitingPool), underlyingBalance);
15 │         emit Withdraw(src, src, src, assets, _amount);
16 │         return _amount;
17 │       } else if(delayed) {
18 │         assets = underlyingBalance;
19 │       } else {
20 │         assets = underlyingBalance + withdrawn;
21 │       }
22 │     }
23 │
24 │     ...
25 │ }
```

Snippet 4.6: Snippet of withdrawUnderlying that withdraws funds from strategies

4.1.6 V-DAV-VUL-006: WithdrawUnsettled will always revert

Severity	Low		Commit	8cc37a7
Type	Logic Error		Status	Fixed
File(s)			WaitingPool.sol	
Location(s)			withdrawUnsettled	

The `WaitingPool` contract implements a queue of users to which the protocol owes funds. As this contract gains funds, the `tryRemove` function will process the queue in order to repay users. In the event that a user cannot be repaid, `withdrawUnsettled` allows users to request their funds only after `tryRemove` has attempted to serve them. However, due to the line below `withdrawUnsettled` will always revert. This is because, as shown below, `tryRemove` sets `people[_index]._settled` to

```
1 function withdrawUnsettled(uint256 _index) external {
2     ...
3     require(!people[_index]._settled && _index < index && people[_index]._address ==
   src, "WaitingPool/already-settled");
4     ...
5 }
```

Snippet 4.7: The following check will always cause `withdrawUnsettled` to revert:

`true if _index < index.`

Impact `withdrawUnsettled` will always revert due to the definition of `tryRemove`

Recommendation   This function no longer seems to serve a purpose after removing the ability to send native tokens, we would recommend removing it altogether.

```
1  function tryRemove() external onlyMasterVault {
2      uint256 balance;
3      uint256 cap = 0;
4      for(uint256 i = index; i < people.length; i++) {
5          balance = getPoolBalance();
6          uint256 userDebt = people[index]._debt;
7          address userAddr = people[index]._address;
8          if(balance >= userDebt && userDebt != 0 && !people[index]._settled && cap <
    capLimit) {
9              totalDebt -= userDebt;
10             people[index]._settled = true;
11             emit WithdrawCompleted(userAddr, userDebt);
12
13             cap++;
14             index++;
15
16             IERC20Upgradeable(underlying).safeTransfer(userAddr, userDebt);
17         } else return;
18     }
19 }
```

Snippet 4.8: The definition of `tryRemove` which always sets `_settled` when processing a user

4.1.7 V-DAV-VUL-007: Payable functions do not credit users

Severity	Low	Commit	8cc37a7
Type	Locked Funds	Status	Fixed
File(s)		depositUnderlying, withdrawUnderlying, deposit, withdraw	
Location(s)		MasterVault, CerosYieldConverterStrategyLs, ...	

Several functions in the protocol are marked as `payable`, allowing them to accept ETH deposits. If a user provides such funds, however, they will not be credited with any shares in return. Instead, the funds will eventually be locked inside the protocol as most of these contracts do not include any logic to interact with native currency.

Impact   This can cause user funds to be locked in the contract.

Recommendation   Either remove the payable modifier or require that `msg.value` is 0 if that is not possible.

Developer Response   We've removed the payable functions.

```
1  function depositUnderlying(uint256 _amount) external payable override nonReentrant
       whenNotPaused onlyOwnerOrProvider returns (uint256 shares) {
2
3          require(_amount > 0, "MasterVault/invalid-amount");
4          address src = msg.sender;
5
6          SafeERC20Upgradeable.safeTransferFrom(IERC20Upgradeable(asset()), src,
       address(this), _amount);
7          shares = _assessFee(_amount, depositFee);
8
9          uint256 waitingPoolDebt = waitingPool.totalDebt();
10         uint256 waitingPoolBalance = IERC20Upgradeable(asset()).balanceOf(address(
       waitingPool));
11         if(waitingPoolDebt > 0 && waitingPoolBalance < waitingPoolDebt) {
12             uint256 waitingPoolDebtDiff = waitingPoolDebt - waitingPoolBalance; //
       @audit-info transfers $$ to debt pool to pay off all debt first
13             uint256 poolAmount = (waitingPoolDebtDiff < shares) ? waitingPoolDebtDiff
        : shares;
14             SafeERC20Upgradeable.safeTransfer(IERC20Upgradeable(asset()), address(
       waitingPool), poolAmount);
15         }
16
17         _mint(src, shares);
18
19         if(allocateOnDeposit == 1) allocate();
20
21         emit Deposit(src, src, _amount, shares); // @audit incorrect event emission
22     }
```

Snippet 4.9: The following code snippet contains no logic for minting shares to a user if they had deposited ETH.

4.1.8 V-DAV-VUL-008: No Check that Strategy Exists when Migrated

Severity	Low	**Commit**	8cc37a7
Type	Data Validation	**Status**	Fixed
File(s)		MasterVault.sol	
Location(s)		migrateStrategy	

The protocol allows strategies to be upgraded or migrated to new versions of the strategy. In doing so, funds from the old strategy are withdrawn it is removed from the set of active strategies. The new strategy is then added in its place. Unlike in addStrategy, however, an existing and already active strategy can be added a second time to the strategies list.

Impact   This could accidentally overwrite a strategy's stored configuration parameters including, most importantly, the amount of debt that has been locked in the strategy.

Recommendation   Specifically disallow migrating old strategies to existing strategies.

```
1  function migrateStrategy(address _oldStrategy, address _newStrategy, uint256
       _newAllocation, Type _class) external onlyOwnerOrManager {
2      require(_oldStrategy != address(0) && _newStrategy != address(0));
3
4      ...
5
6      StrategyParams memory params = StrategyParams({active: true, class: _class,
       allocation: _newAllocation, debt: 0});
7
8      bool isValidStrategy;
9      for(uint256 i = 0; i < strategies.length; i++) {
10         if(strategies[i] == _oldStrategy) {
11             isValidStrategy = true;
12             strategies[i] = _newStrategy;
13             strategyParams[_newStrategy] = params;
14
15             break;
16         }
17     }
18
19     require(isValidStrategy, "MasterVault/invalid-oldStrategy");
20     require(_deactivateStrategy(_oldStrategy),"MasterVault/cannot-deactivate");
21     require(_isValidAllocation(), "MasterVault/>100%");
22
23     emit StrategyMigrated(_oldStrategy, _newStrategy, _newAllocation);
24 }
```

Snippet 4.10: The migrate strategy function which does not check that `_newStrategy` already exists

4.1.9 V-DAV-VUL-009: Lack of Inclusion of End module

Severity	Low		Commit	8cc37a7
Type	Maintainability		Status	Intended Behavior
File(s)				N/A
Location(s)				N/A

Davos's MakerDAO fork doesn't include end - the emergency shutdown module used to call most of the cage functions located throughout. The End's purpose is to coordinate a shutdown where the protocol closes down the system and reimburses stablecoin holders. This is a process that can occur during upgrades (Dai iterations), as well as for security reasons in the event that implementation flaws arise in both in the code and in the design.****

Impact   Not including the end module would cause shutdown to become more difficult to execute.

Recommendation   Consider forking the end module.

Developer Response   Since we have single entry point, we can just halt interaction rather than all of the contracts that were forked from MakerDAO.

4.1.10 V-DAV-VUL-010: Return value should be marked as incomplete

Severity	Low	Commit	8cc37a7
Type	Logic Error	Status	Fixed
File(s)			MasterVault.sol
Location(s)			_withdrawFromStrategy

As MasterVault strategies may not have sufficient funds to fully fulfill a request, it might be the case that a withdraw is partially complete. For this reason, the `_withdrawFromStrategy` function returns a boolean to indicate that the full amount could not be withdrawn. In some cases, however, the function indicates that the withdraw was complete despite no funds being withdrawn as shown below.

```
1  function _withdrawFromStrategy(address _recipient, address _strategy, uint256 _amount
       ) private returns(uint256, bool incomplete) {
2      ...
3
4      if(capacity <= 0 || chargedCapacity > capacity) return (0, false);
5      else if(capacity < _amount) incomplete = true;
6
7      if(params.class == Type.DELAYED && incomplete) return (0, true);
8
9      ...
10
11     emit WithdrawnFromStrategy(_strategy, _amount, chargedCapacity);
12     return (chargedCapacity, incomplete);
13 }
```

Snippet 4.11: Location where an incomplete withdraw is not identified as such

Impact   As some parts of the code rely on this boolean to determine if the withdraw was completed successfully, this could cause funds to be tracked incorrectly.

Recommendation   Only return `false` if a withdraw is completed successfully

4.1.11 V-DAV-VUL-011: Potential for DoS in WaitingPool

Severity	Low		Commit	8cc37a7
Type	Denial of Service		Status	Acknowledged
File(s)			WaitingPool.sol	
Location(s)			tryRemove	

Davos uses a waiting queue for users who have yet to be paid their funds on a withdrawal. Over time, this queue is processed by sending users the funds that they are owed in the order of the queue. While this will work for most ERC20 tokens, the developers should ensure that the underlying asset does not have user callbacks as are used by some custom tokens and ERC20 extensions such as ERC777. If such a currency is used, there is the potential that a single user can perform a denial of service attack on the waiting pool.

```
1  function tryRemove() external onlyMasterVault {
2      uint256 balance;
3      uint256 cap = 0;
4      for(uint256 i = index; i < people.length; i++) {
5          balance = getPoolBalance();
6          uint256 userDebt = people[index]._debt;
7          address userAddr = people[index]._address;
8          if(balance >= userDebt && userDebt != 0 && !people[index]._settled && cap <
   capLimit) {
9              totalDebt -= userDebt;
10             people[index]._settled = true;
11             emit WithdrawCompleted(userAddr, userDebt);
12
13             cap++;
14             index++;
15
16             IERC20Upgradeable(underlying).safeTransfer(userAddr, userDebt);
17         } else return;
18     }
19  }
```

Snippet 4.12: Definition of the tryRemove function

Impact   If there are user callbacks, a user could always revert on the transfer to userAddr. This would cause the entire transaction to revert, preventing users from recovering funds.

Recommendation   Ensure that any underlying assets do not contain callbacks to the user on a transfer.

Developer Response   We are aware of this and we will make sure the underlyings are not ERC777 or add support for them later if needed.

4.1.12 V-DAV-VUL-012: Approve Lazily rather than Eagerly

Severity	Low		Commit	8cc37a7
Type	Access Control		Status	Acknowledged
File(s)				dConverterStrategyLs.sol, YieldConverterStrategySp.sol, CerosRouterLs.sol, CerosRouterLsEth.sol, CerosRout
Location(s)				Multiple

Many of the Davos contracts have to transfer funds between each other. To simplify the logic, it
is a common pattern for contracts to approve other addresses to access all of the stored funds as
shown below. Should there be an error in one of the approved contracts or should one of these
addresses be compromised, this could allow funds to be stolen.

```
1  function initialize(address _destination, address _feeRecipient, address
       _underlyingToken, address _masterVault) external initializer {
2      __BaseStrategy_init(_destination, _feeRecipient, _underlyingToken);
3
4      masterVault = IMasterVault(_masterVault);
5      underlying.approve(address(_destination), type(uint256).max);
6      underlying.approve(address(_masterVault), type(uint256).max);
7  }
```

Snippet 4.13: Location where a strategy approves other addresses to access all of its funds

Recommendation   Rather than approve max funds on initialization, consider instead approving
funds only when necessary.

Developer Response   We are aware of this. The contracts are kept as restricted as possible. We
will keep this in mind for future changes.

4.1.13 V-DAV-VUL-013: Strategy can manipulate Debt

Severity	Low	**Commit**	8cc37a7
Type	Data Validation	**Status**	Fixed
File(s)			MasterVault.sol
Location(s)			_depositToStrategy

The `MasterVault` contract is used to invest stored user funds using various strategies. The amount of invested funds are then tracked so that they can later be retrieved when users perform a withdraw. As some investment strategies may incur fees, however, the amount the entire deposit may not be invested. To accurately track the amount of invested funds, the protocol therefore determines two values: the strategy's capacity, and capacity after fees have been charged. The capacity is then deposited while the capacity with fees is added to a strategy's debt. There is currently no validation on the relationship between these two values though.

```
1 function _depositToStrategy(address _strategy, uint256 _amount) private returns (bool
      success) {
2    ...
3
4    // 'capacity' is total depositable; 'chargedCapacity' is capacity after charging
     fee
5    (uint256 capacity, uint256 chargedCapacity) = IBaseStrategy(_strategy).canDeposit
     (_amount);
6    if(capacity <= 0 || capacity > _amount || chargedCapacity > capacity) return
     false;
7
8    totalDebt += chargedCapacity;
9    strategyParams[_strategy].debt += chargedCapacity;
10
11   SafeERC20Upgradeable.safeTransfer(IERC20Upgradeable(asset()), _strategy, capacity
     );
12   IBaseStrategy(_strategy).deposit{value: msg.value}(capacity);
13
14   emit DepositedToStrategy(_strategy, capacity, chargedCapacity);
15   return true;
16 }
```

<div align="center">

Snippet 4.14: Function used to deposit funds into a strategy

</div>

Impact   As there is no validation between these two values, a strategy's debt may not be accurately tracked. For example, if there is a bug in a strategy, it could be the case that it returns 0 as the capacity after fees, in which case the strategy's debt would not increase.

Recommendation   Validate the allowable loss due to fees to prevent potential errors.

4.1.14 V-DAV-VUL-014: Debt may be canceled with Fees

Severity	Low	**Commit**	8cc37a7
Type	Logic Error	**Status**	Fixed
File(s)			MasterVault.sol
Location(s)			cancelDebt

As the MasterVault may not be able to retrieve all funds at the time a request is withdrawn, Davos places users in a queue to be refunded as funds are made available. As this occurs, managers may cancel the debt to users which will withdraw additional funds from certain strategies and forward those funds to the waiting pool. When calculating the amount of funds that can be sent to the waiting pool, it appears that there is an error in the asset calculation. After performing a withdraw, the total assets in the vault is first queried, which determines the amount of funds in the vault without accumulated fees. After that though, if the withdraw was successful this amount is augmented with the amount withdrawn, increasing the amount to send above the amount of user funds in the pool.

```
1  function cancelDebt(Type _class) public onlyOwnerOrManager {
2      ...
3
4      (withdrawn,,delayed) = _withdrawFromActiveStrategies(address(waitingPool),
       withdrawAmount + 1, _class);
5      uint256 amount = totalAssetInVault();
6      if(withdrawn > 0 && !delayed) amount += withdrawn;
7      SafeERC20Upgradeable.safeTransfer(IERC20Upgradeable(asset()), address(waitingPool
       ), amount);
8
9      ...
10 }
```

Snippet 4.15: Location where `cancelDebt` withdraws from strategies to regain funds

Impact   As `totalAssetInVault` returns the total assets without fees, any additional amount added to this will either come from fees, or will cause the transfer of funds to revert, preventing the debt from being canceled.

Recommendation   Only send the funds reported by `totalAssetInVault`.

4.1.15 V-DAV-VUL-015: Return Value does not Match Documentation

Severity	Low	Commit	8cc37a7
Type	Maintainability	Status	Fixed
File(s)			MasterVault.sol
Location(s)			withdrawUnderlying

When funds are withdrawn from `MasterVault` a withdrawal fee may be charged. According to the inline documentation, it appears that withdraw is intended to return the amount of funds withdrawn after fees are charged. However, currently it returns the amount of funds that the user requested to withdraw.

```
1  /** Withdraw underlying assets via DavosProvider
2   * @param _account receipient
3   * @param _amount underlying assets to withdraw
4   * @return assets underlying assets excluding any fees
5   */
6  function withdrawUnderlying(address _account, uint256 _amount) external payable
       override nonReentrant whenNotPaused onlyOwnerOrProvider returns (uint256 assets)
       {
7      ...
8
9      assets = _assessFee(assets, withdrawalFee);
10     SafeERC20Upgradeable.safeTransfer(IERC20Upgradeable(asset()), _account, assets);
11
12     emit Withdraw(src, src, src, assets, _amount);
13     return _amount;
14  }
```

Snippet 4.16: Location where the code is inconsistent with the documentation

Impact   It is possible that incorrect values may be reported to the user as the return value of this function is later emitted in the Davos protocol.

Recommendation   If the code is correct, update the documentation so that developers don't misuse this function.

4.1.16 V-DAV-VUL-016: Value not Explicitly Returned

Severity	Warning	Commit	8cc37a7
Type	Maintainability	Status	Fixed
File(s)			MasterVault.sol
Location(s)			_deactivateStrategy

Solidity allows functions to return values without an explicit return statement, in which case the default value will be returned.

```solidity
function _deactivateStrategy(address _strategy) private returns(bool success) {
    if (strategyParams[_strategy].debt <= 10) {
        strategyParams[_strategy].active = false;
        strategyParams[_strategy].debt = 0;
        return true;
    }
}
```

Snippet 4.17: Example where the default value is returned on a program path

Recommendation   For clarity, we recommend that all functions explicitly return a value on all program paths rather than relying on default values.

4.1.17 V-DAV-VUL-017: Value can be a Boolean

Severity	Warning	Commit	8cc37a7
Type	Maintainability	Status	Intended Behavior
File(s)		function depositUnderlying	
Location(s)		MasterVault.sol	

The Davos developers declare the `allocateOnDeposit` variable as a `uint256` but constrain it's behavior so that it is effectively a boolean.

```
1  uint256 public allocateOnDeposit;
2
3  ...
4
5  function changeAllocateOnDeposit(uint256 _status) external onlyOwner {
6      require(_status >= 0 && _status < 2, "MasterVault/range-0-or-1");
7      allocateOnDeposit = _status;
8
9      emit AllocationOnDepositChangeed(_status);
10 }
```

Snippet 4.18: The behavior of `allocateOnDeposit` is enforced to behave like a boolean.

Impact Changing this to a boolean renders the check unnecessary and improves code clarity.

Recommendation Implement changes as described above.

Developer Response In the future, we may add additional variable assignments.

4.1.18 V-DAV-VUL-018: State changes should follow CEI pattern

Severity	Warning	Commit	8cc37a7
Type	Best Practices	Status	Fixed
File(s)		CerosRouterLs, CerosRouterLsETH, CerosRouterSp	
Location(s)		claimProfit	

Due to the prevalence of reentrancy attacks, it is considered good practice to follow the Checks-Effects-Interaction (CEI) pattern. Under this pattern functions should first perform any necessary checks and update the internal state of the contract before transferring control outside of the contract. We would recommend that the developers follow this pattern.

```
1 function claimProfit(address _recipient) external nonReentrant {
2             ...
3             s_aMATICc.transfer(_recipient, profit);  // aMATICc
4       s_profits[msg.sender] -= profit;
5             ...
6 }
```

Snippet 4.19: Location where CEI should be used.

Impact   Using the Checks-Effects-Interaction pattern minimizes reentrancy attack risk.

Recommendation   Implement the above by switching the order of the state change and transfer.

4.1.19 V-DAV-VUL-019: Get Token Decimals instead of Hardcoding

Severity	Warning	**Commit**	89cdc07	
Type	Maintainability	**Status**	Fixed	
File(s)		WstETHOracle.sol		
Location(s)		peek		

As no price oracle exists between the `MasterVault` share token and other currencies such as USD, the Davos developers provide one that relies on oracles for existing currencies. However, this contract relies on using hard-coded values for token decimals, which can be error prone if the code is re-used.

```
1  function peek() public view returns (bytes32, bool) {
2      (
3          /*uint80 roundID*/,
4          int price,
5          /*uint startedAt*/,
6          /*uint timeStamp*/,
7          /*uint80 answeredInRound*/
8      ) = priceFeed.latestRoundData();
9      if (price < 0) {
10         return (0, false);
11     }
12
13     // Get stETH equivalent to 1wstETH and multiply with stETH price
14     uint256 stETH = wstETH.getStETHByWstETH(1e18);
15     uint256 wstETHPrice = (stETH * uint(price * (10**10))) / 1e18;
16
17     // Get wstETH equivalent to 1share in MasterVault
18     uint256 wstETH = masterVault.previewRedeem(1e18);
19     uint256 sharePrice = (wstETHPrice * wstETH) / 1e18;
20
21     return (bytes32(sharePrice), true);
22 }
```

Recommendation Make the token and oracle decimals immutable values and initialize them with the proper values from the contracts themselves.

4.1.20 V-DAV-VUL-020: Validate Yield Margin on Initialization

Severity	Warning		Commit	89cdc07
Type	Maintainability		Status	Fixed
File(s)				MasterVault_V2.sol
Location(s)				initialize

The MasterVault_V2 contract stores wstETH liquid staking tokens so that users may utilize them as collateral for Davos. During the time that these tokens are staked with Davos, they will also generate rewards as one wstETH will be worth more stETH. As part of the Davos protocol, they will take a cut of those generated rewards as determined by yieldMargin while they hold the wstETH as collateral. When this value is set on initialization no validation is performed, as shown below, unlike when it is set in changeYieldMargin.

```
1  function initialize(string memory _name, string memory _symbol, uint256 _yieldMargin,
      address _underlying) external initializer {
2      __ERC4626_init(IERC20MetadataUpgradeable(_underlying));
3      __ERC20_init(_name, _symbol);
4      __Ownable_init();
5      __Pausable_init();
6      __ReentrancyGuard_init();
7
8      yieldMargin = _yieldMargin;
9      yieldBalance = 0;
10 }
```

Snippet 4.20: The initialize function with no validation of the yield margin

Impact   There is the potential that this could cause initialization issues.

Recommendation   Perform similar validation in initialize that is performed in changeYieldMargin

4.1.21 V-DAV-VUL-021: No constructor calling _disableInitializers() in Router

Severity	Warning		Commit	8cc37a7
Type	Best Practices		Status	Fixed
File(s)			CerosRouterSP	
Location(s)			constructor	

_disableInitializers() prevents initialization of the implementation contract itself, providing an extra line of defense against attackers looking to initialize the contract as an attack against the proxy.

Impact   If the initializer is not called upon deployment, attackers can call the contract to initialize the contract.

Recommendation   Add the constructor and _disableInitializers(), similar to what many of the other contracts in scope already have.

4.1.22 V-DAV-VUL-022: Validate or Directly use Oracle Decimals

Severity	Warning	Commit	8cc37a7
Type	Data Validation	Status	Fixed
File(s)		MranticOracle.sol	
Location(s)		peek	

Davos implements several price oracles that fetch data from chainlink. While these chainlink oracle pairs have predictable decimals for the returned prices, they can vary depending on the pair used. For example, USD pairs use 8 decimals while ETH pairs use 18 decimals.

```
1  function peek() public view returns (bytes32, bool) {
2      (
3          /*uint80 roundID*/,
4          int price,
5          /*uint startedAt*/,
6          /*uint timeStamp*/,
7          /*uint80 answeredInRound*/
8      ) = priceFeed.latestRoundData();
9      if (price < 0) {
10         return (0, false);
11     }
12     return (bytes32(uint(price * (10**10))), true);
13 }
```

Snippet 4.21: Code that fetches a chainlink price and converts it to 18 decimals

Impact   Should the wrong pair is used, it is possible for the oracles to return incorrect prices.

Recommendation   Either use the pair's decimals directly or validate the pair's decimals on initialization

4.1.23 V-DAV-VUL-023: Perform Item Validation

Severity	Warning	**Commit**	7869f69
Type	Data Validation	**Status**	Acknowledged
File(s)		CallDataRLPReader.sol	
Location(s)		toUintStrict, toUint, beginIteration	

The Davos Bridge encodes transaction data in the RLP format, which is then parsed by the on-chain contracts. Importantly, under this format most items (except for small bytes) are associated with a length indicating the size of the item. Davos's RLP reader uses such lengths to determine the location of items, but currently does not use them to validate user parsing requests, such as when a user requests to convert an item's data to an integer as shown below.

```
1  function toUintStrict(uint256 ptr) internal pure returns (uint256) {
2      // one byte prefix
3      uint256 result;
4      assembly {
5          result := calldataload(add(ptr, 1))
6      }
7      return result;
8  }
```

Snippet 4.22: Location where no validation is performed when reading an integer

Impact The lack of validation could cause the reader to misinterpret data or improperly use data.

Recommendation For functions such as `beginIteration`, ensure that the current item is a list. For items such as `toUint` and `toUintStrict` validate that the items are of an appropriate size.

4.1.24 V-DAV-VUL-024: Funds can potentially get stuck in bridge

Severity	Warning	**Commit**	89cdc07	
Type	Locked Funds	**Status**	Fixed	
File(s)		DavosBridge.sol		
Location(s)		receive()		

The Davos bridge implements a 'primitive' to receive ETH. Funds that get sent in this way, however, do not get accounted for.

```
1  // --- Primitives ---
2      receive() external payable {}
```

Snippet 4.23: The bridge can receive ETH via. this receive function

Impact   As a result, users that send ETH to the bridge will have their funds locked inside the bridge.

Recommendation   In the short term, remove `receive()` since the bridge only seems to be supporting USDC / ERC20 inputs of funds.

In the long term, the team can implement support for bridging ETH via. wETH, eliminating the need to take in native ETH while maintaining the same interface as today.

4.1.25 V-DAV-VUL-025: Incomplete Withdraw Marked as Complete

Severity	Warning	Commit	8cc37a7
Type	Maintainability	Status	Fixed
File(s)		MasterVault.sol	
Location(s)		_withdrawFromActiveStrategies	

When the MasterVault contract requires additional funds, it may need to withdraw funds from its investment strategies using the `_withdrawFromActiveStrategies` function. As the investment strategies may not have sufficient funds, this function may return that the withdrawal request is incomplete, or rather it was unable to retrieve the desired amount of funds. In the current version of the source code, though, when a strategy could not be found to withdraw from, the withdrawal will be marked as complete (or rather not incomplete) due to the fact that the function will return the default value for the boolean.

```
1 function _withdrawFromActiveStrategies(address _recipient, uint256 _amount, Type
     class) private returns(uint256 withdrawn, bool incomplete, bool delayed) {
2     for(uint8 i = 0; i < strategies.length; i++) {
3         if(strategyParams[strategies[i]].active && (strategyParams[strategies[i]].
     class == class || class == Type.ABSTRACT) && strategyParams[strategies[i]].debt
     >= _amount) {
4             _recipient = strategyParams[strategies[i]].class == Type.DELAYED ?
     _recipient : address(this);
5             delayed = strategyParams[strategies[i]].class == Type.DELAYED ? true :
     false;
6             (withdrawn, incomplete) = _withdrawFromStrategy(_recipient, strategies[i],
     _amount);
7         }
8     }
9 }
```

<div align="center">

Snippet 4.24: Source code of the `_withdrawFromActiveStrategies` function

</div>

Impact   Currently callers of this function validate both `incomplete` and the amount withdrawn. If future code does not do so, however, and relies only on the `incomplete` flag, it could cause the protocol to misbehave.

Recommendation   If no withdraw was made, return that it is incomplete.