BASE PROTOCOL

Smart Contract Security Audit

Prepared by: **Halborn**
Date of Engagement: November 25th, 2020
Visit: Halborn.com

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	11/26/2020	Gabi Urrutia
0.2	Document Edits	11/28/2020	Gabi Urrutia
1.0	Final Version	11/30/2020	Gabi Urrutia
1.1	Remediations	12/01/2020	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com

1.1 INTRODUCTION

Base Protocol engaged Halborn to conduct a security assessment on their smart contracts that implement the protocol on the Ethereum blockchain. Base Protocol is a decentralized elastic supply protocol for creating indices of other tokens. It maintains a price peg by adjusting supply directly to and from wallet holders based on a data feed generated by off-chain oracles. The security assessment was scoped to the contract BaseToken, BaseTokenMonetaryPolicy, BaseTokenOrchestrator, Cascade and an audit of the security risk and implications regarding the changes introduced by the development team at Base Protocol prior to its production release shortly following the assessments deadline.

The most important security finding was the re-entrancy vulnerability in Cascade.sol due to the order of the calls into functions. This vulnerability was immediately fixed by Base Protocol Team and tested it again by Halborn auditors.

Overall, the smart contract code is extremely well documented, follows a high-quality software development standard, contains many utilities and automation scripts to support continuous deployment / testing / integration, and does NOT contain any obvious exploitation vectors that Halborn was able to leverage within the timeframe of testing allotted.

Though the outcome of this security audit is satisfactory; due to time and resource constraints, only testing and verification of essential properties were performed to achieve objectives and deliverables set in the scope. It is important to remark the use of the best practices for secure smart contract development. Halborn recommends performing further testing to validate extended safety and correctness in context to the whole set of contracts. External threats, such as economic attacks, oracle attacks, and inter-contract functions and calls should be validated for expected logic and state.

1.2 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose
- Smart Contract manual code read and walkthrough
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual Assessment of use and safety for the critical solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Scanning of solidity files for vulnerabilities, security hotspots, or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Localhost - npx)
- Smart Contract Fuzzing and dynamic state exploitation (Echidna) Symbolic Execution / EVM bytecode security assessment (limited time)

1.3 SCOPE

Code related to:

- BaseToken.sol
- BaseTokenMonetaryPolicy.sol
- BaseTokenOrchestrastor.sol
- Cascade.sol

Specific commit of contract: Commit ID:
48d71fb86d4667c90d2f03fce8a07cce2d7d4b7e
OUT-OF-SCOPE:
External contracts, External Oracles, other smart contracts in the repository or imported by BASE protocol contracts, economic attacks

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	0	3

SECURITY ANALYSIS	RISK LEVEL	Remediation Date
RE-ENTRANCY	Critical	12/30/2020
FOR LOOP OVER DYNAMIC ARRAY	Informational	-
USE OF TX.ORIGIN	Low	-
AVOID USING NOW	Low	-
STATE VARIABLE SHADOWING	Informational	-
STATIC ANALYSIS	Critical	12/30/2020 LOW
AUTOMATED SECURITY SCAN RESULTS	Informational	-

FINDINGS & TECH DETAILS

3.1 RE-ENTRANCY - CRITICAL

Description

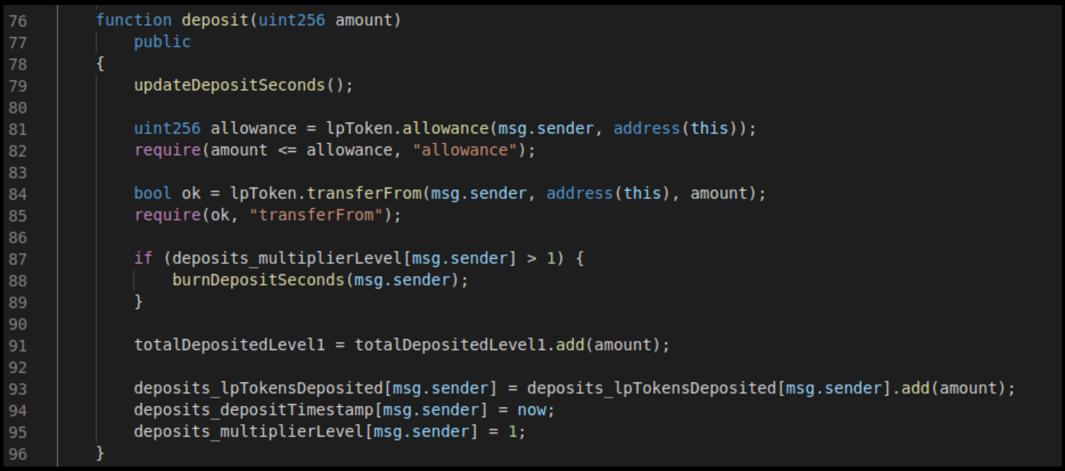
Calling external contracts is dangerous if some functions and variables are called after the external call. An attacker could use a malicious contract to perform a recursive call before calling function and take over the control flow. **transferFrom** and **transfer** are external calls. Values are set and burn only after checking is both are the msg.sender. Thus, an attacker could perform a recursive call to execute malicious code.

Reference:

https://swcregistry.io/docs/SWC-107

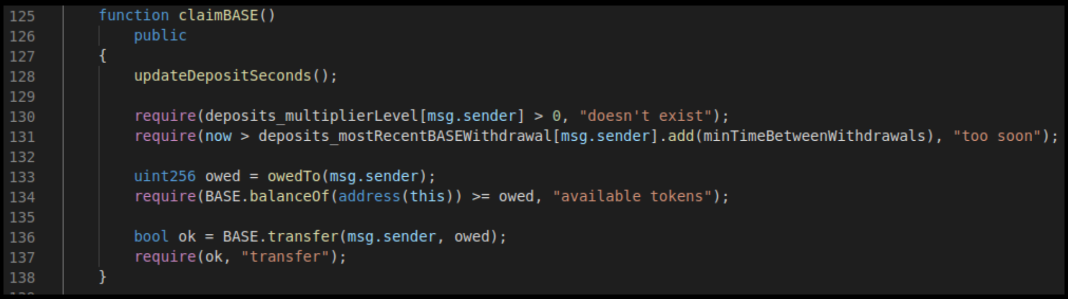
Code Location

Cascade.sol Lines #76-96



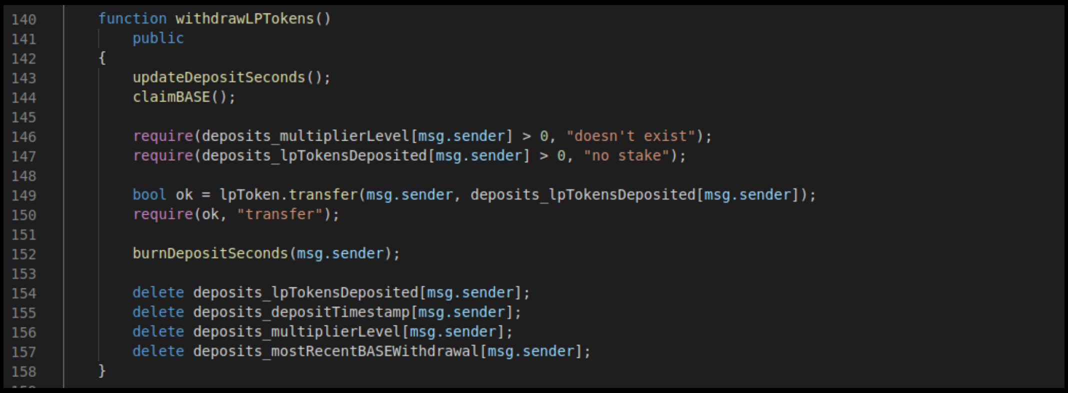
```
76      function deposit(uint256 amount)
77          public
78      {
79          updateDepositSeconds();
80
81          uint256 allowance = lpToken.allowance(msg.sender, address(this));
82          require(amount <= allowance, "allowance");
83
84          bool ok = lpToken.transferFrom(msg.sender, address(this), amount);
85          require(ok, "transferFrom");
86
87          if (deposits_multiplierLevel[msg.sender] > 1) {
88              burnDepositSeconds(msg.sender);
89          }
90
91          totalDepositedLevel1 = totalDepositedLevel1.add(amount);
92
93          deposits_lpTokensDeposited[msg.sender] = deposits_lpTokensDeposited[msg.sender].add(amount);
94          deposits_depositTimestamp[msg.sender] = now;
95          deposits_multiplierLevel[msg.sender] = 1;
96      }
```

Cascade.sol Lines #125-138



```
125     function claimBASE()
126         public
127     {
128         updateDepositSeconds();
129
130         require(deposits_multiplierLevel[msg.sender] > 0, "doesn't exist");
131         require(now > deposits_mostRecentBASEWithdrawal[msg.sender].add(minTimeBetweenWithdrawals), "too soon");
132
133         uint256 owed = owedTo(msg.sender);
134         require(BASE.balanceOf(address(this)) >= owed, "available tokens");
135
136         bool ok = BASE.transfer(msg.sender, owed);
137         require(ok, "transfer");
138     }
```

Cascade.sol Lines #140-158



```
function withdrawLPTokens()
    public
{
    updateDepositSeconds();
    claimBASE();

    require(deposits_multiplierLevel[msg.sender] > 0, "doesn't exist");
    require(deposits_lpTokensDeposited[msg.sender] > 0, "no stake");

    bool ok = lpToken.transfer(msg.sender, deposits_lpTokensDeposited[msg.sender]);
    require(ok, "transfer");

    burnDepositSeconds(msg.sender);

    delete deposits_lpTokensDeposited[msg.sender];
    delete deposits_depositTimestamp[msg.sender];
    delete deposits_multiplierLevel[msg.sender];
    delete deposits_mostRecentBASEWithdrawal[msg.sender];
}
```

Recommendation:

As possible, external calls should be at the end of the function in order to avoiding an attacker take over the control flow.

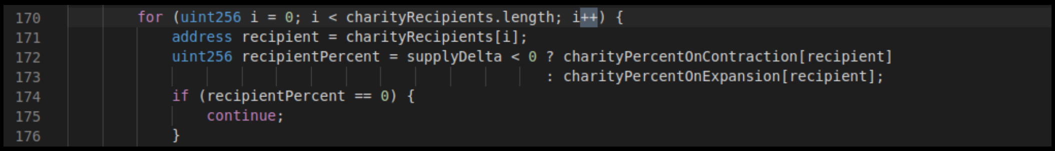
3.2 FOR LOOP OVER DYNAMIC ARRAY - INFORMATIONAL

Description

Calls inside a loop might lead to a denial-of-service attack. The function discovered is a for loop on variable \`i\` that iterates up to the charityRecipients variable. If this integer is evaluated at extremely large numbers, or \`i\` is reset by external calling functions, this can cause a DoS.

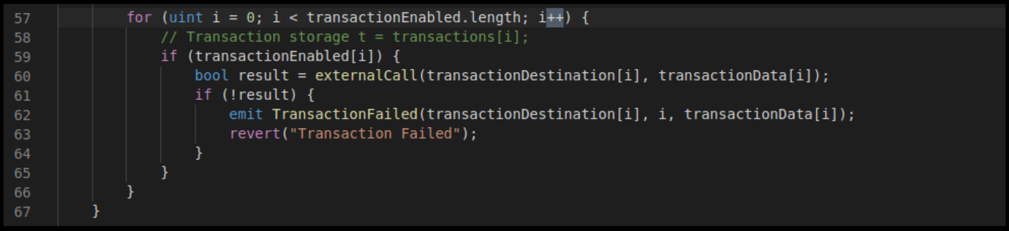
Code Location

BaseTokenMonetaryPolicy.sol Line #170



```
170         for (uint256 i = 0; i < charityRecipients.length; i++) {
171             address recipient = charityRecipients[i];
172             uint256 recipientPercent = supplyDelta < 0 ? charityPercentOnContraction[recipient]
173                                                        : charityPercentOnExpansion[recipient];
174             if (recipientPercent == 0) {
175                 continue;
176             }
```

BaseTokenOrchestrator.sol Line #57



```
57      for (uint i = 0; i < transactionEnabled.length; i++) {
58          // Transaction storage t = transactions[i];
59          if (transactionEnabled[i]) {
60              bool result = externalCall(transactionDestination[i], transactionData[i]);
61              if (!result) {
62                  emit TransactionFailed(transactionDestination[i], i, transactionData[i]);
63                  revert("Transaction Failed");
64              }
65          }
66      }
67  }
```

Recommendation:

If possible, use pull over push strategy for external calls.

3.3 USE OF TX.ORIGIN - LOW

Description

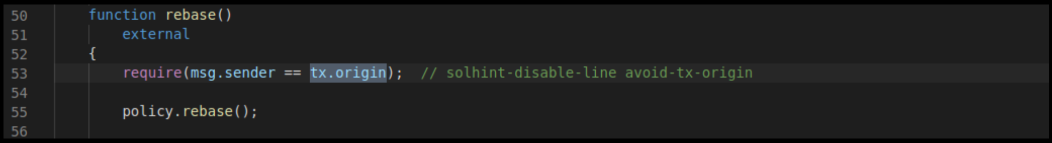
BaseTokenOrchestrator.sol uses tx.origin so that rebase() function can be called by anybody. It is recommended that you use msg.sender instead of tx.origin because if a transaction is made to a malicious wallet, when you check it you will have the origin address and you will not be able to know the address of the malicious wallet. Nevertheless, the use of tx.origin is semi-legitimized for recording who calls the contract most. Furthermore, tx.origin could be used to prevent an address from interacting with your contract because the owner of the address cannot use the contract as an intermediary to circumvent your blocking. Finally, it is important to remark that the use of tx.origin will be deprecated.

Reference:

https://consensys.net/blog/blockchain-development/solidity-best-practices-for-smart-contract-security/

Code Location

BaseTokenOrchestrator.sol Line #53



```
50      function rebase()
51          external
52      {
53          require(msg.sender == tx.origin);  // solhint-disable-line avoid-tx-origin
54
55          policy.rebase();
56
```

Recommendation:

It is recommended not to use tx.origin because a malicious wallet could receive funds and cannot be tracked. However, its use is semi-legitimate in some cases with caution.

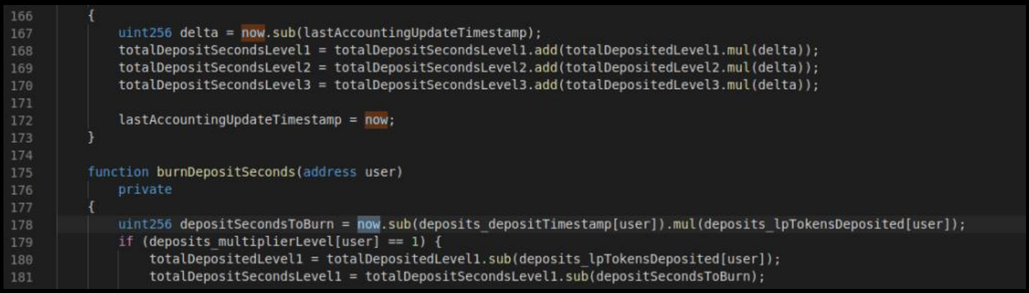
3.4 AVOID USING NOW - LOW

Description:

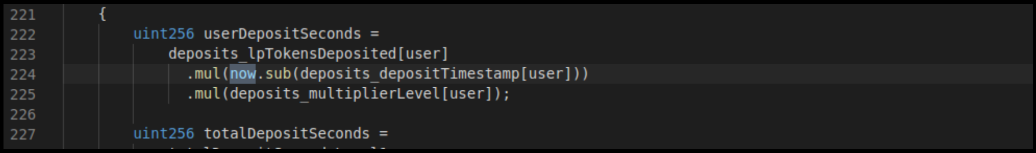
The use of “now” can be influenced by miners to some degree. Miners can manipulate “now” to exploit the contract. Here in BaseTokenMonetaryPolicy.sol contract, “now” is being used in inRebaseWindow() and rebase() methods.

Code Location:

Cascade.sol Lines #167, 172l, 178, 224

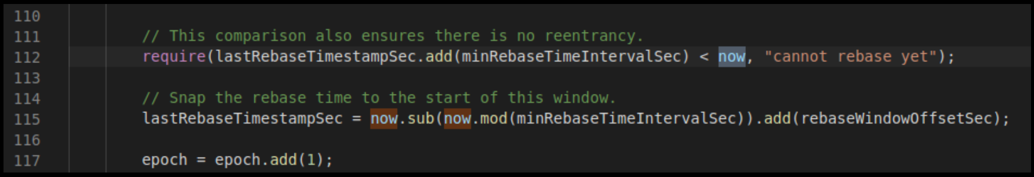


```
166         {
167             uint256 delta = now.sub(lastAccountingUpdateTimestamp);
168             totalDepositSecondsLevel1 = totalDepositSecondsLevel1.add(totalDepositedLevel1.mul(delta));
169             totalDepositSecondsLevel2 = totalDepositSecondsLevel2.add(totalDepositedLevel2.mul(delta));
170             totalDepositSecondsLevel3 = totalDepositSecondsLevel3.add(totalDepositedLevel3.mul(delta));
171
172             lastAccountingUpdateTimestamp = now;
173         }
174
175         function burnDepositSeconds(address user)
176             private
177         {
178             uint256 depositSecondsToBurn = now.sub(deposits_depositTimestamp[user]).mul(deposits_lpTokensDeposited[user]);
179             if (deposits_multiplierLevel[user] == 1) {
180                 totalDepositedLevel1 = totalDepositedLevel1.sub(deposits_lpTokensDeposited[user]);
181                 totalDepositSecondsLevel1 = totalDepositSecondsLevel1.sub(depositSecondsToBurn);
```



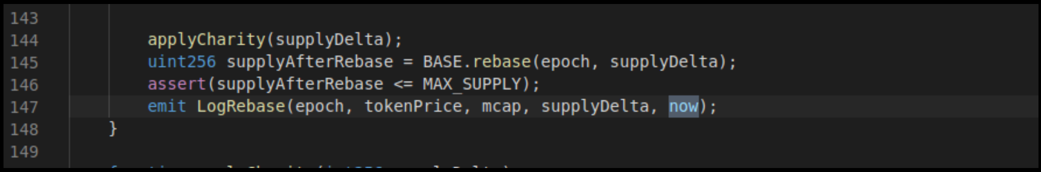
```
221         {
222             uint256 userDepositSeconds =
223                 deposits_lpTokensDeposited[user]
224                 .mul(now.sub(deposits_depositTimestamp[user]))
225                 .mul(deposits_multiplierLevel[user]);
226
227             uint256 totalDepositSeconds =
```

BaseTokenMonetaryPolicy.sol Lines #112,115



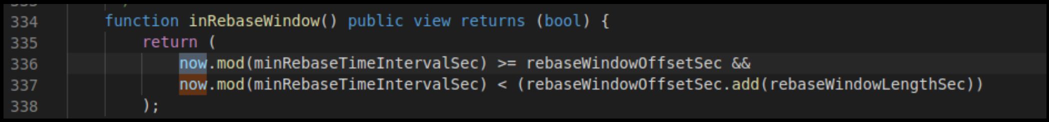
```
110
111             // This comparison also ensures there is no reentrancy.
112             require(lastRebaseTimestampSec.add(minRebaseTimeIntervalSec) < now, "cannot rebase yet");
113
114             // Snap the rebase time to the start of this window.
115             lastRebaseTimestampSec = now.sub(now.mod(minRebaseTimeIntervalSec)).add(rebaseWindowOffsetSec);
116
117             epoch = epoch.add(1);
```

BaseTokenMonetaryPolicy.sol Line #147



```
143
144             applyCharity(supplyDelta);
145             uint256 supplyAfterRebase = BASE.rebase(epoch, supplyDelta);
146             assert(supplyAfterRebase <= MAX_SUPPLY);
147             emit LogRebase(epoch, tokenPrice, mcap, supplyDelta, now);
148         }
149
```

BaseTokenMonetaryPolicy.sol Lines #336-337



```
334         function inRebaseWindow() public view returns (bool) {
335             return (
336                 now.mod(minRebaseTimeIntervalSec) >= rebaseWindowOffsetSec &&
337                 now.mod(minRebaseTimeIntervalSec) < (rebaseWindowOffsetSec.add(rebaseWindowLengthSec))
338             );
```

Recommendation:

Avoid getting relied on use of “now” in require methods because it could be manipulated by miners.

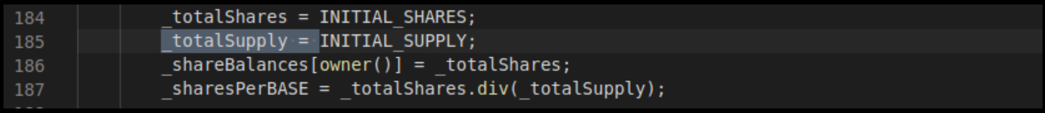
3.5 STATE VARIABLE SHADOWING - INFORMATIONAL

Description:

There are few state variables are getting shadowed by the child contract BaseToken.sol (_totalSupply variable) and ERC20UpgradeSafe.sol (__gap variable).

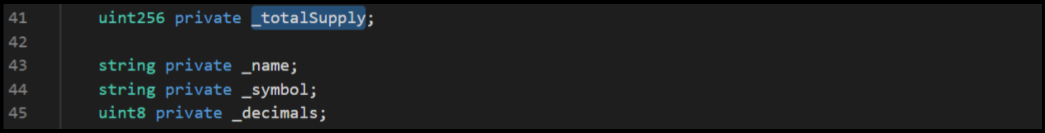
Code Location:

BaseToken.sol Line #185



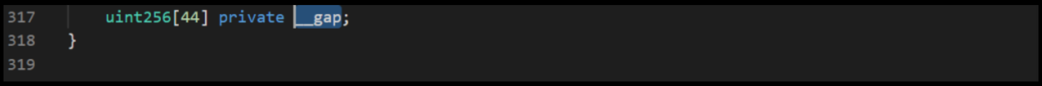
```
184            _totalShares = INITIAL_SHARES;
185            _totalSupply = INITIAL_SUPPLY;
186            _shareBalances[owner()] = _totalShares;
187            _sharesPerBASE = _totalShares.div(_totalSupply);
```

ERC20UpgradeSafe.sol Line #41



```
41      uint256 private _totalSupply;
42
43      string private _name;
44      string private _symbol;
45      uint8 private _decimals;
```

ERC20UpgradeSafe.sol Line #317



```
317      uint256[44] private __gap;
318  }
319
```

Recommendation:

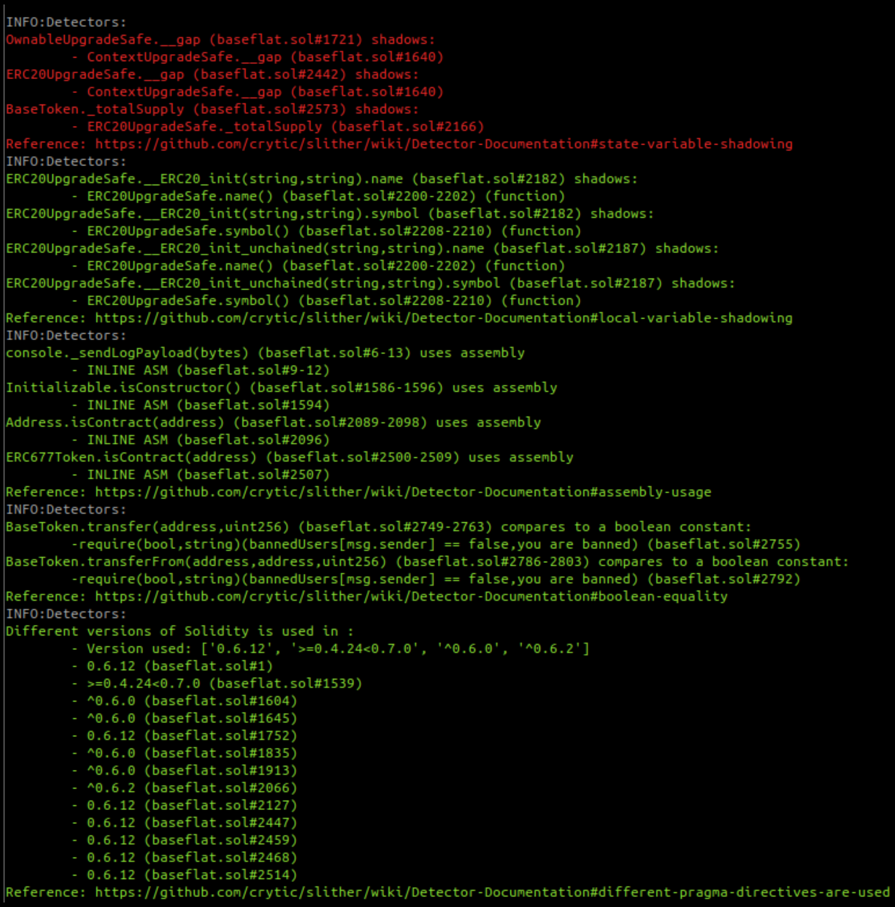
It is recommended to carefully review the variable storage design in the contracts to avoid possible ambiguities.

3.6 STATIC ANALYSIS - CRITICAL

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their abi and binary formats, Slither was run on all contracts (BaseToken.sol, BaseTokenMonetaryPolicy.sol, BaseTokenOrchestrator.sol, Cascade.sol).

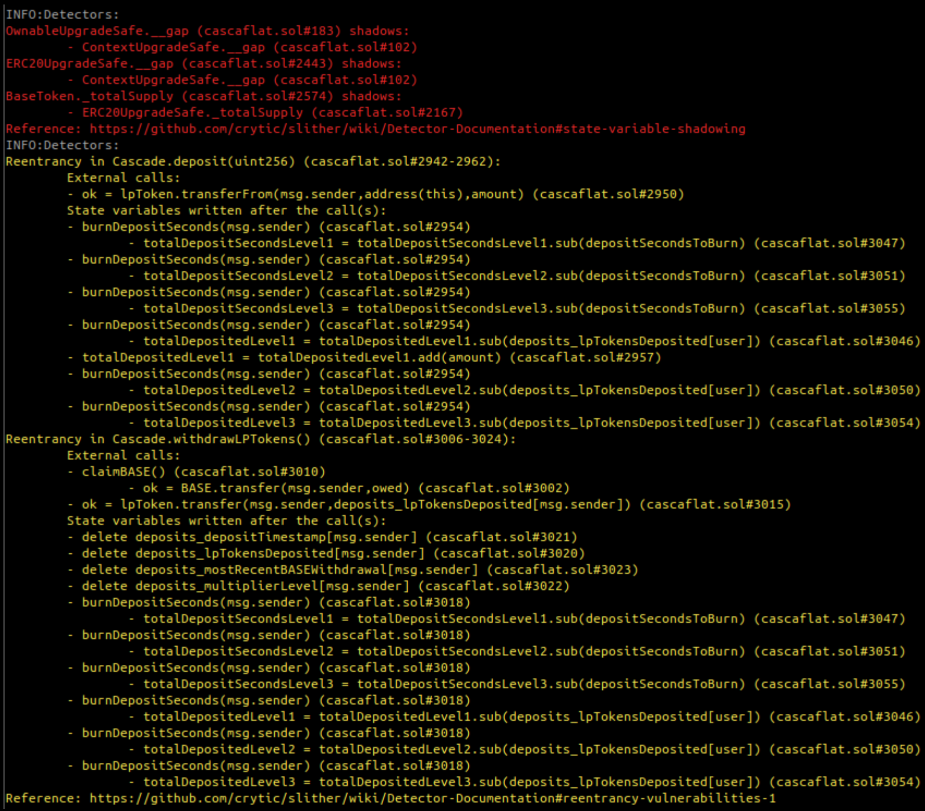
Results:

BaseToken.sol



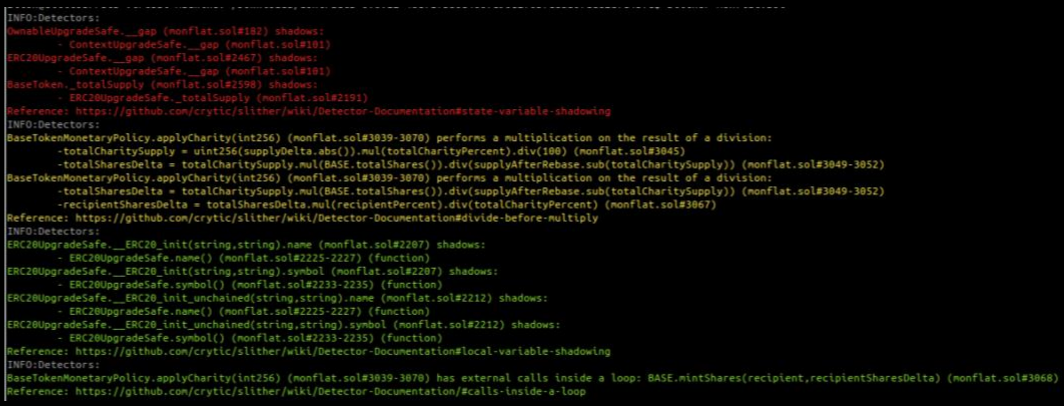
```
INFO:Detectors:
OwnableUpgradeSafe.__gap (baseflat.sol#1721) shadows:
        - ContextUpgradeSafe.__gap (baseflat.sol#1640)
ERC20UpgradeSafe.__gap (baseflat.sol#2442) shadows:
        - ContextUpgradeSafe.__gap (baseflat.sol#1640)
BaseToken._totalSupply (baseflat.sol#2573) shadows:
        - ERC20UpgradeSafe._totalSupply (baseflat.sol#2166)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing
INFO:Detectors:
ERC20UpgradeSafe.__ERC20_init(string,string).name (baseflat.sol#2182) shadows:
        - ERC20UpgradeSafe.name() (baseflat.sol#2200-2202) (function)
ERC20UpgradeSafe.__ERC20_init(string,string).symbol (baseflat.sol#2182) shadows:
        - ERC20UpgradeSafe.symbol() (baseflat.sol#2208-2210) (function)
ERC20UpgradeSafe.__ERC20_init_unchained(string,string).name (baseflat.sol#2187) shadows:
        - ERC20UpgradeSafe.name() (baseflat.sol#2200-2202) (function)
ERC20UpgradeSafe.__ERC20_init_unchained(string,string).symbol (baseflat.sol#2187) shadows:
        - ERC20UpgradeSafe.symbol() (baseflat.sol#2208-2210) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
console._sendLogPayload(bytes) (baseflat.sol#6-13) uses assembly
        - INLINE ASM (baseflat.sol#9-12)
Initializable.isConstructor() (baseflat.sol#1586-1596) uses assembly
        - INLINE ASM (baseflat.sol#1594)
Address.isContract(address) (baseflat.sol#2089-2098) uses assembly
        - INLINE ASM (baseflat.sol#2096)
ERC677Token.isContract(address) (baseflat.sol#2500-2509) uses assembly
        - INLINE ASM (baseflat.sol#2507)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
BaseToken.transfer(address,uint256) (baseflat.sol#2749-2763) compares to a boolean constant:
        -require(bool,string)(bannedUsers[msg.sender] == false,you are banned) (baseflat.sol#2755)
BaseToken.transferFrom(address,address,uint256) (baseflat.sol#2786-2803) compares to a boolean constant:
        -require(bool,string)(bannedUsers[msg.sender] == false,you are banned) (baseflat.sol#2792)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Different versions of Solidity is used in :
        - Version used: ['0.6.12', '>=0.4.24<0.7.0', '^0.6.0', '^0.6.2']
        - 0.6.12 (baseflat.sol#1)
        - >=0.4.24<0.7.0 (baseflat.sol#1539)
        - ^0.6.0 (baseflat.sol#1604)
        - ^0.6.0 (baseflat.sol#1645)
        - 0.6.12 (baseflat.sol#1752)
        - ^0.6.0 (baseflat.sol#1835)
        - ^0.6.0 (baseflat.sol#1913)
        - ^0.6.2 (baseflat.sol#2066)
        - 0.6.12 (baseflat.sol#2127)
        - 0.6.12 (baseflat.sol#2447)
        - 0.6.12 (baseflat.sol#2459)
        - 0.6.12 (baseflat.sol#2468)
        - 0.6.12 (baseflat.sol#2514)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
```

Cascade.sol



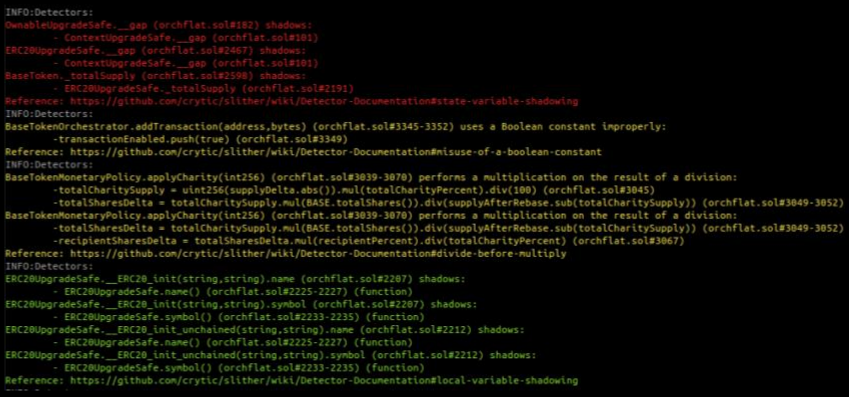
```
INFO:Detectors:
OwnableUpgradeSafe.__gap (cascaflat.sol#183) shadows:
        - ContextUpgradeSafe.__gap (cascaflat.sol#102)
ERC20UpgradeSafe.__gap (cascaflat.sol#2443) shadows:
        - ContextUpgradeSafe.__gap (cascaflat.sol#102)
BaseToken._totalSupply (cascaflat.sol#2574) shadows:
        - ERC20UpgradeSafe._totalSupply (cascaflat.sol#2167)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing
INFO:Detectors:
Reentrancy in Cascade.deposit(uint256) (cascaflat.sol#2942-2962):
        External calls:
        - ok = lpToken.transferFrom(msg.sender,address(this),amount) (cascaflat.sol#2950)
        State variables written after the call(s):
        - burnDepositSeconds(msg.sender) (cascaflat.sol#2954)
                - totalDepositSecondsLevel1 = totalDepositSecondsLevel1.sub(depositSecondsToBurn) (cascaflat.sol#3047)
        - burnDepositSeconds(msg.sender) (cascaflat.sol#2954)
                - totalDepositSecondsLevel2 = totalDepositSecondsLevel2.sub(depositSecondsToBurn) (cascaflat.sol#3051)
        - burnDepositSeconds(msg.sender) (cascaflat.sol#2954)
                - totalDepositSecondsLevel3 = totalDepositSecondsLevel3.sub(depositSecondsToBurn) (cascaflat.sol#3055)
        - burnDepositSeconds(msg.sender) (cascaflat.sol#2954)
                - totalDepositedLevel1 = totalDepositedLevel1.sub(deposits_lpTokensDeposited[user]) (cascaflat.sol#3046)
        - totalDepositedLevel1 = totalDepositedLevel1.add(amount) (cascaflat.sol#2957)
        - burnDepositSeconds(msg.sender) (cascaflat.sol#2954)
                - totalDepositedLevel2 = totalDepositedLevel2.sub(deposits_lpTokensDeposited[user]) (cascaflat.sol#3050)
        - burnDepositSeconds(msg.sender) (cascaflat.sol#2954)
                - totalDepositedLevel3 = totalDepositedLevel3.sub(deposits_lpTokensDeposited[user]) (cascaflat.sol#3054)
Reentrancy in Cascade.withdrawLPTokens() (cascaflat.sol#3006-3024):
        External calls:
        - claimBASE() (cascaflat.sol#3010)
                - ok = BASE.transfer(msg.sender,owed) (cascaflat.sol#3002)
        - ok = lpToken.transfer(msg.sender,deposits_lpTokensDeposited[msg.sender]) (cascaflat.sol#3015)
        State variables written after the call(s):
        - delete deposits_depositTimestamp[msg.sender] (cascaflat.sol#3021)
        - delete deposits_lpTokensDeposited[msg.sender] (cascaflat.sol#3020)
        - delete deposits_mostRecentBASEWithdrawal[msg.sender] (cascaflat.sol#3023)
        - delete deposits_multiplierLevel[msg.sender] (cascaflat.sol#3022)
        - burnDepositSeconds(msg.sender) (cascaflat.sol#3018)
                - totalDepositSecondsLevel1 = totalDepositSecondsLevel1.sub(depositSecondsToBurn) (cascaflat.sol#3047)
        - burnDepositSeconds(msg.sender) (cascaflat.sol#3018)
                - totalDepositSecondsLevel2 = totalDepositSecondsLevel2.sub(depositSecondsToBurn) (cascaflat.sol#3051)
        - burnDepositSeconds(msg.sender) (cascaflat.sol#3018)
                - totalDepositSecondsLevel3 = totalDepositSecondsLevel3.sub(depositSecondsToBurn) (cascaflat.sol#3055)
        - burnDepositSeconds(msg.sender) (cascaflat.sol#3018)
                - totalDepositedLevel1 = totalDepositedLevel1.sub(deposits_lpTokensDeposited[user]) (cascaflat.sol#3046)
        - burnDepositSeconds(msg.sender) (cascaflat.sol#3018)
                - totalDepositedLevel2 = totalDepositedLevel2.sub(deposits_lpTokensDeposited[user]) (cascaflat.sol#3050)
        - burnDepositSeconds(msg.sender) (cascaflat.sol#3018)
                - totalDepositedLevel3 = totalDepositedLevel3.sub(deposits_lpTokensDeposited[user]) (cascaflat.sol#3054)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
```

BaseTokenMonetaryPolicy.sol



```
INFO:Detectors:
OwnableUpgradeSafe.__gap (monflat.sol#182) shadows:
        - ContextUpgradeSafe.__gap (monflat.sol#101)
ERC20UpgradeSafe.__gap (monflat.sol#2467) shadows:
        - ContextUpgradeSafe.__gap (monflat.sol#101)
BaseToken._totalSupply (monflat.sol#2598) shadows:
        - ERC20UpgradeSafe._totalSupply (monflat.sol#2191)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing
INFO:Detectors:
BaseTokenMonetaryPolicy.applyCharity(int256) (monflat.sol#3039-3070) performs a multiplication on the result of a division:
        -totalCharitySupply = uint256(supplyDelta.abs()).mul(totalCharityPercent).div(100) (monflat.sol#3045)
        -totalSharesDelta = totalCharitySupply.mul(BASE.totalShares()).div(supplyAfterRebase.sub(totalCharitySupply)) (monflat.sol#3049-3052)
BaseTokenMonetaryPolicy.applyCharity(int256) (monflat.sol#3039-3070) performs a multiplication on the result of a division:
        -totalSharesDelta = totalCharitySupply.mul(BASE.totalShares()).div(supplyAfterRebase.sub(totalCharitySupply)) (monflat.sol#3049-3052)
        -recipientSharesDelta = totalSharesDelta.mul(recipientPercent).div(totalCharityPercent) (monflat.sol#3067)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
ERC20UpgradeSafe.__ERC20_init(string,string).name (monflat.sol#2207) shadows:
        - ERC20UpgradeSafe.name() (monflat.sol#2225-2227) (function)
ERC20UpgradeSafe.__ERC20_init(string,string).symbol (monflat.sol#2207) shadows:
        - ERC20UpgradeSafe.symbol() (monflat.sol#2233-2235) (function)
ERC20UpgradeSafe.__ERC20_init_unchained(string,string).name (monflat.sol#2212) shadows:
        - ERC20UpgradeSafe.name() (monflat.sol#2225-2227) (function)
ERC20UpgradeSafe.__ERC20_init_unchained(string,string).symbol (monflat.sol#2212) shadows:
        - ERC20UpgradeSafe.symbol() (monflat.sol#2233-2235) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
BaseTokenMonetaryPolicy.applyCharity(int256) (monflat.sol#3039-3070) has external calls inside a loop: BASE.mintShares(recipient,recipientSharesDelta) (monflat.sol#3068)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
```

BaseTokenOrchestrator.sol



```
INFO:Detectors:
OwnableUpgradeSafe.__gap (orchflat.sol#182) shadows:
        - ContextUpgradeSafe.__gap (orchflat.sol#101)
ERC20UpgradeSafe.__gap (orchflat.sol#2467) shadows:
        - ContextUpgradeSafe.__gap (orchflat.sol#101)
BaseToken._totalSupply (orchflat.sol#2598) shadows:
        - ERC20UpgradeSafe._totalSupply (orchflat.sol#2191)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing
INFO:Detectors:
BaseTokenOrchestrator.addTransaction(address,bytes) (orchflat.sol#3345-3352) uses a Boolean constant improperly:
        -transactionEnabled.push(true) (orchflat.sol#3349)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#misuse-of-a-boolean-constant
INFO:Detectors:
BaseTokenMonetaryPolicy.applyCharity(int256) (orchflat.sol#3039-3070) performs a multiplication on the result of a division:
        -totalCharitySupply = uint256(supplyDelta.abs()).mul(totalCharityPercent).div(100) (orchflat.sol#3045)
        -totalSharesDelta = totalCharitySupply.mul(BASE.totalShares()).div(supplyAfterRebase.sub(totalCharitySupply)) (orchflat.sol#3049-3052)
BaseTokenMonetaryPolicy.applyCharity(int256) (orchflat.sol#3039-3070) performs a multiplication on the result of a division:
        -totalSharesDelta = totalCharitySupply.mul(BASE.totalShares()).div(supplyAfterRebase.sub(totalCharitySupply)) (orchflat.sol#3049-3052)
        -recipientSharesDelta = totalSharesDelta.mul(recipientPercent).div(totalCharityPercent) (orchflat.sol#3067)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
ERC20UpgradeSafe.__ERC20_init(string,string).name (orchflat.sol#2207) shadows:
        - ERC20UpgradeSafe.name() (orchflat.sol#2225-2227) (function)
ERC20UpgradeSafe.__ERC20_init(string,string).symbol (orchflat.sol#2207) shadows:
        - ERC20UpgradeSafe.symbol() (orchflat.sol#2233-2235) (function)
ERC20UpgradeSafe.__ERC20_init_unchained(string,string).name (orchflat.sol#2212) shadows:
        - ERC20UpgradeSafe.name() (orchflat.sol#2225-2227) (function)
ERC20UpgradeSafe.__ERC20_init_unchained(string,string).symbol (orchflat.sol#2212) shadows:
        - ERC20UpgradeSafe.symbol() (orchflat.sol#2233-2235) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
```

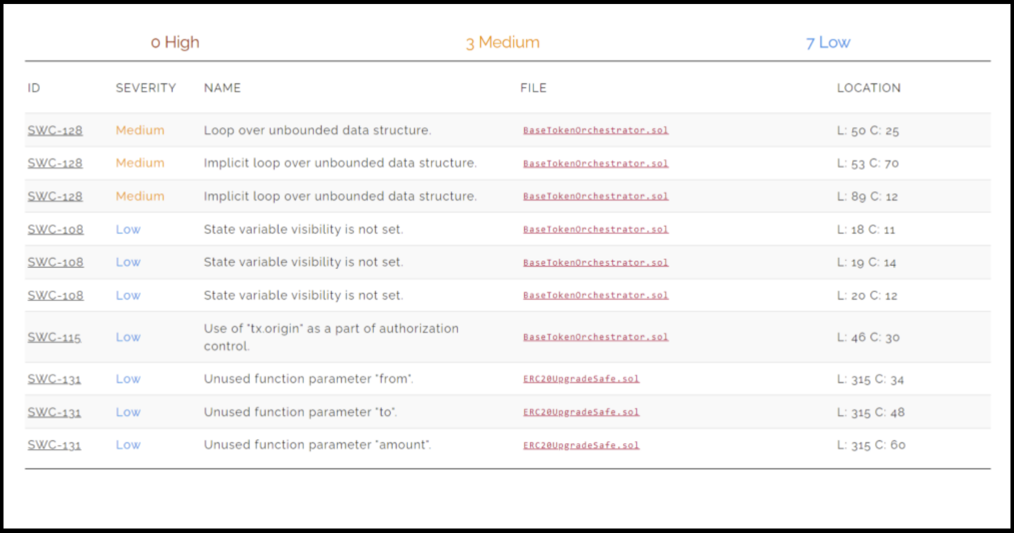
3.7 AUTOMATED SECURITY SCAN - INFORMATIONAL

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruit on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the testers machine and sent the compiled results to the analyzers
to locate any vulnerabilities. Security Detections are only in scope, and the analysis was pointed towards issues with BaseTokenOrchestrator.sol and ERC20UpgradeSafe.sol.

Results:

MythX detected 0 High findings, 3 Medium, and 7 Low.



0 High | 3 Medium | 7 Low

ID	SEVERITY	NAME	FILE	LOCATION
SWC-128	Medium	Loop over unbounded data structure.	BaseTokenOrchestrator.sol	L: 50 C: 25
SWC-128	Medium	Implicit loop over unbounded data structure.	BaseTokenOrchestrator.sol	L: 53 C: 70
SWC-128	Medium	Implicit loop over unbounded data structure.	BaseTokenOrchestrator.sol	L: 89 C: 12
SWC-108	Low	State variable visibility is not set.	BaseTokenOrchestrator.sol	L: 18 C: 11
SWC-108	Low	State variable visibility is not set.	BaseTokenOrchestrator.sol	L: 19 C: 14
SWC-108	Low	State variable visibility is not set.	BaseTokenOrchestrator.sol	L: 20 C: 12
SWC-115	Low	Use of "tx.origin" as a part of authorization control.	BaseTokenOrchestrator.sol	L: 46 C: 30
SWC-131	Low	Unused function parameter "from".	ERC20UpgradeSafe.sol	L: 315 C: 34
SWC-131	Low	Unused function parameter "to".	ERC20UpgradeSafe.sol	L: 315 C: 48
SWC-131	Low	Unused function parameter "amount".	ERC20UpgradeSafe.sol	L: 315 C: 60

THANK YOU FOR CHOOSING

// HALBORN