EGL - Genesis

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **May 24th, 2021 - June 1st, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	05/24/2021	Gokberk Gulgun
0.2	Document Updates	05/28/2021	Gabi Urrutia
1.0	Final Version	06/01/2021	Gabi Urrutia
1.1	Remediation Plan	06/09/2021	Gokberk Gulgun

CONTACTS

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

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

EGL engaged Halborn to conduct a security assessment on their Smart contract beginning on May 24th, 2021 and ending June 1st, 2021.

The security assessment was scoped to the smart contract repository. An audit of the security risk and implications regarding the changes introduced by the development team at EGL prior to its production release shortly following the assessments deadline.

Though this security audit's outcome is satisfactory, only the most essential aspects were tested and verified to achieve objectives and deliverables set in the scope due to time and resource constraints. It is essential to note the use of the best practices for secure smart-contract development.

1.2 AUDIT SUMMARY

The team at Halborn was provided a week for the engagement and assigned two full time security engineers to audit the security of the smart contract. The security engineers are blockchain and smart-contract security experts with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit to achieve the following:

- Ensure that smart contract functions are intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn identified few security risks, and recommends performing further testing to validate extended safety and correctness in context to the whole of contract. External threats, such as economic attacks, oracle attacks, and inter-contract functions and calls should be validated for expected logic and state.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to 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.
- Static Analysis of security for scoped contract, and imported functions.(Slither)
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Symbolic Execution / EVM bytecode security assessment (Manticore)
- Testnet deployment (Truffle, Ganache)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident, and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

Code related to EglGenesis.sol.

Specific commit ID of contract:
ae62f0872f714ab95febe17f53e01957f80ec6fa
Fixed commit ID:
1f3c3c88c4ffc27d97df553ee3c9c16b55a2fcbe

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	4	5

LIKELIHOOD

IMPACT

	(HAL-01)			
(HAL-07) (HAL-08) (HAL-09)	(HAL-02) (HAL-03) (HAL-04)			
(HAL-05) (HAL-06)				

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - FLOATING PRAGMA	Low	SOLVED: 06/08/2021
HAL02 - MISSING ADDRESS VALIDATION	Low	SOLVED: 06/08/2021
HAL03 - MISSING CALCULATION ON THE CONTRIBUTORS COUNT	Low	SOLVED: 06/08/2021
HAL04 - ALLOW WITHDRAW PROGRESS WITHOUT FUNDS	Low	SOLVED: 06/08/2021
HAL05 - OWNER CAN RENOUNCE OWNERSHIP	Informational	SOLVED: 06/08/2021
HAL06 - MISSING VALIDATION ON THE FUNCTION	Informational	RISK ACCEPTED: 06/08/2021
HAL07 - POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	SOLVED: 06/08/2021
HAL08 - BLOCK TIMESTAMP ALIAS USAGE	Informational	NOT APPLICABLE: 06/08/2021
HAL09 - LACK OF VISIBILITY ON THE MAXTHRESHOLD VARIABLE	Informational	RISK ACCEPTED: 06/08/2021
SYMBOLIC EXECUTION SECURITY ASSESMENT	-	-
STATIC ANALYSIS	-	-
AUTOMATED SECURITY SCAN RESULTS	-	-

FINDINGS & TECH DETAILS

3.1 (HAL-01) FLOATING PRAGMA - LOW

Description:

EglGenesis.sol contract use the floating pragma ^0.6.0. Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the **pragma** helps to ensure that contracts do not accidentally get deployed using another pragma, for example, either an outdated pragma version that might introduce bugs that affect the contract system negatively or a recently released pragma version which has not been extensively tested.

Reference: ConsenSys Diligence - Lock pragmas

Code Location:

EglGenesis.sol Line #1

Listing 1: EglGenesis.sol (Lines 1)

```
1 pragma solidity ^0.6.0;
```

- This is an example where the floating pragma is used. ^0.6.0.

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Consider lock the pragma version known bugs for the compiler version. Therefore, it is recommended not to use floating pragma in the production. Apart from just locking the pragma version in the code, the sign (>=) need to be removed. it is possible locked the pragma fixing the version both in truffle-config.js if you use the Truffle framework and

in hardhat.config.js if you use HardHat framework for the deployment.

truffle-config.js

```
// Configure your compilers
compilers: {
  solc: {
     version: "0.7.5",    // Fetch exact version from solc-bin (default: truffle's version)
    // docker: true,        // Use "0.5.1" you've installed locally with docker (default: false)
    // settings: {          // See the solidity docs for advice about optimization and evmVersion
    //  optimizer: {
    //    enabled: false,
    //    runs: 200
    //  },
    //  evmVersion: "byzantium"
    // }
  }
}
};
```

hardhat.config.js

```
/**
 * @type import('hardhat/config').HardhatUserConfig
 */
module.exports = {
  solidity: "0.7.5",
};
```

Remediation Plan:

SOLVED: EGL Team locked pragma version to 0.6.6.

Listing 2: EglGenesis.sol (Lines 1)

```
1 pragma solidity 0.6.6;
2
3 import "@openzeppelin/contracts-upgradeable/access/
      OwnableUpgradeable.sol";
4 import "@openzeppelin/contracts-upgradeable/proxy/Initializable.
      sol";
5 import "@openzeppelin/contracts-upgradeable/utils/
      PausableUpgradeable.sol";
6 import "@openzeppelin/contracts-upgradeable/math/
      SafeMathUpgradeable.sol";
```

3.2 (HAL-02) MISSING ADDRESS VALIDATION - LOW

Description:

In the EglGenesis.sol contract is missing a safety check inside their constructors and multiple functions. Setters of address type parameters should include a zero-address check otherwise contract functionality may become inaccessible or tokens burnt forever.

Code Location:

EglGenesis.sol Line #~103-112

Listing 3: EglGenesis.sol (Lines 103)

```
103     function initialize(address _owner, uint _threshold) public
            initializer {
104         __Context_init_unchained();
105         __Ownable_init_unchained();
106         __Pausable_init_unchained();
107
108         transferOwnership(_owner);
109         canContribute = true;
110         maxThreshold = _threshold;
111         emit Initialized(owner(), canContribute, _threshold);
112     }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Add proper address validation when assigning a value to a variable from user-supplied data. Better yet, address white-listing/black-listing

should be implemented in relevant functions if possible.

For example:

Listing 4: Modifier.sol (Lines 2,3,4)

```
1  modifier validAddress(address addr) {
2      require(addr != address(0), "Address cannot be 0x0");
3      require(addr != address(this), "Address cannot be contract");
4      _;
5  }
```

Remediation Plan:

SOLVED: EGL Team added address validation on the initialize function. initialize function defined instead of a constructor in the contract.

Listing 5: EglGenesis.sol (Lines 2,3)

```
1      function initialize(address _owner, uint _threshold) external
           initializer {
2          require(_owner != address(0), "GENESIS:INVALID_OWNER");
3          require(_owner != address(this), "GENESIS:
               ADDRESS_IS_CONTRACT");
4
5          __Context_init_unchained();
6          __Ownable_init_unchained();
7          __Pausable_init_unchained();
8
9          transferOwnership(_owner);
10         canContribute = true;
11         maxThreshold = _threshold;
12         emit Initialized(owner(), canContribute, _threshold);
13     }
```

3.3 (HAL-03) MISSING CALCULATION ON THE CONTRIBUTORS COUNT - LOW

Description:

During the phase of EglGenesis.sol contract, the user can withdraw contribution. The contract is applied multiple calculations and controls on the related functions. But, contributorsCount is not decreased when the user withdraw funds.

Code Location:

EglGenesis.sol Line #~117

Listing 6: EglGenesis.sol (Lines)

```
117     function withdraw() public whenNotPaused {
118         require(canWithdraw, "GENESIS:WITHDRAW_NOT_ALLOWED");
119         require(contributors[msg.sender].amount > 0, "GENESIS:
                NOT_CONTRIBUTED");
120
121         uint amountToWithdraw = contributors[msg.sender].amount;
122         uint contributorIdx = contributors[msg.sender].idx;
123         delete contributors[msg.sender];
124         delete contributorsList[contributorIdx - 1];
125
126         cumulativeBalance = cumulativeBalance.sub(amountToWithdraw
                );
127         (bool success, ) = msg.sender.call{ value:
                amountToWithdraw}("");
128         require(success, "GENESIS:WITHDRAW_FAILED");
129         emit ContributionWithdrawn(msg.sender, amountToWithdraw,
                now);
130     }
131
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Add proper calculation on the contributorsCount variable. According to workflow, EGL can continue without substraction operation.

Listing 7: Example Remediation (Lines 9)

```
1     function withdraw() public whenNotPaused {
2         require(canWithdraw, "GENESIS:WITHDRAW_NOT_ALLOWED");
3         require(contributors[msg.sender].amount > 0, "GENESIS:
              NOT_CONTRIBUTED");
4         uint amountToWithdraw = contributors[msg.sender].amount;
5         uint contributorIdx = contributors[msg.sender].idx;
6         delete contributors[msg.sender];
7         delete contributorsList[contributorIdx - 1];
8
9         contributorsCount = contributorsCount.sub(1);
10        cumulativeBalance = cumulativeBalance.sub(amountToWithdraw
              );
11        (bool success, ) = msg.sender.call{ value:
              amountToWithdraw}("");
12        require(success, "GENESIS:WITHDRAW_FAILED");
13        emit ContributionWithdrawn(msg.sender, amountToWithdraw,
              now);
14    }
15
```

Remediation Plan:

SOLVED: The main purpose of the the contributorsCount variable is in combination with the contributorsList array. According to workflow, the variable name is changed with absoluteMaxContributorsCount.

3.4 (HAL-04) ALLOW WITHDRAW PROGRESS WITHOUT FUNDS - LOW

Description:

During the phase of EglGenesis.sol contract, only the owner can allow withdraw progress. But, without funds the owner can allow withdraw phase.

Code Location:

EglGenesis.sol Line #~117

Listing 8: EglGenesis.sol (Lines)

```
135     function allowWithdraw() public onlyOwner whenNotPaused {
136         require(cumulativeBalance < maxThreshold, "GENESIS:
                MAX_THRESHOLD_REACHED");
137         require(canContribute, "GENESIS:GENESIS_ENDED");
138         canWithdraw = true;
139         canContribute = false;
140         emit WithdAllowed(msg.sender, now);
141     }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommend to validate allowWithdraw function should not be called without funds.

Listing 9: EglGenesis.sol (Lines 2)

```
2     function allowWithdraw() public onlyOwner whenNotPaused {
3         require(cumulativeBalance > 0, "GENESIS:NO BALANCE");
```

```
4         require(cumulativeBalance < maxThreshold, "GENESIS:
              MAX_THRESHOLD_REACHED");
5         require(canContribute, "GENESIS:GENESIS_ENDED");
6         canWithdraw = true;
7         canContribute = false;
8         emit WithdAllowed(msg.sender, now);
9     }
```

Remediation Plan:

SOLVED: The cumulative balance check has been added into the allowWithdraw function.

Listing 10: Fix (Lines)

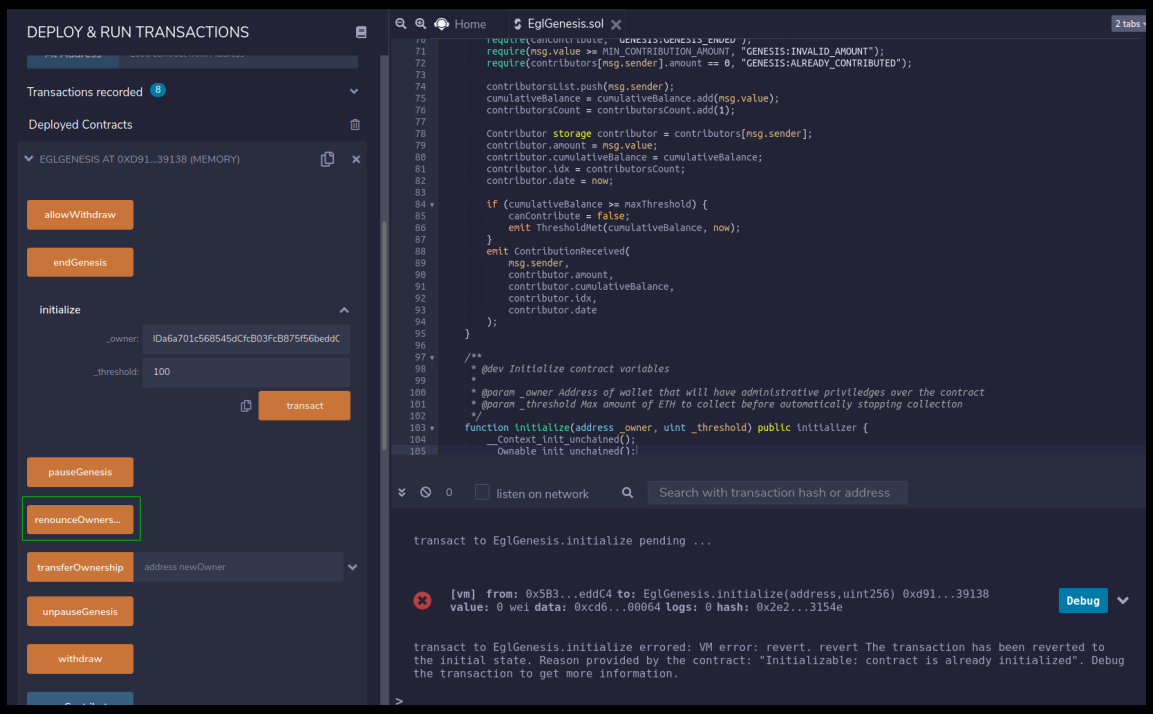
```
1 require(cumulativeBalance > 0, "GENESIS:NO_BALANCE");
```

3.5 (HAL-05) OWNER CAN RENOUNCE OWNERSHIP - INFORMATIONAL

Description:

The Owner of the contract is usually the account which deploys the contract. As a result, the Owner is able to perform some privileged actions. In the EglGenesis.sol smart contracts, the renounceOwnership function is used to renounce being Owner. Otherwise, if the ownership was not transferred before, the contract will never have an Owner, which is dangerous.

Function:



Risk Level:

Likelihood - 1

Impact - 1

Recommendation:

It's recommended that the Owner is not able to call renounceOwnership without transferring the Ownership to other address before. In addition, if a multi-signature wallet is used, calling renounceOwnership function should be confirmed for two or more users. As an other solution, Renounce Ownership functionality can be disabled with the following line.

For example:

Listing 11: Modifier.sol (Lines 3)

```
2  function renounceOwnership() public override onlyOwner {
3          revert("can't renounceOwnership here"); //not possible
               with this smart contract
4      }
```

Remediation Plan:

SOLVED: The contract will not allow owner to renounce ownership. Only transferOwnership function permitted.

Listing 12: Fix (Lines)

```
1      function renounceOwnership() public override onlyOwner {
2          revert("GENESIS:NO_RENOUNCE_OWNERSHIP");
3      }
```

3.6 (HAL-06) MISSING VALIDATION ON THE FUNCTION - INFORMATIONAL

Description:

According to workflow, The owner of the contract can finish genesis progress. In the related function, `require(canContribute, "GENESIS: GENESIS_ENDED");` statement is missing.

Code Location:

EglGenesis.sol Line #~146

Listing 13: EglGenesis.sol (Lines)

```
146     function endGenesis() public onlyOwner whenNotPaused {
147         canContribute = false;
148         (bool success, ) = msg.sender.call{ value:
                cumulativeBalance}("");
149         require(success, "GENESIS: CLOSE_FAILED");
150         emit GenesisEnded(msg.sender, cumulativeBalance, address(
                this).balance, now);
151     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommend to put a `require` statement on the `endGenesis` function.

Listing 14: EglGenesis.sol (Lines 147)

```
146     function endGenesis() public onlyOwner whenNotPaused {
147         require(canContribute, "GENESIS:GENESIS_ENDED");
```

```
148         canContribute = false;
149         (bool success, ) = msg.sender.call{ value:
                cumulativeBalance}("");
150         require(success, "GENESIS: CLOSE_FAILED");
151         emit GenesisEnded(msg.sender, cumulativeBalance, address(
                this).balance, now);
152     }
```

Remediation Plan:

RISK ACCEPTED: In the opinion of the client, It is possible to call endGenesisfunction even if it doesn't hit the maxThreshold. The client wanted that flexibility so that even if the client don't hit their target ETH amount, they could still continue with Genesis if the amount they do have is satisfactory to the owner multisig.

3.7 (HAL-07) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL

Description:

In the public functions, array arguments are immediately copied to memory, while external functions can read directly from calldata. Reading calldata is cheaper than memory allocation. Public functions need to write the arguments to memory because public functions may be called internally. Internal calls are passed internally by pointers to memory. Thus, the function expects its arguments being located in memory when the compiler generates the code for an internal function.

Code Location:

We noticed the use of public functions in the following contract:

- EglGenesis.sol
-

Listing 15: EglGenesis.sol (Lines)

```
37      - EglGenesis.initialize(address,uint256) (contracts/EglGenesis
            .sol#103-112)
38      - EglGenesis.withdraw() (contracts/EglGenesis.sol#117-130)
39      - EglGenesis.allowWithdraw() (contracts/EglGenesis.sol
            #135-141)
40      - EglGenesis.endGenesis() (contracts/EglGenesis.sol#146-151)
41      - EglGenesis.pauseGenesis() (contracts/EglGenesis.sol#156-158)
42      - EglGenesis.unpauseGenesis() (contracts/EglGenesis.sol
            #163-165)
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

Consider declaring external variables instead of public variables. A best practice is to use external if expecting a function to only be called externally and public if called internally. Public functions are always accessible, but external functions are only available to external callers.

Remediation Plan:

SOLVED: The client declared public functions as an external. The necessary changes applied on the relevant functions.

3.8 (HAL-08) BLOCK TIMESTAMP ALIAS USAGE - INFORMATIONAL

Description:

During a manual static review, we noticed the use of now. The contract developers should be aware that his does not mean current time. “now” is an alias for “block.timestamp”. “block.timestamp” can be influenced by miners to a certain degree, so the testers should be warned that this may have some risk if miners collude on time manipulation to influence the price oracles.

Code Location:

- ./contracts/EglGenesis.sol Line #83

Listing 16: EglGenesis.sol (Lines 82)

```
69      receive() external payable whenNotPaused {
70          require(canContribute, "GENESIS:GENESIS_ENDED");
71          require(msg.value >= MIN_CONTRIBUTION_AMOUNT, "GENESIS:
                INVALID_AMOUNT");
72          require(contributors[msg.sender].amount == 0, "GENESIS:
                ALREADY_CONTRIBUTED");
73
74          contributorsList.push(msg.sender);
75          cumulativeBalance = cumulativeBalance.add(msg.value);
76          contributorsCount = contributorsCount.add(1);
77
78          Contributor storage contributor = contributors[msg.sender
                ];
79          contributor.amount = msg.value;
80          contributor.cumulativeBalance = cumulativeBalance;
81          contributor.idx = contributorsCount;
82          contributor.date = now;
83
84          if (cumulativeBalance >= maxThreshold) {
85              canContribute = false;
86              emit ThresholdMet(cumulativeBalance, now);
87          }
```

```
88          emit ContributionReceived(
89              msg.sender,
90              contributor.amount,
91              contributor.cumulativeBalance,
92              contributor.idx,
93              contributor.date
94          );
95      }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

Use block.number instead of block.timestamp or now reduce the influence of miners. If possible, It's recommended to use Oracles.

Remediation plan:

NOT APPLICABLE: the EGL Team considers safe the usage of now because 900 seconds of drift from miners is preferable to other options. Calculating time from the block could be wrong if there is a fork or upgrade - timestamps are less vulnerable to a change in block duration that could occur with Ethereum 2.0 upgrades or hard forks. Use of oracles would create a dependency on the health of a third party service and potentially incur additional fees.

3.9 (HAL-09) LACK OF VISIBILITY ON THE MAXTHRESHOLD VARIABLE - INFORMATIONAL

Description:

During the dynamic analysis, we noticed the visibility of `uint private maxThreshold` variable marked as a private. After an initializing phase, `maxThreshold` variable is not visible through functions.

Code Location:

- `./contracts/EglGenesis.sol` Line #20

Listing 17: EglGenesis.sol (Lines 20)

```
20 uint private maxThreshold;
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

According to workflow, EGL Team should decide about the variable visibility. As an example remediation, Halborn Team suggested the following code.

Listing 18: EglGenesis.sol (Lines 20)

```
20  function getMaxThreshold() external onlyOwner whenNotPaused
        returns (uint){
21          return maxThreshold;
22      }
```

Remediation plan:

RISK ACCEPTED: the EGL Team considers to declaring maxThreshold as a private.

3.10 SYMBOLIC EXECUTION SECURITY ASSESSMENT

Description:

The tool used to perform the symbolic execution for analyzing Smart Contract is Manticore. In general, obtaining all possible states of a Smart Contract by symbolic execution is the main goal for using this tool. Briefly, concrete values are replaced by symbolic values and variables which are used to generate path conditions which are logic formulas that represent the state of the program and the transformations between program states. Some detectors were used in the audit to find some vulnerabilities such as: Integer Overflow, Simple and Advance Reentrancy and Delegate calls.

Results:

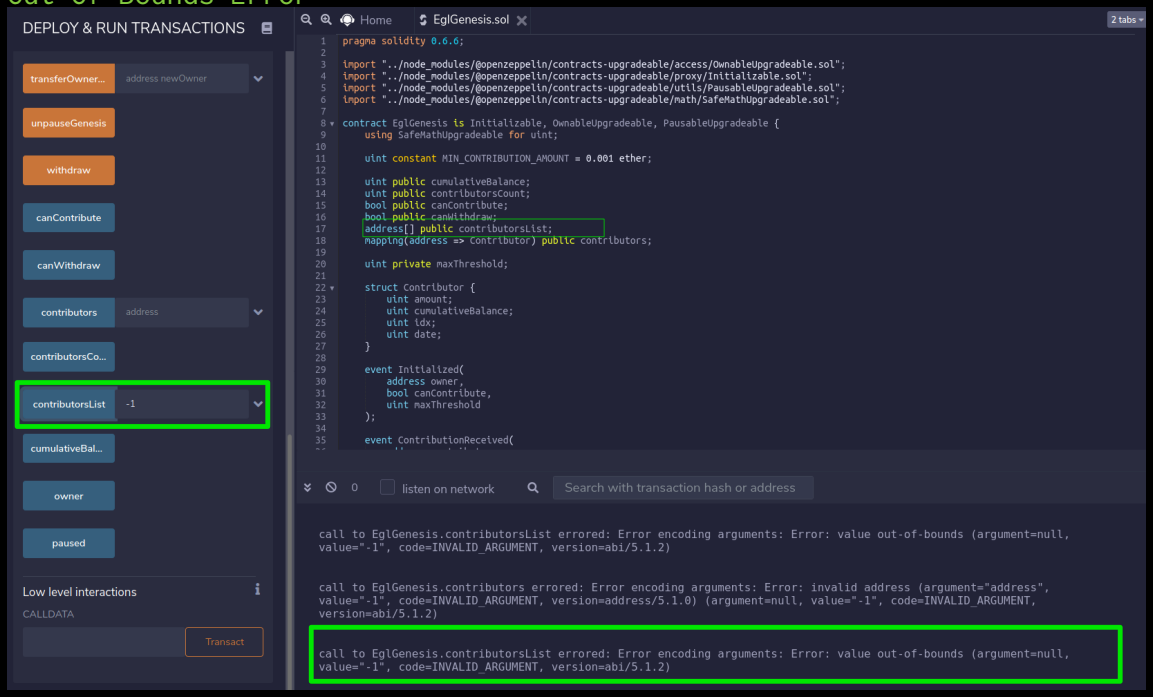
```
2021-06-04 09:46:39,491: [22] m.main:INFO: Beginning analysis
2021-06-04 09:46:39,495: [22] m.e.manticore:INFO: Starting symbolic create contract
2021-06-04 09:46:47,226: [22] m.e.manticore:INFO: Starting symbolic transaction: 0
2021-06-04 09:49:03,591: [22] m.e.detectors:WARNING: INVALID instruction
2021-06-04 09:58:57,295: [22] m.e.manticore:INFO: 7 alive states, 30 terminated states
2021-06-04 09:58:58,342: [183] m.c.plugin:WARNING: Caught will_solve in state None, but failed to capture its initialization
2021-06-04 09:58:58,539: [184] m.c.manticore:INFO: Generated testcase No. 0 - RETURN(2 txs)
2021-06-04 09:58:58,541: [184] m.c.plugin:WARNING: Caught will_solve in state None, but failed to capture its initialization
2021-06-04 09:58:58,644: [186] m.c.manticore:INFO: Generated testcase No. 1 - RETURN(2 txs)
2021-06-04 09:58:58,649: [186] m.c.plugin:WARNING: Caught will_solve in state None, but failed to capture its initialization
2021-06-04 09:58:59,024: [188] m.c.manticore:INFO: Generated testcase No. 2 - RETURN(2 txs)
2021-06-04 09:58:59,029: [188] m.c.plugin:WARNING: Caught will_solve in state None, but failed to capture its initialization
2021-06-04 09:58:59,290: [189] m.c.manticore:INFO: Generated testcase No. 3 - RETURN(2 txs)
2021-06-04 09:58:59,294: [189] m.c.plugin:WARNING: Caught will_solve in state None, but failed to capture its initialization
2021-06-04 09:58:59,412: [192] m.c.manticore:INFO: Generated testcase No. 4 - RETURN(2 txs)
2021-06-04 09:58:59,416: [192] m.c.plugin:WARNING: Caught will_solve in state None, but failed to capture its initialization
2021-06-04 09:58:59,536: [195] m.c.manticore:INFO: Generated testcase No. 5 - RETURN(2 txs)
2021-06-04 09:58:59,540: [195] m.c.plugin:WARNING: Caught will_solve in state None, but failed to capture its initialization
2021-06-04 09:59:00,175: [183] m.c.manticore:INFO: Generated testcase No. 6 - RETURN(2 txs)
2021-06-04 10:00:04,736: [22] m.c.plugin:WARNING: Caught will_solve in state None, but failed to capture its initialization
2021-06-04 10:00:25,267: [22] m.c.manticore:INFO: Results in /share/mcore__4wn7ckg
```

Listing 19: Manticore Analysis (Lines)

```
20 - INVALID instruction -
21   Contract: 0xfeb7f3e0e9e07a37df869581cb75dd9f4055cf16  EVM
        Program counter: 0x8fe
22   Solidity snippet:
23     17  address[] public contributorsList
```

With the results from the Manticore, Halborn Team verified issue through dynamic analysis. As can be seen from the picture below, the finding produced by the Manticore tool was evaluated as false positive.

Out-Of-Bounds Error



The EGL Team can access the address array via a function instead of defining it as public. Thus, the bounds of the array can be checked. This is completely left to the EGL Team.

3.11 STATIC ANALYSIS REPORT

Description:

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. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Results:

EglGenesis.sol

```
INFO:Detectors:
OwnableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#74) shadows:
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#31)
PausableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/PausableUpgradeable.sol#96) shadows:
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#31)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing
INFO:Detectors:
EglGenesis.initialize(address,uint256)._owner (contracts/EglGenesis.sol#103) shadows:
        - OwnableUpgradeable._owner (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#20) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Reentrancy in EglGenesis.endGenesis() (contracts/EglGenesis.sol#146-151):
        External calls:
        - (success) = msg.sender.call{value: cumulativeBalance}() (contracts/EglGenesis.sol#148)
        Event emitted after the call(s):
        - GenesisEnded(msg.sender,cumulativeBalance,address(this).balance,now) (contracts/EglGenesis.sol#150)
Reentrancy in EglGenesis.withdraw() (contracts/EglGenesis.sol#117-130):
        External calls:
        - (success) = msg.sender.call{value: amountToWithdraw}() (contracts/EglGenesis.sol#127)
        Event emitted after the call(s):
        - ContributionWithdrawn(msg.sender,amountToWithdraw,now) (contracts/EglGenesis.sol#129)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
EglGenesis.receive() (contracts/EglGenesis.sol#69-95) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(contributors[msg.sender].amount == 0,GENESIS:ALREADY_CONTRIBUTED) (contracts/EglGenesis.sol#72)
EglGenesis.withdraw() (contracts/EglGenesis.sol#117-130) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(contributors[msg.sender].amount > 0,GENESIS:NOT_CONTRIBUTED) (contracts/EglGenesis.sol#119)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
AddressUpgradeable.isContract(address) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#26-35) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#33)
AddressUpgradeable._verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#147-164) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#156-159)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

```
INFO:Detectors:
Different versions of Solidity is used in :
        - Version used: ['0.6.6', '>=0.4.24<0.8.0', '>=0.6.0<0.8.0', '>=0.6.2<0.8.0']
        - 0.6.6 (contracts/EglGenesis.sol#1)
        - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#3)
        - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/math/SafeMathUpgradeable.sol#3)
        - >=0.4.24<0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/Initializable.sol#4)
        - >=0.6.2<0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#3)
        - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#3)
        - >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/PausableUpgradeable.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Pragma version0.6.6 (contracts/EglGenesis.sol#1) allows old versions
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/math/SafeMathUpgradeable.sol#3) is too complex
Pragma version>=0.4.24<0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/Initializable.sol#4) is too complex
Pragma version>=0.6.2<0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/PausableUpgradeable.sol#3) is too complex
solc-0.6.6 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in EglGenesis.withdraw() (contracts/EglGenesis.sol#117-130):
        - (success) = msg.sender.call{value: amountToWithdraw}() (contracts/EglGenesis.sol#127)
Low level call in EglGenesis.endGenesis() (contracts/EglGenesis.sol#146-151):
        - (success) = msg.sender.call{value: cumulativeBalance}() (contracts/EglGenesis.sol#148)
Low level call in AddressUpgradeable.sendValue(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#53-59):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#57)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#114-121):
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#119)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#139-145):
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#143)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter EglGenesis.initialize(address,uint256)._owner (contracts/EglGenesis.sol#103) is not in mixedCase
Parameter EglGenesis.initialize(address,uint256)._threshold (contracts/EglGenesis.sol#103) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init() (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#27-30) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#32-36) is not in mixedCase
Variable OwnableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#74) is not in mixedCase
Function ContextUpgradeable.__Context_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#17-19) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#21-22) is not in mixedCase
Variable ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#31) is not in mixedCase
Function PausableUpgradeable.__Pausable_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/PausableUpgradeable.sol#33-36) is not in mixedCase
Function PausableUpgradeable.__Pausable_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/PausableUpgradeable.sol#38-40) is not in mixedCase
Variable PausableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/PausableUpgradeable.sol#96) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#28)" inContextUpgradeable (node_modules/@openzeppelin/contracts-upgradeable/utils/Con
textUpgradeable.sol#16-32)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
PausableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/PausableUpgradeable.sol#96) is never used in EglGenesis (contracts/EglGenesis.sol#8-166)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variables
INFO:Detectors:
initialize(address,uint256) should be declared external:
        - EglGenesis.initialize(address,uint256) (contracts/EglGenesis.sol#103-112)
withdraw() should be declared external:
        - EglGenesis.withdraw() (contracts/EglGenesis.sol#117-130)
allowWithdraw() should be declared external:
        - EglGenesis.allowWithdraw() (contracts/EglGenesis.sol#135-141)
endGenesis() should be declared external:
        - EglGenesis.endGenesis() (contracts/EglGenesis.sol#146-151)
pauseGenesis() should be declared external:
        - EglGenesis.pauseGenesis() (contracts/EglGenesis.sol#156-158)
unpauseGenesis() should be declared external:
        - EglGenesis.unpauseGenesis() (contracts/EglGenesis.sol#163-165)
renounceOwnership() should be declared external:
        - OwnableUpgradeable.renounceOwnership() (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#60-63)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:contracts/EglGenesis.sol analyzed (7 contracts with 72 detectors), 43 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Some of the issues identified by the tool are listed in this report as:

HAL04 - POSSIBLE MISUSE OF PUBLIC FUNCTIONS

HAL05 - BLOCK TIMESTAMP ALIAS USAGE

3.12 AUTOMATED SECURITY SCAN RESULTS

Description:

Halborn used automated security scanners to assist with detection of well known security issues, and identify low-hanging fruit on the scoped contract targeted 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 MythX to locate any vulnerabilities. Security Detections are only in scope, and the analysis was pointed towards issues with the EglGenesis.sol

Results:

Report for EglGenesis.sol
https://dashboard.mythx.io/#/console/analyses/bfb70a6b-e82d-4204-8bc5-89eb9516aa20

Line	SWC Title	Severity	Short Description
103	(SWC-000) Unknown	Medium	Function could be marked as external.
117	(SWC-000) Unknown	Medium	Function could be marked as external.
135	(SWC-000) Unknown	Medium	Function could be marked as external.
146	(SWC-000) Unknown	Medium	Function could be marked as external.
148	(SWC-107) Reentrancy	Low	A call to a user-supplied address is executed.
150	(SWC-107) Reentrancy	Low	Read of persistent state following external call.
156	(SWC-000) Unknown	Medium	Function could be marked as external.
163	(SWC-000) Unknown	Medium	Function could be marked as external.

All relevant findings were founded in the manual code review.

THANK YOU FOR CHOOSING

// HALBORN