Smart Contract
Security Audit Report

The SlowMist Security Team received the team's application for smart contract security audit of the Ebunker ETH2 Deposit on 2022.12.12. The following are the details and results of this smart contract security audit:

Contract Name :

BatchDeposit

The contract address :

https://etherscan.io/address/0xa1619Fb8CcC03Ee3c8D543fB8be993764030e028

The audit items and results :

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

NO.	Audit Items	Result
1	Replay Vulnerability	Passed
2	Denial of Service Vulnerability	Passed
3	Race Conditions Vulnerability	Passed
4	Authority Control Vulnerability	Passed
5	Integer Overflow and Underflow Vulnerability	Passed
6	Gas Optimization Audit	Passed
7	Design Logic Audit	Passed
8	Uninitialized Storage Pointers Vulnerability	Passed
9	Arithmetic Accuracy Deviation Vulnerability	Passed
10	"False top-up" Vulnerability	Passed
11	Malicious Event Log Audit	Passed
12	Scoping and Declarations Audit	Passed

NO.	Audit Items	Result
13	Safety Design Audit	Passed
14	Non-privacy/Non-dark Coin Audit	Passed

Audit Result : Passed

Audit Number : 0X002212140001

Audit Date : 2022.12.12 - 2022.12.14

Audit Team : SlowMist Security Team

Summary conclusion : This is a BatchDeposit contract. Users can create the batch transfer to deposit the ETH into the DEPOSIT_CONTRACT_ADDRESS. SafeMath security module is used, which is a recommended approach. The contract does not have the Overflow and the Race Conditions issue.

The source code:

BatchDeposit.sol

```
// SPDX-License-Identifier: MIT
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
pragma solidity 0.5.11;
pragma experimental ABIEncoderV2;

// external dependencies
import './openzeppelin/utils/Address.sol';
import './openzeppelin/math/SafeMath.sol';

import './IDeposit.sol';

/// @notice  Batch ETH2 deposits, uses the official Deposit contract from the ETH
///          Foundation for each atomic deposit. This contract acts as a for loop.
///          Each deposit size will be an optimal 32 ETH.
///
/// @dev     The batch size has an upper bound due to the block gas limit. Each
atomic
///          deposit costs ~62,000 gas. The current block gas-limit is ~12,400,000
```

```
gas.
///
/// Author:  Ebunker Inc. (https://ebunker.io/)
contract BatchDeposit {
    using Address for address payable;
    using SafeMath for uint256;

    /*************** STORAGE VARIABLE DECLARATIONS **************/

    uint256 public constant DEPOSIT_AMOUNT = 32 ether;
    // points at the Beacon Deposit Contract
    address public constant DEPOSIT_CONTRACT_ADDRESS =
0x00000000219ab540356cBB839Cbe05303d7705Fa;    //mainnet
    IDeposit private constant DEPOSIT_CONTRACT = IDeposit(DEPOSIT_CONTRACT_ADDRESS);

    /*************** EVENT DECLARATIONS **************/

    /// @notice  Signals a refund of sent-in Ether that was extra and not required.
    ///
    /// @dev     The refund is sent to the msg.sender.
    ///
    /// @param  to - The ETH address receiving the ETH.
    /// @param  amount - The amount of ETH being refunded.
    event LogSendDepositLeftover(address to, uint256 amount);

    ////////////////////// FUNCTION DECLARATIONS BEGIN //////////////////////

    /******************** PUBLIC FUNCTIONS **********************/

    /// @notice  Empty constructor.
    constructor() public {}

    /// @notice  Fallback function.
    ///
    /// @dev     Used to address parties trying to send in Ether with a helpful
    ///          error message.
    function() external payable {
        revert('#BatchDeposit fallback(): Use the `batchDeposit(...)` function to
send Ether to this contract.');
    }

    /// @notice Submit index-matching arrays that form Phase 0 DepositData objects.
    ///         Will create a deposit transaction per index of the arrays submitted.
    ///
```

```
    /// @param pubkeys - An array of BLS12-381 public keys.
    /// @param withdrawal_credentials - An array of commitment to public key for
withdrawals.
    /// @param signatures - An array of BLS12-381 signatures.
    /// @param deposit_data_roots - An array of the SHA-256 hash of the SSZ-encoded
DepositData object.
    function batchDeposit(
        bytes[] calldata pubkeys,
        bytes[] calldata withdrawal_credentials,
        bytes[] calldata signatures,
        bytes32[] calldata deposit_data_roots
    ) external payable {
        require(
            pubkeys.length == withdrawal_credentials.length &&
                pubkeys.length == signatures.length &&
                pubkeys.length == deposit_data_roots.length,
            "#BatchDeposit batchDeposit(): All parameter array's must have the same
length."
        );
        require(pubkeys.length > 0, "#BatchDeposit batchDeposit(): All parameter
array's must have a length greater than zero.");
        require(
            msg.value >= DEPOSIT_AMOUNT.mul(pubkeys.length),
            '#BatchDeposit batchDeposit(): Ether deposited needs to be at least: 32 *
(parameter `pubkeys[]` length).'
        );
        uint256 deposited = 0;

        // Loop through DepositData arrays submitting deposits
        for (uint256 i = 0; i < pubkeys.length; i++) {
            DEPOSIT_CONTRACT.deposit.value(DEPOSIT_AMOUNT)(pubkeys[i],
withdrawal_credentials[i], signatures[i], deposit_data_roots[i]);
            deposited = deposited.add(DEPOSIT_AMOUNT);
        }
        //SlowMist// It is recommended to replace "assert" with "require" to optimize
Gas
        assert(deposited == DEPOSIT_AMOUNT.mul(pubkeys.length));
        uint256 ethToReturn = msg.value.sub(deposited);
        if (ethToReturn > 0) {
            // Emit `LogSendDepositLeftover` log
            emit LogSendDepositLeftover(msg.sender, ethToReturn);

            // This function doesn't guard against re-entrancy, and we're calling an
            // untrusted address, but in this situation there is no state, etc. to
```

```
            // take advantage of, so re-entrancy guard is unneccesary gas cost.
            // This function uses call.value(), and handles return values/failures by
            // reverting the transaction.
            (msg.sender).sendValue(ethToReturn);
        }
    }
}
```

IDeposit.sol

```
// SPDX-License-Identifier: MIT

pragma solidity 0.5.11;

/// @notice  Interface of the official Deposit contract from the ETH
///          Foundation.
interface IDeposit {
    /// @notice Submit a Phase 0 DepositData object.
    ///
    /// @param pubkey - A BLS12-381 public key.
    /// @param withdrawal_credentials - Commitment to a public key for withdrawals.
    /// @param signature - A BLS12-381 signature.
    /// @param deposit_data_root - The SHA-256 hash of the SSZ-encoded DepositData
object.
    ///                            Used as a protection against malformed input.
    function deposit(
        bytes calldata pubkey,
        bytes calldata withdrawal_credentials,
        bytes calldata signature,
        bytes32 deposit_data_root
    ) external payable;
}
```

SafeMath.sol

```
// SPDX-License-Identifier: MIT
//SlowMist// OpenZeppelin's SafeMath security module is used, which is a recommended
approach
pragma solidity 0.5.11;

/// npm package/version - @openzeppelin/contracts-ethereum-package: 2.5.0
```

```
/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, 'SafeMath: addition overflow');

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, 'SafeMath: subtraction overflow');
    }
```

```
    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom
message on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     *
     * _Available since v2.4.0._
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure
returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but
the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, 'SafeMath: multiplication overflow');

        return c;
    }
```

```
    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, 'SafeMath: division by zero');
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with
custom message on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
     * _Available since v2.4.0._
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure
returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned
integer modulo),
     * Reverts when dividing by zero.
     *
```

```
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, 'SafeMath: modulo by zero');
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned
integer modulo),
     * Reverts with custom message when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
     * _Available since v2.4.0._
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure
returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}
```

Address.sol

```
// SPDX-License-Identifier: MIT

pragma solidity 0.5.11;

/// npm package/version - @openzeppelin/contracts-ethereum-package: 2.5.0

/**
 * @dev Collection of functions related to the address type
```

```
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others, `isContract` will return false for the following
     * types of addresses:
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // According to EIP-1052, 0x0 is the value returned for not-yet created
accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is
returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly {
            codehash := extcodehash(account)
        }
        return (codehash != accountHash && codehash != 0x0);
    }

    /**
     * @dev Converts an `address` into `address payable`. Note that this is
     * simply a type cast: the actual underlying value is not changed.
     *
     * _Available since v2.4.0._
     */
    function toPayable(address account) internal pure returns (address payable) {
        return address(uint160(account));
    }
```

```
    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-
now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-
the-checks-effects-interactions-pattern[checks-effects-interactions pattern].
     *
     * _Available since v2.4.0._
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, 'Address: insufficient balance');

        // solhint-disable-next-line avoid-call-value
        (bool success, ) = recipient.call.value(amount)('');
        require(success, 'Address: unable to send value, recipient may have
reverted');
    }
}
```

Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist