Contract Audit
Smart Fast

Smart Contract
Security Audit Report

Number: 0524093328840

Date: 2021-05-24

Welcome to SmartFast!

0x01 Summary Information

The SmartFast (SF, for short) platform received this smart contract security audit application and audited the contract in May 2021.

It is necessary to declare that SF only issues this report in respect of facts that have occurred or existed before the issuance of this report, and undertakes corresponding responsibilities for this. For the facts that occur or exist in the future, SF is unable to judge the security status of its smart contract, and will not be responsible for it. The security audit analysis and other content made in this report are based on the documents and information provided to smart analysis team by the information provider as of the issuance of this report (referred to as "provided information"). SF hypothesis: There is no missing, tampered, deleted or concealed information in the mentioned information. If the information that has been mentioned is missing, tampered with, deleted, concealed or reflected does not match the actual situation, SmartFast shall not be liable for any losses and adverse effects caused thereby.

Table 1 Contract audit information

Project	Description
Contract name	
Contract type	Ethereum contract
Code language	Solidity
Contract files	BitsForAI.sol
Contract address	
Auditors	Smart analysis team
Audit time	2021-05-24 09:33:28
Audit tool	SmartFast (SF)

Table 1 shows the relevant information of this contract audit in detail. The details and results of the contract security audit will be introduced in detail below.

0x02 Contract Audit Results

2.1 Vulnerability Distribution

The severity of vulnerabilities in this security audit is distributed according to the level of danger:

Table 2 Overview of contract audit vulnerability distribution

Vulnerability security distribution				
High	Medium	Low	Info	Opt
0	21	0	71	1

Figure 1 Vulnerability security distribution map

This security audit found 0 High-severity vulnerabilities, 21 Medium-severity vulnerabilities, 0 Low-severity vulnerabilities, 1 Optimization-severity vulnerabilities, and 71 places that need attention.

2.2 Audit Results

There are 119 test items in this security audit, and the test items are as follows (other unknown security vulnerabilities are not included in the scope of responsibility of this audit):

Table 3 Contract audit items

ID	Pattern	Description	Severity	Confidence	Status/Num
1	abiencoderv2-array	ABI encoding error	High	exactly	Pass
2	array-by-reference	Storage parameter usage	High	exactly	Pass
3	multiple-constructors	Multiple constructors in a contract	High	exactly	Pass

4	names-reused	Check contract name reuse	High	exactly	Pass
5	public-mappings-nested	Struct sets the nested structure of struct	High	exactly	Pass
6	rtlo	Overwritten from right to left	High	exactly	Pass
7	shadowing-state	Check state variable hiding	High	exactly	Pass
8	suicidal	Check if anyone can break the contract	High	exactly	Pass
9	uninitialized-state	Check for uninitialized state variables	High	exactly	Pass
10	uninitialized-storage	Check for uninitialized storage variables	High	exactly	Pass
11	unprotected-upgrade	Break the logic of the contract	High	exactly	Pass
12	visibility	Check visibility level error	High\Info	exactly	Pass
13	redundant-fallback	Check for redundant fallback functions	High\Opt	exactly	Pass
14	arbitrary-send	Check if Ether can be sent to any address	High	probably	Pass
15	continue-in-loop	Continue causes an infinite loop	High	probably	Pass
16	controlled-array-length	Length is allocated directly	High	probably	Pass
17	controlled-delegatecall	The delegate address out of control	High	probably	Pass
18	incorrect-constructor	Check the constructor name error	High	probably	Pass
19	parity-multisig-bug	Check for multi-signature vulnerabilities	High	probably	Pass
20	reentrancy-eth	Re-entry vulnerabilities (Ethereum theft)	High	probably	Pass
21	storage-array	Signed integer array problem	High	probably	Pass
22	weak-prng	Modulo problem of parameters	High	probably	Pass
23	assert-violation	Check for incorrect use of assertions	Medium	exactly	Pass
24	constructor-return	The use of return in the constructor	Medium	exactly	Pass
25	enum-conversion	Scope of enumerated types	Medium	exactly	Pass
26	erc1155-interface	Check the wrong ERC1155 interface	Medium	exactly	Pass
27	erc1410-interface	Check the wrong ERC1410 interface	Medium	exactly	Pass
28	erc20-interface	Check for wrong ERC20 interface	Medium	exactly	Pass
29	erc223-interface	Check the wrong ERC223 interface	Medium	exactly	Pass
30	erc621-interface	Check the wrong ERC621 interface	Medium	exactly	Pass
31	erc721-interface	Check the wrong ERC721 interface	Medium	exactly	Pass
32	erc777-interface	Check the wrong ERC777 interface	Medium	exactly	Pass
33	erc875-interface	Check the wrong ERC875 interface	Medium	exactly	Pass
34	incorrect-equality	Check the strict equality of danger	Medium	exactly	Pass
35	incorrect-signature	Check for incorrect function signatures	Medium	exactly	Pass

36	locked-ether	Whether the contract ether is locked	Medium	exactly	Pass
37	mapping-deletion	Delete the mapped structure problem	Medium	exactly	Pass
38	shadowing-abstract	State variables from the abstract contract	Medium	exactly	Pass
39	tautology	Check for tautologies or contradictions	Medium	exactly	Pass
40	default-return-value	Function returns only default value	Medium	probably	Pass
41	boolean-cst	Check for misuse of Boolean constants	Medium	probably	Pass
42	constant-function-state	Check Constant function to change state	Medium	probably	Medium:8
43	divide-before-multiply	Imprecise order of arithmetic operations	Medium	probably	Pass
44	erc20-approve	ERC-20 advance attack (TOD)	Medium	probably	Pass
45	function-problem	Abnormal termination of contract function	Medium	probably	Pass
46	mul-var-len-arguments	Hash collisions with multiple variable	Medium	probably	Pass
47	reentrancy-no-eth	Re-entry vulnerabilities (no ether theft)	Medium	probably	Pass
48	reused-constructor	Constructor conflict problem in the contract	Medium	probably	Pass
49	tx-origin	Check the dangerous use of tx.origin	Medium	probably	Pass
50	typographical-error	Check for writing errors (=+)	Medium	probably	Pass
51	unchecked-lowlevel	Check for uncensored low-level calls	Medium	probably	Pass
52	unchecked-send	Check unreviewed send	Medium	probably	Pass
53	uninitialized-local	Check for uninitialized local variables	Medium	probably	Pass
54	unused-return	Check if there is an unused return value	Medium	probably	Pass
55	writeto-arbitrarystorage	It can be written to any storage location	Medium	probably	Pass
56	integer-overflow	Check for integer overflow	Medium	probably	Pass
57	costly-loop	Check for too expensive loops	Medium	possibly	Medium:13
58	shift-parameter-mixup	Check reversible shift operation	Medium	possibly	Pass
59	shadowing-builtin	Check the hiding of built-in symbols	Low	exactly	Pass
60	shadowing-function	Check function hiding	Low	exactly	Pass
61	shadowing-local	Check local variable hiding	Low	exactly	Pass
62	transfer-to-zeroaddress	The withdrawal address is 0x0	Low	exactly	Pass
63	uninitialized-fptr-cst	Uninitialized pointers in constructor	Low	exactly	Pass
64	variable-scope	Check the declaration of variables	Low	exactly	Pass
65	void-cst	Calls to unimplemented constructors	Low	exactly	Pass
66	incorrect-modifier	Restore abnormal Modifier	Low	exactly	Pass
67	assemblycall-rewrite	Assemblycall covers the input	Low	probably	Pass

68	block-other-parameters	Hazardous use variables (block.number etc.)	Low	probably	Pass
69	calls-loop	Check the external call in the loop	Low	probably	Pass
70	events-access	The loss of key access control parameters	Low	probably	Pass
71	events-maths	The loss of key arithmetic parameters	Low	probably	Pass
72	extcodesize-invoke	Check Extcodesize call	Low	probably	Pass
73	fllback-outofgas	The fallback function is too complicated	Low	probably	Pass
74	incorrect-blockhash	Incorrect use of Blockhash function	Low	probably	Pass
75	incorrect-inheritance-order	The inherited variables conflict	Low	probably	Pass
76	missing-zero-check	Check the use of zero addresses	Low	probably	Pass
77	reentrancy-benign	Reentrant vulnerabilities (continuous calls)	Low	probably	Pass
78	reentrancy-events	Reentrant vulnerabilities (events)	Low	probably	Pass
79	timestamp	The dangerous use of block.timestamp	Low	probably	Pass
80	signature-malleability	The signature contains an existing signature	Low	possibly	Pass
81	assembly	Unsafe use of assembly	Info	exactly	Info:15
82	assert-state-change	Incorrect use of assert()	Info	exactly	Pass
83	delete-dynamic-arrays	The deletion of the dynamic storage array	Info	exactly	Pass
84	deprecated-standards	Solidity deprecated instructions	Info	exactly	Pass
85	erc20-indexed	ERC20 event parameter is missing indexed	Info	exactly	Pass
86	erc20-throw	ERC20 throws an exception	Info	exactly	Pass
87	length-manipulation	Unsafe operation to check array length	Info	exactly	Info:2
88	low-level-calls	Check low-level calls	Info	exactly	Pass
89	msgvalue-equals-zero	The judgment of msg.value and zero	Info	exactly	Pass
90	naming-convention	The naming follows the Solidity format	Info	exactly	Pass
91	pragma	Undeclared multiple compiled versions	Info	exactly	Pass
92	solc-version	Check for incorrect Solidity version	Info	exactly	Pass
93	unimplemented-functions	Functions overloaded in the contract	Info	exactly	Pass
94	upgrade-050	Code for Solidity 0.5.x upgrade	Info	exactly	Pass
95	function-init-state	State variables initialized by functions	Info	exactly	Pass
96	complex-function	Check complex functions	Info	probably	Pass
97	hardcoded	Check the legitimacy of the address	Info	probably	Info:16
98	overpowered-role	The permissions are too concentrated	Info	probably	Pass
99	reentrancy-limited-events	Reentrancy vulnerabilities (limited events)	Info	probably	Pass

100	reentrancy-limited-gas	Reentry (send and transfer, with eth)	Info	probably	Pass
101	reentrancy-limited-gas-no-eth	Reentrance (send and transfer, no eth)	Info	probably	Pass
102	similar-names	Detect similar variables	Info	probably	Pass
103	too-many-digits	Too many number symbols	Info	probably	Pass
104	private-not-hidedata	Check the use of private visibility	Info	possibly	Info:35
105	safemath	Check the use of SafeMath	Info	possibly	Info:3
106	array-instead-bytes	The byte array can be replaced with bytes	Opt	exactly	Pass
107	boolean-equal	Check comparison with boolean constant	Opt	exactly	Pass
108	code-no-effects	Check for invalid codes	Opt	exactly	Pass
109	constable-states	State variables can be declared as constants	Opt	exactly	Pass
110	event-before-revert	Check if event is called before revert	Opt	exactly	Pass
111	external-function	Public functions can be declared as external	Opt	exactly	Pass
112	extra-gas-inloops	Check for additional gas consumption	Opt	exactly	Pass
113	missing-inheritance	Detect lost inheritance	Opt	exactly	Pass
114	redundant-statements	Detect the use of invalid sentences	Opt	exactly	Pass
115	return-struct	Multiple return values (struct)	Opt	exactly	Pass
116	revert-require	Check Revert in if operation	Opt	exactly	Opt:1
117	send-transfer	Check Transfe to replace Send	Opt	exactly	Pass
118	unused-state	Check unused state variables	Opt	exactly	Pass
119	costly-operations-loop	Expensive operations in the loop	Opt	probably	Pass

0x03 Contract Code

3.1 Code and Vulnerability Lables

In the corresponding position of each contract code, security vulnerabilities and coding specification issues have been marked in the form of comments. The comment labels start with //StFt. For details, please refer to the following contract code content.

```
// File: node_modules\@openzeppelin\contracts\math\SafeMath.sol

pragma solidity ^0.5.0;

/**
 * @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;
  }
}

// File: node_modules\@openzeppelin\contracts\math\Math.sol

pragma solidity ^0.5.0;

/**
 * @dev Standard math utilities missing in the Solidity language.
 */
library Math {
  /**
   * @dev Returns the largest of two numbers.
   */
  function max(uint256 a, uint256 b) internal pure returns (uint256) {
    return a >= b ? a : b;
  }

  /**
   * @dev Returns the smallest of two numbers.
   */
```

```
    function min(uint256 a, uint256 b) internal pure returns (uint256) {
        return a < b ? a : b;
    }

    /**
     * @dev Returns the average of two numbers. The result is rounded towards
     * zero.
     */
    function average(uint256 a, uint256 b) internal pure returns (uint256) {
        // (a + b) / 2 can overflow, so we distribute
        return (a / 2) + (b / 2) + ((a % 2 + b % 2) / 2);
    }
}

// File: node_modules\@openzeppelin\contracts\GSN\Context.sol

pragma solidity ^0.5.0;

/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
contract Context {
    // Empty internal constructor, to prevent people from mistakenly deploying
    // an instance of this contract, which should be used via inheritance.
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see
https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

// File: node_modules\@openzeppelin\contracts\ownership\Ownable.sol

pragma solidity ^0.5.0;

/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
```

```
* the owner.
*/
contract Ownable is Context {
    address private _owner; //StFt //private-not-hidedata

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () internal {
        _owner = _msgSender();
        emit OwnershipTransferred(address(0), _owner);
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view returns (address) {
        return _owner;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner(), "Ownable: caller is not the owner");
        _;
    }

    /**
     * @dev Returns true if the caller is the current owner.
     */
    function isOwner() public view returns (bool) {
        return _msgSender() == _owner;
    }

    /**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions anymore. Can only be called by the current owner.
     *
     * NOTE: Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }
```

```
    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

// File: node_modules\@openzeppelin\contracts\introspection\IERC165.sol

pragma solidity ^0.5.0;

/**
 * @dev Interface of the ERC165 standard, as defined in the
 * https://eips.ethereum.org/EIPS/eip-165[EIP].
 *
 * Implementers can declare support of contract interfaces, which can then be
 * queried by others ({ERC165Checker}).
 *
 * For an implementation, see {ERC165}.
 */
interface IERC165 {
    /**
     * @dev Returns true if this contract implements the interface defined by
     * `interfaceId`. See the corresponding
     * https://eips.ethereum.org/EIPS/eip-165#how-interfaces-are-identified[EIP section]
     * to learn more about how these ids are created.
     *
     * This function call must use less than 30 000 gas.
     */
    function supportsInterface(bytes4 interfaceId) external view returns (bool);
}

// File: node_modules\@openzeppelin\contracts\token\ERC721\IERC721.sol

pragma solidity ^0.5.0;


/**
 * @dev Required interface of an ERC721 compliant contract.
 */
contract IERC721 is IERC165 {
    event Transfer(address indexed from, address indexed to, uint256 indexed tokenId);
    event Approval(address indexed owner, address indexed approved, uint256 indexed tokenId);
    event ApprovalForAll(address indexed owner, address indexed operator, bool approved);

    /**
     * @dev Returns the number of NFTs in `owner`'s account.
     */
    function balanceOf(address owner) public view returns (uint256 balance);

    /**
     * @dev Returns the owner of the NFT specified by `tokenId`.
```

```
    */
    function ownerOf(uint256 tokenId) public view returns (address owner);

    /**
     * @dev Transfers a specific NFT (`tokenId`) from one account (`from`) to
     * another (`to`).
     *
     *
     *
     * Requirements:
     * - `from`, `to` cannot be zero.
     * - `tokenId` must be owned by `from`.
     * - If the caller is not `from`, it must be have been allowed to move this
     * NFT by either {approve} or {setApprovalForAll}.
     */
    function safeTransferFrom(address from, address to, uint256 tokenId) public;
    /**
     * @dev Transfers a specific NFT (`tokenId`) from one account (`from`) to
     * another (`to`).
     *
     * Requirements:
     * - If the caller is not `from`, it must be approved to move this NFT by
     * either {approve} or {setApprovalForAll}.
     */
    function transferFrom(address from, address to, uint256 tokenId) public;
    function approve(address to, uint256 tokenId) public;
    function getApproved(uint256 tokenId) public view returns (address operator);

    function setApprovalForAll(address operator, bool _approved) public;
    function isApprovedForAll(address owner, address operator) public view returns (bool);


    function safeTransferFrom(address from, address to, uint256 tokenId, bytes memory data) public;
}

// File: node_modules\@openzeppelin\contracts\token\ERC721\IERC721Receiver.sol

pragma solidity ^0.5.0;

/**
 * @title ERC721 token receiver interface
 * @dev Interface for any contract that wants to support safeTransfers
 * from ERC721 asset contracts.
 */
contract IERC721Receiver {
    /**
     * @notice Handle the receipt of an NFT
     * @dev The ERC721 smart contract calls this function on the recipient
     * after a {IERC721-safeTransferFrom}. This function MUST return the function selector,
     * otherwise the caller will revert the transaction. The selector to be
     * returned can be obtained as `this.onERC721Received.selector`. This
     * function MAY throw to revert and reject the transfer.
     * Note: the ERC721 contract address is always the message sender.
     * @param operator The address which called `safeTransferFrom` function
     * @param from The address which previously owned the token
     * @param tokenId The NFT identifier which is being transferred
```

```
    * @param data Additional data with no specified format
    * @return bytes4 `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`
    */
    function onERC721Received(address operator, address from, uint256 tokenId, bytes memory data)
    public returns (bytes4);
}

// File: node_modules\@openzeppelin\contracts\utils\Address.sol

pragma solidity ^0.5.5;

/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
    * @dev Returns true if `account` is a contract.
    *
    * This test is non-exhaustive, and there may be false-negatives: during the
    * execution of a contract's constructor, its address will be reported as
    * not containing 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.
    */
    function isContract(address account) internal view returns (bool) { //StFt //constant-function-state
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        // 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 = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d8
5a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) } //StFt //assembly
        return (codehash != 0x0 && codehash != accountHash);
    }

    /**
    * @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-int
eractions-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");
    }
}

// File: node_modules\@openzeppelin\contracts\drafts\Counters.sol

pragma solidity ^0.5.0;


/**
 * @title Counters
 * @author Matt Condon (@shrugs)
 * @dev Provides counters that can only be incremented or decremented by one. This can be used e.g.
        to track the number
 * of elements in a mapping, issuing ERC721 ids, or counting request ids.
 *
 * Include with `using Counters for Counters.Counter;`
 * Since it is not possible to overflow a 256 bit integer with increments of one, `increment` can skip the
        {SafeMath}
 * overflow check, thereby saving gas. This does assume however correct usage, in that the underlying
`_value` is never
 * directly accessed.
 */
library Counters {
    using SafeMath for uint256; //StFt //safemath

    struct Counter {
        // This variable should never be directly accessed by users of the library: interactions must be
restricted to
        // the library's function. As of Solidity v0.5.2, this cannot be enforced, though there is a proposal
to add
        // this feature: see https://github.com/ethereum/solidity/issues/4637
        uint256 _value; // default: 0
    }

    function current(Counter storage counter) internal view returns (uint256) {
```

```
        return counter._value;
    }

    function increment(Counter storage counter) internal {
        counter._value += 1;
    }

    function decrement(Counter storage counter) internal {
        counter._value = counter._value.sub(1);
    }
}

// File: node_modules\@openzeppelin\contracts\introspection\ERC165.sol

pragma solidity ^0.5.0;


/**
 * @dev Implementation of the {IERC165} interface.
 *
 * Contracts may inherit from this and call {_registerInterface} to declare
 * their support of an interface.
 */
contract ERC165 is IERC165 {
    /*
     * bytes4(keccak256('supportsInterface(bytes4)')) == 0x01ffc9a7
     */
    bytes4 private constant _INTERFACE_ID_ERC165 = 0x01ffc9a7; //StFt //private-not-hidedata

    /**
     * @dev Mapping of interface ids to whether or not it's supported.
     */
    mapping(bytes4 => bool) private _supportedInterfaces; //StFt //private-not-hidedata

    constructor () internal {
        // Derived contracts need only register support for their own interfaces,
        // we register support for ERC165 itself here
        _registerInterface(_INTERFACE_ID_ERC165);
    }

    /**
     * @dev See {IERC165-supportsInterface}.
     *
     * Time complexity O(1), guaranteed to always use less than 30 000 gas.
     */
    function supportsInterface(bytes4 interfaceId) external view returns (bool) {
        return _supportedInterfaces[interfaceId];
    }

    /**
     * @dev Registers the contract as an implementer of the interface defined by
     * `interfaceId`. Support of the actual ERC165 interface is automatic and
     * registering its interface id is not required.
     *
     * See {IERC165-supportsInterface}.
     *
```

```
    * Requirements:
    *
    * - `interfaceId` cannot be the ERC165 invalid interface (`0xffffffff`).
    */
   function _registerInterface(bytes4 interfaceId) internal {
      require(interfaceId != 0xffffffff, "ERC165: invalid interface id");
      _supportedInterfaces[interfaceId] = true;
   }
}

// File: node_modules\@openzeppelin\contracts\token\ERC721\ERC721.sol

pragma solidity ^0.5.0;









/**
 * @title ERC721 Non-Fungible Token Standard basic implementation
 * @dev see https://eips.ethereum.org/EIPS/eip-721
 */
contract ERC721 is Context, ERC165, IERC721 {
   using SafeMath for uint256; //StFt //safemath
   using Address for address;
   using Counters for Counters.Counter;

   // Equals to `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`
   // which can be also obtained as `IERC721Receiver(0).onERC721Received.selector`
   bytes4 private constant _ERC721_RECEIVED = 0x150b7a02; //StFt //private-not-hidedata

   // Mapping from token ID to owner
   mapping (uint256 => address) private _tokenOwner; //StFt //private-not-hidedata

   // Mapping from token ID to approved address
   mapping (uint256 => address) private _tokenApprovals; //StFt //private-not-hidedata

   // Mapping from owner to number of owned token
   mapping (address => Counters.Counter) private _ownedTokensCount; //StFt //private-not-hidedata

   // Mapping from owner to operator approvals
   mapping (address => mapping (address => bool)) private _operatorApprovals; //StFt
//private-not-hidedata

   /*
    *     bytes4(keccak256('balanceOf(address)')) == 0x70a08231
    *     bytes4(keccak256('ownerOf(uint256)')) == 0x6352211e
    *     bytes4(keccak256('approve(address,uint256)')) == 0x095ea7b3
    *     bytes4(keccak256('getApproved(uint256)')) == 0x081812fc
    *     bytes4(keccak256('setApprovalForAll(address,bool)')) == 0xa22cb465
    *     bytes4(keccak256('isApprovedForAll(address,address)')) == 0xe985e9c5
    *     bytes4(keccak256('transferFrom(address,address,uint256)')) == 0x23b872dd
    *     bytes4(keccak256('safeTransferFrom(address,address,uint256)')) == 0x42842e0e
```

```
 *     bytes4(keccak256('safeTransferFrom(address,address,uint256,bytes)')) == 0xb88d4fde
 *
 *     => 0x70a08231 ^ 0x6352211e ^ 0x095ea7b3 ^ 0x081812fc ^
 *       0xa22cb465 ^ 0xe985e9c ^ 0x23b872dd ^ 0x42842e0e ^ 0xb88d4fde == 0x80ac58cd
 */
bytes4 private constant _INTERFACE_ID_ERC721 = 0x80ac58cd; //StFt //private-not-hidedata

constructor () public {
    // register the supported interfaces to conform to ERC721 via ERC165
    _registerInterface(_INTERFACE_ID_ERC721);
}

/**
 * @dev Gets the balance of the specified address.
 * @param owner address to query the balance of
 * @return uint256 representing the amount owned by the passed address
 */
function balanceOf(address owner) public view returns (uint256) {
    require(owner != address(0), "ERC721: balance query for the zero address");

    return _ownedTokensCount[owner].current();
}

/**
 * @dev Gets the owner of the specified token ID.
 * @param tokenId uint256 ID of the token to query the owner of
 * @return address currently marked as the owner of the given token ID
 */
function ownerOf(uint256 tokenId) public view returns (address) {
    address owner = _tokenOwner[tokenId];
    require(owner != address(0), "ERC721: owner query for nonexistent token");

    return owner;
}

/**
 * @dev Approves another address to transfer the given token ID
 * The zero address indicates there is no approved address.
 * There can only be one approved address per token at a given time.
 * Can only be called by the token owner or an approved operator.
 * @param to address to be approved for the given token ID
 * @param tokenId uint256 ID of the token to be approved
 */
function approve(address to, uint256 tokenId) public {
    address owner = ownerOf(tokenId);
    require(to != owner, "ERC721: approval to current owner");

    require(_msgSender() == owner || isApprovedForAll(owner, _msgSender()),
        "ERC721: approve caller is not owner nor approved for all"
    );

    _tokenApprovals[tokenId] = to;
    emit Approval(owner, to, tokenId);
}

/**
```

```
 * @dev Gets the approved address for a token ID, or zero if no address set
 * Reverts if the token ID does not exist.
 * @param tokenId uint256 ID of the token to query the approval of
 * @return address currently approved for the given token ID
 */
function getApproved(uint256 tokenId) public view returns (address) {
    require(_exists(tokenId), "ERC721: approved query for nonexistent token");

    return _tokenApprovals[tokenId];
}

/**
 * @dev Sets or unsets the approval of a given operator
 * An operator is allowed to transfer all tokens of the sender on their behalf.
 * @param to operator address to set the approval
 * @param approved representing the status of the approval to be set
 */
function setApprovalForAll(address to, bool approved) public {
    require(to != _msgSender(), "ERC721: approve to caller");

    _operatorApprovals[_msgSender()][to] = approved;
    emit ApprovalForAll(_msgSender(), to, approved);
}

/**
 * @dev Tells whether an operator is approved by a given owner.
 * @param owner owner address which you want to query the approval of
 * @param operator operator address which you want to query the approval of
 * @return bool whether the given operator is approved by the given owner
 */
function isApprovedForAll(address owner, address operator) public view returns (bool) {
    return _operatorApprovals[owner][operator];
}

/**
 * @dev Transfers the ownership of a given token ID to another address.
 * Usage of this method is discouraged, use {safeTransferFrom} whenever possible.
 * Requires the msg.sender to be the owner, approved, or operator.
 * @param from current owner of the token
 * @param to address to receive the ownership of the given token ID
 * @param tokenId uint256 ID of the token to be transferred
 */
function transferFrom(address from, address to, uint256 tokenId) public {
    //solhint-disable-next-line max-line-length
    require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: transfer caller is not owner nor
approved");

    _transferFrom(from, to, tokenId);
}

/**
 * @dev Safely transfers the ownership of a given token ID to another address
 * If the target address is a contract, it must implement {IERC721Receiver-onERC721Received},
 * which is called upon a safe transfer, and return the magic value
 * `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`; otherwise,
 * the transfer is reverted.
```

```
    * Requires the msg.sender to be the owner, approved, or operator
    * @param from current owner of the token
    * @param to address to receive the ownership of the given token ID
    * @param tokenId uint256 ID of the token to be transferred
    */
   function safeTransferFrom(address from, address to, uint256 tokenId) public {
       safeTransferFrom(from, to, tokenId, "");
   }

   /**
    * @dev Safely transfers the ownership of a given token ID to another address
    * If the target address is a contract, it must implement {IERC721Receiver-onERC721Received},
    * which is called upon a safe transfer, and return the magic value
    * `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`; otherwise,
    * the transfer is reverted.
    * Requires the _msgSender() to be the owner, approved, or operator
    * @param from current owner of the token
    * @param to address to receive the ownership of the given token ID
    * @param tokenId uint256 ID of the token to be transferred
    * @param _data bytes data to send along with a safe transfer check
    */
   function safeTransferFrom(address from, address to, uint256 tokenId, bytes memory _data) public {
       require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: transfer caller is not owner nor
approved");
       _safeTransferFrom(from, to, tokenId, _data);
   }

   /**
    * @dev Safely transfers the ownership of a given token ID to another address
    * If the target address is a contract, it must implement `onERC721Received`,
    * which is called upon a safe transfer, and return the magic value
    * `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`; otherwise,
    * the transfer is reverted.
    * Requires the msg.sender to be the owner, approved, or operator
    * @param from current owner of the token
    * @param to address to receive the ownership of the given token ID
    * @param tokenId uint256 ID of the token to be transferred
    * @param _data bytes data to send along with a safe transfer check
    */
   function _safeTransferFrom(address from, address to, uint256 tokenId, bytes memory _data) internal
           {
       _transferFrom(from, to, tokenId);
       require(_checkOnERC721Received(from, to, tokenId, _data), "ERC721: transfer to non
ERC721Receiver implementer");
   }

   /**
    * @dev Returns whether the specified token exists.
    * @param tokenId uint256 ID of the token to query the existence of
    * @return bool whether the token exists
    */
   function _exists(uint256 tokenId) internal view returns (bool) {
       address owner = _tokenOwner[tokenId];
       return owner != address(0);
   }
```

```
    /**
     * @dev Returns whether the given spender can transfer a given token ID.
     * @param spender address of the spender to query
     * @param tokenId uint256 ID of the token to be transferred
     * @return bool whether the msg.sender is approved for the given token ID,
     * is an operator of the owner, or is the owner of the token
     */
    function _isApprovedOrOwner(address spender, uint256 tokenId) internal view returns (bool) {
        require(_exists(tokenId), "ERC721: operator query for nonexistent token");
        address owner = ownerOf(tokenId);
        return (spender == owner || getApproved(tokenId) == spender || isApprovedForAll(owner,
spender));
    }

    /**
     * @dev Internal function to safely mint a new token.
     * Reverts if the given token ID already exists.
     * If the target address is a contract, it must implement `onERC721Received`,
     * which is called upon a safe transfer, and return the magic value
     * `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`; otherwise,
     * the transfer is reverted.
     * @param to The address that will own the minted token
     * @param tokenId uint256 ID of the token to be minted
     */
    function _safeMint(address to, uint256 tokenId) internal {
        _safeMint(to, tokenId, "");
    }

    /**
     * @dev Internal function to safely mint a new token.
     * Reverts if the given token ID already exists.
     * If the target address is a contract, it must implement `onERC721Received`,
     * which is called upon a safe transfer, and return the magic value
     * `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`; otherwise,
     * the transfer is reverted.
     * @param to The address that will own the minted token
     * @param tokenId uint256 ID of the token to be minted
     * @param _data bytes data to send along with a safe transfer check
     */
    function _safeMint(address to, uint256 tokenId, bytes memory _data) internal {
        _mint(to, tokenId);
        require(_checkOnERC721Received(address(0), to, tokenId, _data), "ERC721: transfer to non
ERC721Receiver implementer");
    }

    /**
     * @dev Internal function to mint a new token.
     * Reverts if the given token ID already exists.
     * @param to The address that will own the minted token
     * @param tokenId uint256 ID of the token to be minted
     */
    function _mint(address to, uint256 tokenId) internal {
        require(to != address(0), "ERC721: mint to the zero address");
        require(!_exists(tokenId), "ERC721: token already minted");

        _tokenOwner[tokenId] = to;
```

```
    _ownedTokensCount[to].increment();

    emit Transfer(address(0), to, tokenId);
}

/**
 * @dev Internal function to burn a specific token.
 * Reverts if the token does not exist.
 * Deprecated, use {_burn} instead.
 * @param owner owner of the token to burn
 * @param tokenId uint256 ID of the token being burned
 */
function _burn(address owner, uint256 tokenId) internal {
    require(ownerOf(tokenId) == owner, "ERC721: burn of token that is not own");

    _clearApproval(tokenId);

    _ownedTokensCount[owner].decrement();
    _tokenOwner[tokenId] = address(0);

    emit Transfer(owner, address(0), tokenId);
}

/**
 * @dev Internal function to burn a specific token.
 * Reverts if the token does not exist.
 * @param tokenId uint256 ID of the token being burned
 */
function _burn(uint256 tokenId) internal {
    _burn(ownerOf(tokenId), tokenId);
}

/**
 * @dev Internal function to transfer ownership of a given token ID to another address.
 * As opposed to {transferFrom}, this imposes no restrictions on msg.sender.
 * @param from current owner of the token
 * @param to address to receive the ownership of the given token ID
 * @param tokenId uint256 ID of the token to be transferred
 */
function _transferFrom(address from, address to, uint256 tokenId) internal {
    require(ownerOf(tokenId) == from, "ERC721: transfer of token that is not own");
    require(to != address(0), "ERC721: transfer to the zero address");

    _clearApproval(tokenId);

    _ownedTokensCount[from].decrement();
    _ownedTokensCount[to].increment();

    _tokenOwner[tokenId] = to;

    emit Transfer(from, to, tokenId);
}

/**
 * @dev Internal function to invoke {IERC721Receiver-onERC721Received} on a target address.
 * The call is not executed if the target address is not a contract.
```

```
    *
    * This function is deprecated.
    * @param from address representing the previous owner of the given token ID
    * @param to target address that will receive the tokens
    * @param tokenId uint256 ID of the token to be transferred
    * @param _data bytes optional data to send along with the call
    * @return bool whether the call correctly returned the expected magic value
    */
   function _checkOnERC721Received(address from, address to, uint256 tokenId, bytes memory _data)
       internal returns (bool)
   {
       if (!to.isContract()) {
           return true;
       }

       bytes4 retval = IERC721Receiver(to).onERC721Received(_msgSender(), from, tokenId, _data);
       return (retval == _ERC721_RECEIVED);
   }

   /**
    * @dev Private function to clear current approval of a given token ID.
    * @param tokenId uint256 ID of the token to be transferred
    */
   function _clearApproval(uint256 tokenId) private {
       if (_tokenApprovals[tokenId] != address(0)) {
           _tokenApprovals[tokenId] = address(0);
       }
   }
}

// File: node_modules\@openzeppelin\contracts\token\ERC721\IERC721Enumerable.sol

pragma solidity ^0.5.0;


/**
 * @title ERC-721 Non-Fungible Token Standard, optional enumeration extension
 * @dev See https://eips.ethereum.org/EIPS/eip-721
 */
contract IERC721Enumerable is IERC721 {
    function totalSupply() public view returns (uint256);
    function tokenOfOwnerByIndex(address owner, uint256 index) public view returns (uint256 tokenId);

    function tokenByIndex(uint256 index) public view returns (uint256);
}

// File: node_modules\@openzeppelin\contracts\token\ERC721\ERC721Enumerable.sol

pragma solidity ^0.5.0;




/**
 * @title ERC-721 Non-Fungible Token with optional enumeration extension logic
```

```
 * @dev See https://eips.ethereum.org/EIPS/eip-721
 */
contract ERC721Enumerable is Context, ERC165, ERC721, IERC721Enumerable {
    // Mapping from owner to list of owned token IDs
    mapping(address => uint256[]) private _ownedTokens; //StFt //private-not-hidedata

    // Mapping from token ID to index of the owner tokens list
    mapping(uint256 => uint256) private _ownedTokensIndex; //StFt //private-not-hidedata

    // Array with all token ids, used for enumeration
    uint256[] private _allTokens; //StFt //private-not-hidedata

    // Mapping from token id to position in the allTokens array
    mapping(uint256 => uint256) private _allTokensIndex; //StFt //private-not-hidedata

    /*
     *     bytes4(keccak256('totalSupply()')) == 0x18160ddd
     *     bytes4(keccak256('tokenOfOwnerByIndex(address,uint256)')) == 0x2f745c59
     *     bytes4(keccak256('tokenByIndex(uint256)')) == 0x4f6ccce7
     *
     *     => 0x18160ddd ^ 0x2f745c59 ^ 0x4f6ccce7 == 0x780e9d63
     */
    bytes4 private constant _INTERFACE_ID_ERC721_ENUMERABLE = 0x780e9d63; //StFt
//private-not-hidedata

    /**
     * @dev Constructor function.
     */
    constructor () public {
        // register the supported interface to conform to ERC721Enumerable via ERC165
        _registerInterface(_INTERFACE_ID_ERC721_ENUMERABLE);
    }

    /**
     * @dev Gets the token ID at a given index of the tokens list of the requested owner.
     * @param owner address owning the tokens list to be accessed
     * @param index uint256 representing the index to be accessed of the requested tokens list
     * @return uint256 token ID at the given index of the tokens list owned by the requested address
     */
    function tokenOfOwnerByIndex(address owner, uint256 index) public view returns (uint256) {
        require(index < balanceOf(owner), "ERC721Enumerable: owner index out of bounds");
        return _ownedTokens[owner][index];
    }

    /**
     * @dev Gets the total amount of tokens stored by the contract.
     * @return uint256 representing the total amount of tokens
     */
    function totalSupply() public view returns (uint256) {
        return _allTokens.length;
    }

    /**
     * @dev Gets the token ID at a given index of all the tokens in this contract
     * Reverts if the index is greater or equal to the total number of tokens.
     * @param index uint256 representing the index to be accessed of the tokens list
```

```
 * @return uint256 token ID at the given index of the tokens list
 */
function tokenByIndex(uint256 index) public view returns (uint256) {
    require(index < totalSupply(), "ERC721Enumerable: global index out of bounds");
    return _allTokens[index];
}

/**
 * @dev Internal function to transfer ownership of a given token ID to another address.
 * As opposed to transferFrom, this imposes no restrictions on msg.sender.
 * @param from current owner of the token
 * @param to address to receive the ownership of the given token ID
 * @param tokenId uint256 ID of the token to be transferred
 */
function _transferFrom(address from, address to, uint256 tokenId) internal {
    super._transferFrom(from, to, tokenId);

    _removeTokenFromOwnerEnumeration(from, tokenId);

    _addTokenToOwnerEnumeration(to, tokenId);
}

/**
 * @dev Internal function to mint a new token.
 * Reverts if the given token ID already exists.
 * @param to address the beneficiary that will own the minted token
 * @param tokenId uint256 ID of the token to be minted
 */
function _mint(address to, uint256 tokenId) internal {
    super._mint(to, tokenId);

    _addTokenToOwnerEnumeration(to, tokenId);

    _addTokenToAllTokensEnumeration(tokenId);
}

/**
 * @dev Internal function to burn a specific token.
 * Reverts if the token does not exist.
 * Deprecated, use {ERC721-_burn} instead.
 * @param owner owner of the token to burn
 * @param tokenId uint256 ID of the token being burned
 */
function _burn(address owner, uint256 tokenId) internal {
    super._burn(owner, tokenId);

    _removeTokenFromOwnerEnumeration(owner, tokenId);
    // Since tokenId will be deleted, we can clear its slot in _ownedTokensIndex to trigger a gas
refund
    _ownedTokensIndex[tokenId] = 0;

    _removeTokenFromAllTokensEnumeration(tokenId);
}

/**
 * @dev Gets the list of token IDs of the requested owner.
```

```
    * @param owner address owning the tokens
    * @return uint256[] List of token IDs owned by the requested address
    */
   function _tokensOfOwner(address owner) internal view returns (uint256[] storage) {
       return _ownedTokens[owner];
   }

   /**
    * @dev Private function to add a token to this extension's ownership-tracking data structures.
    * @param to address representing the new owner of the given token ID
    * @param tokenId uint256 ID of the token to be added to the tokens list of the given address
    */
   function _addTokenToOwnerEnumeration(address to, uint256 tokenId) private {
       _ownedTokensIndex[tokenId] = _ownedTokens[to].length;
       _ownedTokens[to].push(tokenId);
   }

   /**
    * @dev Private function to add a token to this extension's token tracking data structures.
    * @param tokenId uint256 ID of the token to be added to the tokens list
    */
   function _addTokenToAllTokensEnumeration(uint256 tokenId) private {
       _allTokensIndex[tokenId] = _allTokens.length;
       _allTokens.push(tokenId);
   }

   /**
    * @dev Private function to remove a token from this extension's ownership-tracking data
structures. Note that
    * while the token is not assigned a new owner, the `_ownedTokensIndex` mapping is _not_ updated:
                this allows for
    * gas optimizations e.g. when performing a transfer operation (avoiding double writes).
    * This has O(1) time complexity, but alters the order of the _ownedTokens array.
    * @param from address representing the previous owner of the given token ID
    * @param tokenId uint256 ID of the token to be removed from the tokens list of the given address
    */
   function _removeTokenFromOwnerEnumeration(address from, uint256 tokenId) private {
       // To prevent a gap in from's tokens array, we store the last token in the index of the token to
delete, and
       // then delete the last slot (swap and pop).

       uint256 lastTokenIndex = _ownedTokens[from].length.sub(1);
       uint256 tokenIndex = _ownedTokensIndex[tokenId];

       // When the token to delete is the last token, the swap operation is unnecessary
       if (tokenIndex != lastTokenIndex) {
           uint256 lastTokenId = _ownedTokens[from][lastTokenIndex];

           _ownedTokens[from][tokenIndex] = lastTokenId; // Move the last token to the slot of the
to-delete token
           _ownedTokensIndex[lastTokenId] = tokenIndex; // Update the moved token's index
       }

       // This also deletes the contents at the last position of the array
       _ownedTokens[from].length--; //StFt //length-manipulation
```

```
        // Note that _ownedTokensIndex[tokenId] hasn't been cleared: it still points to the old slot (now
occupied by
        // lastTokenId, or just over the end of the array if the token was the last one).
    }

    /**
     * @dev Private function to remove a token from this extension's token tracking data structures.
     * This has O(1) time complexity, but alters the order of the _allTokens array.
     * @param tokenId uint256 ID of the token to be removed from the tokens list
     */
    function _removeTokenFromAllTokensEnumeration(uint256 tokenId) private {
        // To prevent a gap in the tokens array, we store the last token in the index of the token to
delete, and
        // then delete the last slot (swap and pop).

        uint256 lastTokenIndex = _allTokens.length.sub(1);
        uint256 tokenIndex = _allTokensIndex[tokenId];

        // When the token to delete is the last token, the swap operation is unnecessary. However, since
this occurs so
        // rarely (when the last minted token is burnt) that we still do the swap here to avoid the gas cost
                of adding
        // an 'if' statement (like in _removeTokenFromOwnerEnumeration)
        uint256 lastTokenId = _allTokens[lastTokenIndex];

        _allTokens[tokenIndex] = lastTokenId; // Move the last token to the slot of the to-delete token
        _allTokensIndex[lastTokenId] = tokenIndex; // Update the moved token's index

        // This also deletes the contents at the last position of the array
        _allTokens.length--; //StFt //length-manipulation
        _allTokensIndex[tokenId] = 0;
    }
}

// File: node_modules\@openzeppelin\contracts\token\ERC721\IERC721Metadata.sol

pragma solidity ^0.5.0;


/**
 * @title ERC-721 Non-Fungible Token Standard, optional metadata extension
 * @dev See https://eips.ethereum.org/EIPS/eip-721
 */
contract IERC721Metadata is IERC721 {
    function name() external view returns (string memory);
    function symbol() external view returns (string memory);
    function tokenURI(uint256 tokenId) external view returns (string memory);
}

// File: node_modules\@openzeppelin\contracts\token\ERC721\ERC721Metadata.sol

pragma solidity ^0.5.0;
```

```
contract ERC721Metadata is Context, ERC165, ERC721, IERC721Metadata {
    // Token name
    string private _name; //StFt //private-not-hidedata

    // Token symbol
    string private _symbol; //StFt //private-not-hidedata

    // Optional mapping for token URIs
    mapping(uint256 => string) private _tokenURIs; //StFt //private-not-hidedata

    /*
     *     bytes4(keccak256('name()')) == 0x06fdde03
     *     bytes4(keccak256('symbol()')) == 0x95d89b41
     *     bytes4(keccak256('tokenURI(uint256)')) == 0xc87b56dd
     *
     *     => 0x06fdde03 ^ 0x95d89b41 ^ 0xc87b56dd == 0x5b5e139f
     */
    bytes4 private constant _INTERFACE_ID_ERC721_METADATA = 0x5b5e139f; //StFt
//private-not-hidedata

    /**
     * @dev Constructor function
     */
    constructor (string memory name, string memory symbol) public {
        _name = name;
        _symbol = symbol;

        // register the supported interfaces to conform to ERC721 via ERC165
        _registerInterface(_INTERFACE_ID_ERC721_METADATA);
    }

    /**
     * @dev Gets the token name.
     * @return string representing the token name
     */
    function name() external view returns (string memory) {
        return _name;
    }

    /**
     * @dev Gets the token symbol.
     * @return string representing the token symbol
     */
    function symbol() external view returns (string memory) {
        return _symbol;
    }

    /**
     * @dev Returns an URI for a given token ID.
     * Throws if the token ID does not exist. May return an empty string.
     * @param tokenId uint256 ID of the token to query
     */
    function tokenURI(uint256 tokenId) external view returns (string memory) {
        require(_exists(tokenId), "ERC721Metadata: URI query for nonexistent token");
        return _tokenURIs[tokenId];
```

```
    }

    /**
     * @dev Internal function to set the token URI for a given token.
     * Reverts if the token ID does not exist.
     * @param tokenId uint256 ID of the token to set its URI
     * @param uri string URI to assign
     */
    function _setTokenURI(uint256 tokenId, string memory uri) internal {
        require(_exists(tokenId), "ERC721Metadata: URI set of nonexistent token");
        _tokenURIs[tokenId] = uri;
    }

    /**
     * @dev Internal function to burn a specific token.
     * Reverts if the token does not exist.
     * Deprecated, use _burn(uint256) instead.
     * @param owner owner of the token to burn
     * @param tokenId uint256 ID of the token being burned by the msg.sender
     */
    function _burn(address owner, uint256 tokenId) internal {
        super._burn(owner, tokenId);

        // Clear metadata (if any)
        if (bytes(_tokenURIs[tokenId]).length != 0) {
            delete _tokenURIs[tokenId];
        }
    }
}

// File: node_modules\@openzeppelin\contracts\token\ERC721\ERC721Full.sol

pragma solidity ^0.5.0;




/**
 * @title Full ERC721 Token
 * @dev This implementation includes all the required and some optional functionality of the ERC721
standard
 * Moreover, it includes approve all functionality using operator terminology.
 *
 * See https://eips.ethereum.org/EIPS/eip-721
 */
contract ERC721Full is ERC721, ERC721Enumerable, ERC721Metadata {
    constructor (string memory name, string memory symbol) public ERC721Metadata(name, symbol) {
        // solhint-disable-previous-line no-empty-blocks
    }
}

// File: node_modules\@openzeppelin\contracts\utils\ReentrancyGuard.sol

pragma solidity ^0.5.0;

/**
```

```
* @dev Contract module that helps prevent reentrant calls to a function.
*
* Inheriting from `ReentrancyGuard` will make the {nonReentrant} modifier
* available, which can be applied to functions to make sure there are no nested
* (reentrant) calls to them.
*
* Note that because there is a single `nonReentrant` guard, functions marked as
* `nonReentrant` may not call one another. This can be worked around by making
* those functions `private`, and then adding `external` `nonReentrant` entry
* points to them.
*/
contract ReentrancyGuard {
    // counter to allow mutex lock with only one SSTORE operation
    uint256 private _guardCounter; //StFt //private-not-hidedata

    constructor () internal {
        // The counter starts at one to prevent changing it from zero to a non-zero
        // value, which is a more expensive operation.
        _guardCounter = 1;
    }

    /**
     * @dev Prevents a contract from calling itself, directly or indirectly.
     * Calling a `nonReentrant` function from another `nonReentrant`
     * function is not supported. It is possible to prevent this from happening
     * by making the `nonReentrant` function external, and make it call a
     * `private` function that does the actual work.
     */
    modifier nonReentrant() {
        _guardCounter += 1;
        uint256 localCounter = _guardCounter;
        _;
        require(localCounter == _guardCounter, "ReentrancyGuard: reentrant call");
    }
}

// File: contracts\provableAPI.sol

//
/*


Copyright (c) 2015-2016 Oraclize SRL
Copyright (c) 2016-2019 Oraclize LTD
Copyright (c) 2019 Provable Things Limited

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.
```

```
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.  IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS 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 SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.

*/
pragma solidity >= 0.5.0 < 0.6.0; // Incompatible compiler version - please select a compiler within the
                    stated pragma range, or use a different version of the provableAPI!

// Dummy contract only used to emit to end-user they are using wrong solc
contract solcChecker {
/* INCOMPATIBLE SOLC: import the following instead: "
github.com/oraclize/ethereum-api/oraclizeAPI_0.4.sol" */ function f(bytes calldata x) external;
}

contract ProvableI {

    address public cbAddress;

    function setProofType(byte _proofType) external;
    function setCustomGasPrice(uint _gasPrice) external;
    function getPrice(string memory _datasource) public returns (uint _dsprice);
    function randomDS_getSessionPubKeyHash() external view returns (bytes32 _sessionKeyHash);
    function getPrice(string memory _datasource, uint _gasLimit) public returns (uint _dsprice);
    function queryN(uint _timestamp, string memory _datasource, bytes memory _argN) public payable
returns (bytes32 _id);
    function query(uint _timestamp, string calldata _datasource, string calldata _arg) external payable
returns (bytes32 _id);
    function query2(uint _timestamp, string memory _datasource, string memory _arg1, string memory
_arg2) public payable returns (bytes32 _id);
    function query_withGasLimit(uint _timestamp, string calldata _datasource, string calldata _arg, uint
_gasLimit) external payable returns (bytes32 _id);
    function queryN_withGasLimit(uint _timestamp, string calldata _datasource, bytes calldata _argN,
uint _gasLimit) external payable returns (bytes32 _id);
    function query2_withGasLimit(uint _timestamp, string calldata _datasource, string calldata _arg1,
string calldata _arg2, uint _gasLimit) external payable returns (bytes32 _id);
}

contract OracleAddrResolverI {
    function getAddress() public returns (address _address);
}
/*

Begin solidity-cborutils

https://github.com/smartcontractkit/solidity-cborutils

MIT License

Copyright (c) 2018 SmartContract ChainLink, Ltd.

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
```

```
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS 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 SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.

*/
library Buffer {

    struct buffer {
        bytes buf;
        uint capacity;
    }

    function init(buffer memory _buf, uint _capacity) internal pure { //StFt //constant-function-state
        uint capacity = _capacity;
        if (capacity % 32 != 0) {
            capacity += 32 - (capacity % 32);
        }
        _buf.capacity = capacity; // Allocate space for the buffer data
        assembly { //StFt //assembly
            let ptr := mload(0x40)
            mstore(_buf, ptr)
            mstore(ptr, 0)
            mstore(0x40, add(ptr, capacity))
        }
    }

    function resize(buffer memory _buf, uint _capacity) private pure {
        bytes memory oldbuf = _buf.buf;
        init(_buf, _capacity);
        append(_buf, oldbuf);
    }

    function max(uint _a, uint _b) private pure returns (uint _max) {
        if (_a > _b) {
            return _a;
        }
        return _b;
    }
    /**
     * @dev Appends a byte array to the end of the buffer. Resizes if doing so
     *      would exceed the capacity of the buffer.
     * @param _buf The buffer to append to.
     * @param _data The data to append.
     * @return The original buffer.
```

```
     *
     */
    function append(buffer memory _buf, bytes memory _data) internal pure returns (buffer memory
_buffer) { //StFt //constant-function-state
        if (_data.length + _buf.buf.length > _buf.capacity) {
            resize(_buf, max(_buf.capacity, _data.length) * 2);
        }
        uint dest;
        uint src;
        uint len = _data.length;
        assembly { //StFt //assembly
            let bufptr := mload(_buf) // Memory address of the buffer data
            let buflen := mload(bufptr) // Length of existing buffer data
            dest := add(add(bufptr, buflen), 32) // Start address = buffer address + buffer length +
sizeof(buffer length)
            mstore(bufptr, add(buflen, mload(_data))) // Update buffer length
            src := add(_data, 32)
        }
        for(; len >= 32; len -= 32) { // Copy word-length chunks while possible
            assembly { //StFt //assembly
                mstore(dest, mload(src))
            }
            dest += 32;
            src += 32;
        }
        uint mask = 256 ** (32 - len) - 1; // Copy remaining bytes
        assembly { //StFt //assembly
            let srcpart := and(mload(src), not(mask))
            let destpart := and(mload(dest), mask)
            mstore(dest, or(destpart, srcpart))
        }
        return _buf;
    }
    /**
     *
     * @dev Appends a byte to the end of the buffer. Resizes if doing so would
     * exceed the capacity of the buffer.
     * @param _buf The buffer to append to.
     * @param _data The data to append.
     * @return The original buffer.
     *
     */
    function append(buffer memory _buf, uint8 _data) internal pure { //StFt //constant-function-state
        if (_buf.buf.length + 1 > _buf.capacity) {
            resize(_buf, _buf.capacity * 2);
        }
        assembly { //StFt //assembly
            let bufptr := mload(_buf) // Memory address of the buffer data
            let buflen := mload(bufptr) // Length of existing buffer data
            let dest := add(add(bufptr, buflen), 32) // Address = buffer address + buffer length +
sizeof(buffer length)
            mstore8(dest, _data)
            mstore(bufptr, add(buflen, 1)) // Update buffer length
        }
    }
    /**
```

```
     *
     * @dev Appends a byte to the end of the buffer. Resizes if doing so would
     * exceed the capacity of the buffer.
     * @param _buf The buffer to append to.
     * @param _data The data to append.
     * @return The original buffer.
     *
     */
    function appendInt(buffer memory _buf, uint _data, uint _len) internal pure returns (buffer memory
_buffer) { //StFt //constant-function-state
        if (_len + _buf.buf.length > _buf.capacity) {
            resize(_buf, max(_buf.capacity, _len) * 2);
        }
        uint mask = 256 ** _len - 1;
        assembly { //StFt //assembly
            let bufptr := mload(_buf) // Memory address of the buffer data
            let buflen := mload(bufptr) // Length of existing buffer data
            let dest := add(add(bufptr, buflen), _len) // Address = buffer address + buffer length +
sizeof(buffer length) + len
            mstore(dest, or(and(mload(dest), not(mask)), _data))
            mstore(bufptr, add(buflen, _len)) // Update buffer length
        }
        return _buf;
    }
}

library CBOR {

    using Buffer for Buffer.buffer;

    uint8 private constant MAJOR_TYPE_INT = 0; //StFt //private-not-hidedata
    uint8 private constant MAJOR_TYPE_MAP = 5; //StFt //private-not-hidedata
    uint8 private constant MAJOR_TYPE_BYTES = 2; //StFt //private-not-hidedata
    uint8 private constant MAJOR_TYPE_ARRAY = 4; //StFt //private-not-hidedata
    uint8 private constant MAJOR_TYPE_STRING = 3; //StFt //private-not-hidedata
    uint8 private constant MAJOR_TYPE_NEGATIVE_INT = 1; //StFt //private-not-hidedata
    uint8 private constant MAJOR_TYPE_CONTENT_FREE = 7; //StFt //private-not-hidedata

    function encodeType(Buffer.buffer memory _buf, uint8 _major, uint _value) private pure {
        if (_value <= 23) {
            _buf.append(uint8((_major << 5) | _value));
        } else if (_value <= 0xFF) {
            _buf.append(uint8((_major << 5) | 24));
            _buf.appendInt(_value, 1);
        } else if (_value <= 0xFFFF) {
            _buf.append(uint8((_major << 5) | 25));
            _buf.appendInt(_value, 2);
        } else if (_value <= 0xFFFFFFFF) {
            _buf.append(uint8((_major << 5) | 26));
            _buf.appendInt(_value, 4);
        } else if (_value <= 0xFFFFFFFFFFFFFFFF) {
            _buf.append(uint8((_major << 5) | 27));
            _buf.appendInt(_value, 8);
        }
    }
```

```
    function encodeIndefiniteLengthType(Buffer.buffer memory _buf, uint8 _major) private pure {
        _buf.append(uint8((_major << 5) | 31));
    }

    function encodeUInt(Buffer.buffer memory _buf, uint _value) internal pure {
        encodeType(_buf, MAJOR_TYPE_INT, _value);
    }

    function encodeInt(Buffer.buffer memory _buf, int _value) internal pure {
        if (_value >= 0) {
            encodeType(_buf, MAJOR_TYPE_INT, uint(_value));
        } else {
            encodeType(_buf, MAJOR_TYPE_NEGATIVE_INT, uint(-1 - _value));
        }
    }

    function encodeBytes(Buffer.buffer memory _buf, bytes memory _value) internal pure {
        encodeType(_buf, MAJOR_TYPE_BYTES, _value.length);
        _buf.append(_value);
    }

    function encodeString(Buffer.buffer memory _buf, string memory _value) internal pure {
        encodeType(_buf, MAJOR_TYPE_STRING, bytes(_value).length);
        _buf.append(bytes(_value));
    }

    function startArray(Buffer.buffer memory _buf) internal pure {
        encodeIndefiniteLengthType(_buf, MAJOR_TYPE_ARRAY);
    }

    function startMap(Buffer.buffer memory _buf) internal pure {
        encodeIndefiniteLengthType(_buf, MAJOR_TYPE_MAP);
    }

    function endSequence(Buffer.buffer memory _buf) internal pure {
        encodeIndefiniteLengthType(_buf, MAJOR_TYPE_CONTENT_FREE);
    }
}
/*

End solidity-cborutils

*/
contract usingProvable {

    using CBOR for Buffer.buffer;

    ProvableI provable;
    OracleAddrResolverI OAR;

    uint constant day = 60 * 60 * 24;
    uint constant week = 60 * 60 * 24 * 7;
    uint constant month = 60 * 60 * 24 * 30;

    byte constant proofType_NONE = 0x00;
    byte constant proofType_Ledger = 0x30;
```

```
    byte constant proofType_Native = 0xF0;
    byte constant proofStorage_IPFS = 0x01;
    byte constant proofType_Android = 0x40;
    byte constant proofType_TLSNotary = 0x10;

    string provable_network_name;
    uint8 constant networkID_auto = 0;
    uint8 constant networkID_morden = 2;
    uint8 constant networkID_mainnet = 1;
    uint8 constant networkID_testnet = 2;
    uint8 constant networkID_consensys = 161;

    mapping(bytes32 => bytes32) provable_randomDS_args;
    mapping(bytes32 => bool) provable_randomDS_sessionKeysHashVerified;

    modifier provableAPI {
        if ((address(OAR) == address(0)) || (getCodeSize(address(OAR)) == 0)) {
            provable_setNetwork(networkID_auto);
        }
        if (address(provable) != OAR.getAddress()) {
            provable = ProvableI(OAR.getAddress());
        }
        _;
    }

    modifier provable_randomDS_proofVerify(bytes32 _queryId, string memory _result, bytes memory
_proof) {
        // RandomDS Proof Step 1: The prefix has to match 'LP\x01' (Ledger Proof version 1)
        require((_proof[0] == "L") && (_proof[1] == "P") && (uint8(_proof[2]) == uint8(1)));
        bool proofVerified = provable_randomDS_proofVerify__main(_proof, _queryId, bytes(_result),
provable_getNetworkName());
        require(proofVerified);
        _;
    }

    function provable_setNetwork(uint8 _networkID) internal returns (bool _networkSet) {
      _networkID; // NOTE: Silence the warning and remain backwards compatible
      return provable_setNetwork();
    }

    function provable_setNetworkName(string memory _network_name) internal {
        provable_network_name = _network_name;
    }

    function provable_getNetworkName() internal view returns (string memory _networkName) {
        return provable_network_name;
    }

    function provable_setNetwork() internal returns (bool _networkSet) {
        if (getCodeSize(0x1d3B2638a7cC9f2CB3D298A3DA7a90B67E5506ed) > 0) { //mainnet //StFt
//hardcoded
            OAR = OracleAddrResolverI(0x1d3B2638a7cC9f2CB3D298A3DA7a90B67E5506ed); //StFt
//hardcoded
            provable_setNetworkName("eth_mainnet");
            return true;
        }
```

```
        if (getCodeSize(0xc03A2615D5efaf5F49F60B7BB6583eaec212fdf1) > 0) { //ropsten testnet //StFt
//hardcoded
            OAR = OracleAddrResolverI(0xc03A2615D5efaf5F49F60B7BB6583eaec212fdf1); //StFt
//hardcoded
            provable_setNetworkName("eth_ropsten3");
            return true;
        }
        if (getCodeSize(0xB7A07BcF2Ba2f2703b24C0691b5278999C59AC7e) > 0) { //kovan testnet //StFt
//hardcoded
            OAR = OracleAddrResolverI(0xB7A07BcF2Ba2f2703b24C0691b5278999C59AC7e); //StFt
//hardcoded
            provable_setNetworkName("eth_kovan");
            return true;
        }
        if (getCodeSize(0x146500cfd35B22E4A392Fe0aDc06De1a1368Ed48) > 0) { //rinkeby testnet //StFt
                        //hardcoded
            OAR = OracleAddrResolverI(0x146500cfd35B22E4A392Fe0aDc06De1a1368Ed48); //StFt
//hardcoded
            provable_setNetworkName("eth_rinkeby");
            return true;
        }
        if (getCodeSize(0xa2998EFD205FB9D4B4963aFb70778D6354ad3A41) > 0) { //goerli testnet //StFt
//hardcoded
            OAR = OracleAddrResolverI(0xa2998EFD205FB9D4B4963aFb70778D6354ad3A41); //StFt
//hardcoded
            provable_setNetworkName("eth_goerli");
            return true;
        }
        if (getCodeSize(0x6f485C8BF6fc43eA212E93BBF8ce046C7f1cb475) > 0) { //ethereum-bridge
//StFt //hardcoded
            OAR = OracleAddrResolverI(0x6f485C8BF6fc43eA212E93BBF8ce046C7f1cb475); //StFt
//hardcoded
            return true;
        }
        if (getCodeSize(0x20e12A1F859B3FeaE5Fb2A0A32C18F5a65555bBF) > 0) { //ether.camp ide
//StFt //hardcoded
            OAR = OracleAddrResolverI(0x20e12A1F859B3FeaE5Fb2A0A32C18F5a65555bBF); //StFt
//hardcoded
            return true;
        }
        if (getCodeSize(0x51efaF4c8B3C9AfBD5aB9F4bbC82784Ab6ef8fAA) > 0) { //browser-solidity
//StFt //hardcoded
            OAR = OracleAddrResolverI(0x51efaF4c8B3C9AfBD5aB9F4bbC82784Ab6ef8fAA); //StFt
//hardcoded
            return true;
        }
        return false;
    }
    /**
     * @dev The following `__callback` functions are just placeholders ideally
     *      meant to be defined in child contract when proofs are used.
     *      The function bodies simply silence compiler warnings.
     */
    function __callback(bytes32 _myid, string memory _result) public {
        __callback(_myid, _result, new bytes(0));
    }
```

```
  function __callback(bytes32 _myid, string memory _result, bytes memory _proof) public {
    _myid; _result; _proof;
    provable_randomDS_args[bytes32(0)] = bytes32(0);
  }

  function provable_getPrice(string memory _datasource) provableAPI internal returns (uint
_queryPrice) {
      return provable.getPrice(_datasource);
  }

  function provable_getPrice(string memory _datasource, uint _gasLimit) provableAPI internal returns
(uint _queryPrice) {
      return provable.getPrice(_datasource, _gasLimit);
  }

  function provable_query(string memory _datasource, string memory _arg) provableAPI internal
returns (bytes32 _id) {
      uint price = provable.getPrice(_datasource);
      if (price > 1 ether + tx.gasprice * 200000) {
         return 0; // Unexpectedly high price
      }
      return provable.query.value(price)(0, _datasource, _arg);
  }

  function provable_query(uint _timestamp, string memory _datasource, string memory _arg)
provableAPI internal returns (bytes32 _id) {
      uint price = provable.getPrice(_datasource);
      if (price > 1 ether + tx.gasprice * 200000) {
         return 0; // Unexpectedly high price
      }
      return provable.query.value(price)(_timestamp, _datasource, _arg);
  }

  function provable_query(uint _timestamp, string memory _datasource, string memory _arg, uint
_gasLimit) provableAPI internal returns (bytes32 _id) {
      uint price = provable.getPrice(_datasource,_gasLimit);
      if (price > 1 ether + tx.gasprice * _gasLimit) {
         return 0; // Unexpectedly high price
      }
      return provable.query_withGasLimit.value(price)(_timestamp, _datasource, _arg, _gasLimit);
  }

  function provable_query(string memory _datasource, string memory _arg, uint _gasLimit)
provableAPI internal returns (bytes32 _id) {
      uint price = provable.getPrice(_datasource, _gasLimit);
      if (price > 1 ether + tx.gasprice * _gasLimit) {
        return 0; // Unexpectedly high price
      }
      return provable.query_withGasLimit.value(price)(0, _datasource, _arg, _gasLimit);
  }

  function provable_query(string memory _datasource, string memory _arg1, string memory _arg2)
provableAPI internal returns (bytes32 _id) {
      uint price = provable.getPrice(_datasource);
      if (price > 1 ether + tx.gasprice * 200000) {
```

```
        return 0; // Unexpectedly high price
    }
    return provable.query2.value(price)(0, _datasource, _arg1, _arg2);
  }

  function provable_query(uint _timestamp, string memory _datasource, string memory _arg1, string
memory _arg2) provableAPI internal returns (bytes32 _id) {
    uint price = provable.getPrice(_datasource);
    if (price > 1 ether + tx.gasprice * 200000) {
        return 0; // Unexpectedly high price
    }
    return provable.query2.value(price)(_timestamp, _datasource, _arg1, _arg2);
  }

  function provable_query(uint _timestamp, string memory _datasource, string memory _arg1, string
memory _arg2, uint _gasLimit) provableAPI internal returns (bytes32 _id) {
    uint price = provable.getPrice(_datasource, _gasLimit);
    if (price > 1 ether + tx.gasprice * _gasLimit) {
        return 0; // Unexpectedly high price
    }
    return provable.query2_withGasLimit.value(price)(_timestamp, _datasource, _arg1, _arg2,
_gasLimit);
  }

  function provable_query(string memory _datasource, string memory _arg1, string memory _arg2,
uint _gasLimit) provableAPI internal returns (bytes32 _id) {
    uint price = provable.getPrice(_datasource, _gasLimit);
    if (price > 1 ether + tx.gasprice * _gasLimit) {
        return 0; // Unexpectedly high price
    }
    return provable.query2_withGasLimit.value(price)(0, _datasource, _arg1, _arg2, _gasLimit);
  }

  function provable_query(string memory _datasource, string[] memory _argN) provableAPI internal
returns (bytes32 _id) {
    uint price = provable.getPrice(_datasource);
    if (price > 1 ether + tx.gasprice * 200000) {
        return 0; // Unexpectedly high price
    }
    bytes memory args = stra2cbor(_argN);
    return provable.queryN.value(price)(0, _datasource, args);
  }

  function provable_query(uint _timestamp, string memory _datasource, string[] memory _argN)
provableAPI internal returns (bytes32 _id) {
    uint price = provable.getPrice(_datasource);
    if (price > 1 ether + tx.gasprice * 200000) {
        return 0; // Unexpectedly high price
    }
    bytes memory args = stra2cbor(_argN);
    return provable.queryN.value(price)(_timestamp, _datasource, args);
  }

  function provable_query(uint _timestamp, string memory _datasource, string[] memory _argN, uint
_gasLimit) provableAPI internal returns (bytes32 _id) {
    uint price = provable.getPrice(_datasource, _gasLimit);
```

```
        if (price > 1 ether + tx.gasprice * _gasLimit) {
            return 0; // Unexpectedly high price
        }
        bytes memory args = stra2cbor(_argN);
        return provable.queryN_withGasLimit.value(price)(_timestamp, _datasource, args, _gasLimit);
    }

    function provable_query(string memory _datasource, string[] memory _argN, uint _gasLimit)
provableAPI internal returns (bytes32 _id) {
        uint price = provable.getPrice(_datasource, _gasLimit);
        if (price > 1 ether + tx.gasprice * _gasLimit) {
            return 0; // Unexpectedly high price
        }
        bytes memory args = stra2cbor(_argN);
        return provable.queryN_withGasLimit.value(price)(0, _datasource, args, _gasLimit);
    }

    function provable_query(string memory _datasource, string[1] memory _args) provableAPI internal
returns (bytes32 _id) {
        string[] memory dynargs = new string[](1);
        dynargs[0] = _args[0];
        return provable_query(_datasource, dynargs);
    }

    function provable_query(uint _timestamp, string memory _datasource, string[1] memory _args)
provableAPI internal returns (bytes32 _id) {
        string[] memory dynargs = new string[](1);
        dynargs[0] = _args[0];
        return provable_query(_timestamp, _datasource, dynargs);
    }

    function provable_query(uint _timestamp, string memory _datasource, string[1] memory _args, uint
_gasLimit) provableAPI internal returns (bytes32 _id) {
        string[] memory dynargs = new string[](1);
        dynargs[0] = _args[0];
        return provable_query(_timestamp, _datasource, dynargs, _gasLimit);
    }

    function provable_query(string memory _datasource, string[1] memory _args, uint _gasLimit)
provableAPI internal returns (bytes32 _id) {
        string[] memory dynargs = new string[](1);
        dynargs[0] = _args[0];
        return provable_query(_datasource, dynargs, _gasLimit);
    }

    function provable_query(string memory _datasource, string[2] memory _args) provableAPI internal
returns (bytes32 _id) {
        string[] memory dynargs = new string[](2);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        return provable_query(_datasource, dynargs);
    }

    function provable_query(uint _timestamp, string memory _datasource, string[2] memory _args)
provableAPI internal returns (bytes32 _id) {
        string[] memory dynargs = new string[](2);
```

```
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        return provable_query(_timestamp, _datasource, dynargs);
    }

    function provable_query(uint _timestamp, string memory _datasource, string[2] memory _args, uint
_gasLimit) provableAPI internal returns (bytes32 _id) {
        string[] memory dynargs = new string[](2);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        return provable_query(_timestamp, _datasource, dynargs, _gasLimit);
    }

    function provable_query(string memory _datasource, string[2] memory _args, uint _gasLimit)
provableAPI internal returns (bytes32 _id) {
        string[] memory dynargs = new string[](2);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        return provable_query(_datasource, dynargs, _gasLimit);
    }

    function provable_query(string memory _datasource, string[3] memory _args) provableAPI internal
returns (bytes32 _id) {
        string[] memory dynargs = new string[](3);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        dynargs[2] = _args[2];
        return provable_query(_datasource, dynargs);
    }

    function provable_query(uint _timestamp, string memory _datasource, string[3] memory _args)
provableAPI internal returns (bytes32 _id) {
        string[] memory dynargs = new string[](3);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        dynargs[2] = _args[2];
        return provable_query(_timestamp, _datasource, dynargs);
    }

    function provable_query(uint _timestamp, string memory _datasource, string[3] memory _args, uint
_gasLimit) provableAPI internal returns (bytes32 _id) {
        string[] memory dynargs = new string[](3);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        dynargs[2] = _args[2];
        return provable_query(_timestamp, _datasource, dynargs, _gasLimit);
    }

    function provable_query(string memory _datasource, string[3] memory _args, uint _gasLimit)
provableAPI internal returns (bytes32 _id) {
        string[] memory dynargs = new string[](3);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        dynargs[2] = _args[2];
        return provable_query(_datasource, dynargs, _gasLimit);
    }
```

```
    function provable_query(string memory _datasource, string[4] memory _args) provableAPI internal
returns (bytes32 _id) {
        string[] memory dynargs = new string[](4);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        dynargs[2] = _args[2];
        dynargs[3] = _args[3];
        return provable_query(_datasource, dynargs);
    }

    function provable_query(uint _timestamp, string memory _datasource, string[4] memory _args)
provableAPI internal returns (bytes32 _id) {
        string[] memory dynargs = new string[](4);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        dynargs[2] = _args[2];
        dynargs[3] = _args[3];
        return provable_query(_timestamp, _datasource, dynargs);
    }

    function provable_query(uint _timestamp, string memory _datasource, string[4] memory _args, uint
_gasLimit) provableAPI internal returns (bytes32 _id) {
        string[] memory dynargs = new string[](4);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        dynargs[2] = _args[2];
        dynargs[3] = _args[3];
        return provable_query(_timestamp, _datasource, dynargs, _gasLimit);
    }

    function provable_query(string memory _datasource, string[4] memory _args, uint _gasLimit)
provableAPI internal returns (bytes32 _id) {
        string[] memory dynargs = new string[](4);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        dynargs[2] = _args[2];
        dynargs[3] = _args[3];
        return provable_query(_datasource, dynargs, _gasLimit);
    }

    function provable_query(string memory _datasource, string[5] memory _args) provableAPI internal
returns (bytes32 _id) {
        string[] memory dynargs = new string[](5);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        dynargs[2] = _args[2];
        dynargs[3] = _args[3];
        dynargs[4] = _args[4];
        return provable_query(_datasource, dynargs);
    }

    function provable_query(uint _timestamp, string memory _datasource, string[5] memory _args)
provableAPI internal returns (bytes32 _id) {
        string[] memory dynargs = new string[](5);
        dynargs[0] = _args[0];
```

```
        dynargs[1] = _args[1];
        dynargs[2] = _args[2];
        dynargs[3] = _args[3];
        dynargs[4] = _args[4];
        return provable_query(_timestamp, _datasource, dynargs);
    }

    function provable_query(uint _timestamp, string memory _datasource, string[5] memory _args, uint
_gasLimit) provableAPI internal returns (bytes32 _id) {
        string[] memory dynargs = new string[](5);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        dynargs[2] = _args[2];
        dynargs[3] = _args[3];
        dynargs[4] = _args[4];
        return provable_query(_timestamp, _datasource, dynargs, _gasLimit);
    }

    function provable_query(string memory _datasource, string[5] memory _args, uint _gasLimit)
provableAPI internal returns (bytes32 _id) {
        string[] memory dynargs = new string[](5);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        dynargs[2] = _args[2];
        dynargs[3] = _args[3];
        dynargs[4] = _args[4];
        return provable_query(_datasource, dynargs, _gasLimit);
    }

    function provable_query(string memory _datasource, bytes[] memory _argN) provableAPI internal
returns (bytes32 _id) {
        uint price = provable.getPrice(_datasource);
        if (price > 1 ether + tx.gasprice * 200000) {
            return 0; // Unexpectedly high price
        }
        bytes memory args = ba2cbor(_argN);
        return provable.queryN.value(price)(0, _datasource, args);
    }

    function provable_query(uint _timestamp, string memory _datasource, bytes[] memory _argN)
provableAPI internal returns (bytes32 _id) {
        uint price = provable.getPrice(_datasource);
        if (price > 1 ether + tx.gasprice * 200000) {
            return 0; // Unexpectedly high price
        }
        bytes memory args = ba2cbor(_argN);
        return provable.queryN.value(price)(_timestamp, _datasource, args);
    }

    function provable_query(uint _timestamp, string memory _datasource, bytes[] memory _argN, uint
_gasLimit) provableAPI internal returns (bytes32 _id) {
        uint price = provable.getPrice(_datasource, _gasLimit);
        if (price > 1 ether + tx.gasprice * _gasLimit) {
            return 0; // Unexpectedly high price
        }
        bytes memory args = ba2cbor(_argN);
```

```
        return provable.queryN_withGasLimit.value(price)(_timestamp, _datasource, args, _gasLimit);
    }

    function provable_query(string memory _datasource, bytes[] memory _argN, uint _gasLimit)
provableAPI internal returns (bytes32 _id) {
        uint price = provable.getPrice(_datasource, _gasLimit);
        if (price > 1 ether + tx.gasprice * _gasLimit) {
            return 0; // Unexpectedly high price
        }
        bytes memory args = ba2cbor(_argN);
        return provable.queryN_withGasLimit.value(price)(0, _datasource, args, _gasLimit);
    }

    function provable_query(string memory _datasource, bytes[1] memory _args) provableAPI internal
returns (bytes32 _id) {
        bytes[] memory dynargs = new bytes[](1);
        dynargs[0] = _args[0];
        return provable_query(_datasource, dynargs);
    }

    function provable_query(uint _timestamp, string memory _datasource, bytes[1] memory _args)
provableAPI internal returns (bytes32 _id) {
        bytes[] memory dynargs = new bytes[](1);
        dynargs[0] = _args[0];
        return provable_query(_timestamp, _datasource, dynargs);
    }

    function provable_query(uint _timestamp, string memory _datasource, bytes[1] memory _args, uint
_gasLimit) provableAPI internal returns (bytes32 _id) {
        bytes[] memory dynargs = new bytes[](1);
        dynargs[0] = _args[0];
        return provable_query(_timestamp, _datasource, dynargs, _gasLimit);
    }

    function provable_query(string memory _datasource, bytes[1] memory _args, uint _gasLimit)
provableAPI internal returns (bytes32 _id) {
        bytes[] memory dynargs = new bytes[](1);
        dynargs[0] = _args[0];
        return provable_query(_datasource, dynargs, _gasLimit);
    }

    function provable_query(string memory _datasource, bytes[2] memory _args) provableAPI internal
returns (bytes32 _id) {
        bytes[] memory dynargs = new bytes[](2);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        return provable_query(_datasource, dynargs);
    }

    function provable_query(uint _timestamp, string memory _datasource, bytes[2] memory _args)
provableAPI internal returns (bytes32 _id) {
        bytes[] memory dynargs = new bytes[](2);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        return provable_query(_timestamp, _datasource, dynargs);
    }
```

```
    function provable_query(uint _timestamp, string memory _datasource, bytes[2] memory _args, uint
_gasLimit) provableAPI internal returns (bytes32 _id) {
        bytes[] memory dynargs = new bytes[](2);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        return provable_query(_timestamp, _datasource, dynargs, _gasLimit);
    }

    function provable_query(string memory _datasource, bytes[2] memory _args, uint _gasLimit)
provableAPI internal returns (bytes32 _id) {
        bytes[] memory dynargs = new bytes[](2);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        return provable_query(_datasource, dynargs, _gasLimit);
    }

    function provable_query(string memory _datasource, bytes[3] memory _args) provableAPI internal
returns (bytes32 _id) {
        bytes[] memory dynargs = new bytes[](3);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        dynargs[2] = _args[2];
        return provable_query(_datasource, dynargs);
    }

    function provable_query(uint _timestamp, string memory _datasource, bytes[3] memory _args)
provableAPI internal returns (bytes32 _id) {
        bytes[] memory dynargs = new bytes[](3);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        dynargs[2] = _args[2];
        return provable_query(_timestamp, _datasource, dynargs);
    }

    function provable_query(uint _timestamp, string memory _datasource, bytes[3] memory _args, uint
_gasLimit) provableAPI internal returns (bytes32 _id) {
        bytes[] memory dynargs = new bytes[](3);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        dynargs[2] = _args[2];
        return provable_query(_timestamp, _datasource, dynargs, _gasLimit);
    }

    function provable_query(string memory _datasource, bytes[3] memory _args, uint _gasLimit)
provableAPI internal returns (bytes32 _id) {
        bytes[] memory dynargs = new bytes[](3);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        dynargs[2] = _args[2];
        return provable_query(_datasource, dynargs, _gasLimit);
    }

    function provable_query(string memory _datasource, bytes[4] memory _args) provableAPI internal
returns (bytes32 _id) {
        bytes[] memory dynargs = new bytes[](4);
```

```
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        dynargs[2] = _args[2];
        dynargs[3] = _args[3];
        return provable_query(_datasource, dynargs);
    }

    function provable_query(uint _timestamp, string memory _datasource, bytes[4] memory _args)
provableAPI internal returns (bytes32 _id) {
        bytes[] memory dynargs = new bytes[](4);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        dynargs[2] = _args[2];
        dynargs[3] = _args[3];
        return provable_query(_timestamp, _datasource, dynargs);
    }

    function provable_query(uint _timestamp, string memory _datasource, bytes[4] memory _args, uint
_gasLimit) provableAPI internal returns (bytes32 _id) {
        bytes[] memory dynargs = new bytes[](4);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        dynargs[2] = _args[2];
        dynargs[3] = _args[3];
        return provable_query(_timestamp, _datasource, dynargs, _gasLimit);
    }

    function provable_query(string memory _datasource, bytes[4] memory _args, uint _gasLimit)
provableAPI internal returns (bytes32 _id) {
        bytes[] memory dynargs = new bytes[](4);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        dynargs[2] = _args[2];
        dynargs[3] = _args[3];
        return provable_query(_datasource, dynargs, _gasLimit);
    }

    function provable_query(string memory _datasource, bytes[5] memory _args) provableAPI internal
returns (bytes32 _id) {
        bytes[] memory dynargs = new bytes[](5);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        dynargs[2] = _args[2];
        dynargs[3] = _args[3];
        dynargs[4] = _args[4];
        return provable_query(_datasource, dynargs);
    }

    function provable_query(uint _timestamp, string memory _datasource, bytes[5] memory _args)
provableAPI internal returns (bytes32 _id) {
        bytes[] memory dynargs = new bytes[](5);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        dynargs[2] = _args[2];
        dynargs[3] = _args[3];
        dynargs[4] = _args[4];
```

```
        return provable_query(_timestamp, _datasource, dynargs);
    }

    function provable_query(uint _timestamp, string memory _datasource, bytes[5] memory _args, uint
_gasLimit) provableAPI internal returns (bytes32 _id) {
        bytes[] memory dynargs = new bytes[](5);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        dynargs[2] = _args[2];
        dynargs[3] = _args[3];
        dynargs[4] = _args[4];
        return provable_query(_timestamp, _datasource, dynargs, _gasLimit);
    }

    function provable_query(string memory _datasource, bytes[5] memory _args, uint _gasLimit)
provableAPI internal returns (bytes32 _id) {
        bytes[] memory dynargs = new bytes[](5);
        dynargs[0] = _args[0];
        dynargs[1] = _args[1];
        dynargs[2] = _args[2];
        dynargs[3] = _args[3];
        dynargs[4] = _args[4];
        return provable_query(_datasource, dynargs, _gasLimit);
    }

    function provable_setProof(byte _proofP) provableAPI internal {
        return provable.setProofType(_proofP);
    }


    function provable_cbAddress() provableAPI internal returns (address _callbackAddress) {
        return provable.cbAddress();
    }

    function getCodeSize(address _addr) view internal returns (uint _size) { //StFt
//constant-function-state
        assembly {
            _size := extcodesize(_addr)
        }
    }

    function provable_setCustomGasPrice(uint _gasPrice) provableAPI internal {
        return provable.setCustomGasPrice(_gasPrice);
    }

    function provable_randomDS_getSessionPubKeyHash() provableAPI internal returns (bytes32
_sessionKeyHash) {
        return provable.randomDS_getSessionPubKeyHash();
    }

    function parseAddr(string memory _a) internal pure returns (address _parsedAddress) {
        bytes memory tmp = bytes(_a);
        uint160 iaddr = 0;
        uint160 b1;
        uint160 b2;
        for (uint i = 2; i < 2 + 2 * 20; i += 2) {
```

```
        iaddr *= 256;
        b1 = uint160(uint8(tmp[i]));
        b2 = uint160(uint8(tmp[i + 1]));
        if ((b1 >= 97) && (b1 <= 102)) {
            b1 -= 87;
        } else if ((b1 >= 65) && (b1 <= 70)) {
            b1 -= 55;
        } else if ((b1 >= 48) && (b1 <= 57)) {
            b1 -= 48;
        }
        if ((b2 >= 97) && (b2 <= 102)) {
            b2 -= 87;
        } else if ((b2 >= 65) && (b2 <= 70)) {
            b2 -= 55;
        } else if ((b2 >= 48) && (b2 <= 57)) {
            b2 -= 48;
        }
        iaddr += (b1 * 16 + b2);
    }
    return address(iaddr);
}

function strCompare(string memory _a, string memory _b) internal pure returns (int _returnCode) {
    bytes memory a = bytes(_a);
    bytes memory b = bytes(_b);
    uint minLength = a.length;
    if (b.length < minLength) {
        minLength = b.length;
    }
    for (uint i = 0; i < minLength; i ++) { //StFt //costly-loop
        if (a[i] < b[i]) {
            return -1;
        } else if (a[i] > b[i]) {
            return 1;
        }
    }
    if (a.length < b.length) {
        return -1;
    } else if (a.length > b.length) {
        return 1;
    } else {
        return 0;
    }
}

function indexOf(string memory _haystack, string memory _needle) internal pure returns (int
_returnCode) {
    bytes memory h = bytes(_haystack);
    bytes memory n = bytes(_needle);
    if (h.length < 1 || n.length < 1 || (n.length > h.length)) {
        return -1;
    } else if (h.length > (2 ** 128 - 1)) {
        return -1;
    } else {
        uint subindex = 0;
        for (uint i = 0; i < h.length; i++) { //StFt //costly-loop
```

```
            if (h[i] == n[0]) {
                subindex = 1;
                while(subindex < n.length && (i + subindex) < h.length && h[i + subindex] ==
n[subindex]) { //StFt //costly-loop
                    subindex++;
                }
                if (subindex == n.length) {
                    return int(i);
                }
            }
        }
        return -1;
    }
}

    function strConcat(string memory _a, string memory _b) internal pure returns (string memory
_concatenatedString) {
        return strConcat(_a, _b, "", "", "");
    }

    function strConcat(string memory _a, string memory _b, string memory _c) internal pure returns
(string memory _concatenatedString) {
        return strConcat(_a, _b, _c, "", "");
    }

    function strConcat(string memory _a, string memory _b, string memory _c, string memory _d)
internal pure returns (string memory _concatenatedString) {
        return strConcat(_a, _b, _c, _d, "");
    }

    function strConcat(string memory _a, string memory _b, string memory _c, string memory _d, string
memory _e) internal pure returns (string memory _concatenatedString) {
        bytes memory _ba = bytes(_a);
        bytes memory _bb = bytes(_b);
        bytes memory _bc = bytes(_c);
        bytes memory _bd = bytes(_d);
        bytes memory _be = bytes(_e);
        string memory abcde = new string(_ba.length + _bb.length + _bc.length + _bd.length +
_be.length);
        bytes memory babcde = bytes(abcde);
        uint k = 0;
        uint i = 0;
        for (i = 0; i < _ba.length; i++) { //StFt //costly-loop
            babcde[k++] = _ba[i];
        }
        for (i = 0; i < _bb.length; i++) { //StFt //costly-loop
            babcde[k++] = _bb[i];
        }
        for (i = 0; i < _bc.length; i++) { //StFt //costly-loop
            babcde[k++] = _bc[i];
        }
        for (i = 0; i < _bd.length; i++) { //StFt //costly-loop
            babcde[k++] = _bd[i];
        }
        for (i = 0; i < _be.length; i++) { //StFt //costly-loop
            babcde[k++] = _be[i];
```

```
    }
    return string(babcde);
}

function safeParseInt(string memory _a) internal pure returns (uint _parsedInt) {
    return safeParseInt(_a, 0);
}

function safeParseInt(string memory _a, uint _b) internal pure returns (uint _parsedInt) {
    bytes memory bresult = bytes(_a);
    uint mint = 0;
    bool decimals = false;
    for (uint i = 0; i < bresult.length; i++) { //StFt //costly-loop
        if ((uint(uint8(bresult[i])) >= 48) && (uint(uint8(bresult[i])) <= 57)) {
            if (decimals) {
                if (_b == 0) break;
                 else _b--;
            }
            mint *= 10;
            mint += uint(uint8(bresult[i])) - 48;
        } else if (uint(uint8(bresult[i])) == 46) { //StFt //revert-require
            require(!decimals, 'More than one decimal encountered in string!');
            decimals = true;
        } else {
            revert("Non-numeral character encountered in string!");
        }
    }
    if (_b > 0) {
        mint *= 10 ** _b;
    }
    return mint;
}

function parseInt(string memory _a) internal pure returns (uint _parsedInt) {
    return parseInt(_a, 0);
}

function parseInt(string memory _a, uint _b) internal pure returns (uint _parsedInt) {
    bytes memory bresult = bytes(_a);
    uint mint = 0;
    bool decimals = false;
    for (uint i = 0; i < bresult.length; i++) { //StFt //costly-loop
        if ((uint(uint8(bresult[i])) >= 48) && (uint(uint8(bresult[i])) <= 57)) {
            if (decimals) {
                if (_b == 0) {
                    break;
                } else {
                    _b--;
                }
            }
            mint *= 10;
            mint += uint(uint8(bresult[i])) - 48;
        } else if (uint(uint8(bresult[i])) == 46) {
            decimals = true;
        }
    }
```

```
    if (_b > 0) {
        mint *= 10 ** _b;
    }
    return mint;
}

function uint2str(uint _i) internal pure returns (string memory _uintAsString) {
    if (_i == 0) {
        return "0";
    }
    uint j = _i;
    uint len;
    while (j != 0) {
        len++;
        j /= 10;
    }
    bytes memory bstr = new bytes(len);
    uint k = len - 1;
    while (_i != 0) {
        bstr[k--] = byte(uint8(48 + _i % 10));
        _i /= 10;
    }
    return string(bstr);
}

function stra2cbor(string[] memory _arr) internal pure returns (bytes memory _cborEncoding) {
    safeMemoryCleaner();
    Buffer.buffer memory buf;
    Buffer.init(buf, 1024);
    buf.startArray();
    for (uint i = 0; i < _arr.length; i++) { //StFt //costly-loop
        buf.encodeString(_arr[i]);
    }
    buf.endSequence();
    return buf.buf;
}

function ba2cbor(bytes[] memory _arr) internal pure returns (bytes memory _cborEncoding) {
    safeMemoryCleaner();
    Buffer.buffer memory buf;
    Buffer.init(buf, 1024);
    buf.startArray();
    for (uint i = 0; i < _arr.length; i++) { //StFt //costly-loop
        buf.encodeBytes(_arr[i]);
    }
    buf.endSequence();
    return buf.buf;
}

function provable_newRandomDSQuery(uint _delay, uint _nbytes, uint _customGasLimit) internal
returns (bytes32 _queryId) {
    require((_nbytes > 0) && (_nbytes <= 32));
    _delay *= 10; // Convert from seconds to ledger timer ticks
    bytes memory nbytes = new bytes(1);
    nbytes[0] = byte(uint8(_nbytes));
    bytes memory unonce = new bytes(32);
```

```
        bytes memory sessionKeyHash = new bytes(32);
        bytes32 sessionKeyHash_bytes32 = provable_randomDS_getSessionPubKeyHash();
        assembly { //StFt //assembly
            mstore(unonce, 0x20)
            /*
             The following variables can be relaxed.
             Check the relaxed random contract at https://github.com/oraclize/ethereum-examples
             for an idea on how to override and replace commit hash variables.
            */
            mstore(add(unonce, 0x20), xor(blockhash(sub(number, 1)), xor(coinbase, timestamp)))
            mstore(sessionKeyHash, 0x20)
            mstore(add(sessionKeyHash, 0x20), sessionKeyHash_bytes32)
        }
        bytes memory delay = new bytes(32);
        assembly { //StFt //assembly
            mstore(add(delay, 0x20), _delay)
        }
        bytes memory delay_bytes8 = new bytes(8);
        copyBytes(delay, 24, 8, delay_bytes8, 0);
        bytes[4] memory args = [unonce, nbytes, sessionKeyHash, delay];
        bytes32 queryId = provable_query("random", args, _customGasLimit);
        bytes memory delay_bytes8_left = new bytes(8);
        assembly { //StFt //assembly
            let x := mload(add(delay_bytes8, 0x20))
            mstore8(add(delay_bytes8_left, 0x27), div(x,
0x100000000000000000000000000000000000000000000000000000000000000))
            mstore8(add(delay_bytes8_left, 0x26), div(x,
0x10000000000000000000000000000000000000000000000000000000000))
            mstore8(add(delay_bytes8_left, 0x25), div(x,
0x1000000000000000000000000000000000000000000000000000000))
            mstore8(add(delay_bytes8_left, 0x24), div(x,
0x100000000000000000000000000000000000000000000000000))
            mstore8(add(delay_bytes8_left, 0x23), div(x,
0x10000000000000000000000000000000000000000000000))
            mstore8(add(delay_bytes8_left, 0x22), div(x,
0x1000000000000000000000000000000000000000000))
            mstore8(add(delay_bytes8_left, 0x21), div(x,
0x100000000000000000000000000000000000000000))
            mstore8(add(delay_bytes8_left, 0x20), div(x,
0x10000000000000000000000000000000000000000))
        }
        provable_randomDS_setCommitment(queryId, keccak256(abi.encodePacked(delay_bytes8_left,
args[1], sha256(args[0]), args[2])));
        return queryId;
    }

    function provable_randomDS_setCommitment(bytes32 _queryId, bytes32 _commitment) internal {
        provable_randomDS_args[_queryId] = _commitment;
    }

    function verifySig(bytes32 _tosignh, bytes memory _dersig, bytes memory _pubkey) internal returns
(bool _sigVerified) {
        bool sigok;
        address signer;
        bytes32 sigr;
        bytes32 sigs;
```

```
        bytes memory sigr_ = new bytes(32);
        uint offset = 4 + (uint(uint8(_dersig[3])) - 0x20);
        sigr_ = copyBytes(_dersig, offset, 32, sigr_, 0);
        bytes memory sigs_ = new bytes(32);
        offset += 32 + 2;
        sigs_ = copyBytes(_dersig, offset + (uint(uint8(_dersig[offset - 1])) - 0x20), 32, sigs_, 0);
        assembly { //StFt //assembly
            sigr := mload(add(sigr_, 32))
            sigs := mload(add(sigs_, 32))
        }
        (sigok, signer) = safer_ecrecover(_tosignh, 27, sigr, sigs);
        if (address(uint160(uint256(keccak256(_pubkey)))) == signer) {
            return true;
        } else {
            (sigok, signer) = safer_ecrecover(_tosignh, 28, sigr, sigs);
            return (address(uint160(uint256(keccak256(_pubkey)))) == signer);
        }
    }

    function provable_randomDS_proofVerify__sessionKeyValidity(bytes memory _proof, uint
_sig2offset) internal returns (bool _proofVerified) {
        bool sigok;
        // Random DS Proof Step 6: Verify the attestation signature, APPKEY1 must sign the sessionKey
from the correct ledger app (CODEHASH)
        bytes memory sig2 = new bytes(uint(uint8(_proof[_sig2offset + 1])) + 2);
        copyBytes(_proof, _sig2offset, sig2.length, sig2, 0);
        bytes memory appkey1_pubkey = new bytes(64);
        copyBytes(_proof, 3 + 1, 64, appkey1_pubkey, 0);
        bytes memory tosign2 = new bytes(1 + 65 + 32);
        tosign2[0] = byte(uint8(1)); //role
        copyBytes(_proof, _sig2offset - 65, 65, tosign2, 1);
        bytes memory CODEHASH = hex"fd94fa71bc0ba10d39d464d0d8f465efeef0a2764e3887fcc9df41d
                                    ed20f505c";
        copyBytes(CODEHASH, 0, 32, tosign2, 1 + 65);
        sigok = verifySig(sha256(tosign2), sig2, appkey1_pubkey);
        if (!sigok) {
            return false;
        }
        // Random DS Proof Step 7: Verify the APPKEY1 provenance (must be signed by Ledger)
        bytes memory LEDGERKEY = hex"7fb956469c5c9b89840d55b43537e66a98dd4811ea0a27224272c
                                    2e5622911e8537a2f8e86a46baec82864e98dd01e9ccc2f8bc5dfc9cbe5a91
a 2 9 0 4 9 8 d d 9 6 e 4" ;
        bytes memory tosign3 = new bytes(1 + 65);
        tosign3[0] = 0xFE;
        copyBytes(_proof, 3, 65, tosign3, 1);
        bytes memory sig3 = new bytes(uint(uint8(_proof[3 + 65 + 1])) + 2);
        copyBytes(_proof, 3 + 65, sig3.length, sig3, 0);
        sigok = verifySig(sha256(tosign3), sig3, LEDGERKEY);
        return sigok;
    }

    function provable_randomDS_proofVerify__returnCode(bytes32 _queryId, string memory _result,
bytes memory _proof) internal returns (uint8 _returnCode) {
        // Random DS Proof Step 1: The prefix has to match 'LP\x01' (Ledger Proof version 1)
        if ((_proof[0] != "L") || (_proof[1] != "P") || (uint8(_proof[2]) != uint8(1))) {
            return 1;
```

```
        }
        bool proofVerified = provable_randomDS_proofVerify__main(_proof, _queryId, bytes(_result),
provable_getNetworkName());
        if (!proofVerified) {
            return 2;
        }
        return 0;
    }

    function matchBytes32Prefix(bytes32 _content, bytes memory _prefix, uint _nRandomBytes) internal
pure returns (bool _matchesPrefix) {
        bool match_ = true;
        require(_prefix.length == _nRandomBytes);
        for (uint256 i = 0; i< _nRandomBytes; i++) {
            if (_content[i] != _prefix[i]) {
                match_ = false;
            }
        }
        return match_;
    }

    function provable_randomDS_proofVerify__main(bytes memory _proof, bytes32 _queryId, bytes
memory _result, string memory _contextName) internal returns (bool _proofVerified) {
        // Random DS Proof Step 2: The unique keyhash has to match with the sha256 of (context name
+ _queryId)
        uint ledgerProofLength = 3 + 65 + (uint(uint8(_proof[3 + 65 + 1])) + 2) + 32;
        bytes memory keyhash = new bytes(32);
        copyBytes(_proof, ledgerProofLength, 32, keyhash, 0);
        if (!(keccak256(keyhash) == keccak256(abi.encodePacked(sha256(abi.encodePacked(_contextNam
                                    e, _queryId)))))) {
            return false;
        }
        bytes memory sig1 = new bytes(uint(uint8(_proof[ledgerProofLength + (32 + 8 + 1 + 32) + 1])) +
                                    2);
        copyBytes(_proof, ledgerProofLength + (32 + 8 + 1 + 32), sig1.length, sig1, 0);
        // Random DS Proof Step 3: We assume sig1 is valid (it will be verified during step 5) and we
verify if '_result' is the _prefix of sha256(sig1)
        if (!matchBytes32Prefix(sha256(sig1), _result, uint(uint8(_proof[ledgerProofLength + 32 + 8])))) {
            return false;
        }
        // Random DS Proof Step 4: Commitment match verification, keccak256(delay, nbytes, unonce,
sessionKeyHash) == commitment in storage.
        // This is to verify that the computed args match with the ones specified in the query.
        bytes memory commitmentSlice1 = new bytes(8 + 1 + 32);
        copyBytes(_proof, ledgerProofLength + 32, 8 + 1 + 32, commitmentSlice1, 0);
        bytes memory sessionPubkey = new bytes(64);
        uint sig2offset = ledgerProofLength + 32 + (8 + 1 + 32) + sig1.length + 65;
        copyBytes(_proof, sig2offset - 64, 64, sessionPubkey, 0);
        bytes32 sessionPubkeyHash = sha256(sessionPubkey);
        if (provable_randomDS_args[_queryId] == keccak256(abi.encodePacked(commitmentSlice1,
sessionPubkeyHash))) { //unonce, nbytes and sessionKeyHash match
            delete provable_randomDS_args[_queryId];
        } else return false;
        // Random DS Proof Step 5: Validity verification for sig1 (keyhash and args signed with the
sessionKey)
        bytes memory tosign1 = new bytes(32 + 8 + 1 + 32);
```

```
        copyBytes(_proof, ledgerProofLength, 32 + 8 + 1 + 32, tosign1, 0);
        if (!verifySig(sha256(tosign1), sig1, sessionPubkey)) {
            return false;
        }
        // Verify if sessionPubkeyHash was verified already, if not.. let's do it!
        if (!provable_randomDS_sessionKeysHashVerified[sessionPubkeyHash]) {
            provable_randomDS_sessionKeysHashVerified[sessionPubkeyHash] =
provable_randomDS_proofVerify__sessionKeyValidity(_proof, sig2offset);
        }
        return provable_randomDS_sessionKeysHashVerified[sessionPubkeyHash];
    }
    /*
     The following function has been written by Alex Beregszaszi (@axic), use it under the terms of the
MIT license
    */
    function copyBytes(bytes memory _from, uint _fromOffset, uint _length, bytes memory _to, uint
_toOffset) internal pure returns (bytes memory _copiedBytes) { //StFt //constant-function-state
        uint minLength = _length + _toOffset;
        require(_to.length >= minLength); // Buffer too small. Should be a better way?
        uint i = 32 + _fromOffset; // NOTE: the offset 32 is added to skip the `size` field of both bytes
variables
        uint j = 32 + _toOffset;
        while (i < (32 + _fromOffset + _length)) { //StFt //costly-loop
            assembly { //StFt //assembly
                let tmp := mload(add(_from, i))
                mstore(add(_to, j), tmp)
            }
            i += 32;
            j += 32;
        }
        return _to;
    }
    /*
     The following function has been written by Alex Beregszaszi (@axic), use it under the terms of the
MIT license
     Duplicate Solidity's ecrecover, but catching the CALL return value
    */
    function safer_ecrecover(bytes32 _hash, uint8 _v, bytes32 _r, bytes32 _s) internal returns (bool
_success, address _recoveredAddress) {
        /*
         We do our own memory management here. Solidity uses memory offset
         0x40 to store the current end of memory. We write past it (as
         writes are memory extensions), but don't update the offset so
         Solidity will reuse it. The memory used here is only needed for
         this context.
         FIXME: inline assembly can't access return values
        */
        bool ret;
        address addr;
        assembly { //StFt //assembly
            let size := mload(0x40)
            mstore(size, _hash)
            mstore(add(size, 32), _v)
            mstore(add(size, 64), _r)
            mstore(add(size, 96), _s)
            ret := call(3000, 1, 0, size, 128, size, 32) // NOTE: we can reuse the request memory because we
```

```
deal with the return code.
            addr := mload(size)
        }
        return (ret, addr);
    }
    /*
     The following function has been written by Alex Beregszaszi (@axic), use it under the terms of the
MIT license
    */
    function ecrecovery(bytes32 _hash, bytes memory _sig) internal returns (bool _success, address
_recoveredAddress) {
        bytes32 r;
        bytes32 s;
        uint8 v;
        if (_sig.length != 65) {
            return (false, address(0));
        }
        /*
         The signature format is a compact form of:
           {bytes32 r}{bytes32 s}{uint8 v}
         Compact means, uint8 is not padded to 32 bytes.
        */
        assembly { //StFt //assembly
            r := mload(add(_sig, 32))
            s := mload(add(_sig, 64))
            /*
             Here we are loading the last 32 bytes. We exploit the fact that
             'mload' will pad with zeroes if we overread.
             There is no 'mload8' to do this, but that would be nicer.
            */
            v := byte(0, mload(add(_sig, 96)))
            /*
              Alternative solution:
              'byte' is not working due to the Solidity parser, so lets
              use the second best option, 'and'
              v := and(mload(add(_sig, 65)), 255)
            */
        }
        /*
         albeit non-transactional signatures are not specified by the YP, one would expect it
         to match the YP range of [27, 28]
         geth uses [0, 1] and some clients have followed. This might change, see:
         https://github.com/ethereum/go-ethereum/issues/2053
        */
        if (v < 27) {
            v += 27;
        }
        if (v != 27 && v != 28) {
            return (false, address(0));
        }
        return safer_ecrecover(_hash, v, r, s);
    }

    function safeMemoryCleaner() internal pure { //StFt //constant-function-state
        assembly { //StFt //assembly
            let fmem := mload(0x40)
```

```
            codecopy(fmem, codesize, sub(msize, fmem))
        }
    }
}
//

// File: contracts\BitsForAI.sol

pragma solidity ^0.5.16;

contract OwnableDelegateProxy { }

contract ProxyRegistry {
    mapping(address => OwnableDelegateProxy) public proxies;
}

contract BitsForAI is ERC721Full, Ownable, ReentrancyGuard, usingProvable {
    using SafeMath for uint; //StFt //safemath
    using Math for uint;

    address artist;

    string private _name = "BitsForAI"; //StFt //private-not-hidedata
    string private _symbol = "BFA"; //StFt //private-not-hidedata
    string private _tokenURI = "https://api.bitsforai.com/tokenid/"; //StFt //private-not-hidedata
    uint private _currentTokenId; //StFt //private-not-hidedata

    uint constant private GWEI = 1000000000; //StFt //private-not-hidedata
    uint private gasPrice      = 4010000000; //many set exactly 4gwei, so adding 0.01 gwei increases
speed much more than expected. //StFt //private-not-hidedata
    uint private gasAmount = 250000; //StFt //private-not-hidedata

    uint constant private BASIS_POINTS = 10000; //StFt //private-not-hidedata

    uint constant private NUM_RANDOM_BYTES_REQUESTED = 2; //The variable `ceiling` should never
be greater than: `(256 ^ NUM_RANDOM_BYTES_REQUESTED) - 1`. //StFt //private-not-hidedata

    address proxyRegistryAddress;

    uint constant mintFeeStart = 8 finney;
    uint constant mintFeeIncrementPer1k = 1 finney;

    uint constant mintEndChanceStart = 0;
    uint constant mintEndChanceAt10k = 10;
    uint constant mintEndChanceAt12k = 25;
    uint constant mintEndChanceAt14k = 50;
    uint constant mintEndChanceAt16k = 75;
    uint constant mintEndChanceAt18k = 100;
    uint constant mintEndChanceAt20k = 1000;

    uint constant jackpotBP = 1000;
    uint constant burnFundBP = 3000;
    uint constant socialBP = 2000;

    bool public isBurnActive = false;
    uint public burnReward = 15 finney;
```

```
bool public isMinting = false;

uint public jackpot;
uint public burnFund;
uint public social;
uint public totalSocialShares;
uint public totalMinted;

uint public finalWeiPerSocialShare;
uint public finalWeiPerJackpotWinner;

address[10] public lastMinters;
uint lastMintersIndex;

uint public endTimer;
uint constant endTimerMaximum  = 1 days;
uint constant endTimerIncrease = 1 minutes;
uint constant endTimerMininum  = 1 hours;

uint public claimTimer;
uint constant claimTime = 1 days;

mapping(address => uint) lastMintersCurrentIndex;

mapping(address => uint) public socialShares;

mapping(bytes32 => uint16) public provableQueryToSeed;
mapping(bytes32 => address) public provableQueryToAddress;
mapping(bytes32 => uint) public provableQueryToTokenId;

mapping(uint => bytes2) public nftBits;
mapping(uint => uint) public nftColorSeed;

event LogMintQuery(address minter, bytes32 queryId, uint seed, uint tokenId);

modifier onlyArtist {
    require(msg.sender == artist);
    _;
}
modifier whenMinting {
    require(isMinting);
    _;
}
modifier whenNotMinting {
    require(!isMinting);
    _;
}

constructor(address _proxyRegistryAddress)  ERC721Full(_name,_symbol) public{
    artist = msg.sender;
    proxyRegistryAddress = _proxyRegistryAddress;
    provable_setProof(proofType_Ledger);
    provable_setCustomGasPrice(gasPrice);
}
```

```
    function mintBits(uint16 seed) public payable nonReentrant whenMinting{
        if(endTimer.min(now)==endTimer){
            //minting timer over.
            EndMinting();
            return;
        }

        require(msg.value.max(getMintFee())==msg.value);

        uint distributable = msg.value.sub(gasAmount.mul(gasPrice));
        jackpot = jackpot.add(GetAmountFromBasisPoints(distributable,jackpotBP));
        burnFund = burnFund.add(GetAmountFromBasisPoints(distributable,burnFundBP));
        social = social.add(GetAmountFromBasisPoints(distributable,jackpotBP));

        totalMinted = totalMinted.add(1);

        if(lastMintersCurrentIndex[msg.sender].max(0)!=0){
            //revival mechanic, if sender is currently already a last minter then swap their place with last
place
            uint swapIndex = lastMintersCurrentIndex[msg.sender].sub(1); //lastminterscurrentindex is 1
indexed, not 0.
            lastMinters[swapIndex] = lastMinters[lastMintersIndex];
            lastMintersCurrentIndex[lastMinters[swapIndex]] = swapIndex.add(1);
        }else{
            //boot last player
            lastMintersCurrentIndex[lastMinters[lastMintersIndex]] = 0;
        }
        lastMinters[lastMintersIndex] = msg.sender;
        lastMintersCurrentIndex[msg.sender] = lastMintersIndex.add(1);//lastminterscurrentindex is 1
indexed, not 0.
        lastMintersIndex = lastMintersIndex.add(1).mod(10);

        endTimer = endTimer.add(endTimerIncrease);
        if(endTimer.max(now.add(endTimerMaximum))==endTimer)
            endTimer = now.add(endTimerMaximum);
        if(endTimer.min(now.add(endTimerMininum))==endTimer)
            endTimer = now.add(endTimerMininum);

        bytes32 queryId = provable_newRandomDSQuery(
            0, //Execution delay
            NUM_RANDOM_BYTES_REQUESTED,
            gasAmount
        );
        emit LogMintQuery(msg.sender, queryId, seed, _currentTokenId);
        provableQueryToSeed[queryId] = seed;
        provableQueryToAddress[queryId] = msg.sender;
        provableQueryToTokenId[queryId] = _currentTokenId;

        _currentTokenId = _currentTokenId.add(1);
    }

    function claimJackpotEarning() public nonReentrant {
        uint index = lastMintersCurrentIndex[msg.sender];
        require(lastMinters[index] == msg.sender);
        uint weiToTransfer = finalWeiPerJackpotWinner;
        delete lastMintersCurrentIndex[msg.sender];
```

```
    delete lastMinters[index];
    msg.sender.transfer(weiToTransfer);
}
function claimSocialEarning() public nonReentrant {
    uint shares = socialShares[msg.sender];
    require(shares.min(1) == 1);
    uint weiToTransfer = finalWeiPerSocialShare.mul(shares);
    delete socialShares[msg.sender];
    msg.sender.transfer(weiToTransfer);
}

function burnForFree(uint tokenId) public{
    require(msg.sender == ownerOf(tokenId));
    _burn(tokenId);
}

function burnForReward(uint tokenId) public{
    require(msg.sender == ownerOf(tokenId));
    require(isBurnActive);
    require(burnFund.max(burnReward)==burnFund);
    _burn(tokenId);
    burnFund = burnFund.sub(burnReward);
    msg.sender.transfer(burnReward);
}

function onlyArtist_adjustGasPrice(uint newGasPrice) public onlyArtist{
    gasPrice = newGasPrice.mul(GWEI);
    provable_setCustomGasPrice(gasPrice);
}
function onlyArtist_adjustGasAmount(uint newGasAmount) public onlyArtist{
    gasAmount = newGasAmount;
}
function onlyArtist_addSocialShares(address addr, uint shares) public onlyArtist{
    totalSocialShares = totalSocialShares.add(shares);
    socialShares[addr] = socialShares[addr].add(shares);
}
function onlyArtist_subSocialShares(address addr, uint shares) public onlyArtist{
    totalSocialShares = totalSocialShares.sub(shares);
    socialShares[addr] = socialShares[addr].sub(shares);
}
function onlyArtist_claimRemainder(uint claimedWei) public onlyArtist{
    require(claimedWei.min(onlyArtist_checkRemainder())==claimedWei);
    msg.sender.transfer(claimedWei);
}
function onlyArtist_checkRemainder() public view onlyArtist returns(uint){
    return address(this).balance.sub(
                jackpot.add(burnFund).add(social)
                );
}
function onlyArtist_postClaimTransfer() public onlyArtist whenNotMinting{
    require(claimTimer.min(now)==claimTimer);
    msg.sender.transfer(address(this).balance);
}
function onlyArtist_changeBaseUIR(string memory newURI) public onlyArtist{
    _tokenURI = newURI;
}
```

```
function onlyArtist_activateBurnFund(uint _burnReward) public onlyArtist{
    burnReward = _burnReward;
    isBurnActive = true;
}
function onlyArtist_deactivateBurnFund() public onlyArtist{
    isBurnActive = false;
}
function onlyArtist_endMinting() public onlyArtist{
    EndMinting();
}
function onlyArtist_startMinting() public onlyArtist whenNotMinting{
    isMinting = true;
    endTimer = now.add(endTimerMaximum);
}
function onlyArtist_legendaryMinting(bytes2 bits, uint level) public onlyArtist{
    require(_currentTokenId.min(9)==_currentTokenId,"Only 10 Legendaries");
    require(level.min(4)==level,"5 levels of legendaries: 0:gold, 1:purple, 2:red, 3:green, 4:blue");
    _safeMint(msg.sender,_currentTokenId);
    nftBits[_currentTokenId] = bits;

    nftColorSeed[_currentTokenId] = level.add(65536); //unique value for Gold legendaries

    _currentTokenId = _currentTokenId.add(1);
    totalMinted = totalMinted.add(1);
}
function onlyArtist_resendForOracleFailure(bytes32 queryId) public onlyArtist{
    uint tokenId = provableQueryToTokenId[queryId];
    uint seed = provableQueryToSeed[queryId];
    address minter = provableQueryToAddress[queryId];
    require(tokenId != 0,"Must have been a real query.");
    delete provableQueryToSeed[queryId];
    delete provableQueryToAddress[queryId];
    delete provableQueryToTokenId[queryId];
    bytes32 newQueryId = provable_newRandomDSQuery(
        0, //Execution delay
        NUM_RANDOM_BYTES_REQUESTED,
        gasAmount
    );
    emit LogMintQuery(minter, newQueryId, seed, tokenId);
    provableQueryToSeed[newQueryId] = uint16(seed);
    provableQueryToAddress[newQueryId] = minter;
    provableQueryToTokenId[newQueryId] = _currentTokenId;

}

//Public utilities
function getMintFee() public view returns (uint){
    return totalMinted.div(1000).mul(mintFeeIncrementPer1k).add(mintFeeStart);
}
function getJackpotChance() public view returns(uint){
    if(totalMinted.min(10000)==totalMinted){
        return mintEndChanceStart;
    }else if(totalMinted.min(12000)==totalMinted){
        return mintEndChanceAt10k;
    }else if(totalMinted.min(14000)==totalMinted){
        return mintEndChanceAt12k;
```

```
    }else if(totalMinted.min(16000)==totalMinted){
        return mintEndChanceAt14k;
    }else if(totalMinted.min(18000)==totalMinted){
        return mintEndChanceAt16k;
    }else if(totalMinted.min(20000)==totalMinted){
        return mintEndChanceAt18k;
    }else{
        return mintEndChanceAt20k;
    }
  }

  /**
   * Provable callback for minting
   */
  function __callback(
    bytes32 _queryId,
    string memory _result,
    bytes memory _proof
  ) public {
    require(msg.sender == provable_cbAddress());

    uint16 seed = provableQueryToSeed[_queryId];
    address minterAddr = provableQueryToAddress[_queryId];
    uint tokenId = provableQueryToTokenId[_queryId];

    nftBits[tokenId] = bytesToBytes2(bytes(_result),0) ^ bytes2(seed);
    uint rand = uint(
          keccak256(abi.encodePacked(_result)) ^ blockhash(block.number-1) ^ bytes32(uint(seed))
        );
    nftColorSeed[tokenId] = rand.mod(65535);

    uint roll = rand.mod(BASIS_POINTS).add(1);
    if(roll.min(getJackpotChance())==roll){
        //Jackpot, game over.
        EndMinting();
    }

    _safeMint(minterAddr,tokenId);

    delete provableQueryToSeed[_queryId];
    delete provableQueryToAddress[_queryId];
    delete provableQueryToTokenId[_queryId];
  }

  /**
 * Override isApprovedForAll to whitelist user's OpenSea proxy accounts to enable gas-less listings.
 */
function isApprovedForAll(
  address owner,
  address operator
)
  public
  view
  returns (bool)
{
  // Whitelist OpenSea proxy contract for easy trading.
```

```
  ProxyRegistry proxyRegistry = ProxyRegistry(proxyRegistryAddress);
  if (address(proxyRegistry.proxies(owner)) == operator) {
      return true;
  }

  return super.isApprovedForAll(owner, operator);
}



  //ERC721 Implementation

  function name() external view returns (string memory){
      return _name;
  }
  function symbol() external view returns (string memory){
      return _symbol;
  }
  function tokenURI(uint tokenId) external view returns(string memory){
      return strConcat(_tokenURI,uint2str(tokenId));
  }





  //Utilities
  function EndMinting() internal {
      isMinting = false;

      if(totalSocialShares == 0){
          finalWeiPerSocialShare = 0;
      }else{
          finalWeiPerSocialShare = social.div(totalSocialShares);
      }

      claimTimer = now.add(claimTime);
      finalWeiPerJackpotWinner = jackpot.div(10);

  }
  function GetAmountFromBasisPoints(uint amt, uint bp) internal pure returns (uint) {
      return amt.mul(bp).div(BASIS_POINTS);
  }
  function bytesToBytes2(bytes memory b, uint offset) private pure returns (bytes2) {
    bytes2 out;

    for (uint i = 0; i < 2; i++) {
      out |= bytes2(b[offset + i] & 0xFF) >> (i * 8);
    }
    return out;
  }
  function uint2str(uint _i) internal pure returns (string memory _uintAsString) {
      if (_i == 0) {
          return "0";
      }
```

```
        uint j = _i;
        uint len;
        while (j != 0) {
            len++;
            j /= 10;
        }
        bytes memory bstr = new bytes(len);
        uint k = len - 1;
        while (_i != 0) {
            bstr[k--] = byte(uint8(48 + _i % 10));
            _i /= 10;
        }
        return string(bstr);
    }
}
```

0x04 Contract Audit Details

4.1 abiencoderv2-array

Vulnerability description

The solc 0.4.7-0.5.10 version contains a compiler error that causes the incorrect use of the ABI encoder.

Applicable scenarios

```
contract A {
    uint[2][3] bad_arr = [[1, 2], [3, 4], [5, 6]];
    /* Array of arrays passed to abi.encode is vulnerable */
    function bad() public {
        bytes memory b = abi.encode(bad_arr);
    }
}
```

When the compiler version is 0.4.7-0.5.10, because abi.encode (bad_arr) is used in bad(), calling bad() will incorrectly encode the array as [[1, 2], [2, 3], [3, 4]], and cause unexpected behavior.

Audit results: 【Pass】

Security advice: none.

4.2 array-by-reference

Vulnerability description

Detect the function (reference) that passes the storage array to the storage function.

Applicable scenarios

```
contract Memory {
    uint[1] public x; // storage
    function f() public {
        f1(x); // update x
        f2(x); // do not update x
```

```
    }
    function f1(uint[1] storage arr) internal { // by reference
        arr[0] = 1;
    }
    function f2(uint[1] arr) internal { // by value
        arr[0] = 2;
    }
}
```

Bob calls f(). It stands to reason that at the end of the call, Bob x[0] is 2, but it is 1. Therefore, Bob used the contract incorrectly.

Audit results: 【Pass】

Security advice: none.

4.3 multiple-constructors

Vulnerability description

Detect multiple constructor definitions in the same contract (using old and new schemes).

Applicable scenarios

```
contract A {
    uint x;
    constructor() public {
        x = 0;
    }
    function A() public {
        x = 1;
    }
    function test() public returns(uint) {
        return x;
    }
}
```

In Solidity 0.4.22, a contract with two constructor schemes will be compiled at the same time. The first constructor will take precedence over the second, which may be unexpected. Therefore, special attention should be paid to Solidity 0.4.22.

Audit results: 【Pass】

Security advice: none.

4.4 names-reused

Vulnerability description

If a code base has two contracts with similar names, the compilation artifact will not contain one of the contracts with duplicate names.

Applicable scenarios

```
contract B{
    constructor() {
```

```
    }
    function ceshi1() {}
}
contract B{
    constructor() {
    }
    function ceshi2() {}
}
```

Bob's code base has two contracts called "ERC20". When the code base is running, only one of the two contracts will be compiled in "build/contracts". As a result, the second contract cannot be analyzed.

Audit results: 【Pass】

Security advice: none.

4.5 public-mappings-nested

Vulnerability description

Before Solidity 0.5, public maps with nested structures (struct sets of struct) returned incorrect values.

Applicable scenarios

```
contract TestNestedStructInMapping {
    // The struct that is nested.
    struct structNested {
        uint dummy;
    }
    // The struct that holds the nested struct.
    struct structMain {
        structNested gamePaymentsSummary;
    }
    // The map that maps a game ID to a specific game.
    mapping(uint256 => structMain) public s_mapOfNestedStructs;
}
```

It does work in 0.4.25. However, adding another variable to the structure still does not make 0.4.25 error, but the generated code is wrong-it only returns 32 bytes instead of 64 bytes. Bob interacts with a publicly mapped contract with a nested structure. The value returned by the mapping is incorrect (64 bytes are returned as 32 bytes), destroying Bob's use.

Audit results: 【Pass】

Security advice: none.

4.6 rtlo

Vulnerability description

Malicious actors can use Unicode characters (U+202E) overlaid from right to left to force the rendering of RTL text and enable users to achieve the true intention of confusing the contract.

Applicable scenarios

```
contract Token
{
   address payable o; // owner
   mapping(address => uint) tokens;
   function withdraw() external returns(uint)
   {
      uint amount = tokens[msg.sender];
      address payable d = msg.sender;
      token/*noitanitsed*/ d, o/*□□□
            /*value */, amount);
   }
   function _withdraw(address payable fee_receiver, address payable destination, uint value) internal
   {
      fee_receiver.transfer(1);
      destination.transfer(value);
   }
}
```

When calling `_withdraw`, `Token` uses right-to-left overwriting characters (the small box above contains overwriting characters). As a result, the fee was erroneously sent to `msg.sender`, and the token balance was sent to the owner.

Audit results: 【Pass】

Security advice: none.

4.7 shadowing-state

Vulnerability description

Robustness allows ambiguous naming of state variables when using inheritance. When there are multiple definitions at the contract and functional level, state variable hiding may also occur within a single contract.

Applicable scenarios

```
contract BaseContract{
   address owner;
   modifier isOwner(){
      require(owner == msg.sender);
      _;
   }
}
contract DerivedContract is BaseContract{
   address owner;
   constructor(){
      owner = msg.sender;
```

```
  }
  function withdraw() isOwner() external{
    msg.sender.transfer(this.balance);
  }
}
```

The `owner` of `BaseContract` will not be specified, and the modifier `isOwner` has no effect. '''

Audit results:【Pass】

Security advice: none.

4.8 suicidal

Vulnerability description

Due to lack of access control or insufficient access control, malicious parties can self-destruct the contract. Calling selfdestruct/suicide lacks protection.

Applicable scenarios

```
contract Suicidal{
  function kill() public{
    selfdestruct(msg.sender);
  }
}
```

Bob calls the "kill" function and breaks the contract.

Audit results:【Pass】

Security advice: none.

4.9 uninitialized-state

Vulnerability description

Uninitialized state variables can lead to intentional or unintentional vulnerabilities.

Applicable scenarios

```
contract Uninitialized{
  address destination;
  function transfer() payable public{
    destination.transfer(msg.value);
  }
}
```

Bob calls "transfer". As a result, the ether is sent to the address "0x0" and is lost.

Audit results:【Pass】

Security advice: none.

4.10 uninitialized-storage

Vulnerability description

Local variables are not initialized. The initialized storage variable will be used as a reference to the first state variable, and key variables can be overwritten.

Applicable scenarios

```
contract Uninitialized{
   address owner = msg.sender;
   struct St{
      uint a;
   }
   function func() {
      St st;
      st.a = 0x0;
   }
}
```

Bob calls `func`. As a result, "owner" is overwritten with "0".

Audit results: 【Pass】

Security advice: none.

4.11 unprotected-upgrade

Vulnerability description

Detect logical contracts that can be broken.

Applicable scenarios

```
contract Buggy is Initializable{
   address payable owner;

   function initialize() external initializer{
      require(owner == address(0));
      owner = msg.sender;
   }
   function kill() external{
      require(msg.sender == owner);
      selfdestruct(owner);
   }
}
```

Buggy is a renewable contract. Anyone can call initialize on the logic contract and destroy the contract.

Audit results: 【Pass】

Security advice: none.

4.12 visibility

Vulnerability description

The default function visibility level in the contract is public, and in interfaces-external, the default visibility level of state variables is internal. In the contract, the fallback function can be external or public. In the interface, all functions should be declared as external functions. Clearly define function visibility to prevent confusion. Specifically, before version 0.5.0: look for fallback functions without explicit visibility declaration and delete them, look for fallback functions that are neither external nor public, look for constructors with external or private visibility, in the interface Find internal and private functions; after version 0.5.0: Find non-external functions in the interface.

Applicable scenarios

```
pragma solidity ^0.4.24;
interface D {
    function foo() private;
    function foo1() public;
}
contract C {
    address kk;
    constructor() private {
    }
    function aa() public{
    }
    function() {
    }
}
```

Audit results: 【Pass】

Security advice: none.

4.13 redundant-fallback

Vulnerability description

Starting from Solidity 0.4.0, contracts that do not have a rollback function will automatically restore the payment, making the payment refusal to roll back redundant. But below 0.4, there is no fallback function that is prone to reentrance loopholes, which is very dangerous.

Applicable scenarios

```
pragma solidity 0.3.24;
contract Crowdsale {
}
```

A malicious user writes a fallback function to attack the contract.

Audit results: 【Pass】

Security advice: none.

4.14 arbitrary-send

Vulnerability description

The call to the function that sends Ether to an arbitrary address has not been reviewed.

Applicable scenarios

```
contract ArbitrarySend{
   address destination;
   function setDestination(){
      destination = msg.sender;
   }
   function withdraw() public{
      destination.transfer(this.balance);
   }
}
```

Bob calls setDestination and withdraw, and as a result, he withdraws the balance of the contract.

Audit results: 【Pass】

Security advice: none.

4.15 continue-in-loop

Vulnerability description

continue will cause the loop judgment condition to fail to increment, resulting in an infinite loop.

Applicable scenarios

```
contract C {
   function f(uint a, uint b) public{
      uint a = 0;
      do {
            continue;
            a++;
      } while(a<10);
   }
}
```

Audit results: 【Pass】

Security advice: none.

4.16 controlled-array-length

Vulnerability description

Detect direct allocation of array length.

Applicable scenarios

```
contract A {
   uint[] testArray; // dynamic size array
   function f(uint usersCount) public {
```

```
        // ...
        testArray.length = usersCount;
        // ...
    }
    function g(uint userIndex, uint val) public {
        // ...
        testArray[userIndex] = val;
        // ...
    }
}
```

Contract storage/state variables are indexed by 256-bit integers. Users can set the array length to 2 ** 256-1 to index all storage slots. In the above example, you can call function f to set the length of the array, and then call function g to control any storage slots needed. Please note that the storage slots here are indexed by the hash of the indexer. Nonetheless, all storage will still be accessible and can be controlled by an attacker.

Audit results: 【Pass】

Security advice: none.

4.17 controlled-delegatecall

Vulnerability description

Delegate the call or call code to an address controlled by the user. The address of Delegatecall is not necessarily trusted, it is still a problem of access control, and the address is not checked.

Applicable scenarios

```
contract Delegatecall{
    function delegate(address to, bytes data){
        to.delegatecall(data);
    }
}
```

Bob calls `delegate` and delegates the execution of the malicious contract to him. As a result, Bob withdraws the funds from the contract and destroys the contract.

Audit results: 【Pass】

Security advice: none.

4.18 incorrect-constructor

Vulnerability description

The constructor is a special function and can only be called once during contract creation. They usually perform key privileged operations, such as setting up contract owners. Before Solidity 0.4.22, the only way to define a

constructor was to create a function with the same name as the contract classthat contained it. If its name does not exactly match the contract name, thefunction intended to be the constructor will become a normal callablefunction. This behavior can sometimes cause security issues, especially whenthe smart contract code is reused with other names but the name of theconstructor has not been changed accordingly.

Applicable scenarios

```
contract Incorrectconstructor {
   address owner;
   function Incorrectconstructo() {
      owner = msg.sender;
   }
   modifier ifowner()
   {
      require(msg.sender == owner);
      _;
   }
   function withdrawmoney() ifowner {
      msg.sender.transfer(address(this).balance);
   }
}
```

Audit results: 【Pass】

Security advice: none.

4.19 parity-multisig-bug

Vulnerability description

Multi-signature vulnerability. Hackers can use the initWallet function to call the initMultiowned function to obtain the identity of the contract owner.

Applicable scenarios

```
contract WalletLibrary_bad is WalletEvents {
   function initWallet(address[] _owners, uint _required, uint _daylimit) {
      initDaylimit(_daylimit);
      initMultiowned(_owners, _required);
   } // kills the contract sending everything to `_to`.
   function initMultiowned(address[] _owners, uint _required) {
      m_numOwners = _owners.length + 1;
      m_owners[1] = uint(msg.sender);
      m_ownerIndex[uint(msg.sender)] = 1;
      for (uint i = 0; i < _owners.length; ++i)
      {
         m_owners[2 + i] = uint(_owners[i]);
         m_ownerIndex[uint(_owners[i])] = 2 + i;
      }
      m_required = _required;
   }
}
```

Audit results: 【Pass】

Security advice: none.

4.20 reentrancy-eth

Vulnerability description

A reentrancy error was detected. This is the reentry of ether. Through re-entry, the account balance can be maliciously withdrawn, resulting in losses. Do not report re-reporting that does not involve Ether (please refer to "reentrancy-no-eth")

Applicable scenarios

```
function withdrawBalance(){
   // send userBalance[msg.sender] Ether to msg.sender
   // if mgs.sender is a contract, it will call its fallback function
   if( ! (msg.sender.call.value(userBalance[msg.sender])() ) ){
      throw;
   }
   userBalance[msg.sender] = 0;
}
```

Bob used the reentrance vulnerability to call `withdrawBalance` multiple times and withdrew more than he originally deposited into the contract.

Audit results: 【Pass】

Security advice: none.

4.21 storage-array

Vulnerability description

The solc 0.4.7-0.5.10 version contains a compiler error that causes the values in the signed integer array to be incorrect.

Applicable scenarios

```
contract A {
   int[3] ether_balances; // storage signed integer array
   function bad0() private {
      // ...
      ether_balances = [-1, -1, -1];
      // ...
   }
}
```

bad0() uses a (storage-allocated) signed integer array state variable to store the ether balance of three accounts. -1 should indicate uninitialized values, but a Solidity error made them 1 and these values can be used by the account.

Audit results: 【Pass】

Security advice: none.

4.22 weak-prng

Vulnerability description

The PRNG is weaker due to the modulus of block.timestamp, now or blockhash. These may be affected by miners to some extent, so they should be avoided.

Applicable scenarios

```
contract Game {
   uint reward_determining_number;
   function guessing() external{
    reward_determining_number = uint256(block.blockhash(10000)) % 10;
   }
}
```

Eve is a miner. Eve calls guessing() and reorders the blocks containing transactions. As a result, Eve won the game.

Audit results: 【Pass】

Security advice: none.

4.23 assert-violation

Vulnerability description

The Solidity assert() function is used to declare invariants. Normally running code will never reach a failed assert statement. An attainable assertion may mean one of two things: 1. There is an error in the contract that allows it to enter an invalid state. 2. The assert statement is used incorrectly, for example to verify input. ; CWE-670: Always incorrect control flow implementation. The Assert condition can be false, and the definition here is wider. It is best to use requere. Generally used at the end of the function to verify whether there is overflow or something.

Applicable scenarios

```
contract Assertviolation [
   function bad(uint a, uint b){
      assert(a>b);
   }
}
```

Audit results: 【Pass】

Security advice: none.

4.24 constructor-return

Vulnerability description

The return statement is used in the constructor of the contract. With return, the deployment process will be different from the intuitive process. For example, the deployed bytecode may not include functions implemented in the source code.

Applicable scenarios

```
pragma solidity 0.4.24;
contract HoneyPot {
   bytes internal constant ID = hex"6020341460085700 5B60008080803031335AF100";
   constructor () public payable {
      bytes memory contract_identifier = ID;
      assembly { return(add(0x20, contract_identifier), mload(contract_identifier)) }
   }
   function withdraw() public payable {
      require(msg.value >= 1 ether);
      msg.sender.transfer(address(this).balance);
   }
}
```

Audit results: 【Pass】

Security advice: none.

4.25 enum-conversion

Vulnerability description

Detect out-of-range enumeration conversion (solc <0.4.5).

Applicable scenarios

```
pragma solidity 0.4.2;
contract Test{
   enum E{a}
   function bug(uint a) public returns(E){
      return E(a);
   }
}
```

An attacker can trigger unexpected behavior by calling bug(1).

Audit results: 【Pass】

Security advice: none.

4.26 erc1155-interface

Vulnerability description

The return value of the "ERC1155" function is incorrect. Interacting with these functions, the contract of solidity version> 0.4.22 will not be executed because of the lack of return value.

Applicable scenarios

```
contract Token{
   function balanceOf(address _owner, uint256 _id) external view returns (bool);
```

```
  //...
}
```

Token.balanceOf does not return the expected uint256. Bob deploys the token. Alice creates a contract to interact with, but assumes that the interface is the correct `ERC1155`. Alice's contract cannot interact with Bob's contract.

Audit results: 【Pass】

Security advice: none.

4.27 erc1410-interface

Vulnerability description

The return value of the "ERC1410" function is incorrect. Interacting with these functions, the contract of solidity version> 0.4.22 will not be executed because of the lack of return value.

Applicable scenarios

```
contract Token{
  function isOperator(address _operator, address _tokenHolder) external view returns (uint256);
  //...
}
```

Token.isOperator does not return the expected boolean value. Bob deploys the token. Alice creates a contract to interact with, but uses the correct `ERC1410` interface implementation. Alice's contract cannot interact with Bob's contract.

Audit results: 【Pass】

Security advice: none.

4.28 erc20-interface

Vulnerability description

The return value of the "ERC20" function is incorrect. Interacting with these functions, the contract of solidity version> 0.4.22 will not be executed because of the lack of return value.

Applicable scenarios

```
contract Token{
  function transfer(address to, uint value) external;
  //...
}
```

Token.transfer does not return the expected boolean value. Bob deploys the token. Alice creates a contract to interact with, but uses the correct `ERC20` interface implementation. Alice's contract cannot interact with Bob's contract.

Audit results: 【Pass】

Security advice: none.

4.29 erc223-interface

Vulnerability description

The return value of the "ERC223" function is incorrect. Interacting with these functions, the contract of solidity version> 0.4.22 will not be executed because of the lack of return value.

Applicable scenarios

```
contract Token{
   function name() constant returns (uint _name);
   //...
}
```

Token.name does not return the expected boolean value. Bob deploys the token. Alice creates a contract to interact with, but uses the correct `ERC223` interface implementation. Alice's contract cannot interact with Bob's contract.

Audit results:【Pass】

Security advice: none.

4.30 erc621-interface

Vulnerability description

The return value of the "ERC621" function is incorrect. Interacting with these functions, the contract of solidity version> 0.4.22 will not be executed because of the lack of return value.

Applicable scenarios

```
contract Token{
   function decreaseSupply(uint value, address from) external;
   //...
}
```

Token.decreaseSupply does not return the expected boolean value. Bob deploys the token. Alice creates a contract to interact with, but uses the correct `ERC621` interface implementation. Alice's contract cannot interact with Bob's contract.

Audit results:【Pass】

Security advice: none.

4.31 erc721-interface

Vulnerability description

The return value of the "ERC721" function is incorrect. Interacting with these functions, the contract of solidity version> 0.4.22 will not be

executed because of the lack of return value.

Applicable scenarios

```
contract Token{
   function ownerOf(uint256 _tokenId) external view returns (bool);
   //...
}
```

Token.ownerOf does not return the expected boolean value. Bob deploys the token. Alice creates a contract to interact with, but uses the correct `ERC721` interface implementation. Alice's contract cannot interact with Bob's contract.

Audit results: 【Pass】

Security advice: none.

4.32 erc777-interface

Vulnerability description

The return value of the "ERC777" function is incorrect. Interacting with these functions, the contract of solidity version> 0.4.22 will not be executed because of the lack of return value.

Applicable scenarios

```
contract Token{
   function defaultOperators() public view returns (address);
   //...
}
```

Token.defaultOperators does not return the expected boolean value. Bob deploys the token. Alice creates a contract to interact with, but uses the correct `ERC777` interface implementation. Alice's contract cannot interact with Bob's contract.

Audit results: 【Pass】

Security advice: none.

4.33 erc875-interface

Vulnerability description

The return value of the "ERC875" function is incorrect. Interacting with these functions, the contract of solidity version> 0.4.22 will not be executed because of the lack of return value.

Applicable scenarios

```
contract Token{
   function balanceOf(address _owner) public view returns (string _balances);
   //...
}
```

Token.balanceOf does not return the expected boolean value. Bob deploys the token. Alice creates a contract to interact with, but uses the correct `ERC875` interface implementation. Alice's contract cannot interact with Bob's contract.

Audit results: 【Pass】

Security advice: none.

4.34 incorrect-equality

Vulnerability description

Using strict equality (== and !=), an attacker can easily manipulate these equality. Specifically: the opponent can forcefully send Ether to any address through selfdestruct() or through mining, thereby invalidating the strict judgment.

Applicable scenarios

```
contract Crowdsale{
    function fund_reached() public returns(bool){
        return this.balance == 100 ether;
    }
}
```

Crowdsale relies on fund_reached to know when to stop the sale of tokens. Bob sends 0.1 ether. As a result, fund_reached is always false, and crowdsale is always true.

Audit results: 【Pass】

Security advice: none.

4.35 incorrect-signature

Vulnerability description

In Solidity, the definition of a function signature is: a basic prototype canonical expression without a data position specifier, that is, a function name with a bracketed parameter type list. The parameter types are separated by a comma without spaces. This means uint256 and int256 should be used instead of uint or int. For example: in bytes4 (keccak256()), signatrue should be unit256 or int256, not uint or int, otherwise the length is not enough.

Applicable scenarios

```
pragma solidity ^0.5.1;
contract Signature {
    function callFoo(address addr, uint value) public returns (bool) {
        bytes memory data = abi.encodeWithSignature("foo(uint)", value);
        (bool status, ) = addr.call(data);
        return status;
```

```
  }
}
```

Audit results: 【Pass】

Security advice: none.

4.36 locked-ether

Vulnerability description

A contract programmed to receive ether (with the payable logo) should implement the method of withdrawing ether, that is, call transfer (recommended), send or call.value at least once.

Applicable scenarios

```
pragma solidity 0.4.24;
contract Locked{
  function receive() payable public{}
}
```

All Ether sent to "Locked" will be lost.

Audit results: 【Pass】

Security advice: none.

4.37 mapping-deletion

Vulnerability description

Deleting in a structure that contains a mapping will not delete the mapping (see Solidity documentation). The remaining data can be used to break the contract.

Applicable scenarios

```
struct BalancesStruct{
  address owner;
  mapping(address => uint) balances;
}
mapping(address => BalancesStruct) public stackBalance;
function remove() internal{
   delete stackBalance[msg.sender];
}
```

Remove() deletes one item of stackBalance. The mapped balances will never be deleted, so remove() will not work properly.

Audit results: 【Pass】

Security advice: none.

4.38 shadowing-abstract

Vulnerability description

Detect hidden state variables in abstract contracts. Unlike shadowing-state, hidden state variables that are not used in the parent contract are detected here.

Applicable scenarios

```
contract BaseContract{
   address owner;
}
contract DerivedContract is BaseContract{
   address owner;
}
```

The owner of BaseContract is in the hidden variable of DerivedContract.

Audit results: 【Pass】

Security advice: none.

4.39 tautology

Vulnerability description

Detecting tautology or contradictory expressions means that if, while, require, and assert conditions are always true or false.

Applicable scenarios

```
contract A {
   function f(uint x) public {
      // ...
      if (x >= 0) { // bad -- always true
         // ...
      }
      // ...
   }
   function g(uint8 y) public returns (bool) {
      // ...
      return (y < 512); // bad!
      // ...
   }
}
```

x is uint256, so x>= 0 will always be true. y is uint8, so y <512 will always be true.

Audit results: 【Pass】

Security advice: none.

4.40 default-return-value

Vulnerability description

If a function is declared to have a return value, but no return value is given to it in the end, a default return value will be generated, and the

default return value and the return value after actual execution may bedifferent.

Applicable scenarios

```
contract C{
   function bad_return() public returns(bool flag){
       address aa = msg.sender;
   }
   function bad_return1() public returns(bool){
       address aa = msg.sender;
   }
}
```

Audit results: 【Pass】

Security advice: none.

4.41 boolean-cst

Vulnerability description

Detect abuse of Boolean constants. Bool variable is used incorrectly, here is the operation of bool variable.

Applicable scenarios

```
contract A {
   function f(uint x) public {
      // ...
      if (false) { // bad!
          // ...
      }
      // ...
   }
   function g(bool b) public returns (bool) {
      // ...
      return (b || true); // bad!
      // ...
   }
}
```

The Boolean constants in the code have very few legal uses. Other uses (as conditions in complex expressions) indicate the persistence of errors or error codes.

Audit results: 【Pass】

Security advice: none.

4.42 constant-function-state

Vulnerability description

Functions declared as constant/pure/view will change their state. Constant/pure/view was not enforced before Solidity 0.5. Starting from Solidity 0.5, calls to constant/pure/view functions use the STATICCALL

opcode, which is restored when the state is modified. As a result, calls toincorrectly labeled functions may catch contracts compiled with Solidity 0.5.

Applicable scenarios: 【Medium:8】

For this pattern, the specific problems in the contract are as follows:

(Medium)function init(buffer memory _buf, uint _capacity) internal pure { (tests/time_2/BitsForAI.sol)#1399

(Medium)function append(buffer memory _buf, bytes memory _data) internal pure returns (buffer memory _buffer) { (tests/time_2/BitsForAI.sol)#1433

(Medium)function append(buffer memory _buf, uint8 _data) internal pure { (tests/time_2/BitsForAI.sol)#1471

(Medium)function appendInt(buffer memory _buf, uint _data, uint _len) internal pure returns (buffer memory _buffer) { (tests/time_2/BitsForAI.sol)#1492

(Medium)function copyBytes(bytes memory _from, uint _fromOffset, uint _length, bytes memory _to, uint _toOffset) internal pure returns (bytes memory _copiedBytes) { (tests/time_2/BitsForAI.sol)#2558

(Medium)function safeMemoryCleaner() internal pure { (tests/time_2/BitsForAI.sol)#2645

(Medium)function isContract(address account) internal view returns (bool) { (tests/time_2/BitsForAI.sol)#427

(Medium)function getCodeSize(address _addr) view internal returns (uint _size) { (tests/time_2/BitsForAI.sol)#2167

Security advice

Ensure that the properties of the contract compiled before Solidity 0.5.0 are correct.

4.43 divide-before-multiply

Vulnerability description

Entities only support integers, so division is often truncated; performing multiplication before division can sometimes avoid loss of precision, and entity integer division may be truncated. As a result, performing multiplication before division may reduce accuracy.

Applicable scenarios

```
contract A {
   function f(uint n) public {
      coins = (oldSupply / n) * interest;
   }
}
```

If n is greater than oldSupply, coins will be zero. For example, oldSupply = 5; n = 10, interest = 2, coins will be zero. If (oldSupply * interest / n) is used, the coin will be 1. In general, it is usually best to rearrange the arithmetic to perform the multiplication before the division, unless a smaller type of restriction makes this operation dangerous.

Audit results: 【Pass】

Security advice: none.

4.44 erc20-approve

Vulnerability description

The approve function of ERC-20 is vulnerable to attack. Using a preemptive attack, the approved tokens can be spent before changing the quota value. This attack is also a type of transaction sequence dependence (TOD).

Applicable scenarios

```
pragma solidity ^0.4.5;
contract StandardToken is ERC20, BasicToken {
   ...
   function approve(address _spender, uint256 _value) public returns (bool) {
      allowed[msg.sender][_spender] = _value;
      Approval(msg.sender, _spender, _value);
      return true;
   }
   ...
}
```

Audit results: 【Pass】

Security advice: none.

4.45 function-problem

Vulnerability description

The function will always end in an abnormal state such as revert(), and it cannot return after the normal execution, indicating that there is a problem with the function design.

Applicable scenarios

```
contract Functionproblem {
   address owner;
   function bad() { //bad
      revert();
```

```
  }
}
```

Audit results: 【Pass】

Security advice: none.

4.46 mul-var-len-arguments

Vulnerability description

abi.encodePacked() In some cases, using multiple variable length parameters may cause hash collisions. Since abi.encodePacked() all elements are packed in order, regardless of whether they belong to an array, you can move elements between arrays, and as long as all elements are in the same order, it will return the same encoding. In the case of signature verification, an attacker can effectively bypass authorization by modifying the position of the element in the previous function call, thereby exploiting this vulnerability. Replay authentication through abi.encodePacked(), because in encodePacked(a,b), if both a and b are variable-length arrays, you can add the last one of a to the first one of b, so that the total order is not Change, you can generate the same value.

Applicable scenarios

```
contract Mulvarlenarguments {
  function addUsers(
    address[] calldata admins,
    address[] calldata regularUsers,
    bytes calldata signature
  )
    external
  {
    bytes32 hash = keccak256(abi.encodePacked(admins, regularUsers));
    address signer = hash.toEthSignedMessageHash().recover(signature);
  }
}
```

Audit results: 【Pass】

Security advice: none.

4.47 reentrancy-no-eth

Vulnerability description

Check that there is reentrancy (there is a situation that reads first, writes later), but there is no transfer of eth.

Applicable scenarios

```
function bug(){
  require(not_called);
  if( ! (msg.sender.call() ) ){
```

```
        throw;
    }
    not_called = False;
}
```

Audit results: 【Pass】

Security advice: none.

4.48 reused-constructor

Vulnerability description

Check whether the same basic constructor is called with parameters from two different positions in the same inheritance hierarchy.

Applicable scenarios

```
pragma solidity ^0.4.0;
contract A{
    uint num = 5;
    constructor(uint x) public{
        num += x;
    }
}
contract B is A{
    constructor() A(2) public { /* ... */ }
}
contract C is A {
    constructor() A(3) public { /* ... */ }
}
contract D is B, C {
    constructor() public { /* ... */ }
}
contract E is B {
    constructor() A(1) public { /* ... */ }
}
```

The constructor of A is called multiple times in D and E: D inherits from B and C, and both constitute A. E only inherits from B, but B and E construct A.

Audit results: 【Pass】

Security advice: none.

4.49 tx-origin

Vulnerability description

If a legitimate user interacts with a malicious contract, the protection based on tx.origin will be abused by the malicious contract.

Applicable scenarios

```
contract TxOrigin {
    address owner = msg.sender;
    function bug() {
        require(tx.origin == owner);
    }
```

```
}
```

Bob is the owner of TxOrigin. Bob calls Eve's contract. Eve's contract is called TxOrigin and bypasses the protection of tx.origin.

Audit results:【Pass】

Security advice: none.

4.50 typographical-error

Vulnerability description

For example, when the intent of a defined operation is to sum a number with a variable (+=), but a typographical error is accidentally defined in the wrong way (= +), a typographical error occurs, and this happens to be effective Operator. Instead of calculating the sum, initialize the variables again.

Applicable scenarios

```
pragma solidity ^0.4.24;
contract TypoOneCommand {
    uint numberOne = 1;
    string numberstring = "";
    function alwaysOne() public {
        numberOne =+ 1; //bad
    }
    function alwaysOne_bad() public {
        numberOne =- 1; //bad
    }
}
```

Audit results:【Pass】

Security advice: none.

4.51 unchecked-lowlevel

Vulnerability description

The low-level call to the external contract failed, and the return value was not judged. When sending ether at the same time, please check the return value and handle the error.

Applicable scenarios

```
contract MyConc{
    function my_func(address payable dst) public payable{
        dst.call.value(msg.value)("");
    }
}
```

The return value of the low-level call is not checked, so if the call fails, the ether will be locked in the contract. If you use low-level calls to block block

operations, consider logging the failed calls.

Audit results:【Pass】

Security advice: none.

4.52 unchecked-send

Vulnerability description

Similar to unchecked-lowlevel, it is explained here that the return value of send and Highlevelcall is not checked.

Applicable scenarios

```
contract MyConc{
  function my_func(address payable dst) public payable{
    dst.send(msg.value);
  }
}
```

The return value of send is not checked, so if the send fails, the ether will be locked in the contract. If you use send to prevent block operations, please consider logging failed send.

Audit results:【Pass】

Security advice: none.

4.53 uninitialized-local

Vulnerability description

Check the local variables that are not initialized.

Applicable scenarios

```
contract Uninitialized is Owner{
  function withdraw() payable public onlyOwner{
    address to;
    to.transfer(this.balance)
  }
}
```

Bob calls transfer. As a result, all ether was sent to the address "0x0" and lost.

Audit results:【Pass】

Security advice: none.

4.54 unused-return

Vulnerability description

The return value of the call is not stored in a local variable or state variable, that is, the calling function may not have any effect.

Applicable scenarios

```
contract MyConc{
   using SafeMath for uint;
   function my_func(uint a, uint b) public{
      a.add(b);
   }
}
```

MyConc will call SafeMath's add, but will not store the result in a. As a result, the calculation has no effect.

Audit results: 【Pass】

Security advice: none.

4.55 writeto-arbitrarystorage

Vulnerability description

The data of the smart contract (for example, the owner of the storage contract) is permanently stored in a storage location (ie, key or address) at the EVM level. The contract is responsible for ensuring that only authorized users or contract accounts can write to sensitive storage locations. If an attacker can write to any storage location of the contract, the authorization check can be easily bypassed. This may allow an attacker to destroy the storage space. For example, by overwriting a field that stores the address of the contract owner.

Applicable scenarios

```
contract Map {
   address public owner;
   uint256[] map;
   function set(uint256 key, uint256 value) public {
      if (map.length <= key) {
         map.length = key + 1;
      }
      map[key] = value;
   }
}
```

Audit results: 【Pass】

Security advice: none.

4.56 integer-overflow

Vulnerability description

When an arithmetic operation reaches the maximum or minimum size of the type, overflow/underflow will occur. For example, if a number is stored in the uint8 type, it means that the number is stored as an 8-bit unsigned number, ranging from 0 to 2^8-1. In computer programming, when an arithmetic

operation attempts to create a value, an integer overflow occurs, and thevalue can be represented by a given number of bits-greater than the maximumvalue or less than the minimum value.

Applicable scenarios

```
contract Intergeroverflow{
   function bad() {
      uint a;
      uint b;
      uint c = a + b;
   }
}
```

Audit results: 【Pass】

Security advice: none.

4.57 costly-loop

Vulnerability description

Ethereum is a very resource-constrained environment. The price of each calculation step is several orders of magnitude higher than the price of the centralized provider. In addition, Ethereum miners impose limits on the total amount of natural gas consumed in the block. If array.length is large enough, the function exceeds the gas limit, and the transaction that calls the function will never be confirmed. If external participants influence array.length, this will become a security issue.

Applicable scenarios: 【Medium:13】

For this pattern, the specific problems in the contract are as follows:

(Medium)for (uint i = 0; i < minLength; i ++) {
(tests/time_2/BitsForAI.sol)#2216

(Medium)for (uint i = 0; i < h.length; i++) {
(tests/time_2/BitsForAI.sol)#2241

(Medium)for (i = 0; i < _ba.length; i++) {
(tests/time_2/BitsForAI.sol)#2278

(Medium)for (i = 0; i < _bb.length; i++) {
(tests/time_2/BitsForAI.sol)#2281

(Medium)for (i = 0; i < _bc.length; i++) {
(tests/time_2/BitsForAI.sol)#2284

(Medium)for (i = 0; i < _bd.length; i++) {
(tests/time_2/BitsForAI.sol)#2287

(Medium)for (i = 0; i < _be.length; i++) {
(tests/time_2/BitsForAI.sol)#2290

(Medium)for (uint i = 0; i < bresult.length; i++) {
(tests/time_2/BitsForAI.sol)#2304

(Medium)for (uint i = 0; i < bresult.length; i++) {
(tests/time_2/BitsForAI.sol)#2333

(Medium)for (uint i = 0; i < _arr.length; i++) {
(tests/time_2/BitsForAI.sol)#2378

(Medium)for (uint i = 0; i < _arr.length; i++) {
(tests/time_2/BitsForAI.sol)#2390

(Medium)subindex < n.length && (i + subindex) < h.length && h[i + subindex] == n[subindex]) { (tests/time_2/BitsForAI.sol)#2244

(Medium)i < (32 + _fromOffset + _length)) {
(tests/time_2/BitsForAI.sol)#2563

Security advice

Please check the dynamic array of the loop carefully. If you find it can be exploited by an attacker, please change it to prevent the contract from executing too many loops and causing gas overflow and rollback.

4.58 shift-parameter-mixup

Vulnerability description

Check whether the value in the shift operation is inverted.

Applicable scenarios

```
contract C {
  function f() internal returns (uint a) {
    assembly {
      a := shr(a, 8)
    }
  }
}
```

The shift statement shifts the constant 8 one bit to the right.

Audit results: 【Pass】

Security advice: none.

4.59 shadowing-builtin

Vulnerability description

Use local variables, state variables, functions, modifiers or events to detect hidden built-in symbols.

Applicable scenarios

```
pragma solidity ^0.4.24;
contract Bug {
   uint now; // Overshadows current time stamp.
   function assert(bool condition) public {
      // Overshadows built-in symbol for providing assertions.
   }
   function get_next_expiration(uint earlier_time) private returns (uint) {
      return now + 259200; // References overshadowed timestamp.
   }
}
```

now is defined as a state variable, and the built-in symbol now is hidden. The assert function eclipses the built-in assert function. Such use of these built-in symbols may lead to unexpected results.

Audit results: 【Pass】

Security advice: none.

4.60 shadowing-function

Vulnerability description

Detect hidden functions.

Applicable scenarios

```
contract BaseContract{
   function aa(uint a,uint b) returns (uint) {
   return a;
   }
}
contract DerivedContract is BaseContract{
   function aa(uint a,uint b) returns (uint) {
   return b;
   }
}
```

The aa function of BaseContract does not work.

Audit results: 【Pass】

Security advice: none.

4.61 shadowing-local

Vulnerability description

Detect hidden local variables.

Applicable scenarios

```
pragma solidity ^0.4.24;
contract Bug {
```

```
    uint owner;
    function sensitive_function(address owner) public {
        // ...
        require(owner == msg.sender);
    }
    function alternate_sensitive_function() public {
        address owner = msg.sender;
        // ...
        require(owner == msg.sender);
    }
}
```

The sensitive_function.owner hides the Bug.owner. As a result, using owner in sensitive_function may be incorrect.

Audit results: 【Pass】

Security advice: none.

4.62 transfer-to-zeroaddress

Vulnerability description

In sensitive functions such as transfer, transferFrom, and transferOwnership, user operations are irreversible. Therefore, it is recommended that developers add a non-zero check for the target address in the implementation of these functions to avoid user misuse and cause loss of user permissions and property damage. If it is transferred to 0x00, it will be difficult for the ether to revert, which will cause the ether to be lost.

Applicable scenarios

```
contract Transfertozeroaddress {
    address owner;
    function bad() {
        address aa = 0x0;
        aa.transfer(msg.value);
    }
    function good() {
        address aa = 0x0;
        if(aa != 0x0) {revert();}
        aa.transfer(msg.value);
    }
}
```

Audit results: 【Pass】

Security advice: none.

4.63 uninitialized-fptr-cst

Vulnerability description

Solc versions 0.4.5-0.4.26 and 0.5.0-0.5.8 contain a compiler error that causes unexpected behavior when calling uninitialized function pointers in

the constructor.

Applicable scenarios

```
contract bad0 {
  constructor() public {
    /* Uninitialized function pointer */
    function(uint256) internal returns(uint256) a;
    a(10);
  }
}
```

Calling a(10) will cause unexpected behavior because the function pointer a is not initialized in the constructor.

Audit results: 【Pass】

Security advice: none.

4.64 variable-scope

Vulnerability description

Check the possible usage of the variable before closing the declaration (because it was declared later, or declared in another scope).

Applicable scenarios

```
contract C {
  function f(uint z) public returns (uint) {
    uint y = x + 9 + z; // 'z' is used pre-declaration
    uint x = 7;
    if (z % 2 == 0) {
      uint max = 5;
      // ...
    }
    // 'max' was intended to be 5, but it was mistakenly declared in a scope and not assigned (so it is
zero).
    for (uint i = 0; i < max; i++) {
      x += 1;
    }
    return x;
  }
}
```

In the above case, the variable x will be used before the declaration, which may lead to unintended consequences. In addition, the for loop uses the variable max, which is declared in a previous scope that may not have been reached before. If the user incorrectly uses the variable before any expected statement assignment, it may lead to unintended consequences. It may also indicate that the user intends to reference other variables. That is, max may not be declared normally, and unexpected situations will occur during subsequent calls.

Audit results: 【Pass】

Security advice: none.

4.65 void-cst

Vulnerability description

Detect calls to unimplemented constructors. The calling contract does not declare a constructor.

Applicable scenarios

```
contract A{}
contract B is A{
   constructor() public A(){}
}
```

When reading the definition of B's constructor, we can assume that A() starts the contract, but no code is executed.

Audit results: 【Pass】

Security advice: none.

4.66 incorrect-modifier

Vulnerability description

If the modifier does not execute _ or restore, the execution of the function will return to the default value, which may mislead the caller.

Applicable scenarios

```
contract A{
   modidfier myModifier(){
      if(false){
       _;
      }
   }
   function set() myModifier returns(uint){
      return 0;
   }
}
```

myModifier will cause the set function to fail to execute.

Audit results: 【Pass】

Security advice: none.

4.67 assemblycall-rewrite

Vulnerability description

It is dangerous to use the inline assembly instruction assemblyCall of the CALL series, which will overwrite the input with the output. If an arbitrary address is called a return value, the return value may be different from the expected value.

Applicable scenarios

```
contract MixinSignatureValidator {
    function isValidWalletSignature(
        bytes32 hash,
        address walletAddress,
        bytes signature
    )internal view returns (bool isValid){
        assembly {
            let cdStart := add(calldata, 32)
            let success := staticcall(
                gas, // forward all gas
                walletAddress, // address of Wallet contract
                cdStart, // pointer to start of input
                mload(calldata), // length of input
                cdStart, // write output over input
                32 // output size is 32 bytes
            )
        }
        return isValid;
    }
}
```

Audit results: 【Pass】

Security advice: none.

4.68 block-other-parameters

Vulnerability description

Contracts usually require access to time values to perform certain types of functions. block.number can let you know the current time or time increment, but in most cases it is not safe to use them. block.number The block time of Ethereum is usually about 14 seconds, so the time increment between blocks can be predicted. However, the lockout time is not fixed and may change due to various reasons (for example, fork reorganization and difficulty coefficient). Since the block time is variable, block.number should not rely on accurate time calculations. The ability to generate random numbers is very useful in various applications. An obvious example is a gambling DApp, where a pseudo-random number generator is used to select the winner. However, creating a sufficiently powerful source of randomness in Ethereum is very challenging. Using blockhash, block.difficulty and other areas is also unsafe because they are controlled by miners. If the stakes are high, the miner can mine a large number of blocks by renting hardware in a short period of time, select the block that needs to obtain the block hash value to win, and then discard all other blocks.

Applicable scenarios

```
contract Otherparameters{
   event Number(uint);
   event Coinbase(address);
   event Difficulty(uint);
   event Gaslimit(uint);
   function bad0() external{
      require(block.number == 20);
      require(block.coinbase == msg.sender);
      require(block.difficulty == 20);
      require(block.gaslimit == 20);
   }
}
```

The randomness of Bob's contract depends on block.number and so on. Eve is a miner who manipulates block.number and so on to use Bob's contract.

Audit results: 【Pass】

Security advice: none.

4.69 calls-loop

Vulnerability description

Check that the key access control ETH is transmitted cyclically. If at least one address cannot receive ETH (for example, it is a contract with a default fallback function), the entire transaction will be restored. Loss of parameters.

Applicable scenarios

```
contract CallsInLoop{
   address[] destinations;
   constructor(address[] newDestinations) public{
      destinations = newDestinations;
   }
   function bad() external{
      for (uint i=0; i < destinations.length; i++){
         destinations[i].transfer(i);
      }
   }
}
```

If one of the destination addresses is restored by the rollback function, bad() will restore all, so all the work is wasted.

Audit results: 【Pass】

Security advice: none.

4.70 events-access

Vulnerability description

Detect missing events of key access control parameters.

Applicable scenarios

```
contract C {
   modifier onlyAdmin {
      if (msg.sender != owner) throw;
      _;
   }
   function updateOwner(address newOwner) onlyAdmin external {
      owner = newOwner;
   }
}
```

updateOwner() has no events, so it is difficult to track these extremely privileged operations off-chain.

Audit results: 【Pass】

Security advice: none.

4.71 events-maths

Vulnerability description

Detect missing events of key arithmetic parameters.

Applicable scenarios

```
contract C {
   modifier onlyOwner {
      if (msg.sender != owner) throw;
      _;
   }
   function setBuyPrice(uint256 newBuyPrice) onlyOwner public {
      buyPrice = newBuyPrice;
   }
   function buy() external {
    ... // buyPrice is used to determine the number of tokens purchased
   }
}
```

setBuyPrice does not record events, so it is difficult to track changes in the purchase price off-chain.

Audit results: 【Pass】

Security advice: none.

4.72 extcodesize-invoke

Vulnerability description

The extcodesize is zero when the contract is deployed, and the attacker can call the victim contract in his own constructor. At this time, using extcodesize to verify is invalid.

Applicable scenarios

```
pragma solidity ^0.4.23;
contract ExtCodeSize {
   // This contract would be 'hacked' if the address saved here is a contract address
```

```
    address public thisIsNotAContract;
    function aContractCannotCallThis() public {
        uint codeSize;
        assembly { codeSize := extcodesize(caller) }
        // If extcodesize returns 0, it means the caller's code length is 0, so, it is not a contract...
        // or maybe not
        require(codeSize == 0);
        thisIsNotAContract = msg.sender;
    }
}
```

Audit results:【Pass】

Security advice: none.

4.73 fllback-outofgas

Vulnerability description

The fallback function of the contract is usually used to receive an eth transfer (restore the withdrawal operation after the transfer fails), but if too complicated logic is implemented in the fallback, the gas may be exhausted, resulting in unsuccessful transfer.

Applicable scenarios

```
contract C {
    function f(uint a, uint b) public{
    uint a = 0;
    do {
            continue;
            a++;
        } while(a<10);
    }
}
```

Audit results:【Pass】

Security advice: none.

4.74 incorrect-blockhash

Vulnerability description

The blockhash function only returns the non-zero value of the last 256 blocks. In addition, for the current block, it always returns 0, that is, blockhash (block number) is always equal to 0. The hash of the current block cannot be queried, only the most recent 256 blocks can be queried, otherwise only the value 0 is returned.

Applicable scenarios

```
pragma solidity 0.4.25;
contract MyContract {
    function currentBlockHash() public view returns(bytes32) {
        return blockhash(block.number);
```

```
  }
}
```

Audit results: 【Pass】

Security advice: none.

4.75 incorrect-inheritance-order

Vulnerability description

The entity supports multiple inheritance, which means that a contract can inherit multiple contracts. Multiple inheritance introduces an ambiguity problem called diamond: if two or more basic contracts define the same function, which one should be called in the sub-contract? Solidity resolves this ambiguity by using reverse C3 linearization, which sets a priority between the underlying contracts. In this way, basic contracts have different priorities, so the order of inheritance is important. Ignoring the order of inheritance may lead to unexpected behavior. Pay attention to the order of contract inheritance, because the inherited contract may have overlapping variables or functions, and the inheritance order determines the level of the contract, which in turn determines the overlapping variables and functions in which contract to use.

Applicable scenarios

```
contract A {
  address owner;
}
contract B {
  address owner;
}
contract C is B,A{}
```

Audit results: 【Pass】

Security advice: none.

4.76 missing-zero-check

Vulnerability description

Check the verification of the zero address.

Applicable scenarios

```
contract C {
  modifier onlyAdmin {
    if (msg.sender != owner) throw;
    _;
  }
  function updateOwner(address newOwner) onlyAdmin external {
    owner = newOwner;
  }
```

```
}
```

Bob calls updateOwner without specifying newOwner, so Bob loses the ownership of the contract.

Audit results:【Pass】

Security advice: none.

4.77 reentrancy-benign

Vulnerability description

A reentrancy error is detected. The main explanation here is that the effect of the reentrancy is the same as calling the function twice.

Applicable scenarios

```
function callme(){
  if( ! (msg.sender.call()() ) ){
    throw;
  }
  counter += 1
}
```

callme contains reentrant vulnerabilities. But reentrancy is benign, because its use has the same effect as two consecutive calls.

Audit results:【Pass】

Security advice: none.

4.78 reentrancy-events

Vulnerability description

A reentrancy error is detected, only reentrance vulnerabilities that can lead to out-of-sequence events are reported here.

Applicable scenarios

```
function bug(Called d){
  counter += 1;
  d.f();
  emit Counter(counter);
}
```

If d() is reentered, Counter events will be displayed in the wrong order, which may cause problems for third parties.

Audit results:【Pass】

Security advice: none.

4.79 timestamp

Vulnerability description

There is a strict comparison with block.timestamp or now in the contract, and miners can benefit from block.timestamp.

Applicable scenarios

```
contract Timestamp{
   event Time(uint);
   modifier onlyOwner {
      require(block.timestamp == 0);
      _;
   }
   function bad0() external{
      require(block.timestamp == 0);
   }
}
```

Bob's contract relies on the randomness of block.timestamp. Eve is a miner who manipulates block.timestamp to take advantage of Bob's contract.

Audit results: 【Pass】

Security advice: none.

4.80 signature-malleability

Vulnerability description

The implementation of the cryptographic signature system in the Ethereum contract usually assumes that the signature is unique, but the signature can be changed without having the private key, and the signature is still valid. The EVM specification defines several so-called "pre-compiled" contracts, one of which ecrecover performs elliptic curve public key recovery. A malicious user can slightly modify the three values v, r and s to create other valid signatures. If the signature is part of the signed message hash, the system that performs signature verification at the contract level may be attacked. Malicious users can create valid signatures to replay previously signed messages. The signature cannot contain the existing signature, which is vulnerable to attack.

Applicable scenarios

```
function transfer(bytes _signature,address _to,uint256 _value,uint256 _gasPrice,uint256 _nonce) public
returns (bool){
   bytes32 txid = keccak256(abi.encodePacked(getTransferHash(_to, _value, _gasPrice, _nonce),
_signature)); //bad
   require(!signatureUsed[txid]);
   address from = recoverTransferPreSigned(_signature, _to, _value, _gasPrice, _nonce);
   require(balances[from] > _value);
   balances[from] -= _value;
   balances[_to] += _value;
   signatureUsed[txid] = true;
```

```
}
```

Audit results: 【Pass】

Security advice: none.

4.81 assembly

Vulnerability description

Inline assembly is to access the Ethereum virtual machine from a low level, which leads to the abandonment of several important security features of Solidity.

Applicable scenarios: 【Info:15】

For this pattern, the specific problems in the contract are as follows:

(Informational)assembly { codehash := extcodehash(account) } (tests/time_2/BitsForAI.sol)#438

(Informational)assembly { (tests/time_2/BitsForAI.sol)#1405

(Informational)assembly { (tests/time_2/BitsForAI.sol)#1440

(Informational)assembly { (tests/time_2/BitsForAI.sol)#1448

(Informational)assembly { (tests/time_2/BitsForAI.sol)#1455

(Informational)assembly { (tests/time_2/BitsForAI.sol)#1475

(Informational)assembly { (tests/time_2/BitsForAI.sol)#1497

(Informational)assembly { (tests/time_2/BitsForAI.sol)#2405

(Informational)assembly { (tests/time_2/BitsForAI.sol)#2417

(Informational)assembly { (tests/time_2/BitsForAI.sol)#2425

(Informational)assembly { (tests/time_2/BitsForAI.sol)#2455

(Informational)assembly { (tests/time_2/BitsForAI.sol)#2564

(Informational)assembly { (tests/time_2/BitsForAI.sol)#2588

(Informational)assembly { (tests/time_2/BitsForAI.sol)#2614

(Informational)assembly { (tests/time_2/BitsForAI.sol)#2646

Security advice

Check the inline assembly instructions carefully to make sure they can run safely, otherwise they need to be replaced.

4.82 assert-state-change

Vulnerability description

Wrong use of assert(). See Solidity best practices.

Applicable scenarios

```
contract A {
 uint s_a;
 function bad() public {
   assert((s_a += 1) > 10);
 }
}
```

The assert in Bad() increments the state variable s_a when checking the condition.

Audit results: 【Pass】

Security advice: none.

4.83 delete-dynamic-arrays

Vulnerability description

Applying delete or .length=0 to dynamically sized storage arrays may cause Out-of-Gas exceptions. Because it will traverse all the data in the array, the gas may be exceeded.

Applicable scenarios

```
contract C {
   uint[] amounts;
   address payable[] addresses;
   function collect(address payable to) external payable {
      amounts.push(msg.value);
      addresses.push(to);
   }
   function pay() external {
      uint length = amounts.length;
      delete amounts;
      delete addresses;
   }
}
```

Audit results: 【Pass】

Security advice: none.

4.84 deprecated-standards

Vulnerability description

Several functions and operators in Solidity are not recommended. Using them will reduce code quality. In the new major version of the Solidity compiler, deprecated functions and operators may cause side effects and compilation errors. Structures deprecated by Solidity after 0.5.0: years, sha3, suicide, throw and constant functions.

Applicable scenarios

```
contract ContractWithDeprecatedReferences {
   // Deprecated: Change block.blockhash() -> blockhash()
   bytes32 globalBlockHash = block.blockhash(0);
   // Deprecated: Change constant -> view
```

```
    function functionWithDeprecatedThrow() public constant {
        // Deprecated: Change msg.gas -> gasleft()
        if(msg.gas == msg.value) {
            // Deprecated: Change throw -> revert()
            throw;
        }
    }
    // Deprecated: Change constant -> view
    function functionWithDeprecatedReferences() public constant {
        // Deprecated: Change sha3() -> keccak256()
        bytes32 sha3Result = sha3("test deprecated sha3 usage");
        // Deprecated: Change callcode() -> delegatecall()
        address(this).callcode();
        // Deprecated: Change suicide() -> selfdestruct()
        suicide(address(0));
    }
}
```

Audit results:【Pass】

Security advice: none.

4.85 erc20-indexed

Vulnerability description

The address parameters of the "Transfer" and "Approval" events of the ERC-20 token standard shall include indexed.

Applicable scenarios

```
contract ERC20Bad {
    // ...
    event Transfer(address from, address to, uint value);
    event Approval(address owner, address spender, uint value);
    // ...
}
```

According to the definition of the ERC20 specification, the first two parameters of the Transfer and Approval events should carry the indexed keyword. If these keywords are not included, the parameter data will be excluded from the bloom filter of the transaction/block. Therefore, external tools searching for these parameters may ignore them and fail to index the logs in this token contract.

Audit results:【Pass】

Security advice: none.

4.86 erc20-throw

Vulnerability description

The function of the ERC-20 token standard should be thrown in the following special circumstances: if there are not enough tokens in the _from account balance to spend, it should be thrown; unless the _from account

deliberately authorizes the sending of messages through some mechanismOtherwise, transferFrom should be thrown.

Applicable scenarios

```
contract SomeToken {
   mapping(address => uint256) balances;
   event Transfer(address indexed _from, address indexed _to, uint256 _value);
   function transfer(address _to, uint _value) public returns (bool) {
      if (_value > balances[msg.sender] || _value > balances[_to] + _value) {
         return false;
      }
      balances[msg.sender] = balances[msg.sender] - _value;
      balances[_to] = balances[_to] + _value;
      emit Transfer(msg.sender, _to, _value);
      return true;
   }
}
```

Audit results:【Pass】

Security advice: none.

4.87 length-manipulation

Vulnerability description

The length of the dynamic array changes directly. In this case, huge arrays may appear, and storage overlap attacks (conflicts with other data in the storage) may result. The operations of "length" are: =, +=, -=, *=, /=, --, etc.

Applicable scenarios: 【Info:2】

For this pattern, the specific problems in the contract are as follows:

(Informational)_ownedTokens[from].length--; (tests/time_2/BitsForAI.sol)#1107

(Informational)_allTokens.length--; (tests/time_2/BitsForAI.sol)#1134

Security advice

Carefully check the operation of the dynamic array to see if the array can be added arbitrarily by the attacker.

4.88 low-level-calls

Vulnerability description

Label low-level methods such as call, delegatecall, and callcode, because these methods are easily exploited by attackers.

Applicable scenarios

```
contract Sender {
   address owner;
   modifier onlyceshi() {
   owner.callcode(bytes4(keccak256("inc()")));
   _;
   }
   function send(address _receiver) payable external {
      _receiver.call.value(msg.value).gas(7777)("");
   }
   function sendceshi(address _receiver) payable external {
      if(_receiver.call.value(msg.value).gas(7777)("")){
      revert();
   }
   }
}
```

Audit results:【Pass】

Security advice: none.

4.89 msgvalue-equals-zero

Vulnerability description

msg.value==0 The check condition is meaningless in most cases.

Applicable scenarios

```
contract A{
   address owner;
   mapping(address => uint256) balances;
   constructor() {
      owner = msg.sender;
   }
   function B() return (uint256){
      if(masg.value == 0) {
         return 0;
      }
      balances[msg.sender] += msg.value;
      return balances[msg.sender];
   }
}
```

Audit results:【Pass】

Security advice: none.

4.90 naming-convention

Vulnerability description

Check whether the naming written in the contract is standardized, because the naming is messy and not easy to understand and manage.

Applicable scenarios

```
contract naming {
   enum Numbers {ONE, TWO}
   enum numbers {ONE, TWO}
   uint constant MY_CONSTANT = 1;
```

```
    uint constant MY_other_CONSTANT = 2;
    uint Var_One = 1;
    uint varTwo = 2;
}
```

Audit results: 【Pass】

Security advice: none.

4.91 pragma

Vulnerability description

The use of different Solidity versions (more than two) in the contract will make the compiler not able to compile according to our ideas.

Applicable scenarios

```
pragma solidity ^0.4.23;
pragma solidity ^0.4.24;
```

Audit results: 【Pass】

Security advice: none.

4.92 solc-version

Vulnerability description

Solidity source files indicate the version of the compiler that can be compiled with them. It is recommended to indicate a clear version, because future versions of the compiler may handle certain language constructs in ways that developers cannot foresee.

Applicable scenarios

```
pragma solidity >=0.4.23 <0.4.25;
```

Audit results: 【Pass】

Security advice: none.

4.93 unimplemented-functions

Vulnerability description

Detect functions that are not implemented on most derivative contracts.

Applicable scenarios

```
interface BaseInterface {
    function f1() external returns(uint);
    function f2() external returns(uint);
}
interface BaseInterface2 {
    function f3() external returns(uint);
}
contract DerivedContract is BaseInterface, BaseInterface2 {
    function f1() external returns(uint){
        return 42;
```

```
  }
}
```

DerivedContract does not implement BaseInterface.f2 or BaseInterface2.f3. As a result, the contract will not be compiled correctly. All unrealized functions must be realized on the contract to be used.

Audit results:【Pass】

Security advice: none.

4.94 upgrade-050

Vulnerability description

Check the code update for the Solidity 0.5.0 version. For example: .call() with more than one parameter, keccak256(...) with more than one parameter, etc.

Applicable scenarios

```
contract Token {
  uint totalSupply;
  function Token() {
    totalSupply = +1e18;
  }
  function () payable {}
}
```

Audit results:【Pass】

Security advice: none.

4.95 function-init-state

Vulnerability description

Detect non-fixed variables (the execution result in the function is related to the execution order in the function) to initialize the state variable. The different initialization sequence of the state variable may lead to different initialization values.

Applicable scenarios

```
contract StateVaribleInitFromDynamicFunction {
  uint public v_al = setval(); // Initialize from function (sets to 77)
  uint public w_al = 7;
  uint public x_al = setval(); // Initialize from function (sets to 88)

  constructor(){
  }
  function setval() public returns(uint) {
    if(w_val == 0) {
      return 4;
    }
    return 5;
```

```
  }
}
```

The value of v_al is initialized to 4, and the value of x_al is initialized to 5. However, the caller believes that these two values should be the same. Therefore, this may cause other errors to occur.

Audit results: 【Pass】

Security advice: none.

4.96 complex-function

Vulnerability description

Complicated functions will consume more gas. When gas is greater than the set gas limit, the transaction will fail to execute, so every call will fail.

Applicable scenarios

```
contract Complex {
  function a() {
    int numberOfSides = 7;
    string shape;
    uint i0 = 0;
    uint i1 = 0;
    uint i2 = 0;
    uint i3 = 0;
    uint i4 = 0;
    uint i5 = 0;
    uint i6 = 0;
    uint i7 = 0;
    uint i8 = 0;
    uint i9 = 0;
    uint i10 = 0;
    ...
  }
}
```

Bob's call to function a() will exceed the gas limit, and the call will never succeed.

Audit results: 【Pass】

Security advice: none.

4.97 hardcoded

Vulnerability description

The contract contains an unknown address, which may be used for some malicious activities. Need to check the hard-coded address and its purpose. The address length is prone to errors, and the length of the address is not enough, it will not report an error, so it is very dangerous to write a

mistake. Here is an identification.

Applicable scenarios: 【Info:16】

For this pattern, the specific problems in the contract are as follows:

(Informational)0x1d3B2638a7cC9f2CB3D298A3DA7a90B67E5506ed) > 0) { //mainnet (tests/time_2/BitsForAI.sol)#1641

(Informational)0x1d3B2638a7cC9f2CB3D298A3DA7a90B67E5506ed); (tests/time_2/BitsForAI.sol)#1642

(Informational)0xc03A2615D5efaf5F49F60B7BB6583eaec212fdf1) > 0) { //ropsten testnet (tests/time_2/BitsForAI.sol)#1646

(Informational)0xc03A2615D5efaf5F49F60B7BB6583eaec212fdf1); (tests/time_2/BitsForAI.sol)#1647

(Informational)0xB7A07BcF2Ba2f2703b24C0691b5278999C59AC7e) > 0) { //kovan testnet (tests/time_2/BitsForAI.sol)#1651

(Informational)0xB7A07BcF2Ba2f2703b24C0691b5278999C59AC7e); (tests/time_2/BitsForAI.sol)#1652

(Informational)0x146500cfd35B22E4A392Fe0aDc06De1a1368Ed48) > 0) { //rinkeby testnet (tests/time_2/BitsForAI.sol)#1656

(Informational)0x146500cfd35B22E4A392Fe0aDc06De1a1368Ed48); (tests/time_2/BitsForAI.sol)#1657

(Informational)0xa2998EFD205FB9D4B4963aFb70778D6354ad3A41) > 0) { //goerli testnet (tests/time_2/BitsForAI.sol)#1661

(Informational)0xa2998EFD205FB9D4B4963aFb70778D6354ad3A41); (tests/time_2/BitsForAI.sol)#1662

(Informational)0x6f485C8BF6fc43eA212E93BBF8ce046C7f1cb475) > 0) { //ethereum-bridge (tests/time_2/BitsForAI.sol)#1666

(Informational)0x6f485C8BF6fc43eA212E93BBF8ce046C7f1cb475); (tests/time_2/BitsForAI.sol)#1667

(Informational)0x20e12A1F859B3FeaE5Fb2A0A32C18F5a65555bBF) > 0) { //ether.camp ide (tests/time_2/BitsForAI.sol)#1670

(Informational)0x20e12A1F859B3FeaE5Fb2A0A32C18F5a65555bBF); (tests/time_2/BitsForAI.sol)#1671

(Informational)0x51efaF4c8B3C9AfBD5aB9F4bbC82784Ab6ef8fAA) > 0) { //browser-solidity (tests/time_2/BitsForAI.sol)#1674

(Informational)0x51efaF4c8B3C9AfBD5aB9F4bbC82784Ab6ef8fAA);
(tests/time_2/BitsForAI.sol)#1675

Security advice

Check carefully whether the address is wrong, and if there is an error, please take the time to correct it.

4.98 overpowered-role

Vulnerability description

This function can only be called from one address, so the system relies heavily on this address. In this case, it may cause undesirable consequences for investors. For example, if the private key of the address is compromised, the account will become unusable and the contract will not function properly.

Applicable scenarios

```
contract Crowdsale {
   address public owner;
   uint rate;
   constructor() {
      owner = msg.sender;
   }
   function setRate(_rate) public onlyOwner {
      rate = _rate;
   }
}
```

Audit results:【Pass】

Security advice: none.

4.99 reentrancy-limited-events

Vulnerability description

If a reentrancy error is detected, only reentry vulnerabilities that can cause out-of-sequence events in the restricted gas of read and transfer are reported here.

Applicable scenarios

```
function bug(Called d){
   uint sendeth = 0;
   msg.sender.send(sendeth):
   emit Counter(counter);
}
```

If you reenter "send", the "counter" events will be displayed in an incorrect order, which may cause problems for third parties.

Audit results:【Pass】

Security advice: none.

4.100 reentrancy-limited-gas

Vulnerability description

In the case of natural gas price changes, "send" and "transfer" cannot prevent re-entry, but only show that re-entry is more difficult.

Applicable scenarios

```
function callme(){
   msg.sender.transfer(balances[msg.sender]):
   balances[msg.sender] = 0;
}
```

In the case of natural gas price changes, "send" and "transfer" cannot prevent reentry.

Audit results: 【Pass】

Security advice: none.

4.101 reentrancy-limited-gas-no-eth

Vulnerability description

Compared with reentrancy-limited-gas, the send and transfer reentry without eth transfer is detected here.

Applicable scenarios

```
function callme(){
   uint sendeth = 0;
   msg.sender.transfer(sendeth):
   balances[msg.sender] = balances[msg.sender] - sendeth;
}
```

In the case of natural gas price changes, "send" and "transfer" cannot prevent reentry.

Audit results: 【Pass】

Security advice: none.

4.102 similar-names

Vulnerability description

Detect variables whose names are too similar.

Applicable scenarios

```
contract SimilarVariables {
   uint similarvariables1 = 1;
   uint similarvariables2 = 2;
   uint similarvariables3 = 3;
}
```

Similar variables make the contract difficult to read.

Audit results: 【Pass】

Security advice: none.

4.103 too-many-digits

Vulnerability description

Words with many numbers are difficult to read and view, and variable names are easy to mislead people.

Applicable scenarios

```
contract MyContract{
   uint 1_ether = 10000000000000000000;
}
```

Although 1_ether looks like 1 ether, it is 10 ether. As a result, it is likely to be used incorrectly.

Audit results:【Pass】

Security advice: none.

4.104 private-not-hidedata

Vulnerability description

Contrary to common understanding, the private modifier does not make variables invisible, and miners can access the code and data of all contracts. Developers must solve the problem of Ethereum's lack of privacy. Although it is private, it can be viewed by miners.

Applicable scenarios: 【Info:35】

For this pattern, the specific problems in the contract are as follows:

(Informational)private _owner; (tests/time_2/BitsForAI.sol)#236

(Informational)private constant _INTERFACE_ID_ERC165 = 0x01ffc9a7; (tests/time_2/BitsForAI.sol)#533

(Informational)private _supportedInterfaces; (tests/time_2/BitsForAI.sol)#538

(Informational)private constant _ERC721_RECEIVED = 0x150b7a02; (tests/time_2/BitsForAI.sol)#594

(Informational)private _tokenOwner; (tests/time_2/BitsForAI.sol)#597

(Informational)private _tokenApprovals; (tests/time_2/BitsForAI.sol)#600

(Informational)private _ownedTokensCount; (tests/time_2/BitsForAI.sol)#603

(Informational)private _operatorApprovals; (tests/time_2/BitsForAI.sol)#606

(Informational)private constant _INTERFACE_ID_ERC721 = 0x80ac58cd; (tests/time_2/BitsForAI.sol)#622

(Informational)private _ownedTokens; (tests/time_2/BitsForAI.sol)#951

(Informational)private _ownedTokensIndex; (tests/time_2/BitsForAI.sol)#954

(Informational)private _allTokens; (tests/time_2/BitsForAI.sol)#957

(Informational)private _allTokensIndex; (tests/time_2/BitsForAI.sol)#960

(Informational)private constant _INTERFACE_ID_ERC721_ENUMERABLE = 0x780e9d63; (tests/time_2/BitsForAI.sol)#969

(Informational)private _name; (tests/time_2/BitsForAI.sol)#1164

(Informational)private _symbol; (tests/time_2/BitsForAI.sol)#1167

(Informational)private _tokenURIs; (tests/time_2/BitsForAI.sol)#1170

(Informational)private constant _INTERFACE_ID_ERC721_METADATA = 0x5b5e139f; (tests/time_2/BitsForAI.sol)#1179

(Informational)private _guardCounter; (tests/time_2/BitsForAI.sol)#1284

(Informational)private constant MAJOR_TYPE_INT = 0; (tests/time_2/BitsForAI.sol)#1512

(Informational)private constant MAJOR_TYPE_MAP = 5; (tests/time_2/BitsForAI.sol)#1513

(Informational)private constant MAJOR_TYPE_BYTES = 2; (tests/time_2/BitsForAI.sol)#1514

(Informational)private constant MAJOR_TYPE_ARRAY = 4; (tests/time_2/BitsForAI.sol)#1515

(Informational)private constant MAJOR_TYPE_STRING = 3; (tests/time_2/BitsForAI.sol)#1516

(Informational)private constant MAJOR_TYPE_NEGATIVE_INT = 1; (tests/time_2/BitsForAI.sol)#1517

(Informational)private constant MAJOR_TYPE_CONTENT_FREE = 7; (tests/time_2/BitsForAI.sol)#1518

(Informational)private _name = "BitsForAI"; (tests/time_2/BitsForAI.sol)#2670

(Informational)private _symbol = "BFA"; (tests/time_2/BitsForAI.sol)#2671

(Informational)private _tokenURI = "https://api.bitsforai.com/tokenid/"; (tests/time_2/BitsForAI.sol)#2672

(Informational)private _currentTokenId; (tests/time_2/BitsForAI.sol)#2673

(Informational)private GWEI = 1000000000; (tests/time_2/BitsForAI.sol)#2675

(Informational)private gasPrice = 4010000000; //many set exactly 4gwei, so adding 0.01 gwei increases speed much more than expected. (tests/time_2/BitsForAI.sol)#2676

(Informational)private gasAmount = 250000; (tests/time_2/BitsForAI.sol)#2677

(Informational)private BASIS_POINTS = 10000; (tests/time_2/BitsForAI.sol)#2679

(Informational)private NUM_RANDOM_BYTES_REQUESTED = 2; //The variable `ceiling` should never be greater than: `(256 ^ NUM_RANDOM_BYTES_REQUESTED) - 1`. (tests/time_2/BitsForAI.sol)#2681

Security advice

Carefully check the writing logic to ensure that the private variables you write can be seen, otherwise you should pay attention to privacy and use encryption and other means.

4.105 safemath

Vulnerability description

SafeMath library is used. It is good to use SafeMath, but if it is modified, it will also cause some loopholes. It should be noted that we call for the use of the safemath library, but care should be taken not to modify it at will.

Applicable scenarios: 【Info:3】

For this pattern, the specific problems in the contract are as follows:

(Informational)using SafeMath for uint256; (tests/time_2/BitsForAI.sol)#496

(Informational)using SafeMath for uint256; (tests/time_2/BitsForAI.sol)#588

(Informational)using SafeMath for uint; (tests/time_2/BitsForAI.sol)#2665

Security advice

Carefully check the relevant functions of the safemath library to ensure that there are no loopholes.

4.106 array-instead-bytes

Vulnerability description

byte[] can be converted to bytes to save gas resources.

Applicable scenarios

```
pragma solidity 0.4.24;
contract C {
   byte[] someVariable;
   ...
}
```

Audit results: 【Pass】

Security advice: none.

4.107 boolean-equal

Vulnerability description

Check the comparison of Boolean constants. There is no need to compare with true and false, so it's superfluous (gas consumption).

Applicable scenarios

```
contract A {
   function f(bool x) public {
      // ...
      if (x == true) { // bad!
       // ...
      }
      // ...
   }
}
```

Boolean constants can be used directly without comparison with true or false.

Audit results: 【Pass】

Security advice: none.

4.108 code-no-effects

Vulnerability description

In Solidity, you can write code that does not produce the desired effect. Currently, the solidity compiler will not return warnings for invalid codes. This can lead to the introduction of "dead" code that cannot perform the expected actions correctly. For example, it is easy to omit the parentheses msg.sender.call.value(address(this).balance)("");, which may cause the function to continue execution without transferring funds to msg.sender.

Applicable scenarios

```
pragma solidity ^0.5.0;
contract Wallet {
   mapping(address => uint) balance;
```

```
    // Withdraw funds from contract
    function withdraw(uint amount) public {
        require(amount <= balance[msg.sender], 'amount must be less than balance');
        uint previousBalance = balance[msg.sender];
        balance[msg.sender] = previousBalance - amount;
        // Attempt to send amount from the contract to msg.sender
        msg.sender.call.value(amount);
    }
}
```

Audit results: 【Pass】

Security advice: none.

4.109 constable-states

Vulnerability description

Constant state variables should be declared as constants to save gas.

Applicable scenarios

```
contract B {
    address public mySistersAddress = 0x999999cf1046e68e36E1aA2E0E07105eDDD1f08E;
    function setUsed(uint a) public {
        if (msg.sender == MY_ADDRESS) {
            used = a;
        myFriendsAddress = 0xc0ffee254729296a45a3885639AC7E10F9d54980;
        }
    }
}
```

Audit results: 【Pass】

Security advice: none.

4.110 event-before-revert

Vulnerability description

The event is called before the exception is thrown, and the Revert rollback will make the event waste gas.

Applicable scenarios

```
contract Callbeforerevert {
    address owner;
    event EventName(address bidder, uint amount);
    function bad() public {
        emit EventName(msg.sender, msg.value);
        revert();
    }
}
```

Audit results: 【Pass】

Security advice: none.

4.111 external-function

Vulnerability description

Functions with public visibility modifiers are not called internally. Changing the visibility level to an external level can improve the readability of the code. In addition, in many cases, functions that use external visibility modifiers cost less gas than functions that use public visibility modifiers.

Applicable scenarios

```
contract ContractWithFunctionCalledSuper {
  function callWithSuper() {
    uint256 i = 0;
  }
}
```

The callWithSuper() function can be declared as external visibility.

Audit results: 【Pass】

Security advice: none.

4.112 extra-gas-inloops

Vulnerability description

Use non-memory array state variables .balance or .length under the condition of for loop or while loop. In this case, each iteration of the loop will consume additional gas.

Applicable scenarios

```
contract NewContract {
  uint[] ss;
  function longLoop() {
    for(uint i = 0; i < ss.length; i++) {
      uint a = ss[i];
      /* ... */
    }
  }
}
```

Audit results: 【Pass】

Security advice: none.

4.113 missing-inheritance

Vulnerability description

Detect lost inheritance.

Applicable scenarios

```
interface ISomething {
  function f1() external returns(uint);
}
contract Something {
  function f1() external returns(uint){
```

```
        return 42;
    }
}
```

Some things should inherit from ISomething.

Audit results: 【Pass】

Security advice: none.

4.114 redundant-statements

Vulnerability description

Detect the use of invalid statements.

Applicable scenarios

```
contract RedundantStatementsContract {
    constructor() public {
        uint; // Elementary Type Name
        bool; // Elementary Type Name
        RedundantStatementsContract; // Identifier
    }
    function test() public returns (uint) {
        uint; // Elementary Type Name
        assert; // Identifier
        test; // Identifier
        return 777;
    }
}
```

Each comment line references the type/identifier, but does nothing on it, so no code is generated for such statements, so it can be deleted.

Audit results: 【Pass】

Security advice: none.

4.115 return-struct

Vulnerability description

Consider using struct instead of multiple return values for internal or private functions, it can improve the readability of the code. Function to pass multiple values using struct.

Applicable scenarios

```
pragma solidity 0.4.24;
contract TestContract {
    function test() internal returns(uint a, address b, bool c, int d) {
        a = 1;
        b = msg.sender;
        c = true;
        d = 2;
    }
}
```

Audit results: 【Pass】

Security advice: none.

4.116 revert-require

Vulnerability description

if (condition) {revert(); or throw;} can be replaced by require(condition) to save resources and gas.

Applicable scenarios: 【Opt:1】

For this pattern, the specific problems in the contract are as follows:

(Optimization)if (uint(uint8(bresult[i])) == 46) {

(tests/time_2/BitsForAI.sol)#2312

Security advice

Use require(condition) instead of constructing if (condition) {revert(); or throw;}.

4.117 send-transfer

Vulnerability description

The recommended way to perform the check of Ether payment is addr.transfer(x). If the transfer fails, an exception is automatically raised.

Applicable scenarios

```
if(!addr.send(42 ether)) {
    revert();
}
```

Audit results: 【Pass】

Security advice: none.

4.118 unused-state

Vulnerability description

Unused variables are allowed in Solidity, and they do not pose direct security issues. The best practice is to avoid them as much as possible: resulting in increased calculations (and unnecessary gas consumption) means errors or incorrect data structures, and usually means poor code quality leads to code noise and reduces code readability.

Applicable scenarios

```
contract A{
    address unused;
    address public unused2;
    address private unused3;
    address unused4;
```

```
    address used;
    function ceshi1 () external{
        unused3 = address(0);
    }
}
```

Audit results: 【Pass】

Security advice: none.

4.119 costly-operations-loop

Vulnerability description

Expensive operations within the loop.

Applicable scenarios

```
contract CostlyOperationsInLoop{
    uint loop_count = 100;
    uint state_variable=0;
    function bad() external{
        for (uint i=0; i < loop_count; i++){
            state_variable++;
        }
    }
    function good() external{
     uint local_variable = state_variable;
     for (uint i=0; i < loop_count; i++){
        local_variable++;
     }
     state_variable = local_variable;
    }
}
```

Due to the expensive SSTOREs, the incremental state variables in the loop will generate a large amount of gas, which may result in insufficient gas.

Audit results: 【Pass】

Security advice: none.

Thanks for using the SmartFast contract audit platform!