
Armors Labs
UU
Smart Contract Audit

UU Audit Summary

Project name : UU Contract

Project address: None

Code URL : https://cn.etherscan.com/address/0xc1b3bc93bf49439c87ea7b7cfb0b42f8a08dad3b#code

Projct target : UU Contract Audit

Test result : PASSED

Audit Info

Audit NO : 0X202102050002

Audit Team : Armors Labs

UU Audit

The UU team asked us to review and audit their UU contract. We looked at the code and now publish our results.

Here is our assessment and recommendations, in order of importance.

Document information

Name	Auditor	Version	Date
UU Audit	Rock ,Hosea, Rushairer	1.0.0	2021-02-05

Review

Audit results

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the UU contract. The above should not be construed as investment advice.

Based on the widely recognized security status of the current underlying blockchain and smart contract, this audit report is valid for 18 months from the date of output.

(Statement: Armors Labs reports only on facts that have occurred or existed before this report is issued and assumes corresponding responsibilities. Armors Labs is not able to determine the security of its smart contracts and is not responsible for any subsequent or existing facts after this report is issued. The security audit analysis and other content of this report are only based on the documents and information provided by the information provider to Armors Labs at the time of issuance of this report (" information provided " for short). Armors Labs postulates that the information provided is not missing, tampered, deleted or hidden. If the information provided is missing, tampered, deleted, hidden or reflected in a way that is not consistent with the actual situation, Armors Labs shall not be responsible for the losses and adverse effects caused.)

Audited target file

file	md5
UU.sol	76d1e40b100f0d3fb79f4ba839a406fc

Vulnerability analysis

Vulnerability distribution

vulnerability level	number
Critical severity	0
High severity	0
Medium severity	0
Low severity	0

Summary of audit results

Vulnerability	status
Re-Entrancy	safe
Arithmetic Over/Under Flows	safe
Unexpected Ether	safe
Delegatecall	safe
Default Visibilities	safe
Entropy Illusion	safe
External Contract Referencing	safe
Short Address/Parameter Attack	safe
Unchecked CALL Return Values	safe
Race Conditions / Front Running	safe
Denial Of Service (DOS)	safe
Block Timestamp Manipulation	safe
Constructors with Care	safe
Unintialised Storage Pointers	safe
Floating Points and Numerical Precision	safe
tx.origin Authentication	safe

Contract code

```
/**
 *Submitted for verification at Etherscan.io on 2021-01-29
*/

// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0;
//pragma experimental ABIEncoderV2;

/**
 * @title Proxy
 * @dev Implements delegation of calls to other contracts, with proper
 * forwarding of return values and bubbling of failures.
 * It defines a fallback function that delegates all calls to the address
 * returned by the abstract _implementation() internal function.
 */
abstract contract Proxy {
  /**
   * @dev Fallback function.
   * Implemented entirely in `_fallback`.
   */
  fallback () payable external {
    _fallback();
  }

  receive () payable external {
    _fallback();
  }

  /**
   * @return The Address of the implementation.
   */
  function _implementation() virtual internal view returns (address);

  /**
   * @dev Delegates execution to an implementation contract.
   * This is a low level function that doesn't return to its internal call site.
   * It will return to the external caller whatever the implementation returns.
   * @param implementation Address to delegate.
   */
  function _delegate(address implementation) internal {
    assembly {
      // Copy msg.data. We take full control of memory in this inline assembly
      // block because it will not return to Solidity code. We overwrite the
      // Solidity scratch pad at memory position 0.
      calldatacopy(0, 0, calldatasize())

      // Call the implementation.
      // out and outsize are 0 because we don't know the size yet.
      let result := delegatecall(gas(), implementation, 0, calldatasize(), 0, 0)

      // Copy the returned data.
      returndatacopy(0, 0, returndatasize())

      switch result
      // delegatecall returns 0 on error.
      case 0 { revert(0, returndatasize()) }
      default { return(0, returndatasize()) }
    }
  }

  /**
   * @dev Function that is run as the first thing in the fallback function.
   * Can be redefined in derived contracts to add functionality.
   * Redefinitions must call super._willFallback().
```

```
   */
  function _willFallback() virtual internal {

  }

  /**
   * @dev fallback implementation.
   * Extracted to enable manual triggering.
   */
  function _fallback() internal {
    if(OpenZeppelinUpgradesAddress.isContract(msg.sender) && msg.data.length == 0 && gasleft() <= 230
        return;
    _willFallback();
    _delegate(_implementation());
  }
}


/**
 * @title BaseUpgradeabilityProxy
 * @dev This contract implements a proxy that allows to change the
 * implementation address to which it will delegate.
 * Such a change is called an implementation upgrade.
 */
abstract contract BaseUpgradeabilityProxy is Proxy {
  /**
   * @dev Emitted when the implementation is upgraded.
   * @param implementation Address of the new implementation.
   */
  event Upgraded(address indexed implementation);

  /**
   * @dev Storage slot with the address of the current implementation.
   * This is the keccak-256 hash of "eip1967.proxy.implementation" subtracted by 1, and is
   * validated in the constructor.
   */
  bytes32 internal constant IMPLEMENTATION_SLOT = 0x360894a13ba1a3210667c828492db98dca3e2076cc3735a92

  /**
   * @dev Returns the current implementation.
   * @return impl Address of the current implementation
   */
  function _implementation() override internal view returns (address impl) {
    bytes32 slot = IMPLEMENTATION_SLOT;
    assembly {
      impl := sload(slot)
    }
  }

  /**
   * @dev Upgrades the proxy to a new implementation.
   * @param newImplementation Address of the new implementation.
   */
  function _upgradeTo(address newImplementation) internal {
    _setImplementation(newImplementation);
    emit Upgraded(newImplementation);
  }

  /**
   * @dev Sets the implementation address of the proxy.
   * @param newImplementation Address of the new implementation.
   */
  function _setImplementation(address newImplementation) internal {
    require(OpenZeppelinUpgradesAddress.isContract(newImplementation), "Cannot set a proxy implementa

    bytes32 slot = IMPLEMENTATION_SLOT;
```

```
    assembly {
      sstore(slot, newImplementation)
    }
  }
}


/**
 * @title BaseAdminUpgradeabilityProxy
 * @dev This contract combines an upgradeability proxy with an authorization
 * mechanism for administrative tasks.
 * All external functions in this contract must be guarded by the
 * `ifAdmin` modifier. See ethereum/solidity#3864 for a Solidity
 * feature proposal that would enable this to be done automatically.
 */
contract BaseAdminUpgradeabilityProxy is BaseUpgradeabilityProxy {
  /**
   * @dev Emitted when the administration has been transferred.
   * @param previousAdmin Address of the previous admin.
   * @param newAdmin Address of the new admin.
   */
  event AdminChanged(address previousAdmin, address newAdmin);

  /**
   * @dev Storage slot with the admin of the contract.
   * This is the keccak-256 hash of "eip1967.proxy.admin" subtracted by 1, and is
   * validated in the constructor.
   */

  bytes32 internal constant ADMIN_SLOT = 0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b

  /**
   * @dev Modifier to check whether the `msg.sender` is the admin.
   * If it is, it will run the function. Otherwise, it will delegate the call
   * to the implementation.
   */
  modifier ifAdmin() {
    if (msg.sender == _admin()) {
      _;
    } else {
      _fallback();
    }
  }

  /**
   * @return The address of the proxy admin.
   */
  function admin() external ifAdmin returns (address) {
    return _admin();
  }

  /**
   * @return The address of the implementation.
   */
  function implementation() external ifAdmin returns (address) {
    return _implementation();
  }

  /**
   * @dev Changes the admin of the proxy.
   * Only the current admin can call this function.
   * @param newAdmin Address to transfer proxy administration to.
   */
  function changeAdmin(address newAdmin) external ifAdmin {
    require(newAdmin != address(0), "Cannot change the admin of a proxy to the zero address");
```

```
    emit AdminChanged(_admin(), newAdmin);
    _setAdmin(newAdmin);
  }

  /**
   * @dev Upgrade the backing implementation of the proxy.
   * Only the admin can call this function.
   * @param newImplementation Address of the new implementation.
   */
  function upgradeTo(address newImplementation) external ifAdmin {
    _upgradeTo(newImplementation);
  }

  /**
   * @dev Upgrade the backing implementation of the proxy and call a function
   * on the new implementation.
   * This is useful to initialize the proxied contract.
   * @param newImplementation Address of the new implementation.
   * @param data Data to send as msg.data in the low level call.
   * It should include the signature and the parameters of the function to be called, as described in
   * https://solidity.readthedocs.io/en/v0.4.24/abi-spec.html#function-selector-and-argument-encoding
   */
  function upgradeToAndCall(address newImplementation, bytes calldata data) payable external ifAdmin
    _upgradeTo(newImplementation);
    (bool success,) = newImplementation.delegatecall(data);
    require(success);
  }

  /**
   * @return adm The admin slot.
   */
  function _admin() internal view returns (address adm) {
    bytes32 slot = ADMIN_SLOT;
    assembly {
      adm := sload(slot)
    }
  }

  /**
   * @dev Sets the address of the proxy admin.
   * @param newAdmin Address of the new proxy admin.
   */
  function _setAdmin(address newAdmin) internal {
    bytes32 slot = ADMIN_SLOT;

    assembly {
      sstore(slot, newAdmin)
    }
  }

  /**
   * @dev Only fall back when the sender is not the admin.
   */
  function _willFallback() virtual override internal {
    require(msg.sender != _admin(), "Cannot call fallback function from the proxy admin");
    //super._willFallback();
  }
}

interface IAdminUpgradeabilityProxyView {
  function admin() external view returns (address);
  function implementation() external view returns (address);
}


/**
```

```
 * @title UpgradeabilityProxy
 * @dev Extends BaseUpgradeabilityProxy with a constructor for initializing
 * implementation and init data.
 */
abstract contract UpgradeabilityProxy is BaseUpgradeabilityProxy {
  /**
   * @dev Contract constructor.
   * @param _logic Address of the initial implementation.
   * @param _data Data to send as msg.data to the implementation to initialize the proxied contract.
   * It should include the signature and the parameters of the function to be called, as described in
   * https://solidity.readthedocs.io/en/v0.4.24/abi-spec.html#function-selector-and-argument-encoding
   * This parameter is optional, if no data is given the initialization call to proxied contract will
   */
  constructor(address _logic, bytes memory _data) public payable {
    assert(IMPLEMENTATION_SLOT == bytes32(uint256(keccak256('eip1967.proxy.implementation')) - 1));
    _setImplementation(_logic);
    if(_data.length > 0) {
      (bool success,) = _logic.delegatecall(_data);
      require(success);
    }
  }

  //function _willFallback() virtual override internal {
    //super._willFallback();
  //}
}


/**
 * @title AdminUpgradeabilityProxy
 * @dev Extends from BaseAdminUpgradeabilityProxy with a constructor for
 * initializing the implementation, admin, and init data.
 */
contract AdminUpgradeabilityProxy is BaseAdminUpgradeabilityProxy, UpgradeabilityProxy {
  /**
   * Contract constructor.
   * @param _logic address of the initial implementation.
   * @param _admin Address of the proxy administrator.
   * @param _data Data to send as msg.data to the implementation to initialize the proxied contract.
   * It should include the signature and the parameters of the function to be called, as described in
   * https://solidity.readthedocs.io/en/v0.4.24/abi-spec.html#function-selector-and-argument-encoding
   * This parameter is optional, if no data is given the initialization call to proxied contract will
   */
  constructor(address _admin, address _logic, bytes memory _data) UpgradeabilityProxy(_logic, _data)
    assert(ADMIN_SLOT == bytes32(uint256(keccak256('eip1967.proxy.admin')) - 1));
    _setAdmin(_admin);
  }

  function _willFallback() override(Proxy, BaseAdminUpgradeabilityProxy) internal {
    super._willFallback();
  }
}


/**
 * @title InitializableUpgradeabilityProxy
 * @dev Extends BaseUpgradeabilityProxy with an initializer for initializing
 * implementation and init data.
 */
abstract contract InitializableUpgradeabilityProxy is BaseUpgradeabilityProxy {
  /**
   * @dev Contract initializer.
   * @param _logic Address of the initial implementation.
   * @param _data Data to send as msg.data to the implementation to initialize the proxied contract.
   * It should include the signature and the parameters of the function to be called, as described in
   * https://solidity.readthedocs.io/en/v0.4.24/abi-spec.html#function-selector-and-argument-encoding
```

```
    * This parameter is optional, if no data is given the initialization call to proxied contract will
    */
  function initialize(address _logic, bytes memory _data) public payable {
    require(_implementation() == address(0));
    assert(IMPLEMENTATION_SLOT == bytes32(uint256(keccak256('eip1967.proxy.implementation')) - 1));
    _setImplementation(_logic);
    if(_data.length > 0) {
      (bool success,) = _logic.delegatecall(_data);
      require(success);
    }
  }
}


/**
 * @title InitializableAdminUpgradeabilityProxy
 * @dev Extends from BaseAdminUpgradeabilityProxy with an initializer for
 * initializing the implementation, admin, and init data.
 */
contract InitializableAdminUpgradeabilityProxy is BaseAdminUpgradeabilityProxy, InitializableUpgradea
  /**
   * Contract initializer.
   * @param _logic address of the initial implementation.
   * @param _admin Address of the proxy administrator.
   * @param _data Data to send as msg.data to the implementation to initialize the proxied contract.
   * It should include the signature and the parameters of the function to be called, as described in
   * https://solidity.readthedocs.io/en/v0.4.24/abi-spec.html#function-selector-and-argument-encoding
   * This parameter is optional, if no data is given the initialization call to proxied contract will
   */
  function initialize(address _admin, address _logic, bytes memory _data) public payable {
    require(_implementation() == address(0));
    InitializableUpgradeabilityProxy.initialize(_logic, _data);
    assert(ADMIN_SLOT == bytes32(uint256(keccak256('eip1967.proxy.admin')) - 1));
    _setAdmin(_admin);
  }

  function _willFallback() override(Proxy, BaseAdminUpgradeabilityProxy) internal {
    super._willFallback();
  }

}


interface IProxyFactory {
    function productImplementation() external view returns (address);
    function productImplementations(bytes32 name) external view returns (address);
}


/**
 * @title ProductProxy
 * @dev This contract implements a proxy that
 * it is deploied by ProxyFactory,
 * and it's implementation is stored in factory.
 */
contract ProductProxy is Proxy {

  /**
   * @dev Storage slot with the address of the ProxyFactory.
   * This is the keccak-256 hash of "eip1967.proxy.factory" subtracted by 1, and is
   * validated in the constructor.
   */
  bytes32 internal constant FACTORY_SLOT = 0x7a45a402e4cb6e08ebc196f20f66d5d30e67285a2a8aa80503fa409e

  function productName() virtual public pure returns (bytes32) {
    return 0x0;
```

```
  }

  /**
   * @dev Sets the factory address of the ProductProxy.
   * @param newFactory Address of the new factory.
   */
  function _setFactory(address newFactory) internal {
    require(OpenZeppelinUpgradesAddress.isContract(newFactory), "Cannot set a factory to a non-contra

    bytes32 slot = FACTORY_SLOT;

    assembly {
      sstore(slot, newFactory)
    }
  }

  /**
   * @dev Returns the factory.
   * @return factory Address of the factory.
   */
  function _factory() internal view returns (address factory) {
    bytes32 slot = FACTORY_SLOT;
    assembly {
      factory := sload(slot)
    }
  }

  /**
   * @dev Returns the current implementation.
   * @return Address of the current implementation
   */
  function _implementation() virtual override internal view returns (address) {
    address factory = _factory();
    if(OpenZeppelinUpgradesAddress.isContract(factory))
        return IProxyFactory(factory).productImplementations(productName());
    else
        return address(0);
  }

}


/**
 * @title InitializableProductProxy
 * @dev Extends ProductProxy with an initializer for initializing
 * factory and init data.
 */
contract InitializableProductProxy is ProductProxy {
  /**
   * @dev Contract initializer.
   * @param factory Address of the initial factory.
   * @param data Data to send as msg.data to the implementation to initialize the proxied contract.
   * It should include the signature and the parameters of the function to be called, as described in
   * https://solidity.readthedocs.io/en/v0.4.24/abi-spec.html#function-selector-and-argument-encoding
   * This parameter is optional, if no data is given the initialization call to proxied contract will
   */
  function initialize(address factory, bytes memory data) public payable {
    require(_factory() == address(0));
    assert(FACTORY_SLOT == bytes32(uint256(keccak256('eip1967.proxy.factory')) - 1));
    _setFactory(factory);
    if(data.length > 0) {
      (bool success,) = _implementation().delegatecall(data);
      require(success);
    }
  }
}
```

```
/**
 * Utility library of inline functions on addresses
 *
 * Source https://raw.githubusercontent.com/OpenZeppelin/openzeppelin-solidity/v2.1.3/contracts/utils
 * This contract is copied here and renamed from the original to avoid clashes in the compiled artifa
 * when the user imports a zos-lib contract (that transitively causes this contract to be compiled an
 * build/artifacts folder) as well as the vanilla Address implementation from an openzeppelin version
 */
library OpenZeppelinUpgradesAddress {
    /**
     * Returns whether the target address is a contract
     * @dev This function will return false if invoked during the constructor of a contract,
     * as the code is not actually created until after the constructor finishes.
     * @param account address of the account to check
     * @return whether the target address is a contract
     */
    function isContract(address account) internal view returns (bool) {
        uint256 size;
        // XXX Currently there is no better way to check if there is a contract in an address
        // than to check the size of the code at that address.
        // See https://ethereum.stackexchange.com/a/14016/36603
        // for more details about how this works.
        // TODO Check this again before the Serenity release, because all addresses will be
        // contracts then.
        // solhint-disable-next-line no-inline-assembly
        assembly { size := extcodesize(account) }
        return size > 0;
    }
}


/**
 * @title Initializable
 *
 * @dev Helper contract to support initializer functions. To use it, replace
 * the constructor with a function that has the `initializer` modifier.
 * WARNING: Unlike constructors, initializer functions must be manually
 * invoked. This applies both to deploying an Initializable contract, as well
 * as extending an Initializable contract via inheritance.
 * WARNING: When used with inheritance, manual care must be taken to not invoke
 * a parent initializer twice, or ensure that all initializers are idempotent,
 * because this is not dealt with automatically as with constructors.
 */
contract Initializable {

  /**
   * @dev Indicates that the contract has been initialized.
   */
  bool private initialized;

  /**
   * @dev Indicates that the contract is in the process of being initialized.
   */
  bool private initializing;

  /**
   * @dev Modifier to use in the initializer function of a contract.
   */
  modifier initializer() {
    require(initializing || isConstructor() || !initialized, "Contract instance has already been init

    bool isTopLevelCall = !initializing;
    if (isTopLevelCall) {
      initializing = true;
```

```
      initialized = true;
    }

    _;

    if (isTopLevelCall) {
      initializing = false;
    }
  }

  /**
   * @dev Modifier to use in the initializer function of a contract when upgrade EVEN times.
   */
  modifier initializerEven() {
    require(initializing || isConstructor() || initialized, "Contract instance has already been initi

    bool isTopLevelCall = !initializing;
    if (isTopLevelCall) {
      initializing = true;
      initialized = false;
    }

    _;

    if (isTopLevelCall) {
      initializing = false;
    }
  }

  /// @dev Returns true if and only if the function is running in the constructor
  function isConstructor() private view returns (bool) {
    // extcodesize checks the size of the code stored in an address, and
    // address returns the current address. Since the code is still not
    // deployed when running a constructor, any checks on its code size will
    // yield zero, making it an effective way to detect if a contract is
    // under construction or not.
    address self = address(this);
    uint256 cs;
    assembly { cs := extcodesize(self) }
    return cs == 0;
  }

  // Reserved storage space to allow for layout changes in the future.
  uint256[50] private ______gap;
}


contract Governable is Initializable {
    address public governor;

    event GovernorshipTransferred(address indexed previousGovernor, address indexed newGovernor);

    /**
     * @dev Contract initializer.
     * called once by the factory at time of deployment
     */
    function initialize(address governor_) virtual public initializer {
        governor = governor_;
        emit GovernorshipTransferred(address(0), governor);
    }

    modifier governance() {
        require(msg.sender == governor);
        _;
    }
```

```
    /**
     * @dev Allows the current governor to relinquish control of the contract.
     * @notice Renouncing to governorship will leave the contract without an governor.
     * It will not be possible to call the functions with the `governance`
     * modifier anymore.
     */
    function renounceGovernorship() public governance {
        emit GovernorshipTransferred(governor, address(0));
        governor = address(0);
    }

    /**
     * @dev Allows the current governor to transfer control of the contract to a newGovernor.
     * @param newGovernor The address to transfer governorship to.
     */
    function transferGovernorship(address newGovernor) public governance {
        _transferGovernorship(newGovernor);
    }

    /**
     * @dev Transfers control of the contract to a newGovernor.
     * @param newGovernor The address to transfer governorship to.
     */
    function _transferGovernorship(address newGovernor) internal {
        require(newGovernor != address(0));
        emit GovernorshipTransferred(governor, newGovernor);
        governor = newGovernor;
    }
}


contract Configurable is Governable {

    mapping (bytes32 => uint) internal config;

    function getConfig(bytes32 key) public view returns (uint) {
        return config[key];
    }
    function getConfig(bytes32 key, uint index) public view returns (uint) {
        return config[bytes32(uint(key) ^ index)];
    }
    function getConfig(bytes32 key, address addr) public view returns (uint) {
        return config[bytes32(uint(key) ^ uint(addr))];
    }

    function _setConfig(bytes32 key, uint value) internal {
        if(config[key] != value)
            config[key] = value;
    }
    function _setConfig(bytes32 key, uint index, uint value) internal {
        _setConfig(bytes32(uint(key) ^ index), value);
    }
    function _setConfig(bytes32 key, address addr, uint value) internal {
        _setConfig(bytes32(uint(key) ^ uint(addr)), value);
    }

    function setConfig(bytes32 key, uint value) external governance {
        _setConfig(key, value);
    }
    function setConfig(bytes32 key, uint index, uint value) external governance {
        _setConfig(bytes32(uint(key) ^ index), value);
    }
    function setConfig(bytes32 key, address addr, uint value) external governance {
        _setConfig(bytes32(uint(key) ^ uint(addr)), value);
    }
}
```

```
// helper methods for interacting with ERC20 tokens and sending ETH that do not consistently return t
library TransferHelper {
    function safeApprove(address token, address to, uint value) internal {
        // bytes4(keccak256(bytes('approve(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x095ea7b3, to, value))
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: APPROVE_F
    }

    function safeTransfer(address token, address to, uint value) internal {
        // bytes4(keccak256(bytes('transfer(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0xa9059cbb, to, value))
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: TRANSFER_
    }

    function safeTransferFrom(address token, address from, address to, uint value) internal {
        // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x23b872dd, from, to, v
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper: TRANSFER_
    }

    function safeTransferETH(address to, uint value) internal {
        (bool success,) = to.call{value:value}(new bytes(0));
        require(success, 'TransferHelper: ETH_TRANSFER_FAILED');
    }
}


/*
 * @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 ContextUpgradeSafe is Initializable {
    // Empty internal constructor, to prevent people from mistakenly deploying
    // an instance of this contract, which should be used via inheritance.

    function __Context_init() internal initializer {
        __Context_init_unchained();
    }

    function __Context_init_unchained() internal initializer {


    }


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

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

    uint256[50] private __gap;
}

/**
```

```
 * @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);
    }
}

/**
 * @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.
     */
    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.
     */
    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.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}


/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
```

```
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256 value);
}


contract UP is IERC20, ContextUpgradeSafe, Configurable {
    using SafeMath for uint;

    string public constant name = 'UU.finance Upvaluing LPT';
    string public constant symbol = 'UP';
    //uint8 public constant decimals = 18;

    address public uu;

    function initialize(address governor_, address uu_) public virtual initializer {
        ContextUpgradeSafe.__Context_init_unchained();
        Governable.initialize(governor_);
```

```
        __UP_init_unchained(uu_);
    }

    function __UP_init_unchained(address uu_) internal initializer {
        uu = uu_;
    }

    //function name() public view virtual returns (string memory) {
    //    return "UU.finance Upvaluing LPT";
    //}
//
    //function symbol() public view virtual returns (string memory) {
    //    return "UP";
    //}

    function decimals() public view virtual returns (uint8) {
        return UU(uu).decimals();
    }

    function totalSupply() external view virtual override returns (uint256) {
        return UU(uu).upTotalSupply();
    }

    function balanceOf(address account) external view virtual override returns (uint256) {
        return UU(uu).upBalanceOf(account);
    }

    function price() external view virtual returns (uint256) {
        return UU(uu).upPrice();
    }

    function allowance(address owner, address spender) public view virtual override returns (uint256)
        return UU(uu).upAllowance(owner, spender);
    }

    bytes32 public constant DOMAIN_TYPEHASH = keccak256("EIP712Domain(string name,uint256 chainId,add
    bytes32 public constant PERMIT_TYPEHASH = keccak256("Permit(address owner,address spender,uint256

    function nonces(address signatory) external view virtual returns (uint256) {
        return UU(uu).nonces(signatory);
    }

    function permit(address owner, address spender, uint rawAmount, uint deadline, uint8 v, bytes32 r
        UU(uu).upPermit_(name, address(this), owner, spender, rawAmount, deadline, v, r, s);
        emit Approval(owner, spender, rawAmount);
    }

    function approve(address spender, uint256 volume) public virtual override returns (bool success)
        success = UU(uu).upApprove_(_msgSender(), spender, volume);
        emit Approval(_msgSender(), spender, volume);
    }

    function increaseAllowance(address spender, uint256 increment) external virtual returns (bool) {
        return approve(spender, allowance(_msgSender(), spender).add(increment));
    }

    function decreaseAllowance(address spender, uint256 decrement) external virtual returns (bool) {
        return approve(spender, allowance(_msgSender(), spender).sub(decrement, "UP: decreased allowa
    }

    function transferFrom(address sender, address recipient, uint256 volume) external virtual overrid
        success = UU(uu).upTransferFrom_(_msgSender(), sender, recipient, volume);
        emit Transfer(sender, recipient, volume);
    }

    function transfer(address recipient, uint256 volume) external virtual override returns (bool succ
```

```
        success = UU(uu).upTransfer_(_msgSender(), recipient, volume);
        emit Transfer(_msgSender(), recipient, volume);
    }

    // Reserved storage space to allow for layout changes in the future.
    uint256[50] private ______gap;
}

contract UUBaseERC20 is IERC20, ContextUpgradeSafe {
    using SafeMath for uint;

    bytes32 public constant DOMAIN_TYPEHASH = keccak256("EIP712Domain(string name,uint256 chainId,add
    bytes32 public constant PERMIT_TYPEHASH = keccak256("Permit(address owner,address spender,uint256
    mapping (address => uint) public nonces;

    string public constant name = 'UU.finance Yielding United USD';
    string public constant symbol = 'UU';
    uint8 public constant decimals = 18;

    address public up;
    uint256 internal _upTotalSupply3;
    mapping (address => uint256) internal _upBalance3Of;
    mapping (address => mapping (address => uint256)) public upAllowance;
    uint256 public upPrice;

    function __UUBaseERC20_init(address up_) internal virtual initializer {
        ContextUpgradeSafe.__Context_init_unchained();
        __UUBaseERC20_init_unchained(up_);
    }

    function __UUBaseERC20_init_unchained(address up_) internal initializer {
        up = up_;
    }

    function totalSupply() public view virtual override returns (uint256) {
        return up2uu(upTotalSupply());
    }

    function balanceOf(address account) external view virtual override returns (uint256) {
        return up2uu(upBalanceOf(account));
    }

    function allowance(address owner, address spender) public view virtual override returns (uint256)
        return up2uu(upAllowance[owner][spender]);
    }

    function permit(address owner, address spender, uint amount, uint deadline, uint8 v, bytes32 r, b
        _upPermit(name, address(this), owner, spender, uu2up(amount), deadline, v, r, s);
        emit Approval(owner, spender, amount);
    }

    function approve(address spender, uint256 amount) public virtual override returns (bool success)
        success = _upApprove(_msgSender(), spender, uu2up(amount));
        emit Approval(_msgSender(), spender, amount);
    }

    function increaseAllowance(address spender, uint256 increment) external virtual returns (bool) {
        return approve(spender, allowance(_msgSender(), spender).add(increment));
    }

    function decreaseAllowance(address spender, uint256 decrement) external virtual returns (bool) {
        return approve(spender, allowance(_msgSender(), spender).sub(decrement, "UU: decreased allowa
    }

    function transferFrom(address sender, address recipient, uint256 amount) external virtual overrid
        success = _upTransferFrom(_msgSender(), sender, recipient, uu2up(amount));
```

```
        emit Transfer(sender, recipient, amount);
    }

    function transfer(address recipient, uint256 amount) external virtual override returns (bool succ
        success = _upTransfer(_msgSender(), recipient, uu2up(amount));
        emit Transfer(_msgSender(), recipient, amount);
    }

    function uu2up(uint256 amount) public view virtual returns (uint256) {
        return amount.mul(uint256(10)**decimals).div(upPrice);
    }

    function up2uu(uint256 vol) public view virtual returns (uint256) {
        return vol.mul(upPrice).div(uint256(10)**decimals);
    }

    function upTotalSupply() public view virtual returns (uint256) {
        return uint112(_upTotalSupply3);
    }

    function upBalanceOf(address account) public view virtual returns (uint256) {
        return uint112(_upBalance3Of[account]);
    }

    function _upPermit(string memory name_, address verifyingContract, address owner, address spender
        uint256 chainId;
        assembly { chainId := chainid() }
        bytes32 domainSeparator = keccak256(abi.encode(DOMAIN_TYPEHASH, keccak256(bytes(name_)), chai
        bytes32 structHash = keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, volume, nonces[own
        bytes32 digest = keccak256(abi.encodePacked("\x19\x01", domainSeparator, structHash));
        address signatory = ecrecover(digest, v, r, s);
        require(signatory != address(0), "UU.permit: invalid signature");
        require(signatory == owner, "UU.permit: unauthorized");
        require(now <= deadline, "UU.permit: signature expired");

        _upApprove(owner, spender, volume);
    }

    function _upApprove(address owner, address spender, uint256 volume) internal virtual returns (boo
        upAllowance[owner][spender] = volume;
        return true;
    }

    function _upTransferFrom(address spender, address sender, address recipient, uint256 volume) inte
        uint256 a = upAllowance[sender][spender];
        if(a != uint256(-1))
            upAllowance[sender][spender] = a.sub(volume);
        return _upTransfer(sender, recipient, volume);
    }

    function _upTransfer(address sender, address recipient, uint256 volume) internal virtual returns
        _upBalance3Of[sender]    = decreaseTAB(_upBalance3Of[sender],    volume);
        _upBalance3Of[recipient] = increaseTAB(_upBalance3Of[recipient], volume);
        return true;
    }

    function increaseTAB(uint256 tab, uint256 increment) internal view virtual returns (uint256) {
        (uint256 t, uint256 a, uint256 b) = unpackTAB(tab);
        a += b * (uint32(now) - t);                                 // + and * never overflows, and - ove
        b += increment;                                             // + never overflows
        t = now;
        return packTAB(t, a, b);
    }

    function decreaseTAB(uint256 tab, uint256 decrement) internal view virtual returns (uint256) {
        (uint256 t, uint256 a, uint256 b) = unpackTAB(tab);
```

```
        a += b * (uint32(now) -t);                                  // + and * never overflows, and - ove
        b = b.sub(decrement);
        t = now;
        return packTAB(t, a, b);
    }

    function packTAB(uint256 timestamp, uint256 coinAge, uint256 balance) internal pure virtual retur
        require(coinAge <= uint144(-1) && balance <= uint112(-1), 'TAB OVERFLOW');
        return timestamp << 224 | coinAge >> 32 << 112 | balance;
    }

    function unpackTAB(uint256 tab) internal pure virtual returns (uint256 timestamp, uint256 coinAge
        timestamp = tab >> 224;
        coinAge   = uint144(tab >> 112 << 32);
        balance   = uint112(tab);
    }

    //struct TAB {
    //    uint32 timestamp;
    //    uint112 coinAge;
    //    uint112 balance;
    //}

    modifier onlyUP {
        require(_msgSender() == up, 'Only be called by UP Token contract');
        _;
    }

    function upPermit_(string memory name_, address verifyingContract, address owner, address spender
        _upPermit(name_, verifyingContract, owner, spender, rawAmount, deadline, v, r, s);
    }

    function upApprove_(address owner, address spender, uint256 volume) external virtual onlyUP retur
        success = _upApprove(owner, spender, volume);
    }

    function upTransferFrom_(address spender, address sender, address recipient, uint256 volume) exte
        success = _upTransferFrom(spender, sender, recipient, volume);
    }

    function upTransfer_(address sender, address recipient, uint256 volume) external virtual onlyUP r
        success = _upTransfer(sender, recipient, volume);
    }

    // Reserved storage space to allow for layout changes in the future.
    uint256[50] private ______gap;
}

contract UUBaseMintable is UUBaseERC20, Configurable {
    using TransferHelper for address;

    bytes32 internal constant _netValueUnit_        = 'netValueUnit';
    bytes32 internal constant _netValueIndexOfLPT_  = 'netValueIndexOfLPT';
    bytes32 internal constant _depositOfLPT_        = 'depositOfLPT';
    bytes32 internal constant _swapOfLPT_           = 'swapOfLPT';
    bytes32 internal constant _lptOfSwap_           = 'lptOfSwap';
    bytes32 internal constant _gaugeOfLPT_          = 'gaugeOfLPT';
    bytes32 internal constant _lptTwapPeriod_       = 'lptTwapPeriod';

    uint256 public upPriceFactor;
    address[] public lpts;
    function lptN() public view returns (uint) {  return lpts.length;  }

    uint internal unlocked;
    modifier lock() {
        require(unlocked == 1, 'UU: LOCKED');
```

```
        unlocked = 0;
        _;
        unlocked = 1;
    }

    mapping (address => uint) public    lptTWAP;
    mapping (address => uint) internal _lptTwapTime;

    function __UUBaseMintable_init(address governor_, address up_) internal virtual initializer {
        ContextUpgradeSafe.__Context_init_unchained();
        UUBaseERC20.__UUBaseERC20_init_unchained(up_);
        Governable.initialize(governor_);
        __UUBaseMintable_init_unchained();
    }

    function __UUBaseMintable_init_unchained() internal initializer {
        _setConfig(_netValueUnit_,    1.0 ether);
        _setConfig(_lptTwapPeriod_,   60 minutes);
        upPriceFactor = 1 ether;
        unlocked = 1;
    }

    function addLPT(address lpt, address swap, address depo, uint nvi, address gauge) virtual public
        IERC20(lpt).totalSupply();
        StableSwap(swap).A();                                   // just for test
        Deposit(depo).calc_withdraw_one_coin(1 ether, int128(nvi));
        for(uint i=0; i<lpts.length; i++)
            require(lpts[i] != lpt, 'the lpt has already added');

        lpts.push(lpt);
        _setConfig(_netValueIndexOfLPT_, lpt, nvi);
        _setConfig(_depositOfLPT_,       lpt, uint(depo));
        _setConfig(_swapOfLPT_,          lpt, uint(swap));
        _setConfig(_lptOfSwap_,          swap, uint(lpt));
        _updateLptTWAP(lpt);

        emit AddLPT(lpt, swap, depo, nvi, gauge);
    }
    event AddLPT(address lpt, address swap, address depo, uint nvi, address gauge);

    function removeLPT(address lpt) virtual external governance {
        require(lptBalance(lpt) == 0, "Can't remove lpt which balanceOf(UU) is not 0");
        for(uint i=0; i<lpts.length; i++) {
            if(lpts[i] == lpt) {
                lpts[i] = lpts[lpts.length-1];
                lpts.pop();
                emit RemoveLPT(lpt);
                return;
            }
        }
        revert('the lpt is not in list');
    }
    event RemoveLPT(address lpt);

    function calcPrice() virtual public view returns (uint) {
        if(upTotalSupply() == 0)
            return upPriceFactor;
        uint amt = 0;
        for(uint i=0; i<lpts.length; i++)
            amt = amt.add(lptBalance(lpts[i]).mul(StableSwap(getConfig(_swapOfLPT_, lpts[i])).get_vir
        return amt.div(upTotalSupply()).mul(upPriceFactor).div(1 ether);
    }

    function _updatePrice() virtual internal {
        upPrice = calcPrice();
        emit UpdatePrice(upPrice, now);
```

```
    }
    event UpdatePrice(uint256 upPrice, uint256 timestamp);

    function _adjustPriceFactor() virtual internal {
        upPriceFactor = upPriceFactor.mul(upPrice).div(calcPrice());
    }

    function updatePrice() virtual public {
        _updatePrice();
        _upTotalSupply3 = increaseTAB(_upTotalSupply3, 0);
    }

    function lastUpdateTimeSpan() virtual public view returns(uint256) {
        return uint32(now - (_upTotalSupply3 >> 224));                          // overflow is desire
    }

    function netValue(address lpt, uint vol) virtual public view returns (uint amt) {
        address deposit = address(getConfig(_depositOfLPT_, lpt));
        require(deposit != address(0), 'Unsupported LPT');
        uint unit = getConfig(_netValueUnit_);
        if(unit == 0)
            unit = 1 ether;
        int128 i = int128(getConfig(_netValueIndexOfLPT_, lpt));
        amt = Deposit(deposit).calc_withdraw_one_coin(unit, i);
        amt = amt.mul(vol).div(unit);
    }

    function netValue(address lpt) virtual public view returns (uint amt) {
        return netValue(lpt, lptBalance(lpt));
    }

    function totalNetValue() virtual public view returns (uint amt) {
        for(uint i=0; i<lpts.length; i++)
            amt = amt.add(netValue(lpts[i]));
    }

    function lptBalance(address lpt) virtual public view returns (uint) {
        address gauge = address(getConfig(_gaugeOfLPT_, lpt));
        if(gauge != address(0))
            return Gauge(gauge).balanceOf(address(this));
        else
            return IERC20(lpt).balanceOf(address(this));
    }

    function lptPrice(address lpt) virtual public view returns (uint) {
        if(totalSupply() == 0)
            return StableSwap(getConfig(_swapOfLPT_, lpt)).get_virtual_price();
        else
            return netValue(lpt, uint256(10)**decimals).mul(totalSupply()).div(totalNetValue());
    }

    function _updateLptTWAP(address lpt) virtual internal {
        uint period = config[_lptTwapPeriod_];
        uint timestamp = _lptTwapTime[lpt];
        if(now > timestamp.add(period))
            lptTWAP[lpt] = lptPrice(lpt);
        else
            lptTWAP[lpt] = timestamp.add(period).sub(now).mul(lptPrice(lpt)).add(now.sub(timestamp).m
    }

    function lpt2uu(address lpt, uint vol) virtual public view returns (uint) {
        return Math.min(lptPrice(lpt), lptTWAP[lpt]).mul(vol).div(uint256(10)**decimals);
    }

    function uu2lpt(uint amt, address lpt) virtual public view returns (uint) {
        return Math.min(amt.mul(uint256(10)**decimals).div(Math.max(lptPrice(lpt), lptTWAP[lpt])), lp
```

```
    }

    function _mint(address to, uint amt) virtual internal returns (uint) {
        uint vol = uu2up(amt);
        _upTotalSupply3   = increaseTAB(_upTotalSupply3,   vol);
        _upBalance3Of[to] = increaseTAB(_upBalance3Of[to], vol);
        emit Transfer(address(0), to, amt);
        return amt;
    }

    function mint(address lpt, uint vol, uint minMint) virtual external lock returns (uint amt) {
        _updatePrice();
        amt = lpt2uu(lpt, vol);
        require(amt >= minMint, 'Slippage screwed you');
        lpt.safeTransferFrom(_msgSender(), address(this), vol);
        address gauge = address(getConfig(_gaugeOfLPT_, lpt));
        if(gauge != address(0)) {
            lpt.safeApprove(gauge, vol);
            Gauge(gauge).deposit(vol);
        }
        _mint(_msgSender(), amt);
        _updateLptTWAP(lpt);
        _adjustPriceFactor();
    }

    function _burn(address from, uint amt) virtual internal returns (uint) {
        uint vol = uu2up(amt);
        _upBalance3Of[from] = decreaseTAB(_upBalance3Of[from], vol);
        _upTotalSupply3     = decreaseTAB(_upTotalSupply3,     vol);
        emit Transfer(from, address(0), amt);
        return amt;
    }

    function burn(uint amt, address lpt, uint minVol) virtual external lock returns (uint vol) {
        _updatePrice();
        vol = uu2lpt(amt, lpt);
        if(vol == lptBalance(lpt))
            amt = lpt2uu(lpt, vol);
        _burn(_msgSender(), amt);
        require(vol >= minVol, 'Slippage screwed you');

        address gauge = address(getConfig(_gaugeOfLPT_, lpt));
        if(gauge != address(0))
            Gauge(gauge).withdraw(vol);
        lpt.safeTransfer(_msgSender(), vol);
        _updateLptTWAP(lpt);
        _adjustPriceFactor();
    }

    // Reserved storage space to allow for layout changes in the future.
    uint256[47] private ______gap;
}

contract UUBaseClaimable is UUBaseMintable {
    bytes32 internal constant _claimToTimeSpan_     = 'claimToTimeSpan';
    bytes32 internal constant _claimToTipRatio_     = 'claimToTipRatio';
    bytes32 internal constant _settleTipRatio_      = 'settleTipRatio';
    bytes32 internal constant _updateTipRatio_      = 'updateTipRatio';
    bytes32 internal constant _updateInterval_      = 'updateInterval';


    address[] public rewards;
    function rewardN() public view returns (uint) {  return rewards.length;  }
    mapping (address => mapping (address => uint256)) internal _claimed3;           // account => rew

    function __UUBaseClaimable_init(address governor_, address up_) internal virtual initializer {
```

```
        ContextUpgradeSafe.__Context_init_unchained();
        UUBaseERC20.__UUBaseERC20_init_unchained(up_);
        Governable.initialize(governor_);
        UUBaseMintable.__UUBaseMintable_init_unchained();
        __UUBaseClaimable_init_unchained();
    }

    function __UUBaseClaimable_init_unchained() internal initializer {
        _setConfig(_claimToTimeSpan_, 7 days);
        _setConfig(_claimToTipRatio_, 0.1 ether);
        _setConfig(_settleTipRatio_,  0.01 ether);
        _setConfig(_updateTipRatio_,  0.01 ether);
        _setConfig(_updateInterval_,  30 minutes);
    }

    function addLPT(address lpt, address swap, address depo, uint nvi, address gauge) virtual overrid
        super.addLPT(lpt, swap, depo, nvi, gauge);
        if(gauge != address(0)) {
            _setConfig(_gaugeOfLPT_,     lpt, uint(gauge));
            tryAddReward(Gauge(gauge).crv_token());
            //address reward = Gauge(gauge).rewarded_token();
            //if(reward != address(0))
            //    tryAddReward(reward);
        }
    }

    function tryAddReward(address reward) virtual public governance returns (bool success) {
        IERC20(reward).totalSupply();               // just for test
        for(uint i=0; i<rewards.length; i++)
            if(rewards[i] == reward)                // 'the reward has already added'
                return false;
        rewards.push(reward);
        emit AddReward(reward);
        return true;
    }
    event AddReward(address reward);

    function removeReward(address reward) virtual external governance returns (uint remain) {
        for(uint i=0; i<lpts.length; i++) {
            address gauge = address(getConfig(_gaugeOfLPT_, lpts[i]));
            require(Gauge(gauge).crv_token() == reward || Gauge(gauge).rewarded_token() == reward, 't
        }
        for(uint i=0; i<rewards.length; i++) {
            if(rewards[i] == reward) {
                rewards[i] = rewards[rewards.length-1];
                rewards.pop();
                remain = IERC20(reward).balanceOf(address(this));
                reward.safeTransfer(_msgSender(), remain);
                emit RemoveReward(reward, remain);
                return remain;
            }
        }
        revert('the lpt is not in list');
    }
    event RemoveReward(address reward, uint remain);

    function deltaCoinAge(uint256 balance3, uint256 claimaed3) virtual internal view returns (uint256
        (uint256 t, uint256 a, uint256 b) = unpackTAB(balance3);
        a += b * (uint32(now) - t);                                 // + and * never overflows, and - ove
        (timestamp, coinAge, claimed) = unpackTAB(claimaed3);
        delta = a.sub(coinAge);
    }

    function claimable(address acct, address reward) virtual public view returns (uint) {
        uint vol = IERC20(reward).balanceOf(address(this));
        if(vol == 0)
```

```
            return 0;
        (uint256 totalDelta, , ,) = deltaCoinAge(_upTotalSupply3, _claimed3[address(-1)][reward]);
        (uint256 delta, , ,) = deltaCoinAge(_upBalance3Of[acct], _claimed3[acct][reward]);
        return vol.mul(delta).div(totalDelta);
    }

    function _claimTo(address to, address reward) virtual internal returns (uint vol, uint tip) {
        vol = IERC20(reward).balanceOf(address(this));
        if(vol == 0)
            return (0, 0);
        (uint256 totalDelta, , uint256 totalCoinAge, uint256 totalClaimed) = deltaCoinAge(_upTotalSup
        (uint256 delta, uint timestamp, uint256 coinAge, uint256 claimed) = deltaCoinAge(_upBalance3O
        vol = vol.mul(delta).div(totalDelta);
        _claimed3[address(-1)][reward] = packTAB(now, totalCoinAge.add(delta), totalClaimed.add(vol))
        if(to != _msgSender()) {
            require(now > timestamp.add(getConfig(_claimToTimeSpan_)), 'not reach claimToTimeSpan');
            tip = vol.mul(getConfig(_claimToTipRatio_)).div(1 ether);
            vol = vol.sub(tip);
            (uint256 t, uint256 a, uint256 c) = unpackTAB(_claimed3[_msgSender()][reward]);
            _claimed3[_msgSender()][reward] = packTAB(t, a, c.add(tip));
        }
        _claimed3[to][reward] = packTAB(now, coinAge.add(delta), claimed.add(vol));
        reward.safeTransfer(to, vol);
        if(to != _msgSender())
            reward.safeTransfer(_msgSender(), tip);
        emit ClaimTo(to, reward, vol, tip, _msgSender());
    }
    event ClaimTo(address indexed to, address indexed reward, uint vol, uint tip, address indexed age

    function claimTo(address to, address reward) virtual external lock returns (uint vol, uint tip) {
        return _claimTo(to, reward);
    }

    function claim(address reward) virtual external lock returns (uint vol) {
        (vol, ) = _claimTo(_msgSender(), reward);
    }

    function claim() virtual external lock {
        for(uint i=0; i<rewards.length; i++)
            _claimTo(_msgSender(), rewards[i]);
    }

    function claimed(address acct, address reward) virtual public view returns (uint) {
        return uint112(_claimed3[acct][reward]);
    }

    function totalClaimed(address reward) virtual public view returns (uint) {
        return claimed(address(-1), reward);
    }

    function settleable(address lpt, uint j) virtual public view returns (address reward, uint vol, u
        address gauge = address(getConfig(_gaugeOfLPT_, lpt));
        if(gauge == address(0))
            return (address(0), 0, 0);
        if(j == 0) {
            reward = Gauge(gauge).crv_token();
            vol = Gauge(gauge).claimable_tokens(address(this));
        } else if(j == 1) {
            reward = Gauge(gauge).rewarded_token();
            vol = Gauge(gauge).claimable_reward(address(this));
        } else if(j == 2) {
            reward = Gauge(Gauge(gauge).reward_contract()).rewarded_token();
            vol = Gauge(gauge).claimable_reward2(address(this));
        }
        uint tipRatio = getConfig(_settleTipRatio_);
        if(lastUpdateTimeSpan() >= getConfig(_updateInterval_))
```

```
            tipRatio = tipRatio.add(getConfig(_updateTipRatio_));
        tip = vol.mul(tipRatio).div(1 ether);
        vol = vol.sub(tip);
    }

    function settle(address lpt, uint j) virtual external lock {
        address reward;
        address reward2;
        uint vol;
        uint vol2;
        address gauge = address(getConfig(_gaugeOfLPT_, lpt));
        if(gauge == address(0))
            return;
        if(j == 0) {
            reward = Gauge(gauge).crv_token();
            vol = IERC20(reward).balanceOf(address(this));
            Minter(Gauge(gauge).minter()).mint(gauge);
            vol = IERC20(reward).balanceOf(address(this)).sub(vol);
        } else if(j >= 1) {
            reward = Gauge(gauge).rewarded_token();
            vol = IERC20(reward).balanceOf(address(this));
            if(j >= 2) {
                reward2 = Gauge(Gauge(gauge).reward_contract()).rewarded_token();
                vol2 = IERC20(reward2).balanceOf(address(this));
            }
            Gauge(gauge).claim_rewards();
            vol = IERC20(reward).balanceOf(address(this)).sub(vol);
            if(j >= 2)
                vol2 = IERC20(reward2).balanceOf(address(this)).sub(vol2);
        }
        uint tipRatio = getConfig(_settleTipRatio_);
        uint interval = getConfig(_updateInterval_);
        if(lastUpdateTimeSpan() >= interval || now >= _lptTwapTime[lpt].add(interval)) {
            updatePrice();
            _updateLptTWAP(lpt);
            tipRatio = tipRatio.add(getConfig(_updateTipRatio_));
        }
        uint tip = vol.mul(tipRatio).div(1 ether);
        reward.safeTransfer(_msgSender(), tip);
        emit Settle(_msgSender(), gauge, reward, vol.sub(tip), tip);
        if(j >= 2) {
            uint tip2 = vol2.mul(tipRatio).div(1 ether);
            reward2.safeTransfer(_msgSender(), tip2);
            emit Settle(_msgSender(), gauge, reward2, vol2.sub(tip2), tip2);
        }
    }
    event Settle(address indexed agent, address indexed gauge, address indexed reward, uint vol, uint

    // Reserved storage space to allow for layout changes in the future.
    uint256[50] private ______gap;
}

contract UU is UUBaseClaimable {
    function initialize(address governor_, address up_) public virtual initializer {
        __UUBaseClaimable_init(governor_, up_);
        unlocked = 1;
    }

    // Reserved storage space to allow for layout changes in the future.
    uint256[50] private ______gap;
}

contract UUSwap {

    //function multi2uu(address swap, uint[] memory amts) virtual public view returns (uint vol) {
    //    address lpt = address(getConfig(_lptOfSwap_, swap));
```

```
//    require(lpt != address(0), 'Unsupported StableSwap contract');
//
//    if(amts.length == 2)
//        vol = StableSwap(swap).calc_token_amount([amts[0], amts[1]], true);
//    else if(amts.length == 3)
//        vol = StableSwap(swap).calc_token_amount([amts[0], amts[1], amts[2]], true);
//    else if(amts.length == 4)
//        vol = StableSwap(swap).calc_token_amount([amts[0], amts[1], amts[2], amts[3]], true);
//    else if(amts.length == 5)
//        vol = StableSwap(swap).calc_token_amount([amts[0], amts[1], amts[2], amts[3], amts[4]],
//    else if(amts.length == 6)
//        vol = StableSwap(swap).calc_token_amount([amts[0], amts[1], amts[2], amts[3], amts[4],
//    else if(amts.length == 7)
//        vol = StableSwap(swap).calc_token_amount([amts[0], amts[1], amts[2], amts[3], amts[4],
//    else if(amts.length == 8)
//        vol = StableSwap(swap).calc_token_amount([amts[0], amts[1], amts[2], amts[3], amts[4],
//    else
//        revert('Unsupported amts.length');
//
//    return lpt2uu(lpt, vol);
//}
//
//function usd2uu(address swap, address usd, uint amt) virtual public view returns (uint) {
//    uint n = getConfig(_NCoinsOfSwap_, swap);
//    require(n >= 2, 'Unsupported StableSwap contract');
//    uint[] memory amts = new uint[](n);
//    for(uint i=0; i<n; i++) {
//        if(StableSwap(swap).underlying_coins(int128(i)) == usd) {
//            amts[i] = amt;
//            return multi2uu(swap, amts);
//        }
//    }
//    revert('The swap do not support the usd');
//}

//function mint(address swap2, uint[] memory amts, uint minMint) virtual public returns (uint amt
//    address lpt = address(getConfig(_lptOfSwap_, swap2));
//    require(lpt != address(0), 'Unsupported StableSwap or Deposit contract');
//    uint vol = lptBalance(lpt);
//
//    if(amts.length == 2)
//        StableSwap(swap2).add_liquidity([amts[0], amts[1]], 0);
//    else if(amts.length == 3)
//        StableSwap(swap2).add_liquidity([amts[0], amts[1], amts[2]], 0);
//    else if(amts.length == 4)
//        StableSwap(swap2).add_liquidity([amts[0], amts[1], amts[2], amts[3]], 0);
//    else if(amts.length == 5)
//        StableSwap(swap2).add_liquidity([amts[0], amts[1], amts[2], amts[3], amts[4]], 0);
//    else if(amts.length == 6)
//        StableSwap(swap2).add_liquidity([amts[0], amts[1], amts[2], amts[3], amts[4], amts[5]],
//    else if(amts.length == 7)
//        StableSwap(swap2).add_liquidity([amts[0], amts[1], amts[2], amts[3], amts[4], amts[5],
//    else if(amts.length == 8)
//        StableSwap(swap2).add_liquidity([amts[0], amts[1], amts[2], amts[3], amts[4], amts[5],
//    else
//        revert('Unsupported amts.length');
//
//    vol = lptBalance(lpt).sub(vol);
//    amt = lpt2uu(lpt, vol);
//    require(amt >= minMint, 'Slippage screwed you');
//    return _mint(_msgSender(), amt);
//}
//
//function mint(address swap2, address usd, uint amt, uint minMint) virtual public returns (uint)
//    uint n = getConfig(_NCoinsOfSwap_, swap2);
//    require(n >= 2, 'Unsupported StableSwap or Deposit contract');
```

```
//    uint[] memory amts = new uint[](n);
//    for(uint i=0; i<n; i++) {
//        if(StableSwap(swap2).coins(int128(i)) == usd || StableSwap(swap2).underlying_coins(int1
//            amts[i] = amt;
//            return mint(swap2, amts, minMint);
//        }
//    }
//    revert('The swap2 do not support the usd');
//}

//function uu2usd(uint amt, address depo, address usd) virtual public view returns (uint) {
//    address lpt = address(getConfig(_lptOfSwap_, depo));
//    require(lpt != address(0), 'Unsupported Deposit contract');
//    uint vol = uu2lpt(amt, lpt);
//
//    int128 n = int128(getConfig(_NCoinsOfSwap_, depo));
//    require(n >= 2, 'Unsupported StableSwap contract');
//    for(int128 i=0; i<n; i++)
//        if(Deposit(depo).underlying_coins(i) == usd)
//            return Deposit(depo).calc_withdraw_one_coin(vol, i);
//
//    revert('The depo do not support the usd');
//}
//
//function uu4multi(address swap, uint[] memory amts) virtual public view returns (uint vol) {
//    address lpt = address(getConfig(_lptOfSwap_, swap));
//    require(lpt != address(0), 'Unsupported StableSwap contract');
//
//    if(amts.length == 2)
//        vol = StableSwap(swap).calc_token_amount([amts[0], amts[1]], false);
//    else if(amts.length == 3)
//        vol = StableSwap(swap).calc_token_amount([amts[0], amts[1], amts[2]], false);
//    else if(amts.length == 4)
//        vol = StableSwap(swap).calc_token_amount([amts[0], amts[1], amts[2], amts[3]], false);
//    else if(amts.length == 5)
//        vol = StableSwap(swap).calc_token_amount([amts[0], amts[1], amts[2], amts[3], amts[4]],
//    else if(amts.length == 6)
//        vol = StableSwap(swap).calc_token_amount([amts[0], amts[1], amts[2], amts[3], amts[4],
//    else if(amts.length == 7)
//        vol = StableSwap(swap).calc_token_amount([amts[0], amts[1], amts[2], amts[3], amts[4],
//    else if(amts.length == 8)
//        vol = StableSwap(swap).calc_token_amount([amts[0], amts[1], amts[2], amts[3], amts[4],
//    else
//        revert('Unsupported amts.length');
//
//    return lpt2uu(lpt, vol);
//}

//function burn(uint amt, address depo, address usd, uint minU) virtual public returns (uint u) {
//    address lpt = address(getConfig(_lptOfSwap_, depo));
//    require(lpt != address(0), 'Unsupported Deposit contract');
//
//    uint vol = uu2lpt(amt, lpt);
//    if(vol == lptBalance(lpt))
//        amt = lpt2uu(lpt, vol);
//    _burn(_msgSender(), amt);
//
//    int128 n = int128(getConfig(_NCoinsOfSwap_, depo));
//    require(n >= 2, 'Unsupported StableSwap contract');
//    for(int128 i=0; i<n; i++) {
//        if(Deposit(depo).underlying_coins(i) == usd) {
//            Deposit(depo).remove_liquidity_one_coin(vol, i, 0);
//            u = IERC20(usd).balanceOf(address(this));
//            require(u >= minU, 'Slippage screwed you');
//            usd.safeTransfer(_msgSender(), u);
//        }
```

```
//    }
//    revert('The depo do not support the usd');
//}
//
//function burn(uint amt, address swap2, uint[] calldata minAmts) virtual external returns (uint[
//    address lpt = address(getConfig(_lptOfSwap_, swap2));
//    require(lpt != address(0), 'Unsupported StableSwap or Deposit contract');
//
//    uint vol = uu2lpt(amt, lpt);
//    if(vol == lptBalance(lpt))
//        amt = lpt2uu(lpt, vol);
//    _burn(_msgSender(), amt);
//
//    if(minAmts.length == 2)
//        StableSwap(swap2).remove_liquidity(vol, [minAmts[0], minAmts[1]]);
//    else if(minAmts.length == 3)
//        StableSwap(swap2).remove_liquidity(vol, [minAmts[0], minAmts[1], minAmts[2]]);
//    else if(minAmts.length == 4)
//        StableSwap(swap2).remove_liquidity(vol, [minAmts[0], minAmts[1], minAmts[2], minAmts[3]
//    else if(minAmts.length == 5)
//        StableSwap(swap2).remove_liquidity(vol, [minAmts[0], minAmts[1], minAmts[2], minAmts[3]
//    else if(minAmts.length == 6)
//        StableSwap(swap2).remove_liquidity(vol, [minAmts[0], minAmts[1], minAmts[2], minAmts[3]
//    else if(minAmts.length == 7)
//        StableSwap(swap2).remove_liquidity(vol, [minAmts[0], minAmts[1], minAmts[2], minAmts[3]
//    else if(minAmts.length == 8)
//        StableSwap(swap2).remove_liquidity(vol, [minAmts[0], minAmts[1], minAmts[2], minAmts[3]
//    else
//        revert('Unsupported minAmts.length');
//
//    amts = new uint[](minAmts.length);
//    for(uint i=0; i<amts.length; i++) {
//        address coin = StableSwap(swap2).coins(int128(i));
//        amt = IERC20(coin).balanceOf(address(this));
//        if(amt > 0) {
//            coin.safeTransfer(_msgSender(), amt);
//            amts[i] = amt;
//        }
//        coin = StableSwap(swap2).underlying_coins(int128(i));
//        amt = IERC20(coin).balanceOf(address(this));
//        if(amt > 0) {
//            coin.safeTransfer(_msgSender(), amt);
//            amts[i] = amts[i].add(amt);
//        }
//    }
//}
//
//function burn(address swap2, uint[] calldata amts, uint maxBurn) virtual external returns (uint
//    address lpt = address(getConfig(_lptOfSwap_, swap2));
//    require(lpt != address(0), 'Unsupported StableSwap or Deposit contract');
//    uint vol = lptBalance(lpt);
//
//    uint maxVol = uint(-1); //uu2lpt(maxBurn, lpt);
//    if(amts.length == 2)
//        StableSwap(swap2).remove_liquidity_imbalance([amts[0], amts[1]], maxVol);
//    else if(amts.length == 3)
//        StableSwap(swap2).remove_liquidity_imbalance([amts[0], amts[1], amts[2]], maxVol);
//    else if(amts.length == 4)
//        StableSwap(swap2).remove_liquidity_imbalance([amts[0], amts[1], amts[2], amts[3]], maxV
//    else if(amts.length == 5)
//        StableSwap(swap2).remove_liquidity_imbalance([amts[0], amts[1], amts[2], amts[3], amts[
//    else if(amts.length == 6)
//        StableSwap(swap2).remove_liquidity_imbalance([amts[0], amts[1], amts[2], amts[3], amts[
//    else if(amts.length == 7)
//        StableSwap(swap2).remove_liquidity_imbalance([amts[0], amts[1], amts[2], amts[3], amts[
//    else if(amts.length == 8)
```

```
    //        StableSwap(swap2).remove_liquidity_imbalance([amts[0], amts[1], amts[2], amts[3], amts[
    //    else
    //        revert('Unsupported amts.length');
    //
    //    vol = vol.sub(lptBalance(lpt));
    //    amtBurn = lpt2uu(lpt, vol);
    //    require(amtBurn <= maxBurn, 'Slippage screwed you');
    //    _burn(_msgSender(), amtBurn);
    //
    //    for(uint i=0; i<amts.length; i++) {
    //        address coin = StableSwap(swap2).coins(int128(i));
    //        uint amt = IERC20(coin).balanceOf(address(this));
    //        if(amt > 0)
    //            coin.safeTransfer(_msgSender(), amt);
    //
    //        coin = StableSwap(swap2).underlying_coins(int128(i));
    //        amt = IERC20(coin).balanceOf(address(this));
    //        if(amt > 0)
    //            coin.safeTransfer(_msgSender(), amt);
    //    }
    //}

}


interface StableSwap {
    function A() external view returns (uint);
    function coins(int128) external view returns (address);
    function underlying_coins(int128) external view returns (address);

    function get_virtual_price() external view returns (uint);

    function calc_token_amount(uint[2] calldata amounts, bool deposit) external view returns (uint);
    function calc_token_amount(uint[3] calldata amounts, bool deposit) external view returns (uint);
    function calc_token_amount(uint[4] calldata amounts, bool deposit) external view returns (uint);
    function calc_token_amount(uint[5] calldata amounts, bool deposit) external view returns (uint);
    function calc_token_amount(uint[6] calldata amounts, bool deposit) external view returns (uint);
    function calc_token_amount(uint[7] calldata amounts, bool deposit) external view returns (uint);
    function calc_token_amount(uint[8] calldata amounts, bool deposit) external view returns (uint);

    function add_liquidity(uint[2] calldata amounts, uint min_mint_amount) external;
    function add_liquidity(uint[3] calldata amounts, uint min_mint_amount) external;
    function add_liquidity(uint[4] calldata amounts, uint min_mint_amount) external;
    function add_liquidity(uint[5] calldata amounts, uint min_mint_amount) external;
    function add_liquidity(uint[6] calldata amounts, uint min_mint_amount) external;
    function add_liquidity(uint[7] calldata amounts, uint min_mint_amount) external;
    function add_liquidity(uint[8] calldata amounts, uint min_mint_amount) external;

    function remove_liquidity(uint amount, uint[2] calldata min_amounts) external;
    function remove_liquidity(uint amount, uint[3] calldata min_amounts) external;
    function remove_liquidity(uint amount, uint[4] calldata min_amounts) external;
    function remove_liquidity(uint amount, uint[5] calldata min_amounts) external;
    function remove_liquidity(uint amount, uint[6] calldata min_amounts) external;
    function remove_liquidity(uint amount, uint[7] calldata min_amounts) external;
    function remove_liquidity(uint amount, uint[8] calldata min_amounts) external;

    function remove_liquidity_imbalance(uint[2] calldata amounts, uint max_burn_amount) external;
    function remove_liquidity_imbalance(uint[3] calldata amounts, uint max_burn_amount) external;
    function remove_liquidity_imbalance(uint[4] calldata amounts, uint max_burn_amount) external;
    function remove_liquidity_imbalance(uint[5] calldata amounts, uint max_burn_amount) external;
    function remove_liquidity_imbalance(uint[6] calldata amounts, uint max_burn_amount) external;
    function remove_liquidity_imbalance(uint[7] calldata amounts, uint max_burn_amount) external;
    function remove_liquidity_imbalance(uint[8] calldata amounts, uint max_burn_amount) external;
}
```

```
interface Deposit {
    function curve() external view returns (address);
    function underlying_coins(int128) external view returns (address);
    function calc_withdraw_one_coin(uint token_amount, int128 i) external view returns (uint);
    function remove_liquidity_one_coin(uint token_amount, int128 i, uint min_uamount) external;
}


interface Gauge {
    function minter()                       external view returns (address);
    function crv_token()                    external view returns (address);
    function rewarded_token()               external view returns (address);
    function reward_contract()              external view returns (address);
    function claimable_tokens(address addr) external view returns (uint);
    function claimable_reward(address addr) external view returns (uint);
    function claimable_reward2(address addr) external view returns (uint);
    function balanceOf(address)             external view returns (uint);

    function deposit(uint _value) external;
    function withdraw(uint _value) external;
    function claim_rewards() external;
}


interface Minter {
    function mint(address gauge) external;
}
```


Analysis of audit results

Re-Entrancy

- **Description:**
  One of the features of Ethereum smart contracts is the ability to call and utilise code of other external contracts. Contracts also typically handle ether, and as such often send ether to various external user addresses. The operation of calling external contracts, or sending ether to an address, requires the contract to submit an external call. These external calls can be hijacked by attackers whereby they force the contract to execute further code (i.e. through a fallback function) , including calls back into itself. Thus the code execution "re-enters" the contract. Attacks of this kind were used in the infamous DAO hack.
- **Detection results:**

  ```
  PASSED!
  ```

- **Security suggestion:**
  no.

Arithmetic Over/Under Flows

- **Description:**
  The Ethereum Virtual Machine (EVM) specifies fixed-size data types for integers. This means that an integer variable, only has a certain range of numbers it can represent. A uint8 for example, can only store numbers in the range [0,255]. Trying to store 256 into a uint8 will result in 0. If care is not taken, variables in Solidity can be exploited if user input is unchecked and calculations are performed which result in numbers that lie outside the range of the data type that stores them.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unexpected Ether

- **Description:**
  Typically when ether is sent to a contract, it must execute either the fallback function, or another function described in the contract. There are two exceptions to this, where ether can exist in a contract without having executed any code. Contracts which rely on code execution for every ether sent to the contract can be vulnerable to attacks where ether is forcibly sent to a contract.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Delegatecall

- **Description:**
  The CALL and DELEGATECALL opcodes are useful in allowing Ethereum developers to modularise their code. Standard external message calls to contracts are handled by the CALL opcode whereby code is run in the context of the external contract/function. The DELEGATECALL opcode is identical to the standard message call, except that the code executed at the targeted address is run in the context of the calling contract along with the fact that msg.sender and msg.value remain unchanged. This feature enables the implementation of libraries whereby developers can create reusable code for future contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Default Visibilities

- **Description:**
  Functions in Solidity have visibility specifiers which dictate how functions are allowed to be called. The visibility determines whether a function can be called externally by users, by other derived contracts, only internally or only externally. There are four visibility specifiers, which are described in detail in the Solidity Docs. Functions default to public allowing users to call them externally. Incorrect use of visibility specifiers can lead to some devestating vulernabilities in smart contracts as will be discussed in this section.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Entropy Illusion

- **Description:**
  All transactions on the Ethereum blockchain are deterministic state transition operations. Meaning that every transaction modifies the global state of the Ethereum ecosystem and it does so in a calculable way with no uncertainty. This ultimately means that inside the blockchain ecosystem there is no source of entropy or randomness. There is no rand() function in Solidity. Achieving decentralised entropy (randomness) is a well established problem and many ideas have been proposed to address this (see for example, RandDAO or using a chain of Hashes as described by Vitalik in this post).
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

External Contract Referencing

- **Description:**
  One of the benefits of the Ethereum global computer is the ability to re-use code and interact with contracts already deployed on the network. As a result, a large number of contracts reference external contracts and in general operation use external message calls to interact with these contracts. These external message calls can mask malicious actors intentions in some non-obvious ways, which we will discuss.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unsolved TODO comments

- **Description:**
  Check for Unsolved TODO comments
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Short Address/Parameter Attack

- **Description:**
  This attack is not specifically performed on Solidity contracts themselves but on third party applications that may interact with them. I add this attack for completeness and to be aware of how parameters can be manipulated in contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unchecked CALL Return Values

- **Description:**
  There a number of ways of performing external calls in solidity. Sending ether to external accounts is commonly performed via the transfer() method. However, the send() function can also be used and, for more versatile external calls, the CALL opcode can be directly employed in solidity. The call() and send() functions return a boolean indicating if the call succeeded or failed. Thus these functions have a simple caveat, in that the transaction that executes these functions will not revert if the external call (intialised by call() or send()) fails, rather the call() or send() will simply return false. A common pitfall arises when the return value is not checked, rather the developer expects a revert to occur.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Race Conditions / Front Running

- **Description:**
  The combination of external calls to other contracts and the multi-user nature of the underlying blockchain gives rise to a variety of potential Solidity pitfalls whereby users race code execution to obtain unexpected states. Re-Entrancy is one example of such a race condition. In this section we will talk more generally about different kinds of race conditions that can occur on the Ethereum blockchain. There is a variety of good posts on this subject, a few are: Ethereum Wiki - Safety, DASP - Front-Running and the Consensus - Smart Contract Best Practices.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Denial Of Service (DOS)

- **Description:**
  This category is very broad, but fundamentally consists of attacks where users can leave the contract inoperable for a small period of time, or in some cases, permanently. This can trap ether in these contracts forever, as was the case with the Second Parity MultiSig hack
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Block Timestamp Manipulation

- **Description:**
  Block timestamps have historically been used for a variety of applications, such as entropy for random numbers (see the Entropy Illusion section for further details), locking funds for periods of time and various state-changing

conditional statements that are time-dependent. Miner's have the ability to adjust timestamps slightly which can prove to be quite dangerous if block timestamps are used incorrectly in smart contracts.

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Constructors with Care

- **Description:**
  Constructors are special functions which often perform critical, privileged tasks when initialising contracts. Before solidity v0.4.22 constructors were defined as functions that had the same name as the contract that contained them. Thus, when a contract name gets changed in development, if the constructor name isn't changed, it becomes a normal, callable function. As you can imagine, this can (and has) lead to some interesting contract hacks.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unintialised Storage Pointers

- **Description:**
  The EVM stores data either as storage or as memory. Understanding exactly how this is done and the default types for local variables of functions is highly recommended when developing contracts. This is because it is possible to produce vulnerable contracts by inappropriately intialising variables.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Floating Points and Numerical Precision

- **Description:**
  As of this writing (Solidity v0.4.24), fixed point or floating point numbers are not supported. This means that floating point representations must be made with the integer types in Solidity. This can lead to errors/vulnerabilities if not implemented correctly.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

tx.origin Authentication

- **Description:**
  Solidity has a global variable, tx.origin which traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in smart contracts leaves the contract vulnerable to a phishing-like attack.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

armors.io

contact@armors.io