Notable

Code reviewed by human, conclusions are for reference only since no runtime checks or detections were made.

The *farming* `contract` takes user's deposit of one token and returns a reward amount of another token every period of time. It's working on a pair of **ERC20** tokens (D, R), where D is what users are going to **deposit**, R is what will be **rewarded** to users after a period of saving of D. Ideally every user should create his own one of `contract Farming` (line 637), it's supposed to generate token R in form of *rewards* based on the *amount* and *time* of savings of D (non-autonomously). Yet the `contract Farming` is like working in a open domain, thus any other users might operate on this contract unexpectedly. Which implies that a user might be able to find one of the highest `public rewardRate` (650), and *deposit* (712) to that one instead of creating his own for a better interest.

1 Suspicious Lines

1.1 Redundant initialization guard points 235

The `modifier initializer` (235) seems trying to ensure the `function initialize` (608) to be invoked once. Which is making no sense after checking it's starting point (675), should thus be removed.

1.2 Useless storage reservation in 265

The reserved `private ______gap` (265) is a waste, it won't serve as layout reservation in a *solidity* `contract`.

1.3 Misusing modifiers

These modifiers were improperly designed and used like `function`s (664, 759), which is opening security hole to the public. Including the wired implemented `function notifyRewardAmount` (780).

1.4 Dangerous public functions

The public functions (712, 719, 780) plus the improperly implemented modifiers (664, 759) together change *private* values yet shared in *public*, thus exposing important fields like the `public rewardRate` to everyone.

2 Verifications

Skipped.

3 Confirmations

Skipped.

4 Conclusions

The `contract Farming` (637) is supposed to protect *LP* (line 654) for all fields being used for calculation from others, specially the reward rate (650) and the farming supply (605).

The farming supply (605) is restricted to the *LP* only (733), which is good. But those misusing `modifer`s, such as `updateReward, checkhalve`, etc., does not have any such restriction from public. Thus people other than the *LP* may put a tiny stake (712) to a high rated *LP* contract for a better profit for the reward token. This is hidden **backdoor**.

Besides, the modifiers (664, 759) are not supposed to update the significant fields in this case. Especially almost without any restrictions and fallbacks. The author seems to made the `contract Farming` in a traditional request-respond (websites) mode, thus imposed many request based field updates. Which is dangerous, at lease in this case.

Other dangerous functions being so, because they open to the public without any concerns for changing states. Or changing significant states from a notification (780).

5 Code Copy

Note: referred lines are highlighted.

```
/**
 *Submitted for verification at BscScan.com on 2021-01-29
 */

// SPDX-License-Identifier: GPL-3.0-or-later

pragma solidity ^0.6.12;

library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others, `isContract` will return false for the following
     * types of addresses:
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies on extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        uint256 size;
        // solhint-disable-next-line no-inline-assembly
        assembly { size := extcodesize(account) }
        return size > 0;
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
```

```
40       * `recipient`, forwarding all available gas and reverting on
    ↪  errors.
41       *
42       * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases
    ↪  the gas cost
43       * of certain opcodes, possibly making contracts go over the
    ↪  2300 gas limit
44       * imposed by `transfer`, making them unable to receive funds
    ↪  via
45       * `transfer`. {sendValue} removes this limitation.
46       *
47       *
    ↪  https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn
    ↪  more].
48       *
49       * IMPORTANT: because control is transferred to `recipient`,
    ↪  care must be
50       * taken to not create reentrancy vulnerabilities. Consider
    ↪  using
51       * {ReentrancyGuard} or the
52       *
    ↪  https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-
    ↪  pattern].
53       */
54      function sendValue(address payable recipient, uint256 amount)
    ↪  internal {
55          require(address(this).balance >= amount, "Address:
    ↪  insufficient balance");
56
57          // solhint-disable-next-line avoid-low-level-calls,
    ↪  avoid-call-value
58          (bool success, ) = recipient.call{ value: amount }("");
59          require(success, "Address: unable to send value,
    ↪  recipient may have reverted");
60      }
61
62      /**
63       * @dev Performs a Solidity function call using a low level
    ↪  `call`. A
64       * plain`call` is an unsafe replacement for a function call:
    ↪  use this
65       * function instead.
66       *
67       * If `target` reverts with a revert reason, it is bubbled up
    ↪  by this
68       * function (like regular Solidity function calls).
```

```
69          *
70          * Returns the raw returned data. To convert to the expected
    ↪   return value,
71          * use
    ↪   https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi
72          *
73          * Requirements:
74          *
75          * - `target` must be a contract.
76          * - calling `target` with `data` must not revert.
77          *
78          * _Available since v3.1._
79          */
80      function functionCall(address target, bytes memory data)
    ↪   internal returns (bytes memory) {
81          return functionCall(target, data, "Address: low-level
    ↪   call failed");
82      }
83
84      /**
85          * @dev Same as
    ↪   {xref-Address-functionCall-address-bytes-}[`functionCall`],
    ↪   but with
86          * `errorMessage` as a fallback revert reason when `target`
    ↪   reverts.
87          *
88          * _Available since v3.1._
89          */
90      function functionCall(address target, bytes memory data,
    ↪   string memory errorMessage) internal returns (bytes memory) {
91          return functionCallWithValue(target, data, 0,
    ↪   errorMessage);
92      }
93
94      /**
95          * @dev Same as
    ↪   {xref-Address-functionCall-address-bytes-}[`functionCall`],
96          * but also transferring `value` wei to `target`.
97          *
98          * Requirements:
99          *
100         * - the calling contract must have an ETH balance of at
    ↪   least `value`.
101         * - the called Solidity function must be `payable`.
102         *
103         * _Available since v3.1._
```

```
104         */
105         function functionCallWithValue(address target, bytes memory
    ↪ data, uint256 value) internal returns (bytes memory) {
106             return functionCallWithValue(target, data, value,
    ↪ "Address: low-level call with value failed");
107         }
108
109         /**
110          * @dev Same as
    ↪ {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValue`],
    ↪ but
111          * with `errorMessage` as a fallback revert reason when
    ↪ `target` reverts.
112          *
113          * _Available since v3.1._
114          */
115         function functionCallWithValue(address target, bytes memory
    ↪ data, uint256 value, string memory errorMessage) internal
    ↪ returns (bytes memory) {
116             require(address(this).balance >= value, "Address:
    ↪ insufficient balance for call");
117             require(isContract(target), "Address: call to
    ↪ non-contract");
118
119             // solhint-disable-next-line avoid-low-level-calls
120             (bool success, bytes memory returndata) = target.call{
    ↪ value: value }(data);
121             return _verifyCallResult(success, returndata,
    ↪ errorMessage);
122         }
123
124         /**
125          * @dev Same as
    ↪ {xref-Address-functionCall-address-bytes-}[`functionCall`],
126          * but performing a static call.
127          *
128          * _Available since v3.3._
129          */
130         function functionStaticCall(address target, bytes memory
    ↪ data) internal view returns (bytes memory) {
131             return functionStaticCall(target, data, "Address:
    ↪ low-level static call failed");
132         }
133
134         /**
```

```
135      * @dev Same as
     ↪ {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
136      * but performing a static call.
137      *
138      * _Available since v3.3._
139      */
140     function functionStaticCall(address target, bytes memory
     ↪ data, string memory errorMessage) internal view returns
     ↪ (bytes memory) {
141         require(isContract(target), "Address: static call to
     ↪ non-contract");
142
143         // solhint-disable-next-line avoid-low-level-calls
144         (bool success, bytes memory returndata) =
     ↪ target.staticcall(data);
145         return _verifyCallResult(success, returndata,
     ↪ errorMessage);
146     }
147
148     /**
149      * @dev Same as
     ↪ {xref-Address-functionCall-address-bytes-}[`functionCall`],
150      * but performing a delegate call.
151      *
152      * _Available since v3.3._
153      */
154     function functionDelegateCall(address target, bytes memory
     ↪ data) internal returns (bytes memory) {
155         return functionDelegateCall(target, data, "Address:
     ↪ low-level delegate call failed");
156     }
157
158     /**
159      * @dev Same as
     ↪ {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
160      * but performing a delegate call.
161      *
162      * _Available since v3.3._
163      */
164     function functionDelegateCall(address target, bytes memory
     ↪ data, string memory errorMessage) internal returns (bytes
     ↪ memory) {
165         require(isContract(target), "Address: delegate call to
     ↪ non-contract");
166
167         // solhint-disable-next-line avoid-low-level-calls
```

```
168         (bool success, bytes memory returndata) =
    ↪  target.delegatecall(data);
169         return _verifyCallResult(success, returndata,
    ↪  errorMessage);
170     }
171
172     function _verifyCallResult(bool success, bytes memory
    ↪  returndata, string memory errorMessage) private pure
    ↪  returns(bytes memory) {
173         if (success) {
174             return returndata;
175         } else {
176             // Look for revert reason and bubble it up if present
177             if (returndata.length > 0) {
178                 // The easiest way to bubble the revert reason is
    ↪  using memory via assembly
179
180                 // solhint-disable-next-line no-inline-assembly
181                 assembly {
182                     let returndata_size := mload(returndata)
183                         revert(add(32, returndata),
    ↪  returndata_size)
184                         }
185             } else {
186                 revert(errorMessage);
187             }
188         }
189     }
190 }
191
192 /**
193  * @dev Standard math utilities missing in the Solidity language.
194  */
195 library Math {
196     /**
197      * @dev Returns the largest of two numbers.
198      */
199     function max(uint256 a, uint256 b) internal pure returns
    ↪  (uint256) {
200         return a >= b ? a : b;
201     }
202
203     /**
204      * @dev Returns the smallest of two numbers.
205      */
```

```
206     function min(uint256 a, uint256 b) internal pure returns
    ↪   (uint256) {
207         return a < b ? a : b;
208     }
209
210     /**
211      * @dev Returns the average of two numbers. The result is
    ↪   rounded towards
212      * zero.
213      */
214     function average(uint256 a, uint256 b) internal pure returns
    ↪   (uint256) {
215         // (a + b) / 2 can overflow, so we distribute
216         return (a / 2) + (b / 2) + ((a % 2 + b % 2) / 2);
217     }
218 }
219
220 contract Initializable {
221
222   /**
223    * @dev Indicates that the contract has been initialized.
224    */
225   bool private initialized;
226
227   /**
228    * @dev Indicates that the contract is in the process of being
    ↪    initialized.
229    */
230   bool private initializing;
231
232   /**
233    * @dev Modifier to use in the initializer function of a
    ↪    contract.
234    */
235   modifier initializer() { //←
236     require(initializing || isConstructor() || !initialized,
    ↪   "Contract instance has already been initialized");
237
238     bool isTopLevelCall = !initializing;
239     if (isTopLevelCall) {
240       initializing = true;
241       initialized = true;
242     }
243
244     _;
245
```

```
246       if (isTopLevelCall) {
247         initializing = false;
248       }
249     }
250
251     /// @dev Returns true if and only if the function is running in
    ↪  the constructor
252     function isConstructor() private view returns (bool) {
253       // extcodesize checks the size of the code stored in an
    ↪  address, and
254       // address returns the current address. Since the code is
    ↪  still not
255       // deployed when running a constructor, any checks on its
    ↪  code size will
256       // yield zero, making it an effective way to detect if a
    ↪  contract is
257       // under construction or not.
258       address self = address(this);
259       uint256 cs;
260       assembly { cs := extcodesize(self) }
261       return cs == 0;
262     }
263
264     // Reserved storage space to allow for layout changes in the
    ↪  future.
265     uint256[50] private ______gap; //←
266   }
267
268   interface IERC20 {
269       /**
270        * @dev Returns the amount of tokens in existence.
271        */
272       function totalSupply() external view returns (uint256);
273
274       /**
275        * @dev Returns the amount of tokens owned by `account`.
276        */
277       function balanceOf(address account) external view returns
    ↪  (uint256);
278
279       /**
280        * @dev Moves `amount` tokens from the caller's account to
    ↪  `recipient`.
281        *
282        * Returns a boolean value indicating whether the operation
    ↪  succeeded.
```

```
283         *
284         * Emits a {Transfer} event.
285         */
286        function transfer(address recipient, uint256 amount) external
    ↪   returns (bool);
287
288        /**
289         * @dev Returns the remaining number of tokens that `spender`
    ↪   will be
290         * allowed to spend on behalf of `owner` through
    ↪   {transferFrom}. This is
291         * zero by default.
292         *
293         * This value changes when {approve} or {transferFrom} are
    ↪   called.
294         */
295        function allowance(address owner, address spender) external
    ↪   view returns (uint256);
296
297        /**
298         * @dev Sets `amount` as the allowance of `spender` over the
    ↪   caller's tokens.
299         *
300         * Returns a boolean value indicating whether the operation
    ↪   succeeded.
301         *
302         * IMPORTANT: Beware that changing an allowance with this
    ↪   method brings the risk
303         * that someone may use both the old and the new allowance by
    ↪   unfortunate
304         * transaction ordering. One possible solution to mitigate
    ↪   this race
305         * condition is to first reduce the spender's allowance to 0
    ↪   and set the
306         * desired value afterwards:
307         *
    ↪   https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
308         *
309         * Emits an {Approval} event.
310         */
311        function approve(address spender, uint256 amount) external
    ↪   returns (bool);
312
313        /**
314         * @dev Moves `amount` tokens from `sender` to `recipient`
    ↪   using the
```

```
315         * allowance mechanism. `amount` is then deducted from the caller's
316         * allowance.
317         *
318         * Returns a boolean value indicating whether the operation succeeded.
319         *
320         * Emits a {Transfer} event.
321         */
322        function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
323
324        /**
325         * @dev Emitted when `value` tokens are moved from one account (`from`) to
326         * another (`to`).
327         *
328         * Note that `value` may be zero.
329         */
330        event Transfer(address indexed from, address indexed to, uint256 value);
331
332        /**
333         * @dev Emitted when the allowance of a `spender` for an `owner` is set by
334         * a call to {approve}. `value` is the new allowance.
335         */
336        event Approval(address indexed owner, address indexed spender, uint256 value);
337    }
338
339    //import "./IERC20.sol";
340    //import "../../math/SafeMath.sol";
341    //import "../../utils/Address.sol";
342
343    /**
344     * @title SafeERC20
345     * @dev Wrappers around ERC20 operations that throw on failure (when the token
346     * contract returns false). Tokens that return no value (and instead revert or
347     * throw on failure) are also supported, non-reverting calls are assumed to be
348     * successful.
349     * To use this library you can add a `using SafeERC20 for IERC20;` statement to your contract,
```

```
350  * which allows you to call the safe operations as
     ↪  `token.safeTransfer(...)`, etc.
351  */
352 library SafeERC20 {
353     using SafeMath for uint256;
354     using Address for address;
355
356     function safeTransfer(IERC20 token, address to, uint256
     ↪  value) internal {
357         _callOptionalReturn(token,
     ↪  abi.encodeWithSelector(token.transfer.selector, to, value));
358     }
359
360     function safeTransferFrom(IERC20 token, address from, address
     ↪  to, uint256 value) internal {
361         _callOptionalReturn(token,
     ↪  abi.encodeWithSelector(token.transferFrom.selector, from, to,
     ↪  value));
362     }
363
364     /**
365      * @dev Deprecated. This function has issues similar to the
     ↪  ones found in
366      * {IERC20-approve}, and its usage is discouraged.
367      *
368      * Whenever possible, use {safeIncreaseAllowance} and
369      * {safeDecreaseAllowance} instead.
370      */
371     function safeApprove(IERC20 token, address spender, uint256
     ↪  value) internal {
372         // safeApprove should only be called when setting an
     ↪  initial allowance,
373         // or when resetting it to zero. To increase and decrease
     ↪  it, use
374         // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
375         // solhint-disable-next-line max-line-length
376         require((value == 0) || (token.allowance(address(this),
     ↪  spender) == 0),
377             "SafeERC20: approve from non-zero to non-zero
     ↪  allowance"
378         );
379         _callOptionalReturn(token,
     ↪  abi.encodeWithSelector(token.approve.selector, spender,
     ↪  value));
380     }
381
```

```
382     function safeIncreaseAllowance(IERC20 token, address spender,
    ↪  uint256 value) internal {
383         uint256 newAllowance = token.allowance(address(this),
    ↪  spender).add(value);
384         _callOptionalReturn(token,
    ↪  abi.encodeWithSelector(token.approve.selector, spender,
    ↪  newAllowance));
385     }
386
387     function safeDecreaseAllowance(IERC20 token, address spender,
    ↪  uint256 value) internal {
388         uint256 newAllowance = token.allowance(address(this),
    ↪  spender).sub(value, "SafeERC20: decreased allowance below
    ↪  zero");
389         _callOptionalReturn(token,
    ↪  abi.encodeWithSelector(token.approve.selector, spender,
    ↪  newAllowance));
390     }
391
392     /**
393      * @dev Imitates a Solidity high-level call (i.e. a regular
    ↪  function call to a contract), relaxing the requirement
394      * on the return value: the return value is optional (but if
    ↪  data is returned, it must not be false).
395      * @param token The token targeted by the call.
396      * @param data The call data (encoded using abi.encode or one
    ↪  of its variants).
397      */
398     function _callOptionalReturn(IERC20 token, bytes memory data)
    ↪  private {
399         // We need to perform a low level call here, to bypass
    ↪  Solidity's return data size checking mechanism, since
400         // we're implementing it ourselves. We use
    ↪  {Address.functionCall} to perform this call, which verifies
    ↪  that
401         // the target address contains contract code and also
    ↪  asserts for success in the low-level call.
402
403         bytes memory returndata =
    ↪  address(token).functionCall(data, "SafeERC20: low-level call
    ↪  failed");
404         if (returndata.length > 0) { // Return data is optional
405             // solhint-disable-next-line max-line-length
406             require(abi.decode(returndata, (bool)), "SafeERC20:
    ↪  ERC20 operation did not succeed");
407         }
```

```
408     }
409 }
410
411 library SafeMath {
412     /**
413      * @dev Returns the addition of two unsigned integers, with
    ↪  an overflow flag.
414      */
415     function tryAdd(uint256 a, uint256 b) internal pure returns
    ↪  (bool, uint256) {
416         uint256 c = a + b;
417         if (c < a) return (false, 0);
418         return (true, c);
419     }
420
421     /**
422      * @dev Returns the substraction of two unsigned integers,
    ↪  with an overflow flag.
423      */
424     function trySub(uint256 a, uint256 b) internal pure returns
    ↪  (bool, uint256) {
425         if (b > a) return (false, 0);
426         return (true, a - b);
427     }
428
429     /**
430      * @dev Returns the multiplication of two unsigned integers,
    ↪  with an overflow flag.
431      */
432     function tryMul(uint256 a, uint256 b) internal pure returns
    ↪  (bool, uint256) {
433         // Gas optimization: this is cheaper than requiring 'a'
    ↪  not being zero, but the
434         // benefit is lost if 'b' is also tested.
435         // See:
    ↪  https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
436         if (a == 0) return (true, 0);
437         uint256 c = a * b;
438         if (c / a != b) return (false, 0);
439         return (true, c);
440     }
441
442     /**
443      * @dev Returns the division of two unsigned integers, with a
    ↪  division by zero flag.
444      */
```

```
445     function tryDiv(uint256 a, uint256 b) internal pure returns
    ↪   (bool, uint256) {
446         if (b == 0) return (false, 0);
447         return (true, a / b);
448     }
449
450     /**
451      * @dev Returns the remainder of dividing two unsigned
    ↪   integers, with a division by zero flag.
452      */
453     function tryMod(uint256 a, uint256 b) internal pure returns
    ↪   (bool, uint256) {
454         if (b == 0) return (false, 0);
455         return (true, a % b);
456     }
457
458     /**
459      * @dev Returns the addition of two unsigned integers,
    ↪   reverting on
460      * overflow.
461      *
462      * Counterpart to Solidity's `+` operator.
463      *
464      * Requirements:
465      *
466      * - Addition cannot overflow.
467      */
468     function add(uint256 a, uint256 b) internal pure returns
    ↪   (uint256) {
469         uint256 c = a + b;
470         require(c >= a, "SafeMath: addition overflow");
471         return c;
472     }
473
474     /**
475      * @dev Returns the subtraction of two unsigned integers,
    ↪   reverting on
476      * overflow (when the result is negative).
477      *
478      * Counterpart to Solidity's `-` operator.
479      *
480      * Requirements:
481      *
482      * - Subtraction cannot overflow.
483      */
```

```
484     function sub(uint256 a, uint256 b) internal pure returns
    ↪   (uint256) {
485         require(b <= a, "SafeMath: subtraction overflow");
486         return a - b;
487     }
488
489     /**
490      * @dev Returns the multiplication of two unsigned integers,
    ↪   reverting on
491      * overflow.
492      *
493      * Counterpart to Solidity's `*` operator.
494      *
495      * Requirements:
496      *
497      * - Multiplication cannot overflow.
498      */
499     function mul(uint256 a, uint256 b) internal pure returns
    ↪   (uint256) {
500         if (a == 0) return 0;
501         uint256 c = a * b;
502         require(c / a == b, "SafeMath: multiplication overflow");
503         return c;
504     }
505
506     /**
507      * @dev Returns the integer division of two unsigned
    ↪   integers, reverting on
508      * division by zero. The result is rounded towards zero.
509      *
510      * Counterpart to Solidity's `/` operator. Note: this
    ↪   function uses a
511      * `revert` opcode (which leaves remaining gas untouched)
    ↪   while Solidity
512      * uses an invalid opcode to revert (consuming all remaining
    ↪   gas).
513      *
514      * Requirements:
515      *
516      * - The divisor cannot be zero.
517      */
518     function div(uint256 a, uint256 b) internal pure returns
    ↪   (uint256) {
519         require(b > 0, "SafeMath: division by zero");
520         return a / b;
521     }
```

```
522
523     /**
524      * @dev Returns the remainder of dividing two unsigned
    ↪  integers. (unsigned integer modulo),
525      * reverting when dividing by zero.
526      *
527      * Counterpart to Solidity's `%` operator. This function uses
    ↪  a `revert`
528      * opcode (which leaves remaining gas untouched) while
    ↪  Solidity uses an
529      * invalid opcode to revert (consuming all remaining gas).
530      *
531      * Requirements:
532      *
533      * - The divisor cannot be zero.
534      */
535     function mod(uint256 a, uint256 b) internal pure returns
    ↪  (uint256) {
536         require(b > 0, "SafeMath: modulo by zero");
537         return a % b;
538     }
539
540     /**
541      * @dev Returns the subtraction of two unsigned integers,
    ↪  reverting with custom message on
542      * overflow (when the result is negative).
543      *
544      * CAUTION: This function is deprecated because it requires
    ↪  allocating memory for the error
545      * message unnecessarily. For custom revert reasons use
    ↪  {trySub}.
546      *
547      * Counterpart to Solidity's `-` operator.
548      *
549      * Requirements:
550      *
551      * - Subtraction cannot overflow.
552      */
553     function sub(uint256 a, uint256 b, string memory
    ↪  errorMessage) internal pure returns (uint256) {
554         require(b <= a, errorMessage);
555         return a - b;
556     }
557
558     /**
```

```
559         * @dev Returns the integer division of two unsigned
    ↪  integers, reverting with custom message on
560         * division by zero. The result is rounded towards zero.
561         *
562         * CAUTION: This function is deprecated because it requires
    ↪  allocating memory for the error
563         * message unnecessarily. For custom revert reasons use
    ↪  {tryDiv}.
564         *
565         * Counterpart to Solidity's `/` operator. Note: this
    ↪  function uses a
566         * `revert` opcode (which leaves remaining gas untouched)
    ↪  while Solidity
567         * uses an invalid opcode to revert (consuming all remaining
    ↪  gas).
568         *
569         * Requirements:
570         *
571         * - The divisor cannot be zero.
572         */
573        function div(uint256 a, uint256 b, string memory
    ↪  errorMessage) internal pure returns (uint256) {
574            require(b > 0, errorMessage);
575            return a / b;
576        }
577
578        /**
579         * @dev Returns the remainder of dividing two unsigned
    ↪  integers. (unsigned integer modulo),
580         * reverting with custom message when dividing by zero.
581         *
582         * CAUTION: This function is deprecated because it requires
    ↪  allocating memory for the error
583         * message unnecessarily. For custom revert reasons use
    ↪  {tryMod}.
584         *
585         * Counterpart to Solidity's `%` operator. This function uses
    ↪  a `revert`
586         * opcode (which leaves remaining gas untouched) while
    ↪  Solidity uses an
587         * invalid opcode to revert (consuming all remaining gas).
588         *
589         * Requirements:
590         *
591         * - The divisor cannot be zero.
592         */
```

```
593     function mod(uint256 a, uint256 b, string memory
    ↪   errorMessage) internal pure returns (uint256) {
594         require(b > 0, errorMessage);
595         return a % b;
596     }
597 }
598
599 contract StakePool is Initializable {
600     using SafeMath for uint256;
601     using SafeERC20 for IERC20;
602
603     IERC20 public depositToken;
604
605     uint256 private _totalSupply; //←
606     mapping(address => uint256) private _balances;
607
608     function initialize(address _token) public initializer { //←
609         depositToken = IERC20(_token);
610     }
611
612     function totalSupply() public view returns (uint256) {
613         return _totalSupply;
614     }
615
616     function balanceOf(address account) public view returns
    ↪   (uint256) {
617         return _balances[account];
618     }
619
620     function _stake(uint256 amount) internal {
621         _totalSupply = _totalSupply.add(amount);
622         _balances[msg.sender] =
    ↪   _balances[msg.sender].add(amount);
623         depositToken.safeTransferFrom(msg.sender, address(this),
    ↪   amount);
624     }
625
626     function _withdraw(uint256 amount) internal {
627         _totalSupply = _totalSupply.sub(amount);
628         _balances[msg.sender] =
    ↪   _balances[msg.sender].sub(amount);
629         depositToken.safeTransfer(msg.sender, amount);
630     }
631 }
632
633 /**
```

```
634  * Yield Token will be halved at each period.
635  */
636
637 contract Farming is StakePool {//←
638     // Yield Token as a reward for stakers
639     IERC20 public rewardToken;
640
641     // Halving period in seconds, should be defined as 7 days or
     ↪  1 week
642     uint256 public halvingPeriod = 604800;
643     // Total reward in 18 decimal
644     uint256 public totalreward;
645     // Starting timestamp for Staking Pool
646     uint256 public starttime;
647     // The timestamp when stakers should be allowed to withdraw
648     uint256 public stakingtime;
649     uint256 public eraPeriod = 0;
650     uint256 public rewardRate = 0; //←
651     uint256 public lastUpdateTime;
652     uint256 public rewardPerTokenStored;
653     uint256 public totalRewards = 0;
654     address private LPaddress; //←
655
656     mapping(address => uint256) public userRewardPerTokenPaid;
657     mapping(address => uint256) public rewards;
658
659     event RewardAdded(uint256 reward);
660     event Staked(address indexed user, uint256 amount);
661     event Withdrawn(address indexed user, uint256 amount);
662     event RewardPaid(address indexed user, uint256 reward);
663
664     modifier updateReward(address account) { //←
665         rewardPerTokenStored = rewardPerToken();
666         lastUpdateTime = lastTimeRewardApplicable();
667         if (account != address(0)) {
668             rewards[account] = earned(account);
669             userRewardPerTokenPaid[account] =
     ↪  rewardPerTokenStored;
670         }
671         _;
672     }
673
674     constructor(address _depositToken, address _rewardToken,
     ↪  uint256 _totalreward, uint256 _starttime, uint256
     ↪  _stakingtime) public {
675         super.initialize(_depositToken); //←
```

```
676         rewardToken = IERC20(_rewardToken);
677         LPaddress = msg.sender;
678         starttime = _starttime;
679         stakingtime = _stakingtime;
680         notifyRewardAmount(_totalreward.mul(50).div(100));
681     }
682 
683     function lastTimeRewardApplicable() public view returns
    ↪  (uint256) {
684         return Math.min(block.timestamp, eraPeriod);
685     }
686 
687     //Reward per LP token
688     function rewardPerToken() public view returns (uint256) {
689         if (totalSupply() == 0) {
690             return rewardPerTokenStored;
691         }
692         return
693             rewardPerTokenStored.add(
694                 lastTimeRewardApplicable()
695                 .sub(lastUpdateTime)
696                 .mul(rewardRate)
697                 .mul(1e18)
698                 .div(totalSupply())
699              );
700      }
701 
702     //Users earned reward
703      function earned(address account) public view returns
    ↪  (uint256) {
704          return
705              balanceOf(account)
706 
    ↪  .mul(rewardPerToken().sub(userRewardPerTokenPaid[account]))
707                  .div(1e18)
708                  .add(rewards[account]);
709      }
710 
711     //User stake LP token
712      function stake(uint256 amount) public
    ↪  updateReward(msg.sender) checkhalve checkStart{ //←
713          require(amount > 0, "ERROR: Cannot stake 0 Token");
714          super._stake(amount);
715          emit Staked(msg.sender, amount);
716      }
717 
```

```
718     //User withdraw staked LP token
719      function withdraw(uint256 amount) public
    ↪  updateReward(msg.sender) checkhalve checkStart stakingTime{
    ↪  // ←
720         require(amount > 0, "ERROR: Cannot withdraw 0");
721         super._withdraw(amount);
722         emit Withdrawn(msg.sender, amount);
723     }
724
725     //User withdraw staked LP token and rewards
726     function exit() external stakingTime{
727         withdraw(balanceOf(msg.sender));
728         _getRewardInternal();
729     }
730
731     //Start by depositing rewards
732     function depositReward() external {
733         require(msg.sender == LPaddress, "ERROR: Cannot
    ↪  deposit?"); //←
734         depositToken.safeTransfer(LPaddress, totalSupply());
735     }
736
737     //User withdraw rewards only
738      function getReward() public updateReward(msg.sender)
    ↪  checkhalve checkStart stakingTime{
739          uint256 reward = earned(msg.sender);
740          if (reward > 0) {
741              rewards[msg.sender] = 0;
742              rewardToken.safeTransfer(msg.sender, reward);
743              emit RewardPaid(msg.sender, reward);
744              totalRewards = totalRewards.add(reward);
745          }
746      }
747
748     //Update rewards
749      function _getRewardInternal() internal
    ↪  updateReward(msg.sender) checkhalve checkStart{
750          uint256 reward = earned(msg.sender);
751          if (reward > 0) {
752              rewards[msg.sender] = 0;
753              rewardToken.safeTransfer(msg.sender, reward);
754              emit RewardPaid(msg.sender, reward);
755              totalRewards = totalRewards.add(reward);
756          }
757      }
758
```

```
759     modifier checkhalve(){ //←
760         if (block.timestamp >= eraPeriod) {
761             totalreward = totalreward.mul(50).div(100);
762 
763             rewardRate = totalreward.div(halvingPeriod);
764             eraPeriod = block.timestamp.add(halvingPeriod);
765             emit RewardAdded(totalreward);
766         }
767         _;
768     }
769 
770     modifier checkStart(){
771         require(block.timestamp > starttime,"ERROR: Not started
    ↪  yet");
772         _;
773     }
774 
775     modifier stakingTime(){
776         require(block.timestamp >= stakingtime,"ERROR:
    ↪  Withdrawals not allowed yet");
777         _;
778     }
779 
780     function notifyRewardAmount(uint256 reward) //←
781         internal
782         updateReward(address(0))
783     {
784         if (block.timestamp >= eraPeriod) {
785             rewardRate = reward.div(halvingPeriod);
786         } else {
787             uint256 remaining = eraPeriod.sub(block.timestamp);
788             uint256 leftover = remaining.mul(rewardRate);
789             rewardRate =
    ↪  reward.add(leftover).div(halvingPeriod);
790         }
791         totalreward = reward;
792         lastUpdateTime = block.timestamp;
793         eraPeriod = block.timestamp.add(halvingPeriod);
794         emit RewardAdded(reward);
795     }
796 }
```