Degree of secrecy ★ | Project team open

Smart Contract Audit Report

Security status

Safe

★ ★ ★ ★ ★

Principal tester: **KnownSec blockchain security research team**

Release notes

Revised content	Time	Revised by	Version number
Written document	20201023	KnownSec blockchain security research team	V1.0

Document information

Document name	Document version number	Document number	Confidentiality level
CryptoAlpacaCity **Smart Contract Audit Report**	V1.0	CRYTOALPACACITY-ZNHY-20201023	Project team open

The statement

KnownSec only issues this report on the facts that have occurred or exist before the issuance of this report, and shall assume the corresponding responsibility therefor.KnownSec is not in a position to judge the security status of its smart contract and does not assume responsibility for the facts that occur or exist after issuance.The security audit analysis and other contents of this report are based solely on the documents and information provided by the information provider to KnownSec as of the issuance of this report.KnownSec assumes that the information provided was not missing, altered, truncated or suppressed.If the information provided is missing, altered, deleted, concealed or reflected in a way inconsistent with the actual situation, KnownSec shall not be liable for any loss or adverse effect caused thereby.

Directory

1. Review .... - 6 -

2. Code vulnerability analysis .... - 8 -

2.1 Vulnerability level distribution .... - 8 -

2.2 Summary of audit results .... - 9 -

3. Business Security Testing .... - 11 -

3.1. Pre-sale contract adoption function[Pass] .... - 11 -

3.2. Cryptoalpaca contract incubation function [Pass] .... - 12 -

3.3. Cryptoalpaca alpaca contract has a new function [Pass] .... - 16 -

3.4. Cryptoalpaca alpaca contract creates alpaca eggs and 0 generation alpaca functions [Pass] .... - 17 -

4. Basic code vulnerability detection .... - 21 -

4.1. Compiler version security [Pass] .... - 21 -

4.2. Redundant code [Pass] .... - 21 -

4.3. Use of safe arithmetic library [Pass] .... - 21 -

4.4. Not recommended encoding [Pass] .... - 21 -

4.5. Reasonable use of require/assert [Pass] .... - 22 -

4.6. fallback function safety [Pass] .... - 22 -

4.7. tx.orgin authentication [Pass] .... - 22 -

4.8. Owner permission control [Pass] .... - 23 -

4.9. Gas consumption detection [Pass] .... - 23 -

4.10. call injection attack [Pass] .... - 23 -

4.11. Low-level function safety [Pass] .... - 24 -

4.12. Vulnerability of additional token issuance [Pass] .... - 24 -

4.13. Access control defect detection [Pass] .... - 24 -

4.14. Numerical overflow detection [Pass] .... - 25 -

4.15. Arithmetic accuracy error [Pass] .... - 25 -

4.16. Wrong use of random number detection [Pass] .... - 26 -

4.17. Unsafe interface use [Pass] .... - 27 -

4.18. Variable coverage [Pass] .... - 27 -

4.19. Uninitialized storage pointer [Pass] .... - 27 -

4.20. Return value call verification [Pass] .... - 28 -

4.21. Transaction order dependency detection [Pass] .... - 28 -

4.22. Timestamp dependent attack [Pass] .... - 29 -

4.23. Denial of service attack detection [Pass] .... - 30 -

4.24. Fake recharge vulnerability detection [Pass] .... - 30 -

4.25. Reentry attack detection [Pass] .... - 31 -

4.26. Replay attack detection [Pass] .... - 31 -

4.27. Rearrangement attack detection [Pass] .... - 31 -

5. Appendix A: Contract code .... - 1 -

6. Appendix B: Vulnerability risk rating criteria .... - 29 -

7. Appendix C: Introduction to vulnerability testing tools .... - 30 -

7.1 Manticore .... - 30 -

7.2 Oyente .... - 30 -

7.3 securify. Sh - 30 -

7.4 Echidna - 30 -

7.5 MAIAN - 31 -

7.6 ethersplay - 31 -

7.7 IDA - evm entry - 31 -

7.8 want - ide - 31 -

7.9 KnownSec Penetration Tester kit - 31 -

1. Review

The effective test time of this report is from October 22, 2020 to October 23, 2020. During this period, the security and standardization of the CryptoAlpacaCity smart contract code will be audited and used as the statistical basis for the report.

In this test, KnownSec engineers conducted a comprehensive analysis of the common vulnerabilities of smart contracts (see Chapter 3), and the comprehensive evaluation was passed.

The results of this smart contract security audit: Pass

Since this test is conducted in a non-production environment, all codes are updated, the test process is communicated with the relevant interface personnel, and relevant test operations are carried out under the control of operational risks, so as to avoid production and operation risks and code security risks in the test process.

The target information of this test:

entry	description
Token name	AlpaToken
Code type	Token code, game contract, Ethereum smart contract code
Code language	solidity

Code file and hash :

The contract documents	MD5
AlpacaPresale.sol	005c7ef61a23611da2fbe7ec6bd9479a
DateControl.sol	ad595266ddf07b79591843711226a1a8
AlpaReward.sol	24c0020a68dde0eaf0245a2621a8523d
AlpaToken.sol	b2048ea4b18e0900c8157da61002121e

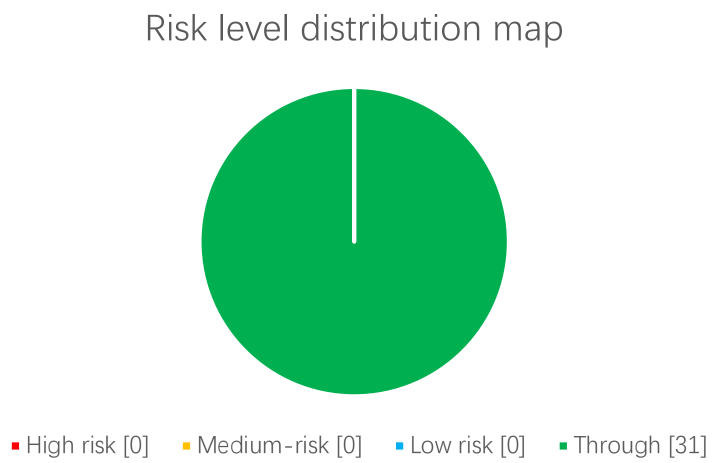
AlpacaBase.sol	93d08d3d0355f4b020652b39516dbdca
AlpacaBreed.sol	ed447768bbe5522ebab390f77c8a767b
AlpacaCore.sol	b0130134c37b62418ffca530b0c8d3b4
AlpacaToken.sol	1fd9fd097e1cdb79c6beae6d268419e0
GeneScience.sol	0c97d5dc35a53e279c931169f7357b9b
IAlpaToken.sol	1740d9f3a067f04c13897b2bb99abb99
ICryptoAlpaca.sol	248c32cebd223683334e2ed1a647bebc
IGeneScience.sol	d207e656c997b805ce4fbfbb44110b03
MasterChef.sol	3a794688ff104499c8577fa84282b273

2. Code vulnerability analysis

2.1 Vulnerability level distribution

This vulnerability risk is calculated by level:

Statistics on the number of security risk levels			
High Risk	Medium Risk	Low Risk	Pass
0	0	0	31



2.2 Summary of audit results

Audit results			
Audit item	Audit content	status	description
Cryptoalpaca contract incubation function Business security testing	Pre-sale contract adoption function	pass	After testing, there are no safety issues.
	Cryptoalpaca contract incubation function	pass	After testing, there are no safety issues.
	Cryptoalpaca contract has a new function	pass	After testing, there are no safety issues.
	Cryptoalpaca contract creates alpaca eggs and 0 generation alpaca functions	pass	After testing, there are no safety issues.
Basic code vulnerability detection	Compiler version security	pass	After testing, there are no safety issues.
	Redundant code	pass	After testing, there are no safety issues.
	Use of safe arithmetic library	pass	After testing, there are no safety issues.
	Not recommended encoding	pass	After testing, there are no safety issues.
	Reasonable use of require/assert	pass	After testing, there are no safety issues.
	fallback function safety	pass	After testing, there are no safety issues.
	tx.orgin authentication	pass	After testing, there are no safety issues.
	Owner permission control	pass	After testing, there are no safety issues.
	Gas consumption detection	pass	After testing, there are no safety issues.
	call injection attack	pass	After testing, there are no safety issues.
	Low-level function safety	pass	After testing, there are no safety issues.

	Vulnerability of additional token issuance	pass	After testing, there are no safety issues.
	Access control defect detection	pass	After testing, there are no safety issues.
	Numerical overflow detection	pass	After testing, there are no safety issues.
	Arithmetic accuracy error	pass	After testing, there are no safety issues.
	Wrong use of random number detection	pass	After testing, there are no safety issues.
	Unsafe interface use	pass	After testing, there are no safety issues.
	Variable coverage	pass	After testing, there are no safety issues.
	Uninitialized storage pointer	pass	After testing, there are no safety issues.
	Return value call verification	pass	After testing, there are no safety issues.
	Transaction order dependency detection	pass	After testing, there are no safety issues.
	Timestamp dependent attack	pass	After testing, there are no safety issues.
	Denial of service attack detection	pass	After testing, there are no safety issues.
	Fake recharge vulnerability detection	pass	After testing, there are no safety issues.
	Reentry attack detection	pass	After testing, there are no safety issues.
	Replay attack detection	pass	After testing, there are no safety issues.
	Rearrangement attack detection	pass	After testing, there are no safety issues.

3. Business Security Testing

3.1. Pre-sale contract adoption function[Pass]

Audit analysis: The adoption function of the cryptoalpaca pre-sale contract is implemented by the adoptAlpaca function in the AlpacaPresale.sol contract file, which is used to adopt alpaca during the pre-sale.

```
function adoptAlpaca(uint256 _count)
    public
    payable
    whenInProgress
    nonReentrant
{//knownsec// Adopt alpaca
    require(_count > 0, "AlpacaPresale: must adopt at least one alpaca");//knownsec// Check quantity
    require(whitelist.contains(msg.sender), "AlpacaPresale: unauthorized");//knownsec// Check the whitelist

    address account = msg.sender;
    uint256 credit = canAdoptCount(account);
    require(//knownsec// Check the number of adoptions
        _count <= credit,
        "AlpacaPresale: adoption count larger than maximum adoption limit"
    );

    require(//knownsec// Verify the transfer limit
        msg.value >= getAdoptionPrice(_count),
        "AlpacaPresale: insufficient funds"
    );

    uint256[] memory ids = new uint256[](_count);
```

```
    uint256[] memory counts = new uint256[](_count);
    for (uint256 i = 0; i < _count; i++) {
        ids[i] = _randRemoveAlpaca();//knownsec// Random Alpaca
        counts[i] = 1;
    }

    accountAddoptionCount[account] += _count;//knownsec// Accumulate the number of adoptions

    cryptoAlpaca.safeBatchTransferFrom(//knownsec// Transfer Alpaca
        address(this),
        account,
        ids,
        counts,
        ""
    );
}
```

Safety advice: None.

3.2. Cryptoalpaca contract incubation function [Pass]

Audit analysis: The incubation function of the cryptoalpaca contract is mainly realized by the hatch and _hatchEgg functions in the AlpacaBreed.sol contract file. In the hatch function, various parameters are checked first, and then the private _hatchEgg function is called to achieve incubation.

```
function hatch(uint256 _matronId, uint256 _sireId)
    external
    override
    payable
    whenNotPaused
```

```
    nonReentrant
    returns (uint256)
{
    address msgSender = msg.sender;

    // Checks for payment.
    require(
        msg.value >= autoCrackingFee,
        "CryptoAlpaca: Required autoCrackingFee not sent"
    );

    // Checks for ALPA payment
    require(
        alpa.allowance(msgSender, address(this)) >=
            _hatchingALPACost(_matronId, _sireId, true),
        "CryptoAlpaca: Required hetching ALPA fee not sent"
    );

    // Checks if matron and sire are valid mating pair
    require(
        _ownerPermittedToBreed(msgSender, _matronId, _sireId),
        "CryptoAlpaca: Invalid permission"
    );

    // Grab a reference to the potential matron
    Alpaca storage matron = alpacas[_matronId];

    // Make sure matron isn't pregnant, or in the middle of a siring cooldown
    require(
        _isReadyToHatch(matron),
        "CryptoAlpaca: Matron is not yet ready to hatch"
    );
```

```
        // Grab a reference to the potential sire
        Alpaca storage sire = alpacas[_sireId];

        // Make sure sire isn't pregnant, or in the middle of a siring cooldown
        require(
            _isReadyToHatch(sire),
            "CryptoAlpaca: Sire is not yet ready to hatch"
        );

        // Test that matron and sire are a valid mating pair.
        require(
            _isValidMatingPair(matron, _matronId, sire, _sireId),
            "CryptoAlpaca: Matron and Sire are not valid mating pair"
        );

        // All checks passed, Alpaca gets pregnant!
        return _hatchEgg(_matronId, _sireId);
}
function _hatchEgg(uint256 _matronId, uint256 _sireId)
        private
        returns (uint256)
{
        // Transfer birthing ALPA fee to this contract
        uint256 alpaCost = _hatchingALPACost(_matronId, _sireId, true);

        uint256 devAmount = alpaCost.mul(devBreedingPercentage).div(100);
        uint256 stakingAmount = alpaCost.mul(100 - devBreedingPercentage).div(
            100
        );
        //knownsec// Calculate processing related costs
        assert(alpa.transferFrom(msg.sender, devAddress, devAmount));
        assert(alpa.transferFrom(msg.sender, stakingAddress, stakingAmount));
```

```
// Grab a reference to the Alpacas from storage.
Alpaca storage sire = alpacas[_sireId];
Alpaca storage matron = alpacas[_matronId];

// refresh hatching multiplier for both parents.
_refreshHatchingMultiplier(sire);
_refreshHatchingMultiplier(matron);

// Determine the lower generation number of the two parents
uint256 parentGen = matron.generation;
if (sire.generation < matron.generation) {//knownsec// Lower generation first
    parentGen = sire.generation;
}

// child generation will be 1 larger than min of the two parents generation;
uint256 childGen = parentGen.add(1);

// Determine when the egg will be cracked
uint256 cooldownEndBlock = (hatchingDuration.div(secondsPerBlock)).add(
    block.number
);

uint256 eggID = _createEgg(//knownnsec// Create alpaca eggs
    _matronId,
    _sireId,
    childGen,
    cooldownEndBlock,
    msg.sender
);

// Emit the hatched event.
emit Hatched(eggID, _matronId, _sireId, cooldownEndBlock);
```

```
    return eggID;
}
```

Safety advice: None.

3.3. Cryptoalpaca alpaca contract has a new function [Pass]

Audit analysis: The shell-breaking function of the cryptoalpaca contract is implemented by the crack function in the AlpacaBreed.sol contract file, which is used to hatch alpaca eggs with a specified id.

```
function crack(uint256 _id) external override nonReentrant {
    // Grab a reference to the egg in storage.
    Alpaca storage egg = alpacas[_id];

    // Check that the egg is a valid alpaca.
    require(egg.birthTime != 0, "CryptoAlpaca: not valid egg");
    require(//knownsec// Check status
        egg.state == AlpacaGrowthState.EGG,
        "CryptoAlpaca: not a valid egg"
    );

    // Check that the matron is pregnant, and that its time has come!
    require(_isReadyToCrack(egg), "CryptoAlpaca: egg cant be cracked yet");//knownsec// Check if it can hatch

    // Grab a reference to the sire in storage.
    Alpaca storage matron = alpacas[egg.matronId];
    Alpaca storage sire = alpacas[egg.sireId];

    // Call the sooper-sekret gene mixing operation.
    (
        uint256 childGene,
```

```
        uint256 childEnergy,
        uint256 generationFactor
    ) = geneScience.mixGenes(//knownsec// Get parameters related to newborn alpaca
        matron.gene,
        sire.gene,
        egg.generation,
        uint256(egg.cooldownEndBlock).sub(1)
    );

    egg.gene = childGene;
    egg.energy = uint32(childEnergy);
    egg.state = AlpacaGrowthState.GROWN;
    egg.cooldownEndBlock = uint64(
        (newBornCoolDown.div(secondsPerBlock)).add(block.number)
    );
    egg.generationFactor = uint64(generationFactor);

    // Send the balance fee to the person who made birth happen.
    if (autoCrackingFee > 0) {//knownsec// Incubation fee
        msg.sender.transfer(autoCrackingFee);
    }

    // emit the born event
    emit BornSingle(_id, childGene, childEnergy);
}
```

Safety advice: None.

3.4. Cryptoalpaca alpaca contract creates alpaca eggs and 0 generation alpaca functions [Pass]

Audit analysis: The function of creating alpaca eggs and generation 0 alpaca in cryptoalpaca contract is only implemented by the _createEgg and _createGen0Alpaca functions in the AlpacaToken.sol contract file, which is used to create alpaca eggs during incubation and the contract owner creates generation 0 sheep camel.

```
function _createEgg(
    uint256 _matronId,
    uint256 _sireId,
    uint256 _generation,
    uint256 _cooldownEndBlock,
    address _owner
) internal returns (uint256) {//knownsec// Check previous generation information
    require(_matronId == uint256(uint32(_matronId)));
    require(_sireId == uint256(uint32(_sireId)));
    require(_generation == uint256(uint16(_generation)));

    Alpaca memory _alpaca = Alpaca({
        gene: 0,
        energy: 0,
        birthTime: uint64(now),
        hatchCostMultiplierEndBlock: 0,
        hatchingCostMultiplier: 1,
        matronId: uint32(_matronId),
        sireId: uint32(_sireId),
        cooldownEndBlock: uint64(_cooldownEndBlock),
        generation: uint16(_generation),
        generationFactor: 0,
        state: AlpacaGrowthState.EGG
    });

    alpacas.push(_alpaca);
    uint256 eggId = alpacas.length - 1;
```

```
    _mint(_owner, eggId, 1, "");

    return eggId;
}
function _createGen0Alpaca(
    uint256 _gene,
    uint256 _energy,
    address _owner
) internal returns (uint256) {//knownsec// Check the maximum energy
    require(_energy <= MAX_GEN0_ENERGY, "CryptoAlpaca: invalid energy");

    Alpaca memory _alpaca = Alpaca({
        gene: _gene,
        energy: uint32(_energy),
        birthTime: uint64(now),
        hatchCostMultiplierEndBlock: 0,
        hatchingCostMultiplier: 1,
        matronId: 0,
        sireId: 0,
        cooldownEndBlock: 0,
        generation: 0,
        generationFactor: GEN0_GENERATION_FACTOR,
        state: AlpacaGrowthState.GROWN
    });

    alpacas.push(_alpaca);
    uint256 newAlpacaID = alpacas.length - 1;

    _mint(_owner, newAlpacaID, 1, "");

    // emit the born event
    emit BornSingle(newAlpacaID, _gene, _energy);
```

```
    return newAlpacaID;
}
```

Safety advice: None.

4. Basic code vulnerability detection

4.1. Compiler version security [Pass]

Check whether a safe compiler version is used in the contract code implementation.

Test result: After testing, the compiler version 0.6.2 is formulated in the smart contract code, and there is no such security issue.

Safety advice: None.

4.2. Redundant code [Pass]

Check whether the contract code implementation contains redundant code.

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

4.3. Use of safe arithmetic library [Pass]

Check whether the SafeMath safe arithmetic library is used in the contract code implementation

Test result: After testing, the SafeMath safe arithmetic library has been used in the smart contract code, and there is no such security problem.

Safety advice: None.

4.4. Not recommended encoding [Pass]

Check whether there is an encoding method that is not officially recommended or abandoned in the contract code implementation

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

4.5. Reasonable use of require/assert [Pass]

Check the rationality of the use of require and assert statements in the contract code implementation

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

4.6. fallback function safety [Pass]

Check whether the fallback function is used correctly in the contract code implementation

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

4.7. tx.orgin authentication [Pass]

tx.origin is a global variable of Solidity that 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 a smart contract makes the contract vulnerable to attacks like phishing.

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

4.8. **Owner permission control** [Pass]

Check whether the owner in the contract code implementation has excessive authority. For example, arbitrarily modify other account balances, etc.

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

4.9. **Gas consumption detection** [Pass]

Check whether the consumption of gas exceeds the maximum block limit

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

4.10. **call injection attack** [Pass]

When the call function is called, strict permission control should be done, or the function called by the call should be written dead.

Detection result: After detection, the smart contract does not use the call function, and this vulnerability does not exist.

Safety advice: None.

4.11. Low-level function safety [Pass]

Check whether there are security vulnerabilities in the use of low-level functions (call/delegatecall) in the contract code implementation

The execution context of the call function is in the called contract; the execution context of the delegatecall function is in the contract that currently calls the function

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

4.12. Vulnerability of additional token issuance [Pass]

Check whether there is a function that may increase the total amount of tokens in the token contract after initializing the total amount of tokens.

Test result: After testing, the smart contract code has the function of issuing additional tokens, but because liquid mining requires additional tokens, it is approved.

Safety advice: None.

4.13. Access control defect detection [Pass]

Different functions in the contract should set reasonable permissions

Check whether each function in the contract correctly uses keywords such as public and private for visibility modification, check whether the contract is correctly defined and use modifier to restrict access to key functions to avoid problems caused by unauthorized access.

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

4.14. **Numerical overflow detection** [Pass]

The arithmetic problems in smart contracts refer to integer overflow and integer underflow.

Solidity can handle up to 256-bit numbers (2^256-1). If the maximum number increases by 1, it will overflow to 0. Similarly, when the number is an unsigned type, 0 minus 1 will underflow to get the maximum digital value.

Integer overflow and underflow are not a new type of vulnerability, but they are especially dangerous in smart contracts. Overflow conditions can lead to incorrect results, especially if the possibility is not expected, which may affect the reliability and safety of the program.

Safety advice: None.

4.15. **Arithmetic accuracy error** [Pass]

As a programming language, Solidity has data structure design similar to ordinary programming languages, such as variables, constants, functions, arrays, functions, structures, etc. There is also a big difference between Solidity and ordinary programming languages-Solidity does not float Point type, and all the numerical calculation results of Solidity will only be integers, there will be no decimals, and it is not allowed to define decimal type data. Numerical calculations in the contract are indispensable, and the design of numerical calculations may cause relative errors. For example, the same level of calculations: 5/2*10=20, and 5*10/2=25, resulting in errors, which are larger in data The error will be larger and more obvious.

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

4.16. Wrong use of random number detection [Pass]

Smart contracts may need to use random numbers. Although the functions and variables provided by Solidity can access values that are obviously unpredictable, such as block.number and block.timestamp, they are usually more public than they appear or are affected by miners. These random numbers are predictable to a certain extent, so malicious users can usually copy it and rely on its unpredictability to attack the function.

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

4.17. Unsafe interface use [Pass]

Check whether unsafe interfaces are used in the contract code implementation

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

4.18. Variable coverage [Pass]

Check whether there are security issues caused by variable coverage in the contract code implementation

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

4.19. Uninitialized storage pointer [Pass]

In solidity, a special data structure is allowed to be a struct structure, and the local variables in the function are stored in storage or memory by default.

The existence of storage (memory) and memory (memory) are two different concepts. Solidity allows pointers to point to an uninitialized reference, while uninitialized local storage will cause variables to point to other storage variables, leading to variable coverage, or even more serious As a consequence, you should avoid initializing struct variables in functions during development.

Test result: After testing, the smart contract code does not use structure, and there is no such problem.

Safety advice: None.

4.20. Return value call verification [Pass]

This problem mostly occurs in smart contracts related to currency transfer, so it is also called silent failed delivery or unchecked delivery.

There are transfer(), send(), call.value() and other currency transfer methods in Solidity, which can all be used to send Ether to an address. The difference is: When the transfer fails, it will be thrown and the state will be rolled back; Only 2300gas will be passed for calling to prevent reentry attacks; false will be returned when send fails; only 2300gas will be passed for calling to prevent reentry attacks; false will be returned when call.value fails to be sent; all available gas will be passed for calling (can be By passing in the gas_value parameter to limit), it cannot effectively prevent reentry attacks.

If the return value of the above send and call.value transfer functions is not checked in the code, the contract will continue to execute the following code, which may lead to unexpected results due to Ether sending failure.

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

4.21. Transaction order dependency detection [Pass]

Since miners always get gas fees through codes that represent externally owned addresses (EOA), users can specify higher fees for faster transactions. Since the Ethereum blockchain is public, everyone can see the content of other people's pending transactions. This means that if a user submits a valuable solution, a malicious user can steal the solution and copy its transaction at a higher fee to preempt the original solution.

Test result: After testing, the security issue does not exist in the smart contract code.

Safety advice: None.

4.22. **Timestamp dependent attack [Pass]**

The timestamp of the data block usually uses the local time of the miner, and this time can fluctuate in the range of about 900 seconds. When other nodes accept a new block, it is only necessary to verify whether the timestamp is later than the previous block and The error with local time is within 900 seconds. A miner can profit from it by setting the timestamp of the block to satisfy the conditions that are beneficial to him as much as possible.

Check whether there are key functions that depend on the timestamp in the contract code implementation

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

4.23. Denial of service attack detection [Pass]

In the world of Ethereum, denial of service is fatal, and a smart contract that has suffered this type of attack may never be able to return to its normal working state. There may be many reasons for the denial of service of the smart contract, including malicious behavior as the transaction recipient, artificially increasing the gas required for computing functions to cause gas exhaustion, abusing access control to access the private component of the smart contract, using confusion and negligence, etc. Wait.

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

4.24. Fake recharge vulnerability detection [Pass]

The transfer function of the token contract uses the if judgment method to check the balance of the transfer initiator (msg.sender). When balances[msg.sender] <value, it enters the else logic part and returns false, and finally no exception is thrown. We believe that only if/else this kind of gentle judgment method is an imprecise coding method in the scene of sensitive functions such as transfer.

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

4.25. Reentry attack detection [Pass]

Re-entry vulnerability is the most famous Ethereum smart contract vulnerability, which once led to the fork of Ethereum (The DAO hack).

The call.value() function in Solidity consumes all the gas it receives when it is used to send Ether. When the call.value() function is called to send Ether before it actually reduces the balance of the sender's account, There is a risk of reentry attacks.

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

4.26. Replay attack detection [Pass]

If the contract involves the need for entrusted management, attention should be paid to the non-reusability of verification to avoid replay attacks

In the asset management system, there are often cases of entrusted management. The principal assigns assets to the trustee for management, and the principal pays a certain fee to the trustee. This business scenario is also common in smart contracts. .

Detection result: After detection, the smart contract does not use the call function, and this vulnerability does not exist.

Safety advice: None.

4.27. Rearrangement attack detection [Pass]

A rearrangement attack refers to a miner or other party trying to "compete" with smart contract participants by inserting their own information into a list or mapping, so that the attacker has the opportunity to store their own information in the contract. in.

Test result: After testing, there are no related vulnerabilities in the smart contract code.

Safety advice: None.

5. Appendix A: Contract code

Source code for this test:

AlpacaPresale.sol

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

pragma solidity =0.6.12;

import "@openzeppelin/contracts/token/ERC1155/IERC1155.sol";
import "@openzeppelin/contracts/token/ERC1155/ERC1155Receiver.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/math/Math.sol";
import "@openzeppelin/contracts/utils/EnumerableSet.sol";
import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";

import "./DateControl.sol";

contract AlpacaPresale is
    Ownable,
    DateControl,
    ReentrancyGuard,
    ERC1155Receiver
{
    using SafeMath for uint256;
    using Math for uint256;
    using EnumerableSet for EnumerableSet.UintSet;
    using EnumerableSet for EnumerableSet.AddressSet;

    /* ========== STATE VARIABLES ========== */

    IERC1155 public cryptoAlpaca;

    uint256 public pricePerAlpaca = 0.01 ether;

    uint256 public maxAdoptionCount = 100;

    // Mapping from address to alpaca count
    mapping(address => uint256) private accountAddoptionCount;

    // Set of alpaca IDs this contract owns
    EnumerableSet.UintSet private presaleAlpacaIDs;

    // Set of address that are approved to purchase alpaca
    EnumerableSet.AddressSet private whitelist;

    /* ========== CONSTRUCTOR ========== */

    constructor(IERC1155 _cryptoAlpaca) public {
        cryptoAlpaca = _cryptoAlpaca;
    }

    /* ========== OWNER ONLY ========== */

    /**
     * @dev Allow owner to change alpaca price
     */
    function addToWhitelist(address[] calldata _addresses) public onlyOwner {
        for (uint256 i = 0; i < _addresses.length; i++) {
            whitelist.add(_addresses[i]);
        }
    }

    /**
     * @dev Allow owner to change alpaca price
     */
    function setPricePerAlpaca(uint256 _price) public onlyOwner {
        pricePerAlpaca = _price;
    }

    /**
     * @dev Allow owner to update maximum number alpaca a given user can adopt
     */
    function setMaxAdoptionCount(uint256 _maxAdoptionCount) public onlyOwner {
        maxAdoptionCount = _maxAdoptionCount;
    }

    /**
     * @dev Allow owner to transfer a alpaca that didn't get adopted during presale
     */
    function reclaim(uint256 _id, address _to) public onlyOwner whenEnded {
```

```
        cryptoAlpaca.safeTransferFrom(address(this), _to, _id, 1, "");
    }

    /**
     * @dev Allow owner to transfer all alpaca that didn't get adopted during presale
     */
    function reclaimAll(address _to) public onlyOwner whenEnded {
        uint256 length = presaleAlpacaIDs.length();
        uint256[] memory ids = new uint256[](length);
        uint256[] memory amount = new uint256[](length);
        for (uint256 i = 0; i < length; i++) {
            ids[i] = presaleAlpacaIDs.at(i);
            amount[i] = 1;
        }

        cryptoAlpaca.safeBatchTransferFrom(address(this), _to, ids, amount, "");
    }

    /**
     * @dev Allows owner to withdrawal the presale balance to an account.
     */
    function withdraw(address payable _to) external onlyOwner {
        _to.transfer(address(this).balance);
    }

    /* ========== EXTERNAL MUTATIVE FUNCTIONS ========== */

    /**
     * @dev Adopt _count number of alpaca
     */
    function adoptAlpaca(uint256 _count)
        public
        payable
        whenInProgress
        nonReentrant
    {//knownsec// Adopt alpaca
        require(_count > 0, "AlpacaPresale: must adopt at least one alpaca");//knownsec// Check quantity
        require(whitelist.contains(msg.sender), "AlpacaPresale: unauthorized");//knownsec// Check the whitelist

        address account = msg.sender;
        uint256 credit = canAdoptCount(account);
        require(//knownsec// Check the number of adoptions
            _count <= credit,
            "AlpacaPresale: adoption count larger than maximum adoption limit"
        );

        require(//knownsec// Verify the transfer limit
            msg.value >= getAdoptionPrice(_count),
            "AlpacaPresale: insufficient funds"
        );

        uint256[] memory ids = new uint256[](_count);
        uint256[] memory counts = new uint256[](_count);
        for (uint256 i = 0; i < _count; i++) {
            ids[i] = _randRemoveAlpaca();//knownsec// Random Alpaca
            counts[i] = 1;
        }

        accountAddoptionCount[account] += _count;//knownsec// Accumulate the number of adoptions

        cryptoAlpaca.safeBatchTransferFrom(//knownsec// Transfer Alpaca
            address(this),
            account,
            ids,
            counts,
            ""
        );
    }

    /* ========== VIEW ========== */

    /**
     * @dev returns if `_account` is whitelisted to adopt alpaca
     */
    function allowedToAdopt(address _account) public view returns (bool) {
        return whitelist.contains(_account);
    }

    /**
     * @dev returns number of _account has adopted presale alpaca
     */
    function getAdoptionCount(address _account) public view returns (uint256) {
        return accountAddoptionCount[_account];
    }

    /**
     * @dev total adoption price if adopt _count many
```

```
     */
    function getAdoptionPrice(uint256 _count) public view returns (uint256) {
        return _count.mul(pricePerAlpaca);
    }

    /**
     * @dev number of presale alpaca this contract owns
     */
    function getPresaleAlpacaCount() public view returns (uint256) {
        return presaleAlpacaIDs.length();
    }

    /**
     * @dev how many more _account can adopt alpaca
     */
    function canAdoptCount(address _account) public view returns (uint256) {
        if (!allowedToAdopt(_account)) {
            return 0;
        }

        uint256 credit = maxAdoptionCount.sub(accountAddoptionCount[_account]);

        uint256 alpacaCount = presaleAlpacaIDs.length();

        return credit.min(alpacaCount);
    }

    /**
     * @dev onERC1155Received implementation per IERC1155Receiver spec
     */
    function onERC1155Received(
        address,
        address,
        uint256 id,
        uint256,
        bytes calldata
    ) external override returns (bytes4) {
        require(//knownsec// Only cryptoAlpaca call
            msg.sender == address(cryptoAlpaca),
            "AlpacaPresale: received alpaca from unauthenticated contract"
        );

        uint256[] memory ids = new uint256[](1);
        ids[0] = id;

        _receivedAlpaca(ids);

        return
            bytes4(
                keccak256(
                    "onERC1155Received(address,address,uint256,uint256,bytes)"
                )
            );
    }

    /**
     * @dev onERC1155BatchReceived implementation per IERC1155Receiver spec
     */
    function onERC1155BatchReceived(
        address,
        address,
        uint256[] calldata ids,
        uint256[] calldata,
        bytes calldata
    ) external override returns (bytes4) {
        require(//knownsec// Only cryptoAlpaca call
            msg.sender == address(cryptoAlpaca),
            "AlpacaPresale: received alpaca from unauthenticated contract"
        );

        _receivedAlpaca(ids);

        return
            bytes4(
                keccak256(
                    "onERC1155BatchReceived(address,address,uint256[],uint256[],bytes)"
                )
            );
    }

    /* ========== PRIVATE ========== */

    /**
     * @dev randomly select and remove a alpaca
     * returns selected alpaca ID
     */
    function _randRemoveAlpaca() private returns (uint256) {
```

```
        require(presaleAlpacaIDs.length() > 0, "No more presale alpaca");

        uint256 totalLength = presaleAlpacaIDs.length();

        uint256 randIndex = uint256(blockhash(block.number - 1));
        randIndex = uint256(keccak256(abi.encodePacked(randIndex, totalLength)))
            .mod(totalLength);

        uint256 randID = presaleAlpacaIDs.at(uint256(randIndex));

        require(presaleAlpacaIDs.remove(randID));

        return randID;
    }

    function _receivedAlpaca(uint256[] memory ids) private {
        for (uint256 i = 0; i < ids.length; i++) {
            presaleAlpacaIDs.add(ids[i]);
        }
    }
}
```

DateControl.sol

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

pragma solidity =0.6.12;

import "@openzeppelin/contracts/access/Ownable.sol";

contract DateControl is Ownable {
    /* ========== STATE VARIABLES ========== */

    uint256 public startBlock;

    uint256 public endBlock;

    /* ========== EXTERNAL MUTATIVE FUNCTIONS ========== */

    function setStartBlock(uint256 _block) external onlyOwner {
        startBlock = _block;
    }

    function setEndBlock(uint256 _block) external onlyOwner {
        endBlock = _block;
    }

    /* ========== MODIFIER ========== */

    modifier whenInProgress() {
        require(block.number >= startBlock, "Event not yet started");
        require(block.number < endBlock, "Event Ended");
        _;
    }

    modifier whenEnded() {
        require(block.number >= endBlock, "Event not yet ended");
        _;
    }
}
```

AlpaReward.sol

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

pragma solidity =0.6.12;// knownsec Specify the compiler version

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";

// AlpaReward
contract AlpaReward is ERC20("AlpaReward", "xALPA") {
    using SafeMath for uint256;// knownsec Specify the compiler version

    /* ========== STATE VARIABLES ========== */

    IERC20 public alpa;

    /* ========== CONSTRUCTOR ========== */

    /**
     * Define the ALPA token contract
     */
```

```
    constructor(IERC20 _alpa) public {
        alpa = _alpa;
    }

    /* ========== EXTERNAL MUTATIVE FUNCTIONS ========== */

    /**
     * Locks ALPA and mints xALPA
     * @param _amount of ALPA to stake
     */
    function enter(uint256 _amount) external {// knownsec External call    lock-up mining
        // Gets the amount of ALPA locked in the contract
        uint256 totalAlpa = alpa.balanceOf(address(this));// knownsec Calculate the balance of this alpa contract

        // Gets the amount of xALPA in existence
        uint256 totalShares = totalSupply();// knownsec Get supply

        // If no xALPA exists, mint it 1:1 to the amount put in
        if (totalShares == 0 || totalAlpa == 0) {// knownsec Deal with no token
            _mint(msg.sender, _amount);// knownsec Mining coins for lock-up miners
        } else {
            // Calculate and mint the amount of xALPA the ALPA is worth. The ratio will change overtime, as
xALPA is burned/minted and ALPA deposited + gained from fees / withdrawn.
            uint256 what = _amount.mul(totalShares).div(totalAlpa);// knownsec Pool ratio calculation
            _mint(msg.sender, what);// knownsec Mining coins for lock-up miners
        }

        // Lock the ALPA in the contract
        alpa.transferFrom(msg.sender, address(this), _amount);// knownsec Deposit to alpa
    }

    /**
     * Claim back your ALPAs.
     * Unclocks the staked + gained ALPA and burns xALPA
     * @param _share amount of xALPA
     */
    function leave(uint256 _share) external {// knownsec External call, take out
        // Gets the amount of xALPA in existence
        uint256 totalShares = totalSupply();// knownsec Get supply

        // Calculates the amount of ALPA the xALPA is worth
        uint256 what = _share.mul(alpa.balanceOf(address(this))).div(
            totalShares
        );// knownsec Use pledged credentials (xALPA) to get the number of withdrawals
        _burn(msg.sender, _share);// knownsec    Reduce supply

        alpa.transfer(msg.sender, what);// knownsec Send tokens
    }
}
```

AlpaToken.sol

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

pragma solidity 0.6.12;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "../interfaces/IAlpaToken.sol";

contract AlpaToken is ERC20("AlpaToken", "ALPA"), IAlpaToken, Ownable {
    /* ========== EXTERNAL MUTATIVE FUNCTIONS ========== */

    /**
     * @dev allow owner to mint
     * @param _to mint token to address
     * @param _amount amount of ALPA to mint
     */
    function mint(address _to, uint256 _amount) external override onlyOwner {// knownsec Administrator use
        _mint(_to, _amount);// knownsec The administrator uses the minting method and can issue additional
    }
}
```

AlpacaBase.sol

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

pragma solidity =0.6.12;

import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/utils/EnumerableMap.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "../interfaces/IGeneScience.sol";
```

```
contract AlpacaBase is Ownable {
    using SafeMath for uint256;

    /* ========== ENUM ========== */

    /**
     * @dev Alpaca can be in one of the two state:
     *
     * EGG - When two alpaca breed with each other, alpaca EGG is created.
     *           `gene` and `energy` are both 0 and will be assigned when egg is cracked
     *
     * GROWN - When egg is cracked and alpaca is born! `gene` and `energy` are determined
     *             in this state.
     */
    enum AlpacaGrowthState {EGG, GROWN}

    /* ========== PUBLIC STATE VARIABLES ========== */

    /**
     * @dev payment required to use cracked if it's done automatically
     * assigning to 0 indicate cracking action is not automatic
     */
    uint256 public autoCrackingFee = 0;

    /**
     * @dev Base breeding ALPA fee
     */
    uint256 public baseHatchingFee = 10e18; // 10 ALPA

    /**
     * @dev ALPA ERC20 contract address
     */
    IERC20 public alpa;

    /**
     * @dev 10% of the breeding ALPA fee goes to `devAddress`
     */
    address public devAddress;

    /**
     * @dev 90% of the breeding ALPA fee goes to `stakingAddress`
     */
    address public stakingAddress;

    /**
     * @dev number of percentage breeding ALPA fund goes to devAddress
     * dev percentage = devBreedingPercentage / 100
     * staking percentage = (100 - devBreedingPercentage) / 100
     */
    uint256 public devBreedingPercentage = 10;

    /**
     * @dev An approximation of currently how many seconds are in between blocks.
     */
    uint256 public secondsPerBlock = 15;

    /**
     * @dev amount of time a new born alpaca needs to wait before participating in breeding activity.
     */
    uint256 public newBornCoolDown = uint256(1 days);

    /**
     * @dev amount of time an egg needs to wait to be cracked
     */
    uint256 public hatchingDuration = uint256(5 minutes);

    /**
     * @dev when two alpaca just bred, the breeding multiplier will doubled to control
     * alpaca's population. This is the amount of time each parent must wait for the
     * breeding multiplier to reset back to 1
     */
    uint256 public hatchingMultiplierCoolDown = uint256(6 hours);

    /**
     * @dev hard cap on the maximum hatching cost multiplier it can reach to
     */
    uint16 public maxHatchCostMultiplier = 16;

    /**
     * @dev Gen0 generation factor
     */
    uint64 public constant GEN0_GENERATION_FACTOR = 10;

    /**
     * @dev maximum gen-0 alpaca energy. This is to prevent contract owner from
     * creating arbitrary energy for gen-0 alpaca
     */
```

```
uint32 public constant MAX_GEN0_ENERGY = 3600;

/**
 * @dev hatching fee increase with higher alpa generation
 */
uint256 public generationHatchingFeeMultiplier = 2;

/**
 * @dev gene science contract address for genetic combination algorithm.
 */
IGeneScience public geneScience;

/* ========== INTERNAL STATE VARIABLES ========== */

/**
 * @dev An array containing the Alpaca struct for all Alpacas in existence. The ID
 * of each alpaca is the index into this array.
 */
Alpaca[] internal alpacas;

/**
 * @dev mapping from AlpacaIDs to an address where alpaca owner approved address to use
 * this alpca for breeding. addrss can breed with this cat multiple times without limit.
 * This will be resetted everytime someone transfered the alpaca.
 */
EnumerableMap.UintToAddressMap internal alpacaAllowedToAddress;

/* ========== ALPACA STRUCT ========== */

/**
 * @dev Everything about your alpaca is stored in here. Each alpaca's appearance
 * is determined by the gene. The energy associated with each alpaca is also
 * related to the gene
 */
struct Alpaca {
    // Theaalpaca genetic code.
    uint256 gene;
    // the alpaca energy level
    uint32 energy;
    // The timestamp from the block when this alpaca came into existence.
    uint64 birthTime;
    // The minimum timestamp alpaca needs to wait to avoid hatching multiplier
    uint64 hatchCostMultiplierEndBlock;
    // hatching cost multiplier
    uint16 hatchingCostMultiplier;
    // The ID of the parents of this alpaca, set to 0 for gen0 alpaca.
    uint32 matronId;
    uint32 sireId;
    // The "generation number" of this alpaca. The generation number of an alpacas
    // is the smaller of the two generation numbers of their parents, plus one.
    uint16 generation;
    // The minimum timestamp new born alpaca needs to wait to hatch egg.
    uint64 cooldownEndBlock;
    // The generation factor buffs alpaca energy level
    uint64 generationFactor;
    // defines current alpaca state
    AlpacaGrowthState state;
}

/* ========== VIEW ========== */

function getTotalAlpaca() external view returns (uint256) {
    return alpacas.length;
}

function _getBaseHatchingCost(uint256 _generation)
    internal
    view
    returns (uint256)
{
    return
        baseHatchingFee.add(
            _generation.mul(generationHatchingFeeMultiplier).mul(1e18)
        );
}

/* ========== OWNER MUTATIVE FUNCTION ========== */

/**
 * @param _hatchingDuration hatching duration
 */
function setHatchingDuration(uint256 _hatchingDuration) external onlyOwner {
    hatchingDuration = _hatchingDuration;
}

/**
 * @param _stakingAddress staking address
```

```
     */
    function setStakingAddress(address _stakingAddress) external onlyOwner {
        stakingAddress = _stakingAddress;
    }

    /**
     * @param _devAddress dev address
     */
    function setDevAddress(address _devAddress) external onlyDev {
        devAddress = _devAddress;
    }

    /**
     * @param _maxHatchCostMultiplier max hatch cost multiplier
     */
    function setMaxHatchCostMultiplier(uint16 _maxHatchCostMultiplier)
        external
        onlyOwner
    {
        maxHatchCostMultiplier = _maxHatchCostMultiplier;
    }

    /**
     * @param _devBreedingPercentage base generation factor
     */
    function setDevBreedingPercentage(uint256 _devBreedingPercentage)
        external
        onlyOwner
    {
        require(
            devBreedingPercentage <= 100,
            "CryptoAlpaca: invalid breeding percentage - must be between 0 and 100"
        );
        devBreedingPercentage = _devBreedingPercentage;
    }

    /**
     * @param _generationHatchingFeeMultiplier multiplier
     */
    function setGenerationHatchingFeeMultiplier(
        uint256 _generationHatchingFeeMultiplier
    ) external onlyOwner {
        generationHatchingFeeMultiplier = _generationHatchingFeeMultiplier;
    }

    /**
     * @param _baseHatchingFee base birthing
     */
    function setBaseHatchingFee(uint256 _baseHatchingFee) external onlyOwner {
        baseHatchingFee = _baseHatchingFee;
    }

    /**
     * @param _newBornCoolDown new born cool down
     */
    function setNewBornCoolDown(uint256 _newBornCoolDown) external onlyOwner {
        newBornCoolDown = _newBornCoolDown;
    }

    /**
     * @param _hatchingMultiplierCoolDown base birthing
     */
    function setHatchingMultiplierCoolDown(uint256 _hatchingMultiplierCoolDown)
        external
        onlyOwner
    {
        hatchingMultiplierCoolDown = _hatchingMultiplierCoolDown;
    }

    /**
     * @dev update how many seconds per blocks are currently observed.
     * @param _secs number of seconds
     */
    function setSecondsPerBlock(uint256 _secs) external onlyOwner {
        secondsPerBlock = _secs;
    }

    /**
     * @dev only owner can update autoCrackingFee
     */
    function setAutoCrackingFee(uint256 _autoCrackingFee) external onlyOwner {
        autoCrackingFee = _autoCrackingFee;
    }

    /**
     * @dev owner can upgrading gene science
     */
```

```
    function setGeneScience(IGeneScience _geneScience) external onlyOwner {
        require(
            _geneScience.isAlpacaGeneScience(),
            "CryptoAlpaca: invalid gene science contract"
        );

        // Set the new contract address
        geneScience = _geneScience;
    }

    /**
     * @dev owner can update ALPA erc20 token location
     */
    function setAlpaContract(IERC20 _alpa) external onlyOwner {
        alpa = _alpa;
    }

    /* ========== MODIFIER ========== */

    /**
     * @dev Throws if called by any account other than the dev.
     */
    modifier onlyDev() {
        require(
            devAddress == _msgSender(),
            "CryptoAlpaca: caller is not the dev"
        );
        _;
    }
}
```

AlpacaBreed.sol

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

pragma solidity =0.6.12;

import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/utils/EnumerableMap.sol";
import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
import "@openzeppelin/contracts/utils/Pausable.sol";

import "./AlpacaToken.sol";
import "../interfaces/ICryptoAlpaca.sol";

contract AlpacaBreed is AlpacaToken, ICryptoAlpaca, ReentrancyGuard, Pausable {
    using SafeMath for uint256;
    using EnumerableMap for EnumerableMap.UintToAddressMap;

    /* ========== EVENTS ========== */

    // The Hatched event is fired when two alpaca successfully hached an egg.
    event Hatched(
        uint256 indexed eggId,
        uint256 matronId,
        uint256 sireId,
        uint256 cooldownEndBlock
    );

    // The GrantedToBreed event is fired whne an alpaca's owner granted
    // addr account to use alpacaId as sire to breed.
    event GrantedToBreed(uint256 indexed alpacaId, address addr);

    /* ========== VIEWS ========== */

    /**
     * Returns all the relevant information about a specific alpaca.
     * @param _id The ID of the alpaca of interest.
     */
    function getAlpaca(uint256 _id)
        external
        override
        view
        returns (
            uint256 id,
            bool isReady,
            uint256 cooldownEndBlock,
            uint256 birthTime,
            uint256 matronId,
            uint256 sireId,
            uint256 hatchingCost,
            uint256 hatchingCostMultiplier,
            uint256 hatchCostMultiplierEndBlock,
            uint256 generation,
            uint256 gene,
            uint256 energy,
```

```
            uint256 state
        )
    {
        Alpaca storage alpaca = alpacas[_id];

        id = _id;
        isReady = (alpaca.cooldownEndBlock <= block.number);
        cooldownEndBlock = alpaca.cooldownEndBlock;
        birthTime = alpaca.birthTime;
        matronId = alpaca.matronId;
        sireId = alpaca.sireId;
        hatchingCost = _getBaseHatchingCost(alpaca.generation);
        hatchingCostMultiplier = alpaca.hatchingCostMultiplier;
        if (alpaca.hatchCostMultiplierEndBlock <= block.number) {
            hatchingCostMultiplier = 1;
        }

        hatchCostMultiplierEndBlock = alpaca.hatchCostMultiplierEndBlock;
        generation = alpaca.generation;
        gene = alpaca.gene;
        energy = alpaca.energy;
        state = uint256(alpaca.state);
    }

    /**
     * @dev Calculating hatching ALPA cost
     */
    function hatchingALPACost(uint256 _matronId, uint256 _sireId)
        external
        view
        returns (uint256)
    {
        return _hatchingALPACost(_matronId, _sireId, false);
    }

    /**
     * @dev Checks to see if a given egg passed cooldownEndBlock and ready to crack
     * @param _id alpaca egg ID
     */

    function isReadyToCrack(uint256 _id) external view returns (bool) {
        Alpaca storage alpaca = alpacas[_id];
        return
            (alpaca.state == AlpacaGrowthState.EGG) &&
            (alpaca.cooldownEndBlock <= uint64(block.number));
    }

    /* ========== EXTERNAL MUTATIVE FUNCTIONS   ========== */

    /**
     * Grants permission to another account to sire with one of your alpacas.
     * @param _addr The address that will be able to use sire for breeding.
     * @param _sireId a alpaca _addr will be able to use for breeding as sire.
     */
    function grandPermissionToBreed(address _addr, uint256 _sireId)
        external
        override
    {
        require(
            isOwnerOf(msg.sender, _sireId),
            "CryptoAlpaca: You do not own sire alpaca"
        );

        alpacaAllowedToAddress.set(_sireId, _addr);
        emit GrantedToBreed(_sireId, _addr);
    }

    /**
     * check if `_addr` has permission to user alpaca `_id` to breed with as sire.
     */
    function hasPermissionToBreedAsSire(address _addr, uint256 _id)
        external
        override
        view
        returns (bool)
    {
        if (isOwnerOf(_addr, _id)) {
            return true;
        }

        return alpacaAllowedToAddress.get(_id) == _addr;
    }

    /**
     * Clear the permission on alpaca for another user to use to breed.
     * @param _alpacaId a alpaca to clear permission .
     */
```

```
function clearPermissionToBreed(uint256 _alpacaId) external override {
    require(
        isOwnerOf(msg.sender, _alpacaId),
        "CryptoAlpaca: You do not own this alpaca"
    );

    alpacaAllowedToAddress.remove(_alpacaId);
}

/**
 * @dev Hatch an baby alpaca egg with two alpaca you own (_matronId and _sireId).
 * Requires a pre-payment of the fee given out to the first caller of crack()
 * @param _matronId The ID of the Alpaca acting as matron
 * @param _sireId The ID of the Alpaca acting as sire
 * @return The hatched alpaca egg ID
 */
function hatch(uint256 _matronId, uint256 _sireId)
    external
    override
    payable
    whenNotPaused
    nonReentrant
    returns (uint256)
{
    address msgSender = msg.sender;

    // Checks for payment.
    require(
        msg.value >= autoCrackingFee,
        "CryptoAlpaca: Required autoCrackingFee not sent"
    );

    // Checks for ALPA payment
    require(
        alpa.allowance(msgSender, address(this)) >=
            _hatchingALPACost(_matronId, _sireId, true),
        "CryptoAlpaca: Required hetching ALPA fee not sent"
    );

    // Checks if matron and sire are valid mating pair
    require(
        _ownerPermittedToBreed(msgSender, _matronId, _sireId),
        "CryptoAlpaca: Invalid permission"
    );

    // Grab a reference to the potential matron
    Alpaca storage matron = alpacas[_matronId];

    // Make sure matron isn't pregnant, or in the middle of a siring cooldown
    require(
        _isReadyToHatch(matron),
        "CryptoAlpaca: Matron is not yet ready to hatch"
    );

    // Grab a reference to the potential sire
    Alpaca storage sire = alpacas[_sireId];

    // Make sure sire isn't pregnant, or in the middle of a siring cooldown
    require(
        _isReadyToHatch(sire),
        "CryptoAlpaca: Sire is not yet ready to hatch"
    );

    // Test that matron and sire are a valid mating pair.
    require(
        _isValidMatingPair(matron, _matronId, sire, _sireId),
        "CryptoAlpaca: Matron and Sire are not valid mating pair"
    );

    // All checks passed, Alpaca gets pregnant!
    return _hatchEgg(_matronId, _sireId);
}

/**
 * @dev egg is ready to crack and give life to baby alpaca!
 * @param _id A Alpaca egg that's ready to crack.
 */
function crack(uint256 _id) external override nonReentrant {
    // Grab a reference to the egg in storage.
    Alpaca storage egg = alpacas[_id];

    // Check that the egg is a valid alpaca.
    require(egg.birthTime != 0, "CryptoAlpaca: not valid egg");
    require(//knownsec// Check status
        egg.state == AlpacaGrowthState.EGG,
        "CryptoAlpaca: not a valid egg"
    );
```

```
        // Check that the matron is pregnant, and that its time has come!
        require(_isReadyToCrack(egg), "CryptoAlpaca: egg cant be cracked yet");//knownsec// Check if it can
hatch

        // Grab a reference to the sire in storage.
        Alpaca storage matron = alpacas[egg.matronId];
        Alpaca storage sire = alpacas[egg.sireId];

        // Call the sooper-sekret gene mixing operation.
        (
            uint256 childGene,
            uint256 childEnergy,
            uint256 generationFactor
        ) = geneScience.mixGenes(//knownsec// Get parameters related to newborn alpaca
            matron.gene,
            sire.gene,
            egg.generation,
            uint256(egg.cooldownEndBlock).sub(1)
        );

        egg.gene = childGene;
        egg.energy = uint32(childEnergy);
        egg.state = AlpacaGrowthState.GROWN;
        egg.cooldownEndBlock = uint64(
            (newBornCoolDown.div(secondsPerBlock)).add(block.number)
        );
        egg.generationFactor = uint64(generationFactor);

        // Send the balance fee to the person who made birth happen.
        if (autoCrackingFee > 0) {//knownsec// Incubation fee
            msg.sender.transfer(autoCrackingFee);
        }

        // emit the born event
        emit BornSingle(_id, childGene, childEnergy);
    }

    /* ========== PRIVATE FUNCTION ========== */

    /**
     * @dev Recalculate the hatchingCostMultiplier for alpaca after breed.
     * If hatchCostMultiplierEndBlock is less than current block number
     * reset hatchingCostMultiplier back to 2, otherwize multiply hatchingCostMultiplier by 2. Also update
     * hatchCostMultiplierEndBlock.
     */
    function _refreshHatchingMultiplier(Alpaca storage _alpaca) private {
        if (_alpaca.hatchCostMultiplierEndBlock < block.number) {
            _alpaca.hatchingCostMultiplier = 2;
        } else {
            uint16 newMultiplier = _alpaca.hatchingCostMultiplier * 2;
            if (newMultiplier > maxHatchCostMultiplier) {
                newMultiplier = maxHatchCostMultiplier;
            }

            _alpaca.hatchingCostMultiplier = newMultiplier;
        }
        _alpaca.hatchCostMultiplierEndBlock = uint64(
            (hatchingMultiplierCoolDown.div(secondsPerBlock)).add(block.number)
        );
    }

    function _ownerPermittedToBreed(
        address _sender,
        uint256 _matronId,
        uint256 _sireId
    ) private view returns (bool) {
        // owner must own matron, othersize not permitted
        if (!isOwnerOf(_sender, _matronId)) {
            return false;
        }

        // if owner owns sire, it's permitted
        if (isOwnerOf(_sender, _sireId)) {
            return true;
        }

        // if sire's owner has given permission to _sender to breed,
        // then it's permitted to breed
        if (alpacaAllowedToAddress.contains(_sireId)) {
            return alpacaAllowedToAddress.get(_sireId) == _sender;
        }

        return false;
    }

    /**
```

```
     * @dev Checks that a given alpaca is able to breed. Requires that the
     * current cooldown is finished (for sires) and also checks that there is
     * no pending pregnancy.
     */
    function _isReadyToHatch(Alpaca storage _alpaca)
        private
        view
        returns (bool)
    {
        return
            (_alpaca.state == AlpacaGrowthState.GROWN) &&
            (_alpaca.cooldownEndBlock < uint64(block.number));
    }

    /**
     * @dev Checks to see if a given alpaca is pregnant and (if so) if the gestation
     * period has passed.
     */

    function _isReadyToCrack(Alpaca storage _egg) private view returns (bool) {
        return
            (_egg.state == AlpacaGrowthState.EGG) &&
            (_egg.cooldownEndBlock < uint64(block.number));
    }

    /**
     * @dev Calculating breeding ALPA cost for internal usage.
     */
    function _hatchingALPACost(
        uint256 _matronId,
        uint256 _sireId,
        bool _strict
    ) private view returns (uint256) {
        uint256 blockNum = block.number;
        if (!_strict) {
            blockNum = blockNum + 1;
        }

        Alpaca storage sire = alpacas[_sireId];
        uint256 sireHatchingBase = _getBaseHatchingCost(sire.generation);
        uint256 sireMultiplier = sire.hatchingCostMultiplier;
        if (sire.hatchCostMultiplierEndBlock < blockNum) {
            sireMultiplier = 1;
        }

        Alpaca storage matron = alpacas[_matronId];
        uint256 matronHatchingBase = _getBaseHatchingCost(matron.generation);
        uint256 matronMultiplier = matron.hatchingCostMultiplier;
        if (matron.hatchCostMultiplierEndBlock < blockNum) {
            matronMultiplier = 1;
        }

        return
            (sireHatchingBase.mul(sireMultiplier)).add(
                matronHatchingBase.mul(matronMultiplier)
            );
    }

    /**
     * @dev Internal utility function to initiate hatching egg, assumes that all breeding
     *  requirements have been checked.
     */
    function _hatchEgg(uint256 _matronId, uint256 _sireId)
        private
        returns (uint256)
    {
        // Transfer birthing ALPA fee to this contract
        uint256 alpaCost = _hatchingALPACost(_matronId, _sireId, true);

        uint256 devAmount = alpaCost.mul(devBreedingPercentage).div(100);
        uint256 stakingAmount = alpaCost.mul(100 - devBreedingPercentage).div(
            100
        );
        //knownsec// Calculate processing related costs
        assert(alpa.transferFrom(msg.sender, devAddress, devAmount));
        assert(alpa.transferFrom(msg.sender, stakingAddress, stakingAmount));

        // Grab a reference to the Alpacas from storage.
        Alpaca storage sire = alpacas[_sireId];
        Alpaca storage matron = alpacas[_matronId];

        // refresh hatching multiplier for both parents.
        _refreshHatchingMultiplier(sire);
        _refreshHatchingMultiplier(matron);

        // Determine the lower generation number of the two parents
        uint256 parentGen = matron.generation;
```

```
        if (sire.generation < matron.generation) {//knownsec// Lower generation first
            parentGen = sire.generation;
        }

        // child generation will be 1 larger than min of the two parents generation;
        uint256 childGen = parentGen.add(1);

        // Determine when the egg will be cracked
        uint256 cooldownEndBlock = (hatchingDuration.div(secondsPerBlock)).add(
            block.number
        );

        uint256 eggID = _createEgg(//knownnsec// 创建羊驼蛋
            _matronId,
            _sireId,
            childGen,
            cooldownEndBlock,
            msg.sender
        );

        // Emit the hatched event.
        emit Hatched(eggID, _matronId, _sireId, cooldownEndBlock);

        return eggID;
    }

    /**
     * @dev Internal check to see if a given sire and matron are a valid mating pair.
     * @param _matron A reference to the Alpaca struct of the potential matron.
     * @param _matronId The matron's ID.
     * @param _sire A reference to the Alpaca struct of the potential sire.
     * @param _sireId The sire's ID
     */
    function _isValidMatingPair(
        Alpaca storage _matron,
        uint256 _matronId,
        Alpaca storage _sire,
        uint256 _sireId
    ) private view returns (bool) {
        // A Aapaca can't breed with itself
        if (_matronId == _sireId) {
            return false;
        }

        // Alpaca can't breed with their parents.
        if (_matron.matronId == _sireId || _matron.sireId == _sireId) {
            return false;
        }
        if (_sire.matronId == _matronId || _sire.sireId == _matronId) {
            return false;
        }

        return true;
    }

    /**
     * @dev openzeppelin ERC1155 Hook that is called before any token transfer
     * Clear any alpacaAllowedToAddress associated to the alpaca
     * that's been transfered
     */
    function _beforeTokenTransfer(
        address,
        address,
        address,
        uint256[] memory ids,
        uint256[] memory,
        bytes memory
    ) internal virtual override {
        for (uint256 i = 0; i < ids.length; i++) {
            if (alpacaAllowedToAddress.contains(ids[i])) {
                alpacaAllowedToAddress.remove(ids[i]);
            }
        }
    }
}
```

AlpacaCore.sol

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

pragma solidity =0.6.12;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "../interfaces/IGeneScience.sol";
import "./AlpacaBreed.sol";
```

```
contract AlpacaCore is AlpacaBreed {
    /**
     * @dev Initializes crypto alpaca contract.
     * @param _alpa ALPA ERC20 contract address
     * @param _devAddress dev address.
     * @param _stakingAddress staking address.
     */
    constructor(
        IERC20 _alpa,
        IGeneScience _geneScience,
        address _devAddress,
        address _stakingAddress
    ) public {
        alpa = _alpa;
        geneScience = _geneScience;
        devAddress = _devAddress;
        stakingAddress = _stakingAddress;

        // start with the mythical genesis alpaca
        _createGen0Alpaca(uint256(-1), 0, msg.sender);
    }

    /* ========== OWNER MUTATIVE FUNCTION ========== */

    /**
     * @dev Allows owner to withdrawal the balance available to the contract.
     */
    function withdrawBalance(uint256 _amount, address payable _to)
        external
        onlyOwner
    {
        _to.transfer(_amount);
    }

    /**
     * @dev pause crypto alpaca contract stops any further hatching.
     */
    function pause() external onlyOwner {
        _pause();
    }

    /**
     * @dev unpause crypto alpaca contract.
     */
    function unpause() external onlyOwner {
        _unpause();
    }
}
```

AlpacaToken.sol

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

pragma solidity =0.6.12;

import "@openzeppelin/contracts/token/ERC1155/ERC1155.sol";
import "./AlpacaBase.sol";

contract AlpacaToken is AlpacaBase, ERC1155("") {
    /* ========== EVENTS ========== */

    /**
     * @dev Emitted when single `alpacaId` alpaca with `gene` and `energy` is born
     */
    event BornSingle(uint256 indexed alpacaId, uint256 gene, uint256 energy);

    /**
     * @dev Equivalent to multiple {BornSingle} events
     */
    event BornBatch(uint256[] alpacaIds, uint256[] genes, uint256[] energy);

    /* ========== VIEWS ========== */

    /**
     * @dev Check if `_alpacaId` is owned by `_account`
     */
    function isOwnerOf(address _account, uint256 _alpacaId)
        public
        view
        returns (bool)
    {
        return balanceOf(_account, _alpacaId) == 1;
    }

    /* ========== OWNER MUTATIVE FUNCTION ========== */
```

```
/**
  * @dev Allow contract owner to update URI to look up all alpaca metadata
  */
function setURI(string memory _newuri) external onlyOwner {
        _setURI(_newuri);
}

/**
  * @dev Allow contract owner to create generation 0 alpaca with `_gene`,
  *      `_energy` and transfer to `owner`
  *
  * Requirements:
  *
  * - `_energy` must be less than or equal to MAX_GEN0_ENERGY
  */
function createGen0Alpaca(
        uint256 _gene,
        uint256 _energy,
        address _owner
) external onlyOwner {
        address alpacaOwner = _owner;
        if (alpacaOwner == address(0)) {
                alpacaOwner = owner();
        }

        _createGen0Alpaca(_gene, _energy, alpacaOwner);
}

/**
  * @dev Equivalent to multiple {createGen0Alpaca} function
  *
  * Requirements:
  *
  * - all `_energies` must be less than or equal to MAX_GEN0_ENERGY
  */
function createGen0AlpacaBatch(
        uint256[] memory _genes,
        uint256[] memory _energies,
        address _owner
) external onlyOwner {
        address alpacaOwner = _owner;
        if (alpacaOwner == address(0)) {
                alpacaOwner = owner();
        }

        _createGen0AlpacaBatch(_genes, _energies, _owner);
}

/* ========== INTERNAL ALPA GENERATION ========== */

/**
  * @dev Create an alpaca egg. Egg's `gene` and `energy` will assigned to 0
  * initially and won't be determined until egg is cracked.
  */
function _createEgg(
        uint256 _matronId,
        uint256 _sireId,
        uint256 _generation,
        uint256 _cooldownEndBlock,
        address _owner
) internal returns (uint256) {//knownsec// Check previous generation information
        require(_matronId == uint256(uint32(_matronId)));
        require(_sireId == uint256(uint32(_sireId)));
        require(_generation == uint256(uint16(_generation)));

        Alpaca memory _alpaca = Alpaca({
                gene: 0,
                energy: 0,
                birthTime: uint64(now),
                hatchCostMultiplierEndBlock: 0,
                hatchingCostMultiplier: 1,
                matronId: uint32(_matronId),
                sireId: uint32(_sireId),
                cooldownEndBlock: uint64(_cooldownEndBlock),
                generation: uint16(_generation),
                generationFactor: 0,
                state: AlpacaGrowthState.EGG
        });

        alpacas.push(_alpaca);
        uint256 eggId = alpacas.length - 1;

        _mint(_owner, eggId, 1, "");

        return eggId;
}
```

```
/**
 * @dev Internal gen-0 alpaca creation function
 *
 * Requirements:
 *
 * - `_energy` must be less than or equal to MAX_GEN0_ENERGY
 */
function _createGen0Alpaca(
        uint256 _gene,
        uint256 _energy,
        address _owner
) internal returns (uint256) {//knownsec// Check the maximum energy
        require(_energy <= MAX_GEN0_ENERGY, "CryptoAlpaca: invalid energy");

        Alpaca memory _alpaca = Alpaca({
                gene: _gene,
                energy: uint32(_energy),
                birthTime: uint64(now),
                hatchCostMultiplierEndBlock: 0,
                hatchingCostMultiplier: 1,
                matronId: 0,
                sireId: 0,
                cooldownEndBlock: 0,
                generation: 0,
                generationFactor: GEN0_GENERATION_FACTOR,
                state: AlpacaGrowthState.GROWN
        });

        alpacas.push(_alpaca);
        uint256 newAlpacaID = alpacas.length - 1;

        _mint(_owner, newAlpacaID, 1, "");

        // emit the born event
        emit BornSingle(newAlpacaID, _gene, _energy);

        return newAlpacaID;
}

/**
 * @dev Internal gen-0 alpaca batch creation function
 *
 * Requirements:
 *
 * - all `_energies` must be less than or equal to MAX_GEN0_ENERGY
 */
function _createGen0AlpacaBatch(
        uint256[] memory _genes,
        uint256[] memory _energies,
        address _owner
) internal returns (uint256[] memory) {
        require(
                _genes.length > 0,
                "CryptoAlpaca: must pass at least one genes"
        );
        require(
                _genes.length == _energies.length,
                "CryptoAlpaca: genes and energy length mismatch"
        );

        uint256 alpacaIdStart = alpacas.length;
        uint256[] memory ids = new uint256[](_genes.length);
        uint256[] memory amount = new uint256[](_genes.length);

        for (uint256 i = 0; i < _genes.length; i++) {
                require(
                        _energies[i] <= MAX_GEN0_ENERGY,
                        "CryptoAlpaca: invalid energy"
                );

                Alpaca memory _alpaca = Alpaca({
                        gene: _genes[i],
                        energy: uint32(_energies[i]),
                        birthTime: uint64(now),
                        hatchCostMultiplierEndBlock: 0,
                        hatchingCostMultiplier: 1,
                        matronId: 0,
                        sireId: 0,
                        cooldownEndBlock: 0,
                        generation: 0,
                        generationFactor: GEN0_GENERATION_FACTOR,
                        state: AlpacaGrowthState.GROWN
                });

                alpacas.push(_alpaca);
                ids[i] = alpacaIdStart + i;
                amount[i] = 1;
```

```
        }
        _mintBatch(_owner, ids, amount, "");
        emit BornBatch(ids, _genes, _energies);
        return ids;
    }
}
```

GeneScience.sol

```
// SPDX-License-Identifier: MIT
pragma solidity =0.6.12;
import "../interfaces/IGeneScience.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
contract GeneScience is IGeneScience {
    using SafeMath for uint256;// knownsec   How to import SafeMath
    /* ========== STATE VARIABLES ========== */
    uint256 private constant _maskLast8Bits = uint256(0xff);
    uint256 private constant _maskFirst248Bits = uint256(~0xff);
    uint256 private constant BASE_GENERATION_FACTOR = 10;
    uint256 private constant ENERGY_BUFF = 2;
    /* ========== CONSTRUCTOR ========== */
    constructor() public {}
    /* ========== VIEWS ========== */
    /**
     * Conform to IGeneScience
     */
    function isAlpacaGeneScience() external override pure returns (bool) {// knownsec External call, return forever
true
        return true;
    }
    struct LocalStorage {
        uint8[48] genes1Array;
        uint8[48] genes2Array;
        uint8[48] babyArray;
        uint8 swap;
        uint256 rand;
        uint256 randomN;
        uint256 traitPos;
        uint256 randomIndex;
        uint256 baseEnergy;
        bool applyEnergyBuff;
    }
    /**
     * @dev given genes of alpaca 1 & 2, return a genetic combination
     * @param _genes1 genes of matron
     * @param _genes2 genes of sire
     * @param _generation child generation
     * @param _targetBlock target block child is intended to be born
     * @return gene child gene
     * @return energy energy associated with the gene
     * @return generationFactor buffs child energy, higher the generation larger the generationFactor
     *     energy = gene energy * generationFactor
     */
    function mixGenes(
        uint256 _genes1,
        uint256 _genes2,
        uint256 _generation,
        uint256 _targetBlock
    )
        external
        override
        view
        returns (
            uint256 gene,
            uint256 energy,
            uint256 generationFactor
        )// knownsec External call Gene hybrid algorithm Return mixed birth genetic parameters gene, energy,
generationFactor
    {
```

```
LocalStorage memory store; // knownsec Instantiate LocalStorage object
require(block.number > _targetBlock);

// Try to grab the hash of the "target block". This should be available the vast
// majority of the time (it will only fail if no-one calls giveBirth() within 256
// blocks of the target block, which is about 40 minutes. Since anyone can call
// giveBirth() and they are rewarded with ether if it succeeds, this is quite unlikely.)
store.randomN = uint256(blockhash(_targetBlock));// knownsec Initialize randomN

if (store.randomN == 0) {// knownsec Handle the case of unsuccessful initialization of randomN
    // We don't want to completely bail if the target block is no-longer available,
    // nor do we want to just use the current block's hash (since it could allow a
    // caller to game the random result). Compute the most recent block that has the
    // the same value modulo 256 as the target block. The hash for this block will
    // still be available, and  –  while it can still change as time passes  –  it will
    // only change every 40 minutes. Again, someone is very likely to jump in with
    // the giveBirth() call before it can cycle too many times.
    _targetBlock =
        (block.number & _maskFirst248Bits) +
        (_targetBlock & _maskLast8Bits);

    // The computation above could result in a block LARGER than the current block,
    // if so, subtract 256.
    if (_targetBlock >= block.number) _targetBlock -= 256;

    store.randomN = uint256(blockhash(_targetBlock));
}

// generate 256 bits of random, using as much entropy as we can from
// sources that can't change between calls.
store.randomN = uint256(
    keccak256(
        abi.encodePacked(
            store.randomN,
            _genes1,
            _genes2,
            _generation,
            _targetBlock,
            block.timestamp,
            block.difficulty
        )
    )
);

store.randomIndex = 0;// knownsec Initialize randomIndex
store.genes1Array = _decode(_genes1);// knownsec Initialize genes1Array
store.genes2Array = _decode(_genes2);// knownsec Initialize genes2Array

// iterate all 12 characteristics
for (uint256 i = 0; i < 12; i++) {
    // pick 4 traits for characteristic i
    uint256 j;
    for (j = 3; j >= 1; j--) {
        store.traitPos = (i * 4) + j;

        store.rand = _sliceNumber(store.randomN, 2, store.randomIndex); // 0~3
        store.randomIndex += 2;

        // 1/4 of a chance of gene swapping forward towards expressing.
        if (store.rand == 0) {
            // do it for parent 1
            store.swap = store.genes1Array[store.traitPos];
            store.genes1Array[store.traitPos] = store.genes1Array[store
                .traitPos - 1];
            store.genes1Array[store.traitPos - 1] = store.swap;
        }

        store.rand = _sliceNumber(store.randomN, 2, store.randomIndex); // 0~3
        store.randomIndex += 2;

        if (store.rand == 0) {
            // do it for parent 2
            store.swap = store.genes2Array[store.traitPos];
            store.genes2Array[store.traitPos] = store.genes2Array[store
                .traitPos - 1];
            store.genes2Array[store.traitPos - 1] = store.swap;
        }
    }
}

uint8 prevEnergyType;
store.applyEnergyBuff = true;
for (store.traitPos = 0; store.traitPos < 48; store.traitPos++) {
    store.rand = _sliceNumber(store.randomN, 1, store.randomIndex); // 0 ~ 1
    store.randomIndex += 1;

    // 50% pick from store.genes1Array
```

```
                if (store.rand == 0) {
                    store.babyArray[store.traitPos] = uint8(
                        store.genes1Array[store.traitPos]
                    );
                } else {
                    store.babyArray[store.traitPos] = uint8(
                        store.genes2Array[store.traitPos]
                    );
                }

                /**
                 * Checks for energy buff
                 * Energy buff only check for dominant gene (store.traitPos % 4 == 0) and only first 5 dominant traits (5 traits * 4 gene/traits    = 20 gene)
                 * Apply ENERGY_BUFF IFF each dominant trait & 8 > 1 and all equal.
                 * For example:
                 *      [[*3*, 9, 4, 2], [*11*, 13, 6, 42], [*3*, 24, 16, 1], [*19*, 5, 6, 8], [], ...]
                 *          3 & 8      =      11 & 8      =      3 & 8      =      19 & 8      = 3 (>2)
                 */
                if (store.traitPos % 4 == 0) {
                    uint8 dominantGene = store.babyArray[store.traitPos];

                    // short circuit energy buff if already failed
                    if (store.applyEnergyBuff && store.traitPos < 20) {
                        // a trait type is dominant gene mod 8
                        uint8 energyType = dominantGene % 8;

                        // energy buff only applicable to energy type greater than 1
                        if (energyType < 2) {
                            store.applyEnergyBuff = false;
                        } else {
                            if (store.traitPos != 0) {
                                store.applyEnergyBuff =
                                    energyType == prevEnergyType;
                            }
                            prevEnergyType = energyType;
                        }
                    }

                    if (dominantGene < 8) {
                        store.baseEnergy += 1;
                    } else if (dominantGene < 16) {
                        store.baseEnergy += 5;
                    } else if (dominantGene < 24) {
                        store.baseEnergy += 10;
                    } else {
                        store.baseEnergy += 15;
                    }
                }
            }

            store.rand = _sliceNumber(store.randomN, 2, store.randomIndex); // 0 ~ 3
            store.randomIndex += 1;

            generationFactor = _calculateGenerationFactor(
                store.rand,
                _generation,
                store.applyEnergyBuff
            );
            energy = store.baseEnergy.mul(generationFactor);
            require(energy == uint256(uint64(energy)));
            gene = _encode(store.babyArray);
        }

        /* ========== PRIVATE METHOD ========== */

        /**
         * given a number get a slice of any bits, at certain offset
         * @param _n a number to be sliced
         * @param _nbits how many bits long is the new number
         * @param _offset how many bits to skip
         */
        function _sliceNumber(
            uint256 _n,
            uint256 _nbits,
            uint256 _offset
        ) private pure returns (uint256) {// knownsec    Private method, offset slice
            // mask is made by shifting left an offset number of times
            uint256 mask = uint256((2**_nbits) - 1) << _offset;
            // AND n with mask, and trim to max of _nbits bits
            return uint256((_n & mask) >> _offset);
        }

        /**
         * Get a 5 bit slice from an input as a number
         * @param _input bits, encoded as uint
         * @param _slot from 0 to 50
```

```
    */
    function _get5Bits(uint256 _input, uint256 _slot)
        private
        pure
        returns (uint8)// knownsec Private method to obtain 5-bit slice data
    {
        return uint8(_sliceNumber(_input, uint256(5), _slot * 5));
    }

    /**
     * Parse a Alpaca gene and returns all of 12 "trait stack" that makes the characteristics
     * @param _genes alpaca gene
     * @return the 48 traits that composes the genetic code, logically divided in stacks of 4, where only the first trait of each stack may express
     */
    function _decode(uint256 _genes) private pure returns (uint8[48] memory) {// knownsec Private method to unravel the alpaca gene
        uint8[48] memory traits;
        uint256 i;
        for (i = 0; i < 48; i++) {
            traits[i] = _get5Bits(_genes, i);
        }
        return traits;
    }

    /**
     * Given an array of traits return the number that represent genes
     */

    function _encode(uint8[48] memory _traits)
        private
        pure
        returns (uint256 _genes)// knownsec Private method, encoding gene characteristic value
    {
        _genes = 0;
        for (uint256 i = 0; i < 48; i++) {
            _genes = _genes << 5;
            // bitwise OR trait with _genes
            _genes = _genes | _traits[47 - i];
        }
        return _genes;
    }

    /**
     * calculate child generation factor
     */
    function _calculateGenerationFactor(
        uint256 _rand,
        uint256 _generation,
        bool _applyEnergyBuff
    ) private pure returns (uint256 _generationFactor) {// knownsec Private method, child factor generation algorithm
        _generationFactor = BASE_GENERATION_FACTOR.add(
            uint256(2).mul(_generation)
        );

        if (_rand == 0) {
            _generationFactor = _generationFactor.sub(1);
        } else if (_rand == 1) {
            _generationFactor = _generationFactor.add(1);
        }

        if (_applyEnergyBuff) {
            _generationFactor = _generationFactor.mul(ENERGY_BUFF);
        }
    }
}
```

IAlpaToken.sol

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

pragma solidity 0.6.12;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

interface IAlpaToken is IERC20 {
    function mint(address _to, uint256 _amount) external;
}
```

ICryptoAlpaca.sol

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

pragma solidity =0.6.12;
```

```
import "@openzeppelin/contracts/token/ERC1155/IERC1155.sol";

interface ICryptoAlpaca is IERC1155 {
    function getAlpaca(uint256 _id)
        external
        view
        returns (
            uint256 id,
            bool isReady,
            uint256 cooldownEndBlock,
            uint256 birthTime,
            uint256 matronId,
            uint256 sireId,
            uint256 hatchingCost,
            uint256 hatchingCostMultiplier,
            uint256 hatchCostMultiplierEndBlock,
            uint256 generation,
            uint256 gene,
            uint256 energy,
            uint256 state
        );

    function hasPermissionToBreedAsSire(address _addr, uint256 _id)
        external
        view
        returns (bool);

    function grandPermissionToBreed(address _addr, uint256 _sireId) external;

    function clearPermissionToBreed(uint256 _alpacaId) external;

    function hatch(uint256 _matronId, uint256 _sireId)
        external
        payable
        returns (uint256);

    function crack(uint256 _id) external;
}
```

IGeneScience.sol

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

pragma solidity =0.6.12;

interface IGeneScience {
    function isAlpacaGeneScience() external pure returns (bool);

    /**
     * @dev given genes of alpaca 1 & 2, return a genetic combination
     * @param genes1 genes of matron
     * @param genes2 genes of sire
     * @param generation child generation
     * @param targetBlock target block child is intended to be born
     * @return gene child gene
     * @return energy energy associated with the gene
     * @return generationFactor buffs child energy, higher the generation larger the generationFactor
     *   energy = gene energy * generationFactor
     */
    function mixGenes(
        uint256 genes1,
        uint256 genes2,
        uint256 generation,
        uint256 targetBlock
    )
        external
        view
        returns (
            uint256 gene,
            uint256 energy,
            uint256 generationFactor
        );
}
```

MasterChef.sol

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

pragma solidity 0.6.12;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC1155/ERC1155Receiver.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
import "@openzeppelin/contracts/utils/EnumerableSet.sol";
```

```
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "../interfaces/IAlpaToken.sol";
import "../interfaces/ICryptoAlpaca.sol";

// MasterChef is the master of ALPA.
contract MasterChef is Ownable, ERC1155Receiver {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    /* ========== EVENTS ========== */

    event Deposit(address indexed user, uint256 indexed pid, uint256 amount);// knownsec  Deposit event

    event Withdraw(address indexed user, uint256 indexed pid, uint256 amount);// knownsec Withdrawal event

    /* ========== STRUCT ========== */

    // Info of each user.
    struct UserInfo {// knownsec  User structure, account balance, income
        // How many LP tokens the user has provided.
        uint256 amount;
        // Reward debt. What have been paid so far
        uint256 rewardDebt;
    }

    struct UserGlobalInfo {// knownsec User global parameters, alpacaID, income
        // alpaca associated
        uint256 alpacaID;
        // alpaca energy
        uint256 alpacaEnergy;
    }

    // Info of each pool.
    struct PoolInfo {// knownsec Pool information
        // Address of LP token contract.
        IERC20 lpToken;
        // How many allocation points assigned to this pool. ALPAs to distribute per block.
        uint256 allocPoint;
        // Last block number that ALPAs distribution occurs.
        uint256 lastRewardBlock;
        // Accumulated ALPAs per share per energy, times 1e12. See below.
        uint256 accAlpaPerShare;
        // Accumulated Share
        uint256 accShare;// knownsec share cumulative
    }

    /* ========== STATES ========== */

    // The ALPA ERC20 token
    IAlpaToken public alpa;

    // Crypto alpaca contract
    ICryptoAlpaca public cryptoAlpaca;// knownsec Encrypted Alpaca contract

    // dev address.
    address public devaddr;

    // number of ALPA tokens created per block.
    uint256 public alpaPerBlock;// knownsec Alpa produced by each block

    // Energy if user does not have any alpaca that boost the LP pool
    uint256 public constant EMPTY_ALPACA_ENERGY = 1;// knownsec alpaca incentive

    // Info of each pool.
    PoolInfo[] public poolInfo;// knownsec Pool information initialization

    // Info of each user that stakes LP tokens.
    mapping(uint256 => mapping(address => UserInfo)) public userInfo;// knownsec User pledge table

    // Info of each user that stakes LP tokens.
    mapping(address => UserGlobalInfo) public userGlobalInfo;// knownsec User pledge table

    // Total allocation poitns. Must be the sum of all allocation points in all pools.
    uint256 public totalAllocPoint = 0;

    // The block number when ALPA mining starts.
    uint256 public startBlock;

    /* ========== CONSTRUCTOR ========== */

    constructor(
        IAlpaToken _alpa,
        ICryptoAlpaca _cryptoAlpaca,
        address _devaddr,
        uint256 _alpaPerBlock,
        uint256 _startBlock
```

```
    ) public {
        alpa = _alpa;
        cryptoAlpaca = _cryptoAlpaca;
        devaddr = _devaddr;
        alpaPerBlock = _alpaPerBlock;
        startBlock = _startBlock;
    }

    /* ========== PUBLIC ========== */

    /**
     * @dev get number of LP pools
     */
    function poolLength() external view returns (uint256) {// knownsec External call to obtain the number of pool
information (poolInfo)
        return poolInfo.length;
    }

    /**
     * @dev Add a new lp to the pool. Can only be called by the owner.
     * DO NOT add the same LP token more than once. Rewards will be messed up if you do.
     */
    function add(
        uint256 _allocPoint,
        IERC20 _lpToken,
        bool _withUpdate
    ) public onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
        }
        uint256 lastRewardBlock = block.number > startBlock
            ? block.number
            : startBlock;
        totalAllocPoint = totalAllocPoint.add(_allocPoint);
        poolInfo.push(
            PoolInfo({
                lpToken: _lpToken,
                allocPoint: _allocPoint,
                lastRewardBlock: lastRewardBlock,
                accAlpaPerShare: 0,
                accShare: 0
            })
        );
    }

    /**
     * @dev Update the given pool's ALPA allocation point. Can only be called by the owner.
     */
    function set(
        uint256 _pid,
        uint256 _allocPoint,
        bool _withUpdate
    ) public onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
        }
        totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(
            _allocPoint
        );
        poolInfo[_pid].allocPoint = _allocPoint;
    }

    /**
     * @dev View `_user` pending ALPAs for a given `_pid` LP pool.
     */
    function pendingAlpa(uint256 _pid, address _user)
        external
        view
        returns (uint256)
    {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        UserGlobalInfo storage userGlobal = userGlobalInfo[msg.sender];

        uint256 accAlpaPerShare = pool.accAlpaPerShare;
        uint256 lpSupply = pool.lpToken.balanceOf(address(this));

        if (block.number > pool.lastRewardBlock && lpSupply != 0) {
            uint256 multiplier = _getMultiplier(
                pool.lastRewardBlock,
                block.number
            );
            uint256 alpaReward = multiplier
                .mul(alpaPerBlock)
                .mul(pool.allocPoint)
                .div(totalAllocPoint);
```

```
            accAlpaPerShare = accAlpaPerShare.add(
                alpaReward.mul(1e12).div(pool.accShare)
            );
        }
        return
            user
                .amount
                .mul(_safeUserAlpacaEnergy(userGlobal))
                .mul(accAlpaPerShare)
                .div(1e12)
                .sub(user.rewardDebt);
    }

    /**
     * @dev Update reward variables for all pools. Be careful of gas spending!
     */
    function massUpdatePools() public {
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            updatePool(pid);
        }
    }

    /**
     * @dev Update reward variables of the given pool to be up-to-date.
     */
    function updatePool(uint256 _pid) public {
        PoolInfo storage pool = poolInfo[_pid];
        if (block.number <= pool.lastRewardBlock) {
            return;
        }

        uint256 lpSupply = pool.lpToken.balanceOf(address(this));
        if (lpSupply == 0) {
            pool.lastRewardBlock = block.number;
            return;
        }

        uint256 multiplier = _getMultiplier(pool.lastRewardBlock, block.number);
        uint256 alpaReward = multiplier
            .mul(alpaPerBlock)
            .mul(pool.allocPoint)
            .div(totalAllocPoint);

        alpa.mint(devaddr, alpaReward.div(10));
        alpa.mint(address(this), alpaReward);

        pool.accAlpaPerShare = pool.accAlpaPerShare.add(
            alpaReward.mul(1e12).div(pool.accShare)
        );
        pool.lastRewardBlock = block.number;
    }

    /**
     * @dev Retrieve caller's Alpaca.
     */
    function retrieve() public {// knownsec Public method to retrieve the caller's Alpaca
        UserGlobalInfo storage userGlobal = userGlobalInfo[msg.sender];// knownsec Query the caller userGlobal object in the userGlobalInfo table
        require(// knownsec alpacaID is 0 processing
            userGlobal.alpacaID != 0,
            "MasterChef: you do not have any alpaca"
        );

        for (uint256 pid = 0; pid < poolInfo.length; pid++) {// knownsec Pool information traversal
            UserInfo storage user = userInfo[pid][msg.sender];

            if (user.amount > 0) {// knownsec Traverse until the user has a balance in the pool, then process
                PoolInfo storage pool = poolInfo[pid];
                updatePool(pid);
                uint256 pending = user
                    .amount
                    .mul(userGlobal.alpacaEnergy)
                    .mul(pool.accAlpaPerShare)
                    .div(1e12)
                    .sub(user.rewardDebt);
                if (pending > 0) {
                    _safeAlpaTransfer(msg.sender, pending);
                }

                user.rewardDebt = user
                    .amount
                    .mul(EMPTY_ALPACA_ENERGY)
                    .mul(pool.accAlpaPerShare)
                    .div(1e12);

                pool.accShare = pool.accShare.sub(
```

```
                    (userGlobal.alpacaEnergy.sub(1)).mul(user.amount)
                );
            }
        }
        uint256 prevAlpacaID = userGlobal.alpacaID;
        userGlobal.alpacaID = 0;
        userGlobal.alpacaEnergy = 0;

        cryptoAlpaca.safeTransferFrom(
            address(this),
            msg.sender,
            prevAlpacaID,
            1,
            ""
        );
    }

    /**
     * @dev Deposit LP tokens to MasterChef for ALPA allocation.
     */
    function deposit(uint256 _pid, uint256 _amount) public {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        UserGlobalInfo storage userGlobal = userGlobalInfo[msg.sender];
        updatePool(_pid);

        if (user.amount > 0) {
            uint256 pending = user
                .amount
                .mul(_safeUserAlpacaEnergy(userGlobal))
                .mul(pool.accAlpaPerShare)
                .div(1e12)
                .sub(user.rewardDebt);
            if (pending > 0) {
                _safeAlpaTransfer(msg.sender, pending);
            }
        }

        if (_amount > 0) {
            pool.lpToken.safeTransferFrom(
                address(msg.sender),
                address(this),
                _amount
            );
            user.amount = user.amount.add(_amount);
            pool.accShare = pool.accShare.add(
                _safeUserAlpacaEnergy(userGlobal).mul(_amount)
            );
        }

        user.rewardDebt = user
            .amount
            .mul(_safeUserAlpacaEnergy(userGlobal))
            .mul(pool.accAlpaPerShare)
            .div(1e12);
        emit Deposit(msg.sender, _pid, _amount);
    }

    /**
     * @dev Withdraw LP tokens from MasterChef.
     */
    function withdraw(uint256 _pid, uint256 _amount) public {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        require(user.amount >= _amount, "MasterChef: invalid amount");

        UserGlobalInfo storage userGlobal = userGlobalInfo[msg.sender];

        updatePool(_pid);
        uint256 pending = user
            .amount
            .mul(_safeUserAlpacaEnergy(userGlobal))
            .mul(pool.accAlpaPerShare)
            .div(1e12)
            .sub(user.rewardDebt);
        if (pending > 0) {
            _safeAlpaTransfer(msg.sender, pending);
        }
        if (_amount > 0) {
            user.amount = user.amount.sub(_amount);
            pool.lpToken.safeTransfer(address(msg.sender), _amount);
            pool.accShare = pool.accShare.sub(
                _safeUserAlpacaEnergy(userGlobal).mul(_amount)
            );
        }

        user.rewardDebt = user
```

```
                .amount
                .mul(_safeUserAlpacaEnergy(userGlobal))
                .mul(pool.accAlpaPerShare)
                .div(1e12);
            emit Withdraw(msg.sender, _pid, _amount);
        }

        /* ========== PRIVATE ========== */

        function _safeUserAlpacaEnergy(UserGlobalInfo storage userGlobal)// knownsec Private method Get user AlpacaEnergy
            private
            view
            returns (uint256)
        {
            if (userGlobal.alpacaEnergy == 0) {
                return EMPTY_ALPACA_ENERGY;
            }
            return userGlobal.alpacaEnergy;
        }

        // Safe alpa transfer function, just in case if rounding error causes pool to not have enough ALPAs.
        function _safeAlpaTransfer(address _to, uint256 _amount) private {// knownsec Private method transfer out of alpa
            uint256 alpaBal = alpa.balanceOf(address(this));// knownsec Get contract alpa balance
            if (_amount > alpaBal) {// knownsec Not enough balance
                alpa.transfer(_to, alpaBal);
            } else {
                alpa.transfer(_to, _amount);
            }
        }

        // Return reward multiplier over the given _from to _to block.
        function _getMultiplier(uint256 _from, uint256 _to)
            private
            pure
            returns (uint256)
        {
            return _to.sub(_from);
        }

        /* ========== EXTERNAL DEV MUTATION ========== */

        // Update dev address by the previous dev.
        function setDev(address _devaddr) external onlyDev {// knownsec dev is available, change dev address
            devaddr = _devaddr;
        }

        /* ========== EXTERNAL OWNER MUTATION ========== */000

        // Update number of ALPA to mint per block
        function setAlpaPerBlock(uint256 _alpaPerBlock) external onlyOwner {// knownsec The administrator is available, change mining revenue
            alpaPerBlock = _alpaPerBlock;
        }

        /* ========== ERC1155Receiver ========== */

        /**
         * @dev onERC1155Received implementation per IERC1155Receiver spec
         */
        function onERC1155Received(
            address,
            address _from,
            uint256 _id,
            uint256,
            bytes calldata
        ) external override returns (bytes4) {
            require(
                msg.sender == address(cryptoAlpaca),
                "MasterChef: received alpaca from unauthenticated contract"
            );

            require(_id != 0, "MasterChef: invalid alpaca");

            UserGlobalInfo storage userGlobal = userGlobalInfo[_from];

            // Fetch alpaca energy
            (, , , , , , , , , , , uint256 energy, ) = cryptoAlpaca.getAlpaca(_id);
            require(energy > 0, "MasterChef: invalid alpaca energy");

            for (uint256 i = 0; i < poolInfo.length; i++) {// knownsec Traverse the _from address balance income in the pool for distribution
                UserInfo storage user = userInfo[i][_from];

                if (user.amount > 0) {
                    PoolInfo storage pool = poolInfo[i];
```

```
                updatePool(i);

                uint256 pending = user
                    .amount
                    .mul(_safeUserAlpacaEnergy(userGlobal))
                    .mul(pool.accAlpaPerShare)
                    .div(1e12)
                    .sub(user.rewardDebt);
                if (pending > 0) {
                    _safeAlpaTransfer(_from, pending);
                }// knownsec Calculate the reward to be transferred and transfer
                // Update user reward debt with new energy
                user.rewardDebt = user
                    .amount
                    .mul(energy)
                    .mul(pool.accAlpaPerShare)
                    .div(1e12);

                pool.accShare = pool.accShare.add(energy.mul(user.amount)).sub(
                    _safeUserAlpacaEnergy(userGlobal).mul(user.amount)
                );// knownsec Update accshare
            }
        }

        // update user global// knownsec Update useGlobal information
        uint256 prevAlpacaID = userGlobal.alpacaID;
        userGlobal.alpacaID = _id;
        userGlobal.alpacaEnergy = energy;

        // Give original owner the right to breed// knownsec Issue of reproduction rights
        cryptoAlpaca.grandPermissionToBreed(_from, _id);

        if (prevAlpacaID != 0) {
            // Transfer alpaca back to owner
            cryptoAlpaca.safeTransferFrom(
                address(this),
                _from,
                prevAlpacaID,
                1,
                ""
            );
        }

        return
            bytes4(
                keccak256(
                    "onERC1155Received(address,address,uint256,uint256,bytes)"
                )
            );
    }

    /**
     * @dev onERC1155BatchReceived implementation per IERC1155Receiver spec
     * User should not send using batch.
     */
    function onERC1155BatchReceived(
        address,
        address,
        uint256[] memory,
        uint256[] memory,
        bytes memory
    ) external override returns (bytes4) {
        return "";
    }

    /* ========== MODIFIER ========== */

    modifier onlyDev() {
        require(devaddr == _msgSender(), "Masterchef: caller is not the dev");// knownsec Access control
        _;
    }
}
```

6. Appendix B: Vulnerability risk rating criteria

Smart contract vulnerability rating standards	
Vulnerability rating	Vulnerability rating description
High-risk vulnerabilities	Vulnerabilities that can directly cause the loss of token contracts or user funds, such as: value overflow loopholes that can cause the value of tokens to zero, fake recharge loopholes that can cause exchanges to lose tokens, and can cause contract accounts to lose ETH or tokens. Access loopholes, etc.;
Mid-risk vulnerability	Vulnerabilities that can cause loss of ownership of token contracts, such as: access control defects of key functions, call injection leading to bypassing of access control of key functions, etc.;
Low-risk vulnerabilities	Vulnerabilities that can cause the token contract to not work properly, such as: denial of service vulnerability caused by sending ETH to malicious addresses, and denial of service vulnerability caused by exhaustion of gas.

7. Appendix C: Introduction to vulnerability testing tools

7.1 Manticore

A Manticore is a symbolic execution tool for analyzing binary files and smart contracts. A Manticore consists of a symbolic Ethereum virtual machine (EVM), an EVM disassembler/assembler, and a convenient interface for automatic compilation and analysis of the Solarium body.It also incorporates Ethersplay, a Bit of Traits of Bits visual disassembler for EVM bytecode, for visual analysis. Like binaries, Manticore provides a simple command-line interface and a Python API for analyzing EVM bytecode.

7.2 Oyente

Oyente is a smart contract analysis tool that can be used to detect common bugs in smart contracts, such as reentrancy, transaction ordering dependencies, and so on.More conveniently, Oyente's design is modular, so this allows power users to implement and insert their own inspection logic to check the custom properties in their contracts.

7.3 securify. Sh

Securify verifies the security issues common to Ethereum's smart contracts, such as unpredictability of trades and lack of input verification, while fully automated and analyzing all possible execution paths, and Securify has a specific language for identifying vulnerabilities that enables the securities to focus on current security and other reliability issues at all times.

7.4 Echidna

Echidna is a Haskell library designed for fuzzy testing EVM code.

7.5 MAIAN

MAIAN is an automated tool used to find holes in Ethereum's smart contracts. MAIAN processes the bytecode of the contract and tries to set up a series of transactions to find and confirm errors.

7.6 ethersplay

Ethersplay is an EVM disassembler that includes correlation analysis tools.

7.7 IDA - evm entry

Ida-evm is an IDA processor module for the Ethereum Virtual Machine (EVM).

7.8 want - ide

Remix is a browser-based compiler and IDE that allows users to build ethereum contracts and debug transactions using Solarium language.

7.9 KnownSec Penetration Tester kit

KnownSec penetration tester's toolkit, developed, collected and used by KnownSec penetration tester engineers, contains batch automated testing tools, self-developed tools, scripts or utilization tools, etc. dedicated to testers.