知道创宇

密级 ★ | 项目组公开

智能合约审计报告

安全状态

安全

★ ★ ★ ★ ★

主测人： 知道创宇区块链安全研究团队

版本说明

修订内容	时间	修订者	版本号
编写文档	20201023	知道创宇区块链安全研究团队	V1.0

文档信息

文档名称	文档版本	文档编号	保密级别
CryptoAlpacaCity 智能合约审计报告	V1.0	CRYTOALPACACITY-ZNHY-20201023	项目组公开

声明

创宇仅就本报告出具前已经发生或存在的事实出具本报告，并就此承担相应责任。对于出具以后发生或存在的事实，创宇无法判断其智能合约安全状况，亦不对此承担责任。本报告所作的安全审计分析及其他内容，仅基于信息提供者截至本报告出具时向创宇提供的文件和资料。创宇假设:已提供资料不存在缺失、被篡改、删减或隐瞒的情形。如已提供资料信息缺失、被篡改、删减、隐瞒或反映的情况与实际情况不符的，创宇对由此而导致的损失和不利影响不承担任何责任。

目录

1. 综述

本次报告有效测试时间是从 2020 年 10 月 22 日开始到 2020 年 10 月 23 日结束，在此期间针对 **CryptoAlpacaCity 智能合约代码**的安全性和规范性进行审计并以此作为报告统计依据。

此次测试中，知道创宇工程师对智能合约的常见漏洞（见第三章节）进行了全面的分析，综合评定为**通过**。

本次智能合约安全审计结果： 通过

由于本次测试过程在非生产环境下进行，所有代码均为最新备份，测试过程均与相关接口人进行沟通，并在操作风险可控的情况下进行相关测试操作，以规避测试过程中的生产运营风险、代码安全风险。

本次测试的目标信息：

条目	描述
Token 名称	AlpaToken
代码类型	代币代码、游戏合约、以太坊智能合约代码
代码语言	solidity

合约文件及哈希：

合约文件	MD5
AlpacaPresale.sol	005c7ef61a23611da2fbe7ec6bd9479a
DateControl.sol	ad595266ddf07b79591843711226a1a8
AlpaReward.sol	24c0020a68dde0eaf0245a2621a8523d
AlpaToken.sol	b2048ea4b18e0900c8157da61002121e
AlpacaBase.sol	93d08d3d0355f4b020652b39516dbdca
AlpacaBreed.sol	ed447768bbe5522ebab390f77c8a767b

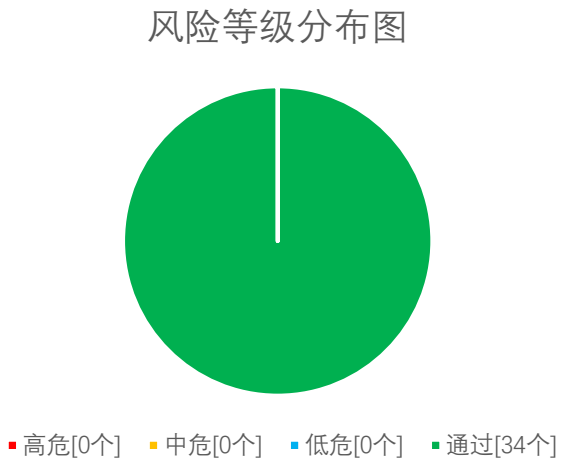
AlpacaCore.sol	b0130134c37b62418ffca530b0c8d3b4
AlpacaToken.sol	1fd9fd097e1cdb79c6beae6d268419e0
GeneScience.sol	0c97d5dc35a53e279c931169f7357b9b
IAlpaToken.sol	1740d9f3a067f04c13897b2bb99abb99
ICryptoAlpaca.sol	248c32cebd223683334e2ed1a647bebc
IGeneScience.sol	d207e656c997b805ce4fbfbb44110b03
MasterChef.sol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

2. 代码漏洞分析

2.1 漏洞等级分布

本次漏洞风险按等级统计：

安全风险等级个数统计表			
高危	中危	低危	通过
0	0	0	34



2.2 审计结果汇总说明

审计结果			
审计项目	审计内容	状态	描述
业务安全性检测	预售合约领养功能	通过	经检测，不存在安全问题。
	加密羊驼合约孵化功能	通过	经检测，不存在安全问题。
	加密羊驼合约破壳新生功能	通过	经检测，不存在安全问题。
	加密羊驼合约创建羊驼蛋及 0 代羊驼功能	通过	经检测，不存在安全问题。
代码基本漏洞检测	编译器版本安全	通过	经检测，不存在该安全问题。
	冗余代码	通过	经检测，不存在该安全问题。
	安全算数库的使用	通过	经检测，不存在该安全问题。
	不推荐的编码方式	通过	经检测，不存在该安全问题。
	require/assert 的合理使用	通过	经检测，不存在该安全问题。
	fallback 函数安全	通过	经检测，不存在该安全问题。
	tx.orgin 身份验证	通过	经检测，不存在该安全问题。
	owner 权限控制	通过	经检测，不存在该安全问题。
	gas 消耗检测	通过	经检测，不存在该安全问题。
	call 注入攻击	通过	经检测，不存在该安全问题。
	低级函数安全	通过	经检测，不存在该安全问题。
	增发代币漏洞	通过	经检测，不存在该安全问题。
	访问控制缺陷检测	通过	经检测，不存在该安全问题。
	数值溢出检测	通过	经检测，不存在该安全问题。
	算数精度误差	通过	经检测，不存在该安全问题。
	错误使用随机数检测	通过	经检测，不存在该安全问题。

	不安全的接口使用	通过	经检测，不存在该安全问题。
	变量覆盖	通过	经检测，不存在该安全问题。
	未初始化的存储指针	通过	经检测，不存在该安全问题。
	返回值调用验证	通过	经检测，不存在该安全问题。
	交易顺序依赖检测	通过	经检测，不存在该安全问题。
	时间戳依赖攻击	通过	经检测，不存在该安全问题。
	拒绝服务攻击检测	通过	经检测，不存在该安全问题。
	假充值漏洞检测	通过	经检测，不存在该安全问题。
	重入攻击检测	通过	经检测，不存在该安全问题。
	重放攻击检测	通过	经检测，不存在该安全问题。
	重排攻击检测	通过	经检测，不存在该安全问题。

3. 业务安全性检测

3.1. 预售合约领养功能【通过】

审计分析：加密羊驼预售合约的领养函数由 AlpacaPresale.sol 合约文件中的 adoptAlpaca 函数实现，用于预售期间领养羊驼。

```
function adoptAlpaca(uint256 _count)
    public
    payable
    whenInProgress
    nonReentrant
{//knownsec// 领养羊驼
    require(_count > 0, "AlpacaPresale: must adopt at least one alpaca");//knownsec// 校验数量
    require(whitelist.contains(msg.sender), "AlpacaPresale: unauthorized");//knownsec// 校验白名单

    address account = msg.sender;
    uint256 credit = canAdoptCount(account);
    require(//knownsec// 校验领养数
        _count <= credit,
        "AlpacaPresale: adoption count larger than maximum adoption limit"
    );

    require(//knownsec// 校验转账额度
        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// 随机羊驼
            counts[i] = 1;
        }

        accountAddoptionCount[account] += _count;//knownsec// 累加记录领养数

        cryptoAlpaca.safeBatchTransferFrom(//knownsec// 转移羊驼
            address(this),
            account,
            ids,
            counts,
            ""
        );
    }
```

安全建议：无。

3.2. 加密羊驼合约孵化功能【通过】

审计分析：加密羊驼合约的孵化功能主要由 AlpacaBreed.sol 合约文件中的 hatch 和_hatchEgg 函数实现，在 hatch 函数中先校验各种参数情况，通过后再调用私有的_hatchEgg 函数实现孵化。

```
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// 计算处理相关费用
        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//  优先取较低代
        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;
}
```

安全建议：无。

3.3. 加密羊驼合约破壳新生功能【通过】

审计分析：加密羊驼合约的破壳新生功能由 AlpacaBreed.sol 合约文件中的 crack 函数实现，用于孵化出指定 id 的羊驼蛋。

```
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// 校验状态
        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// 校验是否能孵化出

    // 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// 获取新生羊驼相关参数
        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//  孵化费
        msg.sender.transfer(autoCrackingFee);
    }

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

安全建议：无。

3.4. 加密羊驼合约创建羊驼蛋及 0 代羊驼功能【通过】

审计分析：加密羊驼合约创建羊驼蛋及 0 代羊驼功能只要由 AlpacaToken.sol 合约文件中代_createEgg 和_createGen0Alpaca 函数实现，用于孵化时创建羊驼蛋，以及合约 owner 创建 0 代羊驼。

```
function _createEgg(
    uint256 _matronId,
    uint256 _sireId,
    uint256 _generation,
```

```
        uint256 _cooldownEndBlock,
        address _owner
    ) internal returns (uint256) {//knownsec//  校验上代信息
        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//  校验最大 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;
}
```

安全建议：无。

4. 代码基本漏洞检测

4.1. 编译器版本安全【通过】

检查合约代码实现中是否使用了安全的编译器版本

检测结果：经检测，智能合约代码中制定了编译器版本 0.6.12，不存在该安全问题。

安全建议：无。

4.2. 冗余代码【通过】

检查合约代码实现中是否包含冗余代码

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.3. 安全算数库的使用【通过】

检查合约代码实现中是否使用了 SafeMath 安全算数库

检测结果：经检测，智能合约代码中已使用 SafeMath 安全算数库，不存在该安全问题。

安全建议：无。

4.4. 不推荐的编码方式【通过】

检查合约代码实现中是否有官方不推荐或弃用的编码方式

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.5. require/assert 的合理使用【通过】

检查合约代码实现中 require 和 assert 语句使用的合理性

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.6. fallback 函数安全【通过】

检查合约代码实现中是否正确使用 fallback 函数

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.7. tx.origin 身份验证【通过】

tx.origin 是 Solidity 的一个全局变量，它遍历整个调用栈并返回最初发送调用（或事务）的帐户的地址。在智能合约中使用此变量进行身份验证会使合约容易受到类似网络钓鱼的攻击。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.8. owner 权限控制【通过】

检查合约代码实现中的 owner 是否具有过高的权限。例如，任意修改其他账户余额等。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.9. gas 消耗检测【通过】

检查 gas 的消耗是否超过区块最大限制

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.10. call 注入攻击【通过】

call 函数调用时，应该做严格的权限控制，或直接写死 call 调用的函数。

检测结果：经检测，智能合约未使用 call 函数，不存在此漏洞。

安全建议：无。

4.11. 低级函数安全【通过】

检查合约代码实现中低级函数（call/delegatecall）的使用是否存在安全漏洞

call 函数的执行上下文是在被调用的合约中；而 delegatecall 函数的执行上下文是在当前调用该函数的合约中

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.12. 增发代币漏洞【通过】

检查在初始化代币总量后，代币合约中是否存在可能使代币总量增加的函数。

检测结果：经检测，智能合约代码中存在增发代币的功能，由于业务需要增发代币，故通过。

安全建议：无。

4.13. 访问控制缺陷检测【通过】

合约中不同函数应设置合理的权限

检查合约中各函数是否正确使用了 public、private 等关键词进行可见性修饰，检查合约是否正确定义并使用了 modifier 对关键函数进行访问限制，避免越权导致的问题。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.14. 数值溢出检测【通过】

智能合约中的算数问题是指整数溢出和整数下溢。

Solidity 最多能处理 256 位的数字（2^256-1），最大数字增加 1 会溢出得到 0。同样，当数字为无符号类型时，0 减去 1 会下溢得到最大数字值。

整数溢出和下溢不是一种新类型的漏洞，但它们在智能合约中尤其危险。溢出情况会导致不正确的结果，特别是如果可能性未被预期，可能会影响程序的可靠性和安全性。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.15. 算术精度误差【通过】

Solidity 作为一门编程语言具备和普通编程语言相似的数据结构设计，比如：变量、常量、函数、数组、函数、结构体等等，Solidity 和普通编程语言也有一个较大的区别——Solidity 没有浮点型，且 Solidity 所有的数值运算结果都只会是整数，不会出现小数的情况，同时也不允许定义小数类型数据。合约中的数值运算必不可少，而数值运算的设计有可能造成相对误差，例如同级运算：5/2*10=20，而 5*10/2=25，从而产生误差，在数据更大时产生的误差也会更大，更明显。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.16. 错误使用随机数【通过】

智能合约中可能需要使用随机数，虽然 Solidity 提供的函数和变量可以访问明显难以预测的值，如 block.number 和 block.timestamp，但是它们通常或者比看起来更公开，或者受到矿工的影响，即这些随机数在一定程度上是可预测的，所以恶意用户通常可以复制它并依靠其不可预知性来攻击该功能。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.17. 不安全的接口使用【通过】

检查合约代码实现中是否使用了不安全的接口

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.18. 变量覆盖【通过】

检查合约代码实现中是否存在变量覆盖导致的安全问题

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.19. 未初始化的储存指针【通过】

在 solidity 中允许一个特殊的数据结构为 struct 结构体，而函数内的局部变量默认使用 storage 或 memory 储存。

而存在 storage(存储器)和 memory(内存)是两个不同的概念，solidity 允许指针指向一个未初始化的引用，而未初始化的局部 stroage 会导致变量指向其他储存变量，导致变量覆盖，甚至其他更严重的后果，在开发中应该避免在函数中初始化 struct 变量。

检测结果：经检测，智能合约代码不使用结构体，不存在该问题。

安全建议：无。

4.20. 返回值调用验证【通过】

此问题多出现在和转币相关的智能合约中，故又称作静默失败发送或未经检查发送。

在 Solidity 中存在 transfer()、send()、call.value()等转币方法，都可以用于向某一地址发送 Ether，其区别在于： transfer 发送失败时会 throw，并且进行状态回滚；只会传递 2300gas 供调用，防止重入攻击；send 发送失败时会返回 false；只会传递 2300gas 供调用，防止重入攻击；call.value 发送失败时会返回 false；

传递所有可用 gas 进行调用（可通过传入 gas_value 参数进行限制），不能有效防止重入攻击。

如果在代码中没有检查以上 send 和 call.value 转币函数的返回值，合约会继续执行后面的代码，可能由于 Ether 发送失败而导致意外的结果。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.21. 交易顺序依赖【通过】

由于矿工总是通过代表外部拥有地址（EOA）的代码获取 gas 费用，因此用户可以指定更高的费用以便更快地开展交易。由于以太坊区块链是公开的，每个人都可以看到其他人未决交易的内容。这意味着，如果某个用户提交了一个有价值的解决方案，恶意用户可以窃取该解决方案并以较高的费用复制其交易，以抢占原始解决方案。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.22. 时间戳依赖攻击【通过】

数据块的时间戳通常来说都是使用矿工的本地时间，而这个时间大约能有 900 秒的范围波动，当其他节点接受一个新区块时，只需要验证时间戳是否晚于之前的区块并且与本地时间误差在 900 秒以内。一个矿工可以通过设置区块的时间戳来尽可能满足有利于他的条件来从中获利。

检查合约代码实现中是否存在有依赖于时间戳的关键功能

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.23. 拒绝服务攻击【通过】

在以太坊的世界中，拒绝服务是致命的，遭受该类型攻击的智能合约可能永远无法恢复正常工作状态。导致智能合约拒绝服务的原因可能有很多种，包括在作为交易接收方时的恶意行为，人为增加计算功能所需 gas 导致 gas 耗尽，滥用访问控制访问智能合约的 private 组件，利用混淆和疏忽等等。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.24. 假充值漏洞【通过】

在代币合约的 transfer 函数对转账发起人(msg.sender)的余额检查用的是 if 判断方式，当 balances[msg.sender] < value 时进入 else 逻辑部分并 return false，最终没有抛出异常，我们认为仅 if/else 这种温和的判断方式在 transfer 这类敏感函数场景中是一种不严谨的编码方式。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.25. 重入攻击检测【通过】

重入漏洞是最著名的以太坊智能合约漏洞，曾导致了以太坊的分叉（The DAO hack）。

Solidity 中的 call.value()函数在被用来发送 Ether 的时候会消耗它接收到的所有 gas，当调用 call.value()函数发送 Ether 的操作发生在实际减少发送者账户的余额之前时，就会存在重入攻击的风险。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.26. 重放攻击检测【通过】

合约中如果涉及委托管理的需求，应注意验证的不可复用性，避免重放攻击

在资产管理体系中，常有委托管理的情况，委托人将资产给受托人管理，委托人支付一定的费用给受托人。这个业务场景在智能合约中也比较普遍。。

检测结果：经检测，智能合约未使用 call 函数，不存在此漏洞。

安全建议：无。

4.27. 重排攻击检测【通过】

重排攻击是指矿工或其他方试图通过将自己的信息插入列表(list)或映射(mapping)中来与智能合约参与者进行“竞争”，从而使攻击者有机会将自己的信息存储到合约中。

检测结果:经检测，智能合约代码中不存在相关漏洞。

安全建议:无。

5. 附录 A：合约代码

本次测试代码来源：

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//  领养羊驼
        require(_count > 0, "AlpacaPresale: must adopt at least one alpaca");//knownsec//  校验数量
        require(whitelist.contains(msg.sender), "AlpacaPresale: unauthorized");//knownsec//  校验白名单

        address account = msg.sender;
        uint256 credit = canAdoptCount(account);
        require(//knownsec//  校验领养数
                _count <= credit,
                "AlpacaPresale: adoption count larger than maximum adoption limit"
        );

        require(//knownsec//  校验转账额度
                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//  随机羊驼
                counts[i] = 1;
        }

        accountAddoptionCount[account] += _count;//knownsec//  累加记录领养数

        cryptoAlpaca.safeBatchTransferFrom(//knownsec//  转移羊驼
                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//  仅 cryptoAlpaca 调用
                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//  仅 cryptoAlpaca 调用
                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  指定编译器版本

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  导入 SafeMath 的方法

    /* ========== 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 外部调用  锁仓挖矿
        // Gets the amount of ALPA locked in the contract
        uint256 totalAlpa = alpa.balanceOf(address(this));// knownsec 计算本 alpa 合约余额

        // Gets the amount of xALPA in existence
        uint256 totalShares = totalSupply();// knownsec 获取供应量

        // If no xALPA exists, mint it 1:1 to the amount put in
        if (totalShares == 0 || totalAlpa == 0) {// knownsec 处理无 token 情况
            _mint(msg.sender, _amount);// knownsec 为锁仓挖矿者铸币
        } 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 池比例计算
            _mint(msg.sender, what);// knownsec 为锁仓挖矿者铸币
        }

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

    /**
     * Claim back your ALPAs.
     * Unclocks the staked + gained ALPA and burns xALPA
     * @param _share amount of xALPA
     */
    function leave(uint256 _share) external {// knownsec 外部调用  取出
        // Gets the amount of xALPA in existence
        uint256 totalShares = totalSupply();// knownsec 获取供应量

        // Calculates the amount of ALPA the xALPA is worth
        uint256 what = _share.mul(alpa.balanceOf(address(this))).div(
            totalShares
        );// knownsec 利用质押凭据（xALPA）获取取出数
        _burn(msg.sender, _share);// knownsec 减少供应

        alpa.transfer(msg.sender, what);// knownsec 发送代币
    }
}
```

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 管理员使用
        _mint(_to, _amount);// knownsec 管理员使用 铸币方法  可增发
    }
}
```

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//  校验状态
            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// 校验是否能孵化出

        // 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// 获取新生羊驼相关参数
            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// 孵化费
            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// 计算处理相关费用
    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//  优先取较低代
                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// 校验上代信息
    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// 校验最大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  导入 SafeMath 的方法

    /* ========== 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  外部调用  返回永真
        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  外部调用  基因混合算法  返回混合的出生基因参数gene、energy、generationFactor
    {
        LocalStorage memory store; // knownsec 实例化 LocalStorage 对象
```

```
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 初始化 randomN

if (store.randomN == 0) {// knownsec 处理未成功初始化 randomN 情况
    // 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 初始化 randomIndex
store.genes1Array = _decode(_genes1);// knownsec 初始化 genes1Array
store.genes2Array = _decode(_genes2);// knownsec 初始化 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 私有方法 偏移切片
            // 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 私有方法 获取 5 个比特切片数据
    {
        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 私有方法 解个羊驼基因
        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 私有方法 编码基因特征值
    {
        _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 私有方法 子代因子生成算法
        _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 存款事件

    event Withdraw(address indexed user, uint256 indexed pid, uint256 amount);// knownsec 取钱事件

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

    // Info of each user.
    struct UserInfo {// knownsec 用户结构体 账户余额、收益
        // How many LP tokens the user has provided.
        uint256 amount;
        // Reward debt. What have been paid so far
        uint256 rewardDebt;
    }

    struct UserGlobalInfo {// knownsec 用户全局参数 alpacaID、收益
        // alpaca associated
        uint256 alpacaID;
        // alpaca energy
        uint256 alpacaEnergy;
    }

    // Info of each pool.
    struct PoolInfo {// knownsec 池子信息
        // 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 累计
    }

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

    // The ALPA ERC20 token
    IAlpaToken public alpa;

    // Crypto alpaca contract
    ICryptoAlpaca public cryptoAlpaca;// knownsec 加密的 Alpaca 合约

    // dev address.
    address public devaddr;

    // number of ALPA tokens created per block.
    uint256 public alpaPerBlock;// knownsec 每个区块产出的 alpa

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

    // Info of each pool.
    PoolInfo[] public poolInfo;// knownsec 池信息初始化

    // Info of each user that stakes LP tokens.
    mapping(uint256 => mapping(address => UserInfo)) public userInfo;// knownsec 用户质押表

    // Info of each user that stakes LP tokens.
    mapping(address => UserGlobalInfo) public userGlobalInfo;// knownsec 用户质押表

    // 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 外部调用 获取池信息（poolInfo）数量
        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  公开方法 检索调用者的 Alpaca
        UserGlobalInfo storage userGlobal = userGlobalInfo[msg.sender];// knownsec userGlobalInfo 表中查
询调用者 userGlobal 对象
        require(// knownsec alpacaID 为 0 处理
            userGlobal.alpacaID != 0,
            "MasterChef: you do not have any alpaca"
        );

        for (uint256 pid = 0; pid < poolInfo.length; pid++) {// knownsec  池信息遍历
            UserInfo storage user = userInfo[pid][msg.sender];

            if (user.amount > 0) {// knownsec  遍历到用户在该池有余额则进行处理
                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 私有方法 获取用户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 私有方法 转出alpa
        uint256 alpaBal = alpa.balanceOf(address(this));// knownsec 获取合约alpa余额
        if (_amount > alpaBal) {// knownsec 余额不够处置
            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 可用 变更dev地址
        devaddr = _devaddr;
    }

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

    // Update number of ALPA to mint per block
    function setAlpaPerBlock(uint256 _alpaPerBlock) external onlyOwner {// knownsec 管理员可用 改变挖矿收益
        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 遍历池中_from地址余额收益进行发放
            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  计算待转移的奖励并转移
                    // 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  更新 accshare
                }
            }

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

            // Give original owner the right to breed// knownsec  繁殖权利发放
            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  权限控制
            _;
        }
}
```

6. 附录 B：安全风险评级标准

智能合约漏洞评级标准	
漏洞评级	**漏洞评级说明**
高危漏洞	能直接造成代币合约或用户资金损失的漏洞，如：能造成代币价值归零的数值溢出漏洞、能造成交易所损失代币的假充值漏洞、能造成合约账户损失 ETH 或代币的重入漏洞等； 能造成代币合约归属权丢失的漏洞，如：关键函数的访问控制缺陷、call 注入导致关键函数访问控制绕过等； 能造成代币合约无法正常工作的漏洞，如：因向恶意地址发送 ETH 导致的拒绝服务漏洞、因 gas 耗尽导致的拒绝服务漏洞。
中危漏洞	需要特定地址才能触发的高风险漏洞，如代币合约拥有者才能触发的数值溢出漏洞等；非关键函数的访问控制缺陷、不能造成直接资金损失的逻辑设计缺陷等。
低危漏洞	难以被触发的漏洞、触发之后危害有限的漏洞，如需要大量 ETH 或代币才能触发的数值溢出漏洞、触发数值溢出后攻击者无法直接获利的漏洞、通过指定高 gas 触发的事务顺序依赖风险等。

7. 附录 C：智能合约安全审计工具简介

6.1 Manticore

Manticore 是一个分析二进制文件和智能合约的符号执行工具，Manticore 包含一个符号以太坊虚拟机（EVM），一个 EVM 反汇编器/汇编器以及一个用于自动编译和分析 Solidity 的方便界面。它还集成了 Ethersplay，用于 EVM 字节码的 Bit of Traits of Bits 可视化反汇编程序，用于可视化分析。 与二进制文件一样，Manticore 提供了一个简单的命令行界面和一个用于分析 EVM 字节码的 Python API。

6.2 Oyente

Oyente 是一个智能合约分析工具，Oyente 可以用来检测智能合约中常见的 bug，比如 reentrancy、事务排序依赖等等。更方便的是，Oyente 的设计是模块化的，所以这让高级用户可以实现并插入他们自己的检测逻辑，以检查他们的合约中自定义的属性。

6.3 securify.sh

Securify 可以验证以太坊智能合约常见的安全问题，例如交易乱序和缺少输入验证，它在全自动化的同时分析程序所有可能的执行路径，此外，Securify 还具有用于指定漏洞的特定语言，这使 Securify 能够随时关注当前的安全性和其他可靠性问题。

6.4 Echidna

Echidna 是一个为了对 EVM 代码进行模糊测试而设计的 Haskell 库。

6.5 MAIAN

MAIAN 是一个用于查找以太坊智能合约漏洞的自动化工具，Maian 处理合

约的字节码，并尝试建立一系列交易以找出并确认错误。

6.6 ethersplay

ethersplay 是一个 EVM 反汇编器，其中包含了相关分析工具。

6.7 ida-evm

ida-evm 是一个针对以太坊虚拟机（EVM）的 IDA 处理器模块。

6.8 Remix-ide

Remix 是一款基于浏览器的编译器和 IDE，可让用户使用 Solidity 语言构建以太坊合约并调试交易。

6.9 知道创宇区块链安全审计人员专用工具包

知道创宇渗透测试人员专用工具包，由知道创宇渗透测试工程师研发，收集和使用，包含专用于测试人员的批量自动测试工具，自主研发的工具、脚本或利用工具等。

知道创宇

北京知道创宇信息技术股份有限公司

咨询电话 +86(10)400 060 9587

邮　　箱 sec@knownsec.com

官　　网 www.knownsec.com

地　　址 北京市 朝阳区 望京 SOHO T2-B座-2509