In PixelmonEvolution smart contract, users usually user interact with the evolvePixelmon and claimPixelmon methods. Slither shows a warning for the claimPixelmon method for reentrancy issue.

```
Reentrancy in PixelmonEvolution.evolvePixelmon(uint256[],uint256[],uint256[],uint256,uint256,uint256,bytes)
(contracts/PixelmonEvolution.sol#181-253):
    External calls:
    - SERUM_CONTRACT.safeBatchTransferFrom(msg.sender,BURNER_ADDRESS,serumIds,serumAmounts,)
(contracts/PixelmonEvolution.sol#225)
    - PIXELMON_CONTRACT.mintEvolvedPixelmon(address(this),evolutionStage) (contracts/PixelmonEvolution.sol#234)
    State variables written after the call(s):
    - nextEvolvePixelmonId ++ (contracts/PixelmonEvolution.sol#236)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
```

Code block:

```
///@dev Evolving New Pixelmon to this Smart Contract and Saving Token Information As Staked
PIXELMON_CONTRACT.mintEvolvedPixelmon(address(this), evolutionStage);
addStakedTokenInformation(nextEvolvePixelmonId, stakedFor, msg.sender);
nextEvolvePixelmonId++;
```

Here, we are calling the mintEvolvedPixelmon of an external smart contract (Pixelmon). After that, we update a state variable “nextEvolvePixelmonId”.

Concern for this sequence:

1. If there exists any call from the “mintEvolvedPixelmon” to the same “evolvePixelmon” method then there could be a reentrancy issue.

Why this scenario will not occur:

1. The Pixelmon smart contract is deployed already. And from the “mintEvolvedPixelmon” method there is no call to the evolve pixelmon method.
2. On the other hand, we have a modifier for reentrancy attacks and also have “noContract” validation for the “evolvePixelmon” method. So, there will not be any reentrancy attack and no external method will not be able to call the “evolvePixelmon” method. We also have an attacker smart contract in the unit test scenario and we have tested this modifier validation.

The full method is given below:

```
    /// @notice Burns Serum token and mint evolved Pixelmon token
    /// @dev Owner of Pixelmon token and Serum token can call with proper signature
    /// @param pixelmonTokenIds List of Pixelmon tokenId
    /// @param serumIds List of Serum token Id
    /// @param serumAmounts List of Serum token amount
    /// @param evolutionStage Evolution stage for minting evolve Pixelmon
    /// @param nonce Nonce for the transaction to evolve pixelmon
    /// @param stakedFor Staking time period in second
    /// @param signature Signature from signer wallet
    function evolvePixelmon(
        uint256[] memory pixelmonTokenIds,
        uint256[] memory serumIds,
        uint256[] memory serumAmounts,
        uint256 evolutionStage,
        uint256 nonce,
        uint256 stakedFor,
        bytes calldata signature
    ) external nonReentrant noContracts {
        if (serumIds.length != serumAmounts.length) {
            revert InvalidData();
```

```solidity
}

uint256 totalEvolveAmount;
for (uint256 index = 0; index < serumAmounts.length; index++) {
    totalEvolveAmount += serumAmounts[index];
}

if (totalEvolveAmount != pixelmonTokenIds.length) {
    revert InvalidAmount();
}

if (nonce != nonces[msg.sender]) {
    revert InvalidNonce();
}
unchecked {
    nonces[msg.sender]++;
}

address signer = recoverSignerFromSignature(
    pixelmonTokenIds,
    serumIds,
    serumAmounts,
    evolutionStage,
    nonce,
    stakedFor,
    msg.sender,
    signature
);
```

```
        if (signer != SIGNER) {
            revert InvalidSignature();
        }

        SERUM_CONTRACT.safeBatchTransferFrom(msg.sender, BURNER_ADDRESS, serumIds, serumAmounts, "");

        uint256 evolvedTokenStartingId = nextEvolvePixelmonId;

        for (uint256 index = 0; index < pixelmonTokenIds.length; index = _uncheckedInc(index)) {
            uint256 tokenId = pixelmonTokenIds[index];
            evolutionPair[tokenId] = nextEvolvePixelmonId;

            ///@dev Evolving New Pixelmon to this Smart Contract and Saving Token Information As Staked
            PIXELMON_CONTRACT.mintEvolvedPixelmon(address(this), evolutionStage);
            addStakedTokenInformation(nextEvolvePixelmonId, stakedFor, msg.sender);
            nextEvolvePixelmonId++;

            ///@dev Transfering the E1 Pixelmon to this Smart Contract and Saving Token Information As Staked
            PIXELMON_CONTRACT.safeTransferFrom(msg.sender, address(this), tokenId, "");
            addStakedTokenInformation(tokenId, stakedFor, msg.sender);
        }

        emit PixelmonBatchEvolve(
            msg.sender,
            nonce,
            pixelmonTokenIds,
            serumIds,
            serumAmounts,
            evolutionStage,
            evolvedTokenStartingId,
            "pixelmon evolved"
        );
    }
```