Resultados del Análisis

Información del Contrato

Dirección: `0x698b1d54e936b9f772b8f58447194bbc82ec1933`

ContractName: PEEZY

CompilerVersion: v0.8.23+commit.f704f362

CompilerType: solc

OptimizationUsed: 0

Runs: 200

EVMVersion: Default

LicenseType: Unlicense

Proxy: 0

SwarmSource:
ipfs://9f842dab930e316f4eadb31a7ec3a00daa2dec8d2770c1ff2518abfddadef21c

balance: 0.012227541702609092

txCount: 2566

contractAddress: 0x698b1d54e936b9f772b8f58447194bbc82ec1933

contractCreator: 0xe8c129d6349cd9347f100f5c789b4476147f5449

txHash:
0xcc106634c3372bb984ece989e1f59fb1bae3ba8536dcbd7e7ba033c9f3f27cf0

blockNumber: 21066647

timestamp: 1730148863

Análisis de Vulnerabilidades

Absolutamente. Procederé a realizar un análisis de seguridad del contrato inteligente `PEEZY` proporcionado.

Auditoría de Seguridad del Contrato Inteligente PEEZY

Versión del Compilador Solidity: 0.8.23
Contratos Analizados: `Context`, `IERC20`, `SafeMath`, `Ownable`, `PEEZY`.

Resumen Ejecutivo

El contrato `PEEZY` implementa un token ERC20 con funcionalidades adicionales como comisiones por transferencia (impuestos), un mecanismo de swap automático de tokens recolectados por ETH, listas de exclusión de comisiones, listas de bots, y límites de transacción y cartera. La propiedad del contrato está gestionada por un `Ownable` estándar. Se han identificado varias áreas de preocupación, desde riesgos de centralización hasta posibles vulnerabilidades que podrían ser explotadas o llevar a un comportamiento no deseado del contrato.

Vulnerabilidades Encontradas

A continuación, se detallan las vulnerabilidades y observaciones encontradas:

1. Riesgo de Sándwich/MEV (Manipulación de Precio) en Swaps Automáticos
* **Severidad:** Major
* **Descripción Técnica:** La función `swapTokensForEth` llama a `uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens` con el parámetro `amountOutMin` fijado en `0`. Esto significa que el contrato aceptará cualquier cantidad de ETH a cambio de los tokens, por mínima que sea. Un atacante (por ejemplo, un minero o un bot de MEV) puede observar esta transacción en el mempool y ejecutar transacciones antes (front-running) y después (back-running) para manipular el precio del par en Uniswap, forzando al contrato a realizar el swap a una tasa muy desfavorable.
* **Ubicación:** `PEEZY.sol#L205-L214`
```solidity
function swapTokensForEth(uint256 tokenAmount) private lockTheSwap {
// ...
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
tokenAmount,
0, // <--- amountOutMin is 0
path,
address(this),
block.timestamp
);
}
```
* **Riesgo:** Pérdida de valor para el protocolo (y, por ende, para los tenedores de tokens o el `_taxWallet`) debido a swaps realizados a tasas subóptimas. Los

fondos destinados a marketing, desarrollo, etc., se verían mermados.
* **Propuesta de Mitigación:**
1. Calcular un `amountOutMin` razonable basado en el precio actual del oráculo (por ejemplo, TWAP de Uniswap si es posible, o una consulta reciente al par, aunque esto último es susceptible a manipulación en el mismo bloque).
2. Permitir que una cuenta privilegiada (idealmente un multisig o un bot off-chain seguro) establezca un `amountOutMin` dinámico o un porcentaje de slippage aceptable.
3. Considerar el uso de agregadores de DEX o soluciones anti-MEV si la complejidad y el coste de gas son aceptables.
Como mínimo, se debería establecer un slippage, por ejemplo, del 95% del valor esperado:
```solidity
// Requiere una función para obtener el precio o estimar la salida
// uint expectedAmountOut = getExpectedETHForTokens(tokenAmount);
// uint amountOutMin = expectedAmountOut * 95 / 100; // Ejemplo: 5% slippage
// uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(..., amountOutMin, ...);
```
Dado que obtener un precio on-chain seguro es complejo, una alternativa pragmática para swaps automáticos es aceptar un cierto riesgo o hacerlos menos predecibles.

2. Centralización de Privilegios del Propietario (`owner`)
* **Severidad:** Medium
* **Descripción Técnica:** El `owner` del contrato tiene poderes significativos que pueden alterar drásticamente la tokenómica y el funcionamiento del contrato post-despliegue. Estos incluyen:
* `removeLimits()`: Puede eliminar `_maxTxAmount` y `_maxWalletSize`.
* `removeTransferTax()`: Puede establecer el `_transferTax` a `0`.
* `addBots(address[] memory bots_)` y `delBots(address[] memory notbot)`: Puede añadir o quitar direcciones de la lista de bots, impidiendo o permitiendo sus transacciones. Esto podría usarse para bloquear competidores o usuarios específicos.
* `openTrading()`: Controla cuándo se inicia el trading y se añade la liquidez inicial.
* **Ubicación:** Funciones `removeLimits`, `removeTransferTax`, `addBots`, `delBots`, `openTrading`.
* **Riesgo:** Un propietario malicioso o comprometido podría abusar de estos privilegios. Por ejemplo, podría eliminar límites y tasas justo antes de vender una gran cantidad de tokens, o añadir direcciones a la lista de bots para impedir que vendan. Aunque común en tokens de este tipo, es un riesgo de confianza.
* **Propuesta de Mitigación:**
1. **Timelock:** Implementar un contrato `Timelock` donde todas las acciones privilegiadas deban pasar por un período de retraso antes de ejecutarse, dando tiempo a la comunidad para reaccionar.
2. **Multisig:** Transferir la propiedad a una cartera multisig controlada por varios individuos de confianza.
3. **Renuncia de Propiedad Parcial:** Considerar la posibilidad de que algunas

funciones críticas (como `addBots`) se vuelvan inmutables o se eliminen después de un período inicial si ya no son necesarias. La función `renounceOwnership()` existe, pero esto eliminaría toda capacidad de gestión.

3. Centralización de Privilegios de `_taxWallet`
* **Severidad:** Medium
* **Descripción Técnica:** La dirección `_taxWallet` (inicialmente el `owner`) tiene privilegios para:
* `reduceFee(uint256 _newFee)`: Puede reducir `_finalBuyTax` y `_finalSellTax`. Notablemente, la condición `require(_newFee <= _finalBuyTax && _newFee <= _finalSellTax)` significa que las tasas finales solo pueden reducirse, no aumentarse, mediante esta función.
* `manualSwap()`: Puede forzar un swap de todos los tokens del contrato por ETH.
* `manualsend()`: Puede forzar el envío de todo el ETH del contrato a `_taxWallet`.
* Recibe todos los ETH recolectados de las comisiones.
* **Ubicación:** Funciones `reduceFee`, `manualSwap`, `manualsend`. Variable `_taxWallet`.
* **Riesgo:** Si la dirección `_taxWallet` se ve comprometida, o si es controlada por una entidad maliciosa, puede manipular las tasas (solo a la baja con `reduceFee`) y drenar los fondos acumulados (ETH y tokens) del contrato a través de `manualSwap` y `manualsend`.
* **Propuesta de Mitigación:**
1. **Multisig:** Asegurar que `_taxWallet` sea una cartera multisig.
2. **Transparencia:** Si `_taxWallet` es una EOA, comunicar claramente quién la controla y bajo qué condiciones se utilizarán sus funciones.
3. Considerar que `_taxWallet` no pueda ser el mismo que el `owner` si se quiere una separación de poderes, aunque actualmente se inicializa al `owner`.

4. Mecanismo de Swap de Comisiones Ineficiente y Potencialmente Costoso en Gas
* **Severidad:** Medium
* **Descripción Técnica:** La función `_transfer`, cuando activa el swap automático de comisiones (`swapTokensForEth`), lo hace con la cantidad `min(amount, min(contractTokenBalance, _maxTaxSwap))`. El `amount` aquí es el `amount` de la transferencia que disparó el swap. Si este `amount` es muy pequeño (por ejemplo, 1 wei de token), pero el `contractTokenBalance` es grande y supera `_taxSwapThreshold`, el contrato solo swapeará una cantidad mínima de tokens (1 wei en este ejemplo).
* **Ubicación:** `PEEZY.sol#L175`
```solidity
swapTokensForEth(min(amount, min(contractTokenBalance, _maxTaxSwap)));
```
* **Riesgo:** Esto puede llevar a swaps muy pequeños y frecuentes si las transacciones que los disparan son pequeñas. Cada swap incurre en costes de gas. Si los swaps son demasiado pequeños, el gas gastado podría ser significativo en relación al valor de ETH obtenido, haciendo el mecanismo de recolección de comisiones ineficiente. El contrato podría acumular muchos tokens de comisión sin poder swapearlos eficientemente si las transacciones que lo activan son

consistentemente pequeñas.
* **Propuesta de Mitigación:** Cambiar la lógica para swapear una cantidad basada en el balance acumulado en el contrato, no en el `amount` de la transacción que lo dispara:
```solidity
// En _transfer, dentro de la lógica de swap:
uint256 tokensToSwap = min(contractTokenBalance, _maxTaxSwap); // No usar 'amount' de la tx
if (tokensToSwap > 0) { // Añadir un cheque para evitar swaps de 0
swapTokensForEth(tokensToSwap);
}
```

5. Uso de `transfer()` para Enviar ETH
* **Severidad:** Minor
* **Descripción Técnica:** Las funciones `sendETHToFee` (privada, usada internamente) y `manualsend` (pública, llamada por `_taxWallet`) utilizan `_taxWallet.transfer(amount)` para enviar ETH. El método `.transfer()` reenvía una cantidad fija de gas (2300 gas). Si `_taxWallet` es un contrato inteligente que requiere más gas en su función `receive()` o `fallback()`, la transferencia fallará.
* **Ubicación:** `PEEZY.sol#L219` (sendETHToFee), `PEEZY.sol#L268` (manualsend)
```solidity
function sendETHToFee(uint256 amount) private {
_taxWallet.transfer(amount);
}
```
* **Riesgo:** Si `_taxWallet` es un contrato (por ejemplo, un multisig Gnosis Safe que no tiene una función `receive` explícita pero usa `fallback` para aceptar ETH, o un contrato de vesting/distribución), la transferencia de ETH podría fallar consistentemente, bloqueando los fondos ETH en el contrato PEEZY. Dado que `_taxWallet` se inicializa con `_msgSender()` (el deployer), es probable que sea una EOA inicialmente, pero podría cambiarse a un contrato si la propiedad (y por ende el `_taxWallet` inicial) se transfiere a un contrato.
* **Propuesta de Mitigación:** Utilizar `.call{value: amount}("")` para enviar ETH, que reenvía todo el gas disponible y permite verificar el éxito de la llamada.
```solidity
function sendETHToFee(uint256 amount) private {
(bool success, ) = _taxWallet.call{value: amount}("");
require(success, "Failed to send ETH");
}
```
Esto es especialmente importante para `manualsend` y cualquier lógica que envíe ETH a `_taxWallet`.

6. Redundancia de SafeMath con Solidity 0.8.x
* **Severidad:** Informational
* **Descripción Técnica:** El contrato utiliza la biblioteca `SafeMath` para operaciones aritméticas. A partir de la versión 0.8.0 de Solidity, las comprobaciones de desbordamiento (overflow) y subdesbordamiento

(underflow) están incorporadas directamente en el compilador, por lo que `SafeMath` es redundante.
* **Ubicación:** Uso de `.add()`, `.sub()`, `.mul()`, `.div()` en todo el contrato.
* **Riesgo:** No representa un riesgo de seguridad directo, pero añade complejidad innecesaria al código y un ligero aumento en el coste de despliegue y gas en tiempo de ejecución en comparación con el uso de operadores nativos.
* **Propuesta de Mitigación:** Eliminar el uso de `SafeMath` y utilizar los operadores aritméticos estándar de Solidity (`+`, `-`, `*`, `/`). El compilador 0.8.23 ya se encargará de revertir en caso de overflow/underflow.

7. Lógica de Impuestos Potencialmente Confusa para Transferencias entre Carteras
* **Severidad:** Minor (Potencial para Comportamiento Inesperado)
* **Descripción Técnica:** En la función `_transfer`, la lógica para calcular `taxAmount` para transferencias que no involucran al par Uniswap (es decir, transferencias de cartera a cartera) depende del valor de `_buyCount`.
```solidity
if (from != owner() && to != owner()) {
// ...
if(_buyCount==0){ // Aplica si _buyCount es CERO
taxAmount = amount.mul((_buyCount>_reduceBuyTaxAt)?
_finalBuyTax:_initialBuyTax).div(100);
// Como _buyCount es 0, (_buyCount > _reduceBuyTaxAt) es false.
// taxAmount = amount.mul(_initialBuyTax).div(100);
}
if(_buyCount>0){ // Aplica si _buyCount es MAYOR A CERO
taxAmount = amount.mul(_transferTax).div(100);
}
// ... esta taxAmount se sobreescribe si es una tx con el pair
}
```
La variable `_buyCount` solo se incrementa en las compras desde el par Uniswap (`from == uniswapV2Pair`). Esto significa:
* Todas las transferencias de cartera a cartera (que no son desde/hacia el owner) *antes de la primera compra desde el par Uniswap* estarán sujetas al `_initialBuyTax`.
* Todas las transferencias de cartera a cartera *después de la primera compra desde el par Uniswap* estarán sujetas al `_transferTax`.
* **Ubicación:** `PEEZY.sol#L147-L153`
* **Riesgo:** Este comportamiento puede no ser intuitivo para los usuarios. Podrían esperar que `_transferTax` se aplique a todas las transferencias entre carteras desde el principio, o que `_initialBuyTax` solo se aplique a las primeras compras reales. Esto podría llevar a confusión o a que los usuarios paguen una tasa diferente a la esperada en transferencias P2P tempranas.
* **Propuesta de Mitigación:**
1. Clarificar esta lógica en la documentación del proyecto.
2. Si la intención es que `_transferTax` se aplique a todas las transferencias P2P (no relacionadas con el par), la lógica debería simplificarse para no depender de `_buyCount` para estas transferencias, o inicializar `_buyCount` de una manera

que refleje la política deseada desde el inicio. Por ejemplo, si se quiere `_transferTax` siempre para P2P:
```solidity
// Dentro de if (from != owner() && to != owner())
// Y antes de la lógica de compra/venta con el par
taxAmount = amount.mul(_transferTax).div(100); // Default para P2P

// Luego la lógica de compra/venta que sobreescribe taxAmount si aplica
if (from == uniswapV2Pair && ...) { /* ... */ }
if (to == uniswapV2Pair && ...) { /* ... */ }
```

8. La Reducción de Impuestos de Venta Depende de `_buyCount`
* **Severidad:** Informational
* **Descripción Técnica:** En la lógica de venta (`to == uniswapV2Pair`), la tasa de impuesto aplicada (`_initialSellTax` o `_finalSellTax`) depende de si `_buyCount` ha superado `_reduceSellTaxAt`.
```solidity
if(to == uniswapV2Pair && from!= address(this) ){
taxAmount = amount.mul((_buyCount>_reduceSellTaxAt)?
_finalSellTax:_initialSellTax).div(100);
}
```
* **Ubicación:** `PEEZY.sol#L166-L168`
* **Riesgo:** Es una decisión de diseño que la reducción de la tasa de venta dependa del número de compras. No es una vulnerabilidad, pero es un comportamiento específico que debe ser comprendido por los usuarios. Normalmente, se esperaría que la reducción de la tasa de venta dependa de un contador de ventas o de un número de bloques/tiempo transcurrido.
* **Propuesta de Mitigación:** Asegurar que esta mecánica esté claramente documentada y comunicada a la comunidad para evitar sorpresas sobre cuándo se reducen las tasas de venta.

9. La función `reduceFee` tiene una Lógica Restrictiva
* **Severidad:** Informational
* **Descripción Técnica:** La función `reduceFee` permite a `_taxWallet` modificar `_finalBuyTax` y `_finalSellTax`. Sin embargo, la condición `require(_newFee <= _finalBuyTax && _newFee <= _finalSellTax)` implica que estas tasas solo pueden ser reducidas, y ambas deben ser reducidas al mismo valor `_newFee`. No pueden aumentarse ni ajustarse independientemente una vez que `_taxWallet` decide usar esta función.
* **Ubicación:** `PEEZY.sol#L252-L257`
```solidity
function reduceFee(uint256 _newFee) external{
require(_msgSender()==_taxWallet);
require(_newFee<=_finalBuyTax && _newFee<=_finalSellTax); // Restricción
_finalBuyTax=_newFee;
_finalSellTax=_newFee;
}
```

```
* **Riesgo:** Limita la flexibilidad de `_taxWallet` para ajustar las tasas finales. Si se necesita un ajuste al alza o independiente, no sería posible a través de esta función. Esto podría ser intencional para proteger a los usuarios de aumentos de tasas.
* **Propuesta de Mitigación:** Si se requiere mayor flexibilidad, la condición podría modificarse o eliminarse, pero esto aumentaría el poder de `_taxWallet`. La lógica actual es más segura desde la perspectiva del usuario contra aumentos inesperados de "tasas finales". Confirmar que este comportamiento es el deseado.

**10. Dirección del Router de Uniswap Hardcodeada**
* **Severidad:** Informational
* **Descripción Técnica:** La dirección del router de Uniswap V2 está hardcodeada como `0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D`.
* **Ubicación:** `PEEZY.sol#L240`
```solidity
uniswapV2Router =
IUniswapV2Router02(0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);
```
* **Riesgo:** Esto vincula el contrato a la red Ethereum Mainnet (o redes de prueba que usen esta misma dirección para el router). Si se quisiera desplegar en otra cadena con un router diferente, o si Uniswap V2 fuera actualizado o reemplazado de una manera que cambie la dirección del router, esta parte del contrato necesitaría ser modificada y redesplegada.
* **Propuesta de Mitigación:** Para mayor flexibilidad, la dirección del router podría pasarse como un argumento del constructor o ser configurable por el `owner` (con las debidas precauciones de seguridad como Timelock). Sin embargo, para un token destinado a una cadena específica, esto es una práctica común.

11. Posible Código Muerto o Innecesario en `openTrading`
* **Severidad:** Informational
* **Descripción Técnica:** En la función `openTrading`, se aprueba al router de Uniswap para gastar los tokens LP del par (`uniswapV2Pair`) que pertenecen al contrato PEEZY:
`IERC20(uniswapV2Pair).approve(address(uniswapV2Router), type(uint).max);`
Sin embargo, la liquidez añadida a través de `uniswapV2Router.addLiquidityETH` especifica que los tokens LP resultantes se envían al `owner()`:
`uniswapV2Router.addLiquidityETH{...}(..., owner(), block.timestamp);`
Por lo tanto, el contrato PEEZY en sí mismo no debería poseer tokens LP del par principal inmediatamente después de `addLiquidityETH`, haciendo que la aprobación mencionada sea innecesaria o para un caso de uso no obvio (por ejemplo, si el contrato PEEZY adquiere tokens LP de otra manera).
* **Ubicación:** `PEEZY.sol#L244`
* **Riesgo:** No representa un riesgo de seguridad. Simplemente podría ser código innecesario que aumenta ligeramente el coste de gas de `openTrading`.
* **Propuesta de Mitigación:** Si no hay un escenario en el que el contrato PEEZY necesite gestionar directamente sus propios tokens LP (del par principal), esta
```

línea podría eliminarse. Si es para un caso futuro, debería documentarse.

Conclusión

El contrato `PEEZY` implementa una serie de características comunes en tokens con comisiones (tax tokens). Si bien la lógica central de ERC20 es estándar, las funcionalidades adicionales introducen complejidades y riesgos.
Las principales preocupaciones son el **riesgo de MEV/sándwich** en los swaps automáticos (Major) y los **poderes centralizados** en manos del `owner` y `_taxWallet` (Medium). La **ineficiencia del mecanismo de swap de comisiones** (Medium) también debería abordarse para optimizar la recolección de ETH.

Se recomienda encarecidamente abordar las vulnerabilidades de severidad "Major" y "Medium" antes de un despliegue en producción o, si ya está desplegado, comunicar claramente estos riesgos a los usuarios y considerar estrategias de mitigación post-despliegue (como el uso de multisigs para `owner` y `_taxWallet`). Las observaciones de severidad "Minor" e "Informational" deberían revisarse para mejorar la claridad, eficiencia y robustez del contrato.

Dado que el contrato ya está desplegado y auditado, es posible que algunas de estas observaciones hayan sido consideradas decisiones de diseño con riesgos aceptados por el equipo del proyecto. Sin embargo, desde una perspectiva de seguridad estricta, los puntos señalados son válidos.

Información del Contrato

Dirección: `0xb29e475b69f843046a757747943c00dce8a3d982`

ContractName: SPECTRE

CompilerVersion: v0.8.27+commit.40a35a09

CompilerType: solc

OptimizationUsed: 1

Runs: 200

EVMVersion: Default

LicenseType: Unlicense

Proxy: 0

SwarmSource:
ipfs://bc7b07ac82e098a00481f41c78126613ddc04e0c2921656b4b34c380e7b05cdd

balance: 0.00001

txCount: 781

contractAddress: 0xb29e475b69f843046a757747943c00dce8a3d982

contractCreator: 0x1fd1a7b908d271a6683e8ef19a2c602604a9fac1

txHash:
0x797e8709ecb93ba8a0bd914b5cd72ee40ec16800866ed65fb9b82e5753abdac3

blockNumber: 20842902

timestamp: 1727451455

Análisis de Vulnerabilidades

Absolutamente. Procederé a realizar un análisis de seguridad exhaustivo del contrato inteligente `SPECTRE` proporcionado.

Análisis de Seguridad del Contrato Inteligente `SPECTRE`

Versión del Compilador: `0.8.27`

He identificado las siguientes vulnerabilidades y puntos de mejora en el contrato `SPECTRE`:

1. Riesgo de Slippage Extremo en la Creación de Liquidez (`openTrading`)

* **Severidad:** Crítica
* **Descripción Técnica:** La función `openTrading` llama a `uniswapV2Router.addLiquidityETH` con los parámetros `amountTokenMin` y `amountETHMin` establecidos en `0`. Esto significa que la transacción de adición de liquidez se ejecutará sin importar cuán desfavorable sea el precio, aceptando cualquier cantidad de tokens de liquidez (LP) a cambio de los tokens y ETH aportados.
* **Ubicación o fragmento de código afectado:**
```solidity
function openTrading() external onlyOwner {
// ...
uniswapV2Router.addLiquidityETH{value: address(this).balance}(
address(this),
balanceOf(address(this)),
0, // amountTokenMin = 0
0, // amountETHMin = 0
owner(),
block.timestamp
);
// ...
}
```
* **Riesgo que representa en el entorno de la red Ethereum:** Un atacante (MEV

bot) puede anticipar (front-run) la llamada a `openTrading`. El atacante podría:
1. Crear el par con una cantidad mínima de liquidez justo antes de que `openTrading` sea ejecutada.
2. Cuando `openTrading` se ejecute, la gran cantidad de tokens y ETH del contrato se añadirá a un pool con una proporción de precios ya manipulada por el atacante.
3. Debido a `amountTokenMin = 0` y `amountETHMin = 0`, la transacción se completará, pero el `owner()` recibirá una cantidad irrisoria de tokens LP.
4. El atacante puede entonces retirar su liquidez inicial junto con una porción significativa de los fondos aportados por el contrato, resultando en una pérdida sustancial de la liquidez inicial del proyecto.
* **Propuesta de mitigación específica:**
* Calcular y establecer valores mínimos aceptables para `amountTokenMin` y `amountETHMin` basados en el ratio esperado y un porcentaje de slippage tolerable. Esto requiere obtener el precio de reserva del par o una estimación razonable antes de llamar a `addLiquidityETH`.
* Alternativamente, considere un mecanismo de adición de liquidez en dos pasos o el uso de un servicio especializado para mitigar el front-running en la creación de liquidez.

2. Riesgo de Slippage en Swaps de Impuestos (`swapTokensForEth`)

* **Severidad:** Mayor
* **Descripción Técnica:** La función `swapTokensForEth` llama a `uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens` con el parámetro `amountOutMin` establecido en `0`. Esto significa que el contrato está dispuesto a aceptar cualquier cantidad de ETH a cambio de los tokens que está intercambiando, sin importar cuán bajo sea el tipo de cambio.
* **Ubicación o fragmento de código afectado:**
```solidity
function swapTokensForEth(uint256 tokenAmount) private lockTheSwap {
// ...
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
tokenAmount,
0, // amountOutMin = 0
path,
address(this),
block.timestamp
);
}
```
* **Riesgo que representa en el entorno de la red Ethereum:** Un atacante (MEV bot) puede hacer un "sandwich attack" a esta transacción. Cuando el contrato intenta intercambiar tokens por ETH, el atacante puede:
1. Comprar ETH (haciendo subir su precio relativo a los tokens) justo antes de la transacción del contrato.
2. Dejar que la transacción del contrato se ejecute a un tipo de cambio desfavorable (debido a `amountOutMin = 0`).
3. Vender el ETH (bajando su precio) justo después, obteniendo un beneficio a

expensas del contrato (y, por ende, de los tenedores de tokens, ya que se reduce la cantidad de ETH obtenida para la billetera de marketing/desarrollo).
* **Propuesta de mitigación específica:**
* Calcular un `amountOutMin` razonable antes de llamar a la función de swap. Esto se puede hacer consultando el precio actual en el par de Uniswap y aplicando un deslizamiento máximo aceptable (por ejemplo, 1-3%).
* Se puede usar una librería de oráculo o llamar directamente a `getAmountsOut` del router de Uniswap para estimar la cantidad de ETH que se debería recibir.

3. Lógica de Impuestos Inicial Potencialmente Problemática en `_transfer`

* **Severidad:** Media
* **Descripción Técnica:** En la función `_transfer`, existe un bloque de código que calcula `taxAmount` si `_buyCount == 0`.
```solidity
if (_buyCount == 0) {
taxAmount = amount
.mul(
(_buyCount > _reduceBuyTaxAt) // Esta condición siempre será falsa si _buyCount == 0
? _finalBuyTax
: _initialBuyTax
)
.div(100);
}
```
Posteriormente, si la transferencia es una compra desde el par de Uniswap (`from == uniswapV2Pair && ...`), `taxAmount` se recalcula y sobrescribe. Lo mismo ocurre si es una venta al par de Uniswap. Sin embargo, si la transferencia no es ni una compra ni una venta al par de Uniswap (por ejemplo, una transferencia entre dos EOAs) y `_buyCount` es `0` (es decir, antes de la primera compra desde Uniswap), la `taxAmount` calculada en el primer bloque persistirá. Esto podría llevar a que transferencias P2P sean tasadas con `_initialBuyTax` si ocurren antes de cualquier compra en Uniswap, lo cual podría no ser la intención.
* **Ubicación o fragmento de código afectado:**
```solidity
function _transfer(
address from,
address to,
uint256 amount
) private {
// ...
if (
from != owner() &&
to != owner() &&
from != _spectreMultiSegWallet &&
to != _spectreMultiSegWallet
) {
// ...
```

```
if (_buyCount == 0) { // <--- Bloque problemático
taxAmount = amount
.mul(
(_buyCount > _reduceBuyTaxAt)
? _finalBuyTax
: _initialBuyTax
)
.div(100);
}

if (
from == uniswapV2Pair && // ... Lógica de compra que sobrescribe taxAmount
) {
// ...
taxAmount = amount.mul( ... ).div(100);
// ...
}

if (to == uniswapV2Pair && from != address(this)) { // ... Lógica de venta que sobrescribe taxAmount
taxAmount = amount.mul( ... ).div(100);
}
// ... Si no es compra ni venta LP, y _buyCount == 0, la taxAmount del primer if persiste.
}
// ...
}
```

* **Riesgo que representa en el entorno de la red Ethereum:** Podría causar que los usuarios paguen impuestos inesperadamente en transferencias que no son de compra/venta en DEX, si estas ocurren antes de la primera compra registrada desde el par de Uniswap. Esto podría llevar a una mala experiencia de usuario y confusión.
* **Propuesta de mitigación específica:**
* Aclarar la intención de este bloque. Si la intención es solo aplicar impuestos a las compras/ventas de Uniswap, entonces la asignación inicial a `taxAmount` debería ocurrir *dentro* de los bloques condicionales que identifican una compra o venta de Uniswap.
* Si la intención es aplicar un impuesto a *todas* las transferencias (excepto las excluidas) solo cuando `_buyCount == 0`, entonces la lógica actual es correcta pero debe ser documentada claramente. Sin embargo, esto es un comportamiento inusual.
* La forma más común y clara es inicializar `taxAmount = 0` y solo asignarle un valor dentro de las condiciones específicas de compra/venta de DEX.

4. Centralización de Poder y Potencial de Abuso

* **Severidad:** Media
* **Descripción Técnica:** El `owner` del contrato y la dirección `_spectreWallet` (que

es el deployer inicial y es inmutable) tienen poderes significativos sobre el contrato.
* `owner` puede:
* Llamar a `renounceOwnership()`, lo que haría inoperables todas las funciones `onlyOwner`.
* Llamar a `removeSpectreLimits()` para eliminar `_maxTxAmount` y `_maxWalletSize`.
* Llamar a `addBots()` y `delBots()` para incluir en listas negras/blancas direcciones, bloqueando sus transacciones.
* Llamar a `openTrading()` (una sola vez) para iniciar el trading.
* Llamar a `updateSpectreTaxWallet()` para cambiar la dirección `_spectreMultiSegWallet` que recibe los fondos de impuestos.
* `_spectreWallet` (deployer original, inmutable) puede:
* Llamar a `reduceSpectreSellTaxFee()` para reducir el impuesto de venta (hasta un mínimo de `_finalSellTax` actual, que a su vez podría ser 0 si `_finalSellTax` se establece en 0).
* Llamar a `sendTokensToSpectreMultiSeg()` para retirar cualquier token ERC20 (que no sea el propio token del contrato) que haya sido enviado accidentalmente al contrato, a la dirección `_spectreMultiSegWallet`.
* Llamar a `swapTaxToETH()` para forzar un intercambio de los tokens del contrato acumulados por impuestos a ETH.
* Llamar a `sendETHToSpectreMultiSeg()` para retirar todo el ETH del contrato a la dirección `_spectreMultiSegWallet`.
* **Ubicación o fragmento de código afectado:** Múltiples funciones con modificadores `onlyOwner` y `onlySpectreWallet`.
* **Riesgo que representa en el entorno de la red Ethereum:** Si la(s) clave(s) privada(s) de la dirección `owner` y/o `_spectreWallet` se ven comprometidas, o si las entidades que las controlan actúan de manera maliciosa, podrían:
* Cambiar la billetera de impuestos a una controlada por el atacante y drenar los fondos de impuestos.
* Censurar usuarios añadiéndolos a la lista de bots.
* Manipular los parámetros del token (límites de transacción, impuestos de venta) de forma perjudicial.
* Drenar todos los tokens (que no sean SPCTR) y ETH del contrato.
La inmutabilidad de `_spectreWallet` significa que el deployer original siempre conservará estos privilegios, incluso si la propiedad del contrato se transfiere.
* **Propuesta de mitigación específica:**
* Para una mayor descentralización, considere implementar un sistema de gobernanza basado en tokens o un contrato Timelock para cambios críticos en los parámetros o la ejecución de funciones privilegiadas.
* Si la centralización es intencionada, debe ser comunicada claramente a los usuarios para que comprendan los riesgos de confianza asociados.
* Documentar claramente la distinción de roles entre `owner` y `_spectreWallet` y las implicaciones de la inmutabilidad de `_spectreWallet`.

5. Posible Agotamiento de Gas en Funciones de Gestión de Bots (`addBots`, `delBots`)

* **Severidad:** Menor

* **Descripción Técnica:** Las funciones `addBots` y `delBots` iteran sobre un array (`bots_` o `notbot`) para actualizar el estado de cada dirección. Si se proporciona un array muy grande, la transacción podría consumir más gas del límite del bloque, causando que la transacción falle (DoS para la función).
* **Ubicación o fragmento de código afectado:**

```solidity
function addBots(address[] memory bots_) public onlyOwner {
for (uint256 i = 0; i < bots_.length; i++) {
bots[bots_[i]] = true;
}
}

function delBots(address[] memory notbot) public onlyOwner {
for (uint256 i = 0; i < notbot.length; i++) {
bots[notbot[i]] = false;
}
}
```

* **Riesgo que representa en el entorno de la red Ethereum:** El `owner` podría verse incapacitado para añadir o eliminar bots si la lista es demasiado grande para ser procesada en una sola transacción. Esto es principalmente un riesgo para la operatividad del `owner`, no directamente para los fondos de los usuarios, a menos que la gestión de bots sea crítica para la seguridad.
* **Propuesta de mitigación específica:**
* Implementar un mecanismo de paginación o procesamiento por lotes, donde el `owner` pueda llamar a la función múltiples veces con subconjuntos más pequeños del array.
* Establecer un límite superior razonable en la longitud del array que se puede procesar en una sola llamada.

6. Uso Redundante de `SafeMath` con Solidity 0.8.x

* **Severidad:** Informativa
* **Descripción Técnica:** El contrato utiliza la librería `SafeMath` para operaciones aritméticas. A partir de la versión 0.8.0 de Solidity, las operaciones aritméticas (+, -, *) ya incluyen comprobaciones de desbordamiento (overflow) y subdesbordamiento (underflow) por defecto, revirtiendo la transacción si ocurren. La división por cero también revierte de forma nativa.
* **Ubicación o fragmento de código afectado:** Todas las instancias de `a.add(b)`, `a.sub(b)`, `a.mul(b)`, `a.div(b)`.
* **Riesgo que representa en el entorno de la red Ethereum:** No representa un riesgo de seguridad directo. Sin embargo, añade una ligera sobrecarga de gas en cada operación matemática y aumenta el tamaño del bytecode del contrato innecesariamente.
* **Propuesta de mitigación específica:**
* Considerar eliminar el uso de `SafeMath` para las operaciones de suma, resta y multiplicación, y utilizar los operadores nativos de Solidity (por ejemplo, `c = a + b;`).
* Para la división, `a.div(b)` de `SafeMath` proporciona un mensaje de error

personalizado, lo cual puede ser útil, pero `a / b` nativo también revertirá en caso de división por cero. La elección aquí es entre un mensaje de error personalizado y un ligero ahorro de gas.

Consideraciones Adicionales (Informativas):

* **Licencia SPDX:** El contrato usa `UNLICENSE`. Aunque válido, las licencias permisivas más comunes como `MIT` son ampliamente reconocidas y preferidas en el ecosistema.
* **Constantes vs. Inmutables:** Algunas variables como `_initialBuyTax`, `_initialSellTax`, etc., son `private constant`. Esto es correcto. `_spectreWallet` es `immutable`, lo cual también es una buena práctica ya que se establece en el constructor y no cambia.
* **Visibilidad de Funciones:** La visibilidad de las funciones parece ser apropiada en general (e.g. `private` para `_transfer`, `_approve`).
* **Mecanismo `lockTheSwap`:** El modificador `lockTheSwap` es una protección estándar contra reentradas en la función `swapTokensForEth`, lo cual es bueno.
* **ETH en el Contrato:** El contrato puede recibir ETH a través de la función `receive() external payable {}` y a través de los swaps. La función `sendETHToMultiSeg` (y su uso en `swapTaxToETH` y `sendETHToSpectreMultiSeg`) gestiona la extracción de este ETH. El fallback a `_spectreWallet.transfer(amount)` en caso de fallo del `.call` a `_spectreMultiSegWallet` es un buen mecanismo de seguridad para evitar que el ETH quede bloqueado si `_spectreMultiSegWallet` es un contrato que no puede recibir ETH.

Conclusión sobre la Seguridad General:

El contrato `SPECTRE` implementa funcionalidades comunes en tokens con mecanismos de impuestos y liquidez automática. Las vulnerabilidades más significativas identificadas se relacionan con el **riesgo de slippage extremo** durante la creación de liquidez inicial y durante los swaps de tokens de impuestos, lo cual puede llevar a pérdidas financieras directas para el proyecto. La **lógica de impuestos inicial** también presenta una ambigüedad que podría llevar a un comportamiento inesperado. Los **riesgos de centralización** son inherentes al diseño con un `owner` y una `_spectreWallet` privilegiados, y deben ser asumidos por los usuarios. Las demás incidencias son de menor impacto o de carácter informativo.

Recomiendo encarecidamente abordar las vulnerabilidades de severidad Crítica y Mayor antes de cualquier despliegue o, si ya está desplegado, comunicar los riesgos y considerar una migración o actualización si es factible. La versión del compilador `0.8.27` mitiga los riesgos de overflow/underflow aritmético, lo cual es positivo.

Información del Contrato

Dirección: `0x9eaebd7e73d97e78c77fab743e6ffa1b550e224c`

ContractName: REXAS_FINANCE

CompilerVersion: v0.8.26+commit.8a97fa7a

CompilerType: solc

OptimizationUsed: 0

Runs: 200

EVMVersion: Default

LicenseType: MIT

Proxy: 0

SwarmSource:
ipfs://91548d3bbbe8c29c0bfedceb505659a7efd56906d6660c2922826162eaa53a25

balance: 15.45672100445071

txCount: 1385

contractAddress: 0x9eaebd7e73d97e78c77fab743e6ffa1b550e224c

contractCreator: 0x48ee57b01d226d1e1efae1fb15f45d9a11bd018a

txHash:
0x496ab249dcb7f77f2160da1af76438b4b3e82093ff2ce5abcf96b2c4d6ea2ac5

blockNumber: 20700162

timestamp: 1725729431

Análisis de Vulnerabilidades

Absolutamente. Procederé a realizar un análisis de seguridad exhaustivo del contrato inteligente `REXAS_FINANCE` proporcionado.

Análisis de Seguridad del Contrato Inteligente REXAS_FINANCE

Versión del Compilador Solidity: `^0.8.20`
Esta versión incluye protecciones integradas contra desbordamientos aritméticos (overflow/underflow), lo que hace que el uso de `SafeMath` para operaciones aritméticas básicas (suma, resta, multiplicación, división) sea redundante, aunque su uso para mensajes de error personalizados en `sub` sigue siendo una práctica válida.

Resumen General:
El contrato implementa un token ERC20 estándar con funcionalidades adicionales como una lista blanca (`whitelist`), la capacidad de habilitar el trading (`enableTrading`), y funciones de utilidad para el propietario (`removeStuckEth`, `removeStuckToken`). La propiedad del contrato se gestiona mediante un

mecanismo similar al `Ownable` de OpenZeppelin. El código es generalmente claro y sigue patrones comunes. Sin embargo, se han identificado algunos puntos que merecen atención.

A continuación, se detallan las vulnerabilidades y observaciones encontradas:

Vulnerabilidades Detectadas:

1. **Vulnerabilidad: Centralización Excesiva de Poder del Propietario (Owner)**
* **Severidad:** Major
* **Descripción Técnica:** El propietario (`owner`) del contrato tiene privilegios significativos que, si se abusa de ellos o si la clave privada del propietario se ve comprometida, podrían tener un impacto severo en el token y sus tenedores. Específicamente, el `owner` puede:
* Habilitar el trading (`enableTrading`).
* Añadir o remover direcciones de la `whitelist` a discreción (`setWhitelist`). Esto permite al `owner` eximir a cualquier cuenta de la restricción de `trading` incluso si está deshabilitado para el público general, o viceversa, potencialmente bloquear transferencias de usuarios no exentos si el trading está deshabilitado.
* Retirar todos los ETH enviados al contrato (`removeStuckEth`).
* Retirar cualquier token ERC20 enviado al contrato (`removeStuckToken`).
* Renunciar a la propiedad (`renounceOwnership`), lo que podría dejar funciones críticas inoperables si no se transfiere a un nuevo propietario activo.
* Transferir la propiedad a una nueva dirección (`transferOwnership`).
* **Ubicación o fragmento de código afectado:**
```solidity
modifier onlyOwner() { ... }
function enableTrading() external onlyOwner { ... }
function setWhitelist(address _user, bool _exmpt) external onlyOwner{ ... }
function removeStuckEth(address _receiver) public onlyOwner { ... }
function removeStuckToken(address _token, address _receiver, uint256 _amount) public onlyOwner { ... }
function renounceOwnership() public virtual onlyOwner { ... }
function transferOwnership(address newOwner) public virtual onlyOwner { ... }
```
* **Riesgo que representa en el entorno de la red Ethereum:** Un propietario malicioso o una clave de propietario comprometida pueden centralizar el control sobre aspectos cruciales del token, manipular la capacidad de comerciar para usuarios específicos, y extraer fondos (ETH u otros tokens) que estén en posesión del contrato. Esto socava la descentralización y la confianza de los usuarios.
* **Propuesta de mitigación específica:**
* **Multisig:** Transferir la propiedad del contrato a una billetera multi-firma (multisig) donde las acciones críticas requieran la aprobación de múltiples partes.
* **Timelock:** Implementar un contrato `Timelock` para las funciones sensibles. Las invocaciones a estas funciones se pondrían en cola y solo se ejecutarían después de un período de retraso predefinido, dando tiempo a la comunidad para auditar y reaccionar ante cambios propuestos.

* **Separación de Privilegios:** Si es posible, separar los roles. Por ejemplo, un rol para gestionar la whitelist y otro para gestionar fondos de emergencia, en lugar de un único `owner` todopoderoso.
* **Documentación Clara:** Si se mantienen estos poderes, documentar claramente estos riesgos para los usuarios y la comunidad.

2. **Vulnerabilidad: Desviación del Estándar ERC20 en Transferencias de Monto Cero**
* **Severidad:** Minor
* **Descripción Técnica:** La función interna `_transfer` (utilizada por `transfer` y `transferFrom`) incluye el requisito `require(amount > 0, "Rexas Finance: Amount must be greater than zero");`. El estándar EIP-20 especifica que las transferencias de valor cero deben tratarse como transferencias normales y deben emitir el evento `Transfer`. Revertir en transferencias de monto cero es una desviación del estándar.
* **Ubicación o fragmento de código afectado:**
```solidity
function _transfer(address from, address to, uint256 amount) private {
// ...
require(amount > 0, "Rexas Finance: Amount must be greater than zero");
// ...
}
```
* **Riesgo que representa en el entorno de la red Ethereum:** Aunque el impacto financiero directo es nulo, esta desviación puede causar problemas de interoperabilidad con otras dApps, exchanges o herramientas que esperan un comportamiento estrictamente conforme al EIP-20. Podrían fallar integraciones que intenten realizar transferencias de valor cero.
* **Propuesta de mitigación específica:**
* Eliminar la condición `require(amount > 0, ...)`. Las transferencias de cero tokens deben permitirse y emitir el evento `Transfer` correspondiente, según el EIP-20. Si se desea prevenir "spam" de eventos, esto debe considerarse como una característica del diseño, pero no como un cumplimiento estricto del estándar.

3. **Vulnerabilidad: Potencial Condición de Carrera (Race Condition) en `approve` (Clásico de ERC20)**
* **Severidad:** Informational / Minor
* **Descripción Técnica:** Como en muchas implementaciones de ERC20, la función `approve` es susceptible a una condición de carrera. Si un usuario aprueba una cantidad X a un gastador y luego decide cambiarla a Y enviando una nueva transacción `approve(spender, Y)`, un gastador malicioso que observe la transacción original podría intentar gastar la cantidad X antes de que la segunda transacción (aprobando Y) se procese.
* **Ubicación o fragmento de código afectado:**
```solidity
function approve(
address spender,
uint256 amount
) public override returns (bool) {
```

_approve(_msgSender(), spender, amount);
return true;
}
```

* **Riesgo que representa en el entorno de la red Ethereum:** Un atacante podría gastar una cantidad mayor a la que el usuario finalmente pretendía, si logra que su transacción `transferFrom` se procese entre las dos transacciones `approve` del usuario (o antes de la que reduce el allowance).
* **Propuesta de mitigación específica:**
* El contrato ya implementa `increaseAllowance` y `decreaseAllowance`, que son las mitigaciones estándar recomendadas por OpenZeppelin para este problema.
* Educar a los usuarios para que, al cambiar un allowance existente, primero lo establezcan en 0 y luego al nuevo valor, o utilicen `increaseAllowance`/`decreaseAllowance`. Dado que las mitigaciones están presentes, la severidad se considera baja, pero es importante que los usuarios sean conscientes.

4. **Vulnerabilidad: Interacción Externa en `removeStuckToken` sin Patrón Checks-Effects-Interactions Estricto**
* **Severidad:** Informational / Minor
* **Descripción Técnica:** La función `removeStuckToken` realiza una llamada externa `IERC20(_token).transfer(_receiver, _amount)`. Si bien esta función es `onlyOwner`, y el `owner` controla los parámetros `_token`, `_receiver` y `_amount`, es una buena práctica seguir el patrón Checks-Effects-Interactions. En este caso, no hay cambios de estado *después* de la llamada externa dentro de esta función, lo que minimiza el riesgo de reentrada directa que afecte a `removeStuckToken`. Sin embargo, si el token `_token` es malicioso o tiene un comportamiento inesperado (por ejemplo, una "token tax" al transferir o una bomba de gas), podría afectar la transacción.
* **Ubicación o fragmento de código afectado:**
```solidity
function removeStuckToken(address _token, address _receiver, uint256 _amount) public onlyOwner {
IERC20(_token).transfer(_receiver, _amount);
}
```

* **Riesgo que representa en el entorno de la red Ethereum:** El riesgo principal es bajo debido al `onlyOwner`. Si el token `_token` es malicioso y reentrante, podría intentar llamar a otras funciones del contrato `REXAS_FINANCE`. Sin embargo, no podría eludir el modificador `onlyOwner` en funciones protegidas. Un riesgo más plausible es que la llamada a `_token.transfer` consuma todo el gas o falle por razones internas del token `_token`, haciendo que la función `removeStuckToken` falle.
* **Propuesta de mitigación específica:**
* Para una mayor robustez contra comportamientos inesperados de tokens externos (aunque sea llamado por el `owner`), considerar el uso de un modificador `nonReentrant` si se anticipan interacciones más complejas o si se modificara estado después de la llamada externa.
* Dado que el `owner` controla los parámetros, se asume que seleccionará tokens
```

y receptores confiables. Mantener la función simple como está es aceptable, pero se señala por completitud y buenas prácticas generales.

Observaciones Adicionales (No clasificadas como vulnerabilidades directas):

* **Uso de `SafeMath` con Solidity `^0.8.20`:**
Como se mencionó, Solidity `^0.8.0` y versiones posteriores incluyen comprobaciones de desbordamiento y subdesbordamiento de forma nativa. El uso de `SafeMath` para `add`, `mul`, `div` es redundante y añade un ligero coste de gas. El uso de `SafeMath.sub` con mensajes de error personalizados es una elección de diseño válida. Para un contrato nuevo, se podría omitir `SafeMath` para las operaciones básicas.

* **Lógica de `trading` y `whitelist`:**
La lógica `if (!whitelist[from] && !whitelist[to]) { require(trading, ...); }` permite que las transferencias ocurran si `trading` está habilitado, O si el remitente (`from`) está en la `whitelist`, O si el receptor (`to`) está en la `whitelist`. Esto es flexible y permite al `owner` facilitar transferencias específicas (por ejemplo, a exchanges, o para airdrops) antes de habilitar el trading público. Esta funcionalidad es poderosa y su uso debe ser transparente para la comunidad.

* **Función `receive() external payable {}`:**
Esta función permite al contrato recibir ETH directamente. Es necesaria si, por ejemplo, el contrato interactúa con un DEX y necesita recibir ETH como resultado de un swap de tokens por ETH donde el contrato mismo es el destinatario. Su presencia es benigna y a menudo necesaria para ciertos casos de uso DeFi. La función `removeStuckEth` permite al `owner` recuperar estos ETH.

Conclusión:

El contrato `REXAS_FINANCE` implementa un token ERC20 con características de gestión comunes. La principal preocupación de seguridad radica en la **centralización de poderes en la figura del `owner` (Severidad: Major)**, lo cual es un riesgo común en muchos proyectos de tokens pero debe ser gestionado con cuidado (multisig, timelock) para generar confianza. La desviación del estándar ERC20 respecto a las **transferencias de monto cero (Severidad: Minor)** es un problema de cumplimiento que podría afectar la interoperabilidad. La **condición de carrera en `approve` (Severidad: Informational/Minor)** es un problema inherente al diseño de `approve` en ERC20, mitigado por la presencia de `increase/decreaseAllowance`. La interacción externa en `removeStuckToken` es de bajo riesgo pero se menciona por adherencia a las mejores prácticas.

Se recomienda encarecidamente abordar la centralización de poder y corregir la desviación del estándar ERC20. La versión del compilador utilizada (`^0.8.20`) es moderna y ayuda a prevenir una clase entera de errores aritméticos. En general,

con las mitigaciones propuestas, el contrato puede alcanzar un buen nivel de seguridad.

← Página Anterior Inicio