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

Análisis de Seguridad del Contrato `PEEZY`

Resumen Ejecutivo
El contrato implementa un token ERC-20 con lógica adicional de trading (taxes, transferTax), gestión de bots, límites de transacciones y funcionalidades de swap para sostener la economía del token. Está basado en Solidity 0.8.23 (donde los desbordamientos y subdesbordamientos aritméticos están chequeados nativamente). El contrato contiene lógica compleja de impuestos, control de trading y swap, y funciones administrativas.

Vulnerabilidades y Debilidades Identificadas

---

1. Exceso de privilegios y control administrativo [Major]
Descripción técnica:
Las siguientes funciones privilegiadas (`addBots`, `delBots`, `removeLimits`, `removeTransferTax`, `reduceFee`, `manualSwap`, `manualsend`, `openTrading`) están protegidas con `onlyOwner` o comprobaciones de `_taxWallet`. El owner puede modificar drásticamente los parámetros del token (por ejemplo, eliminar todas las restricciones de trading/transacciones con `removeLimits` o deshabilitar por completo el impuesto con `removeTransferTax`) y manipular la lista de "bots" (blacklist).

Fragmento relevante:
```solidity
function removeLimits() external onlyOwner {
_maxTxAmount = _tTotal;
_maxWalletSize = _tTotal;
emit MaxTxAmountUpdated(_tTotal);
}
function removeTransferTax() external onlyOwner {
_transferTax = 0;
emit TransferTaxUpdated(0);
}
function addBots(address[] memory bots_) public onlyOwner { ... }
function delBots(address[] memory notbot) public onlyOwner { ... }
```

Riesgo:
- **Rug pull:** El owner puede modificar el comportamiento del token en detrimento de los usuarios, eliminar límites de balances/transacciones o activar/desactivar impuestos de manera arbitraria.
- **Blacklist:** El owner puede bloquear forzosamente las cuentas que desee, impidiendo que transfieran tokens y, por tanto, confiscando de facto sus fondos.

Mitigación:
- Reducción paulatina y posterior renuncia a la propiedad (por ejemplo, transferir ownership a `address(0)` una vez lanzado el token para mitigar el riesgo de

manipulación posterior).
- Documentación transparente del potencial de poder administrativo en la fase inicial del proyecto.
- Alternativamente, limitar los rangos de parámetros que puede ajustar el owner y/o implementar un governance más restrictivo.

---

2. Manipulación/Acaparamiento de fondos vía `swapTokensForEth` y `sendETHToFee` [Medium]
Descripción técnica:
Las funciones `swapTokensForEth`, `manualSwap` y el cobro automático de tasas trasladan los fondos (ETH recolectado por tasas) a `_taxWallet`, que **es arbitraria y controlada por el owner**, sin restricción del uso ni rendición de cuentas.

Fragmentos relevantes:
```solidity
address payable private _taxWallet;
// ...
function sendETHToFee(uint256 amount) private {
_taxWallet.transfer(amount);
}
// En _transfer() y en manualSwap(), manualsend()
```

Riesgo:
El owner/taxWallet puede apropiarse de todos los fondos recaudados por cada swap. No hay mecanismos de transparencia ni rendición de cuentas para asegurar un uso legítimo. Esto puede ser un vector de *rug* o apropiación indebida de fondos recolectados de la comunidad.

Mitigación:
- Declarar públicamente el destino de los fondos.
- Implementar medidas de auditoría o mecanismos automáticos de distribución de los fondos para limitar el control unilateral.
- Divulgar clarísimamente a los usuarios/inversores que todos los fondos de las tasas van a parar a una dirección arbitraria.

---

3. Exclusión arbitraria en la recaudación de tasas (tax exemption/whitelisting) [Medium]
Descripción técnica:
La asignación de exclusión en `_isExcludedFromFee` está establecida solo en el constructor, pero **el owner puede cambiar `_taxWallet` a su antojo** (ya que este es un campo modificable indirectamente), y no hay función para modificar la lista después.

Riesgo:

El owner puede excluir direcciones del pago de tasas coliando en su propio beneficio, aunque actualmente el contrato no expone una función para cambiar `_isExcludedFromFee`.

Mitigación:
- Si se decide que nunca cambiarán las direcciones exentas, considerar hacer la variable `immutable` o eliminar el mecanismo si no va a ser usado.
- Si se requiere flexibilidad, transparentar funciones administrativas para modificar la lista.

---

4. Potencial Denegación de Servicio (DoS) vía blacklisting [Medium]
Descripción técnica:
El owner puede añadir cualquier dirección a la lista de bots y bloquearla permanentemente de transferencias (incluyendo el futuro de pools de liquidez, bridges, exchanges o cualquier usuario inocente).

Fragmento:
```solidity
function addBots(address[] memory bots_) public onlyOwner { ... }
modifier (en _transfer): require(!bots[from] && !bots[to]);
```

Riesgo:
DoS irrecuperable para cualquier dirección (incluido el router de un DEX, bridges, proveedores de liquidez, etc.), lo que puede interrumpir el funcionamiento normal del token en exchanges/descentralizados al antojo del propietario.

Mitigación:
- Limitar esta capacidad una vez que el contrato esté suficientemente propagado (por ejemplo, bloqueando la lista después de lanzar).
- Transparencia extrema con la comunidad sobre los criterios de inclusión/exclusión como "bot".

---

5. No hay protección contra MEV/sniping/bot trading en el lanzamiento [Informational]
Descripción técnica:
Aunque hay una lógica de whitelist y límites para maxTx/maxWallet, no existe un mecanismo más refinado (anti-bot, anti-sniping, cooldown, etc.) más allá del simple blacklist-control manual.

Riesgo:
Durante los primeros bloques tras el lanzamiento, bots pueden evadir límites si el owner no reacciona velozmente, apropiándose de liquidez/acciones clave.

Mitigación:
Implementar protecciones automatizadas (TPS control, time-locks, anti-MEV) en la lógica inicial para bloquear bots anticiapadamente durante el lanzamiento.

---

6. Falta de protección frente a potencial "honeypot" [Informational]
Descripción técnica:
Debido a la complejidad en los impuestos y en cómo se calcula la variable `_buyCount`, puede generarse confusión/errores, llevando a tasas del 50% para ciertas transferencias si las condiciones no se evalúan correctamente. No parece intencionado, pero merece atención.

En concreto:
```solidity
if(_buyCount==0) { ... }
if(_buyCount>0) { taxAmount = amount.mul(_transferTax).div(100);}
...
if (from == uniswapV2Pair && ... ) {
...
_buyCount++;
}
```
Esto hace que transferencias generales (no de compra/venta Uniswap), reciban un tax arbitrario inicialmente alto según la secuencia de compras.

Riesgo:
Hasta que no haya una compra, cualquier transferencia paga `_transferTax` (50%) lo cual puede inutilizar el token, ya que transferencias de usuario a usuario pueden perder la mitad de su valor.

Mitigación:
- Comprobar y testear exhaustivamente la inicialización y el flujo de `buyCount`. Documentar el comportamiento esperado.
- Ajustar la lógica para no penalizar innecesariamente casos de uso legítimo (por ejemplo, transferencias normales previas a la primera compra).

---

7. Deficiencias menores en diseño (usabilidad, gas, view/pure, etc.) [Minor]
- `SafeMath` es redundante en Solidity >=0.8.x. Aunque no es inseguro, incrementa el gas inútilmente y oscurece el código.
- La función `manualsend()` podría lanzar si el balance en ETH es 0.
- Variables internas relevantes son privadas, dificultando la transparencia para herramientas automáticas externas.

No Se Detectan Vulnerabilidades Críticas

- No existen fallas reentrantes; las únicas llamadas externas a transferencias de ETH están siempre tras bloqueos de estado y transferencias de token/ETH gestionadas por Uniswap.
- No hay potencial de desbordamientos/subdesbordamientos graves (por uso de Solidity >=0.8.0 y redundante SafeMath).
- No hay errores lógicos inmediatos en cálculos de balance o verificaciones de acceso.

Resumen: Riesgo y Seguridad General

- El mayor riesgo real y probado es el **poder administrativo absoluto**: blacklist, ajuste de parémetros e incautación del treasury. Si el ownership se mantiene, hay potencial de abuso y DoS, aunque el código en sí no es vulnerable a exploits externos ni reentrancy ni manipulación de lógica de Uniswap.
- No hay bugs clásicos de ERC-20, no hay transferencias de ETH indebidas o backdoor en el sentido tradicional.

Recomendaciones Generales

1. **Renunciar a ownership** cuando se considere el contrato suficientemente maduro.
2. **Transparencia radical** sobre uso y destino de tasas (ETH) y lógica del blacklist para proteger la legitimidad del proyecto y la confianza de la comunidad.
3. Eliminar/limitar lógica de administración crítica y hacer públicas las direcciones claves del treasury.
4. Eliminar bloques de código innecesario (SafeMath redundante) y mejorar comentarios/documentación para auditores futuros.

Conclusión

El contrato no presenta vulnerabilidades técnicas críticas propias del lenguaje o la lógica de transferencias, pero sí contiene privilegios administrativos mayores que permiten al owner tomar acciones dañinas o abusivas hacia los usuarios, lo que debe ser estrictamente mitigado vía transferencia/renuncia de ownership y transparencia. No se observan puertas traseras técnicas invisibles.

Si necesitas un análisis comparativo con una auditoría específica publicada

en Etherscan, puedes proporcionar su resumen para contrastar los puntos aquí analizados.

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

Auditoría de Seguridad – Contrato `SPECTRE`

Compilador: Solidity `0.8.27`
Librerías: Uso de SafeMath innecesario para Solidity >=0.8.0, pero funcional.
Rol especial: `_spectreMultiSegWallet`, `_spectreWallet`
Funciones administrativas: solo `owner` o `onlySpectreWallet`
Exchange: Integración UniswapV2

---

1. Vulnerabilidades encontradas

1.1 Pérdida de control admin/irreversibilidad tras `renounceOwnership`

Severidad: Major

Descripción:
La función `renounceOwnership` permite que el propietario establezca el owner a `address(0)`. Sin embargo, algunas funciones críticas (`removeSpectreLimits`, `addBots`, `delBots`, `updateSpectreTaxWallet`, `openTrading`) requieren `onlyOwner`. Una vez renunciado el ownership, ¡nadie podrá ejecutar nunca más estas funciones administrativas! Esto puede bloquear la gobernanza y la gestión de emergencias del token.

Ubicación:
```solidity
function renounceOwnership() public virtual onlyOwner {
emit OwnershipTransferred(_owner, address(0));
_owner = address(0);
}
```

Riesgo:
En caso de un incidente (ejemplo: un bot toma control de trading), no será posible actualizar la blacklist, añadir/quitar bots, o indeed actualizar la wallet de fees. No hay función `transferOwnership`, así que no se puede recuperar el control.

Mitigación:
Valorar la incorporación de una función `transferOwnership`, o restringir/eliminar `renounceOwnership` a favor de una revocación reversible (ej.: dos pasos, delay, o `pendingOwner`). Notificar a los usuarios del peligro de perder control admin tras renunciar ownership.

1.2 Uso innecesario de SafeMath (Redundancia)

Severidad: Informational

Descripción:
Solidity 0.8.x tiene overflow/underflow checks por defecto; no es necesario el uso de la librería SafeMath, lo que añade redundancia de bytecode.

Ubicación:
- Uso en todo el contrato.

Riesgo:
Aumento en tamaño de despliegue, sin impacto funcional.

Mitigación:
Eliminar el uso de SafeMath. No impacta la seguridad, pero sí la optimización/gas.

1.3 Transferencia de tokens externos vía función administrativa sin whitelist de tokens

Severidad: Medium

Descripción:
La función `sendTokensToSpectreMultiSeg` permite transferir cualquier token ERC20 desde el contrato a la wallet de fees. Esto podría incluir accidentalmente a tokens críticos o tokens enviados a este contrato como protección o por error (ej. airdrops maliciosos).

Ubicación:
```solidity
function sendTokensToSpectreMultiSeg(address tokenAddress)
external onlySpectreWallet returns (bool success) { ... }
```

Riesgo:
Teóricamente, si un usuario envía tokens a este contrato (por ejemplo, LP-tokens, stablecoins, otros ERC20), el dueño puede drenar esos tokens, lo que puede ser nocivo si no hay intencionalidad y si el owner/fundador abusa o liquida activos ajenos. No afecta al token principal.

Mitigación:
Añadir una whitelist de tokens permitidos explícitamente para withdraw; en su defecto, dejar claro el riesgo para los usuarios.

1.4 Bot list centralizada sin transparencia/on-chain event

Severidad: Minor

Descripción:
El owner puede agregar o eliminar cualquier wallet a la lista de bots. No se emiten eventos ni hay mecanismos on-chain para auditar cambios, lo que otorga mucho poder sin transparencia al owner.

Ubicación:

- `addBots`, `delBots` (no emiten eventos)

Riesgo:
Riesgo de censura arbitraria, manipulación de trading/lists. Los usuarios no sabrán on-chain cuándo ni quién fue bloqueado/desbloqueado.

Mitigación:
Emitir eventos en cada alteración y documentar cuál es el punto de contacto para reclamaciones si se marca por error a una dirección.

1.5 Función `openTrading` solo ejecutable una vez, irreversible

Severidad: Minor

Descripción:
`openTrading()` solo puede llamarse una vez; si hay un error durante la ejecución del proceso de apertura (ej. transacción de liquidez falla), no es posible volver a intentarlo. Además, la función aprueba el par de Uniswap V2 usando `type(uint256).max` sin control posterior ni opción de revocación.

Ubicación:
```solidity
function openTrading() external onlyOwner {
require(!tradingOpen, "trading is already open");
// ...
uniswapV2Pair = ...createPair...;
uniswapV2Router.addLiquidityETH{value: address(this).balance}(...);
// ...
tradingOpen = true;
}
```

Riesgo:
Riesgo moderado (deployment). En la práctica, ya desplegado, pero si el deployer comete un error, puede bloquear el trading sin remediarlo.

Mitigación:
Generalmente bajo, pero es mejor implementar mecanismos de fallback o permitir un relanzamiento de la apertura si la primera falla (no crítico por ya estar desplegado y auditado, pero sí relevante en desarrollo).

1.6 Función `removeSpectreLimits` elimina límites de transferencia y de wallet

Severidad: Informational

Descripción:
Permite al owner remover todos los limits a la cantidad de tokens en una wallet o transacción. El contrato no notifica on-chain el cambio salvo por un evento mínimo, ni ofrece posibilidad de restaurar los límites.

Ubicación:
```solidity
function removeSpectreLimits() external onlyOwner
```

Riesgo:
No es crítico, pero los usuarios deben ser advertidos de que el owner puede alterar (o eliminar para siempre) las restricciones anti-whale. No afecta la seguridad per se, pero sí la confianza.

Mitigación:
Documentar y alertar visiblemente en interfaces públicas y scripts de comunidad.

1.7 Error de lógica en la condición de asignación de impuestos en `_transfer` (probable bug, *verificado*)

Severidad: Medium

Descripción:
En la función `_transfer`, la primera condición:
```solidity
if (_buyCount == 0) {
taxAmount = amount.mul(
(_buyCount > _reduceBuyTaxAt)
? _finalBuyTax
: _initialBuyTax
).div(100);
}
```
Pero, según la lógica, *solo asigna impuesto cuando `_buyCount == 0`*. Luego, hay otras comprobaciones en la sección de compra y venta que ya recalculan `taxAmount`, sobrescribiendo el valor anterior.

Riesgo:
El bloque es innecesario, nunca impacta salvo posibilidad de bug futuro (introducción de caminos incontrolados). En la implementación actual, parece redundante pero no dañino, ya que los bloques de compra/venta lo sobreescriben.

Mitigación:
Eliminar esta línea de código redundante para prevenir errores de lógica futuros

y mejorar legibilidad.

1.8 Potencial Denegación de Servicio en retiros ETH de fallback

Severidad: Minor

Descripción:
La función `sendETHToMultiSeg` intenta transferir el saldo ETH a `_spectreMultiSegWallet`. Si falla, hace fallback a `_spectreWallet`. Si ambos fallan (ej.: wallets con fallback reverting), los fondos pueden quedar bloqueados.

Ubicación:
```solidity
(bool success, ) = _spectreMultiSegWallet.call{value: amount}("");
if (!success) {
_spectreWallet.transfer(amount);
}
```

Riesgo:
Probabilidad baja salvo que ambas wallets sean "maliciosas" o sin lógica fallback, pero posible lockeo de ETH en el contrato.

Mitigación:
Mejorar lógica de fallback y monitoreo de estados de transferencia. Considerar la posibilidad de varios intentos, o mecanismo pull-over-push.

1.9 No hay mecanismos anti-"honeypot" ni control de transferencias por el usuario

Severidad: Informational

Descripción:
El contrato no impide transferencias a él mismo, ni valida explícitamente a los destinatarios, ni protege contra los honeypots (aunque permite vender siempre que no seas bot, no hay trampas de bloqueo de sell). Es relevante para el análisis (pues usuarios esperan "libre comercio"), pero no es estrictamente una vulnerabilidad del contrato.

---

2. NO vulnerabilidades detectadas (descartadas e inspección crítica)

Overflow/Underflow:
No posible por Solidity >=0.8.0

Reentrancy:
No hay función externa vulnerable; swaps de Uniswap usan lockTheSwap.

Apropiación de fondos propios/no hay MINT:
Suministro es fijo, sin funciones de minteo o quema. Sin riesgos de dilución.

Control de fee:
El owner puede *bajar* el fee de venta, no incrementarlo arbitrariamente.

Transparencia inicial/preminado:
Asignación del 22% a multiSegWallet y el resto al contrato (para liquidity/fair launch). Control aceptable y claramente especificado.

No hay ninguna función "emergencia"/setBalance/manipulación directa de balances:
Solo vía approve, transfer, etc.

---

3. Resumen de riesgos y recomendaciones principales

#	Severidad	Riesgo clave	Recomendación
1	Major	Renuncia de ownership irreversible y funciones admin bloqueadas	Añadir transferOwnership/mecanismo reversible – documentar riesgos.
2	Medium	Edición/sustracción de tokens ERC20 ajenos por owner sin whitelist	Whitelistear tokens, documentar en interfaz.
3	Medium	Códigos redundantes/lógica confusa puede inducir a bugs	Quitar líneas innecesarias, simplificar `_transfer`.
4	Minor	Bot list arbitraria e invisible	Emitir evento `BotStatusChanged(address,bool)`.
5	Minor	Lógicas de fallback en transferencias ETH pueden fallar	Más robustez en fallback, monitoreo manual.
6	Informational	Uso de SafeMath redundante	Eliminar en próximas versiones, sin prisa.

---

4. **Conclusión profesional**

En resumen:
El contrato `SPECTRE`, según el código presentado y la versión activa desplegada analiza, no tiene vulnerabilidades críticas asociadas a pérdida de fondos, control arbitrario de fees, mint infinito, ni honeypot. La lógica de trading, límites anti-ballena y la configuración de impuestos es razonablemente segura.

Las amenazas principales provienen de la centralización del control admin (con la opción de renunciar sin retorno y el control de whitelists de bots/tokens externos), así como algunos detalles de transparencia y robustez en operativa interna.

Recomendación:
No se detectan vulnerabilidades de seguridad críticas explotables, pero sí varias áreas de mejora en transparencia, robustez y gobernanza. Si el Ownership se mantiene seguro/no se pierde el control admin y los roles se ejercen de buena fe, el riesgo para usuarios es acotado. Se sugiere, sin embargo, mayor transparencia on-chain y documentación clara para la comunidad y holders.

Este análisis ha sido realizado con un enfoque crítico, imparcial y técnico, adecuado para contrastar con auditorías profesionales en Etherscan y siguiendo las mejores prácticas para contratos ERC20+con mecanismos de swap/tax.

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

A continuación, presento un **análisis exhaustivo y crítico** del contrato proporcionado, enfocado en vulnerabilidades reales, su impacto y mitigaciones recomendadas.

Análisis de Seguridad Completo

1. Uso innecesario de SafeMath en Solidity >=0.8.0

Severidad: Informational
Descripción:
El contrato utiliza la biblioteca `SafeMath`, que es redundante en Solidity 0.8.x y versiones superiores, ya que el compilador incorpora comprobaciones automáticas de *overflow/underflow* y lanza excepciones en caso de error aritmético.

Fragmento afectado:
```solidity
using SafeMath for uint256;
// Todas las operaciones .add(), .sub(), etc. en el contrato
```

Riesgo:
No representa un riesgo de seguridad directo, pero incrementa el tamaño de bytecode y puede confundir a los auditores o desarrolladores.

Mitigación:
Eliminar la utilización de `SafeMath` y sus llamadas asociadas para simplificar el código y reducir el *bytecode* desplegado.

---

2. Funciones de rescate de tokens y ETH sin validaciones

Severidad: Minor
Descripción:
Las funciones `removeStuckEth` y `removeStuckToken` permiten al propietario retirar ETH o cualquier token de cualquier dirección, sin restricciones adicionales ni validaciones sobre el destino, token específico o límites.

Fragmento afectado:
```solidity
function removeStuckEth(address _receiver) public onlyOwner {
payable(_receiver).transfer(address(this).balance);
}

function removeStuckToken(address _token, address _receiver, uint256 _amount)
public onlyOwner {
IERC20(_token).transfer(_receiver, _amount);
}
```
Riesgo:
- **removeStuckEth:** Es un patrón común para rescatar ETH atascado en el contrato, pero podría ser usado para retirar fondos recaudados, por ejemplo, en una venta. No hay validación del receptor o restricción temporal.
- **removeStuckToken:** Permite que el propietario retire *cualquier* token, incluso tokens legítimamente depositados por usuarios por error, lo que puede interpretarse como un mecanismo de rescate o posible abuso si no es comunicado adecuadamente.

Mitigación:
- Solo permitir rescate de tokens diferentes al nativo (`RXS`) para evitar retiro accidental/malicioso del suministro principal.
- Añadir eventos específicos y documentar públicamente el fin y límites del mecanismo.
- Considerar restricciones temporales (cooldowns) en el rescate para mayor transparencia.

---

3. Propietario puede activar la *whitelist* en cualquier dirección

Severidad: Medium
Descripción:
La función `setWhitelist` permite al propietario añadir o remover cualquier dirección de la *whitelist*. Cualquiera en la whitelist puede transferir tokens en cualquier momento, incluso cuando el “trading” está deshabilitado. Por lo tanto, el propietario podría incluirse a sí mismo y a otras cuentas para operar libremente antes del lanzamiento público (*trading* habilitado).

Fragmento afectado:
```solidity
function setWhitelist(address _user, bool _exmpt) external onlyOwner {
whitelist[_user] = _exmpt;
}
```
Riesgo:
- Se presta a prácticas injustas si el dueño del contrato agrega cuentas **bot** o

propias para vender/comprar antes de lanzar el token al gran público, dando una desventaja a los usuarios.
- Podría facilitar la manipulación de liquidez inicial, distribuciones privadas u otras acciones previas al lanzamiento.

Mitigación:
- Registrar (event/log) cada cambio de whitelist con detalle.
- Limitar la función `setWhitelist` al constructor o a una ventana temporal corta pre-lanzamiento, después de lo cual la whitelist no debería ser editable.
- Identificar públicamente las direcciones en whitelist en la documentación y/o auditar su uso tras el despliegue.

---

4. No se filtra la transferencia accidental del propio token RXS en removeStuckToken

Severidad: Minor
Descripción:
`removeStuckToken` permite transferirse a sí mismo tokens del contrato. Si por alguna razón, mediante una acción inesperada, el contrato posee tokens `RXS` bloqueados (por ejemplo, por error de usuario enviando tokens directamente), el owner puede autoretirarlos. Si bien no parece haber fuentes externas de RXS al contrato, es una puerta abierta a posibles malentendidos.

Mitigación:
- Añadir una restricción para bloquear el rescate del token propio:
```solidity
require(_token != address(this), "No rescue of RXS token");
```
---

5. *RenounceOwnership* deja el contrato permanentemente sin propietario

Severidad: Minor
Descripción:
Tras ejecutar `renounceOwnership`, no es posible ningún control de funciones `onlyOwner` (incluyendo rescates y gestión de whitelist).

Riesgo/Apreciación Adicional:
Es el comportamiento esperado para muchos estándares de *Ownable*, pero podría tener implicaciones si el rescate de ETH/tokens o la gestión whitelist es requerida después.

Mitigación:
Asegurarse de que la renuncia de propiedad solo se ejecute cuando no se requieran más acciones administrativas. Dejarlo claramente documentado.

---

6. No hay control de "blacklist" o limitación de transferencia (lo cual es positivo)

Nota:
Esta ausencia **no** es una vulnerabilidad (más bien es positivo para la transparencia). Solo se bloquean transferencias antes de habilitar el trading, salvo whitelist; después, cualquier cuenta puede operar.

---

7. Flujos de transferencias relevantes y *front-running* (ausencia de antiMEV/anti-bot):

Severidad: Informational
Descripción:
No están implementadas limitaciones para operar ante MEV, bots o snipers al iniciar el trading. El owner podría jugar con la whitelist, pero no hay otras restricciones.

Mitigación:
Implementar protección contra MEV/bots solo si es intencional en el diseño (por ejemplo, limitar tamaño de órdenes, retrasos, etc.), lo cual aquí no es el caso, y por tanto, no es una vulnerabilidad real.

CONCLUSIÓN FINAL

El contrato es **bastante simple y directo** como estándar ERC20 con un mecanismo de whitelist y protección previa al lanzamiento, sin tax/flex-fee u otros mecanismos potencialmente críticos.
No se observan las vulnerabilidades graves más habituales en tokens (como *reentrancy*, *allowance double-spend*, *backdoors ocultos*, *mint/inflación*, cambio arbitrario de balances, o manipulaciones críticas), ni mecanismos de fee que puedan ser subidos a valores confiscatorios.

Resumido:
- **No hay vulnerabilidades de gravedad crítica o mayor.**
- Existen puntos menores a mejorar para transparencia/opinabilidad y reducir superficie de abuso.
- El contrato es seguro según estándares de la versión del compilador y las prácticas modernas en tokens simples.

RESUMEN DE HALLAZGOS

#	Vulnerabilidad	Severidad	Mitigación clave
1	Uso innecesario de SafeMath (>=0.8.x)	Informational	Eliminar SafeMath
2	Funciones de rescate sin validaciones	Minor	Bloquear rescate del token propio
3	Whitelist editable tras el lanzamiento	Medium	Limitar edición post-lanzamiento
4	removeStuckToken permite RXS	Minor	Bloquear token propio
5	Renuncia de ownership permanente	Minor	Documentar intención

Si deseas una comparación línea por línea o revisiones para forks posteriores, indícalo y se ajusta el análisis al contexto específico.

← Página Anterior Inicio