Contrato EthenaMinting.sol do protocolo da Ethena

Mythril não encontrou nada.
Slither encontrou esses resultados:

Categoria / Alerta	Detecção do Slither	Realmente problema?	Observação
Arbitrary from em `safeTransferFrom`	`_transferCollateral` e `_transferEthCollateral` usam `token.safeTransferFrom(benefactor, ...)`.	Potencial risco, mas não necessariamente bug.	Aqui o `benefactor` é passado como parâmetro e provavelmente o usuário aprova antes. O Slither marca sempre como risco, mas é comportamento esperado em dApps de custódia.
Envio de ETH para endereços arbitrários	Funções chamam `.call{value: amount}()`.	Potencial risco de reentrância.	Porém o contrato usa `ReentrancyGuard`, então não é vulnerabilidade real. Falso positivo parcial.
External calls dentro de loop	`_transferEthCollateral` faz `.call` em loop.	Risco real de DoS (se algum endereço falhar, pode travar o loop).	Aqui o alerta faz sentido, mesmo com `ReentrancyGuard`.

Uso de `block.timestamp`	`verifyOrder` compara `block.timestamp` > `order.expiry`.	Válido, mas comum.	É prática aceitável em sistemas de expiração. Só seria problema se dependesse de alta precisão.
Múltiplas versões de Solidity	Usa `0.8.20`, `^0.8.0` e `^0.8.1` em libs.	Falso positivo.	Isso vem do OpenZeppelin; não é risco real.
Versão do compilador com issues conhecidas	`0.8.20` listada com bugs no changelog da Solidity.	Falso positivo prático.	Slither sempre marca versões com bugs conhecidos, mas esses bugs são raros e não necessariamente exploráveis.
Low-level calls (`.call`, `delegatecall`)	Detectou em várias libs (`SafeERC20`, `Address`).	Falso positivo.	Essas chamadas são da própria OpenZeppelin, que já trata segurança.
Naming convention (`mixedCase`)	Variáveis/funções não seguem padrão.	Apenas estilo.	Não afeta segurança.
State vars não usados (`ROUTE_TYPE`, `EIP712_DOMAIN_TYPEHASH`)	Constantes nunca usadas.	Verdadeiro, mas apenas lixo de código.	Não traz risco, mas pode limpar.

Contrato Lido.sol

Mythril encontrou somente alertas de outros contratos que são importados para o contrato Lido.sol e não encontrou nenhum resultado do Lido.sol:

Slither encontrou esses resultados:

Alerta (detector)	Severidade típica	Falso positivo ?	Resumo rápido (contrato/linha)
Constant functions using assembly	Informational	Provável FP (benigno)	`StakeLimitUtils._constGasMin` é `view` mas usa `assembly` (otimização de gás; não implica mutação). `contracts/0.4.24/lib/StakeLimitUtils.sol#224-230`.
Constant functions using assembly	Informational	Provável FP (benigno)	`ECDSA.recover` usa `assembly` para extrair campos da assinatura; padrão em libs. `contracts/common/lib/ECDSA.sol#50-58`.
Constant functions using assembly	Informational	Provável FP (benigno)	`SignatureUtils.isValidSignature` com `assembly` em fu comum em checagens de assinatura. `contracts/common/lib/SignatureUtils.sol#29-57`.
Constant functions using assembly	Informational	Provável FP (benigno)	`SignatureUtils._hasCode` faz `extcodesize` via `assembly` se há código; padrão. `contracts/common/lib/SignatureUtils.sol#59-64`.
Uninitialized local variables	Medium	Precisa revisão	`Lido.deposit(...).[ilegível]` nunca inicializada. Se nun ruído; se lida antes de set — bug lógico. `contracts/0.4.24/Lido.sol#706`.
Uninitialized local variables	Medium	Provável FP/baixo impacto	`StakeLimitUtils.calculateCurrentStakeLimit(...).` `IncPerBlock` nunca inicializada; costuma ser só variável morta código). `contracts/0.4.24/lib/StakeLimitUtils.sol#9`
Uninitialized local variables	Medium	Precisa revisão	`Lido._handleOracleReport(...).reportContext` nunca está em fluxo crítico de oráculo → vale checar uso. `contracts/0.4.24/Lido.sol#1186`.
Unused return	Medium	Provável FP	`StETHPermit.eip712Domain()` ignora valor retornado de `IEIP712SETH.eip712Domain(...)`; frequentemente é um “pi reverter se falhar. `contracts/0.4.24/StETHPermit.sol#14 (chama #149)`.

Unused return	Medium	Provável não-FP (revisar)	`Lido._collectRewardsAndProcessWithdrawals(...)` ig de `ILidoExecutionLayerRewardsVault.withdrawReward` retorno sinaliza valor/sucesso, ignorar pode afetar a contabilidad recompensas. `contracts/0.4.24/Lido.sol#832-865 (ch`

Esses problemas são somente do `Lido.sol`?

Não. Dos **9** achados, **apenas 3** estão em `Lido.sol` (um “uninitialized” em `deposit`, um “uninitialized” em `_handleOracleReport` e um “unused return” em `_collectRewardsAndProcessWithdrawals`). Os demais vêm de **libs auxiliares**: `StakeLimitUtils.sol`, `ECDSA.sol`, `SignatureUtils.sol`, `StETHPermit.sol`.

Contrato DelegationManager.sol do protocolo EigenLayer

Mythril não encontrou nenhum resultado

Slither encontrou esses resultados:

Categoria/ Alerta	Detecção do slither	Realmente é um problema?	Observação
Variável de estado não inicializada	`DelegationManagerStorage._depositScalingFactor` nunca é inicializada e é usada em várias funções (ex.: `decreaseDelegatedShares`, `_delegate`, `_removeSharesAndQueueWithdrawal`, `_increaseDelegation`, `depositScalingFactor`, `getWithdrawableShares`, `convertToDepositShares`).	Potencialmente	Se o fator de escala ficar `0` (valor default), pode zerar conversões/cálculos. Verifique se há escrita segura antes do uso (p.ex., em `_increaseDelegation`) e considere `require(factor != 0)` onde necessário.
Retorno ignorado (add)	Em `_removeSharesAndQueueWithdrawal(...)`, o retorno de `_stakerQueuedWithdrawalRoots[staker].add(withdrawalRoot)` é ignorado.	Depende / Pode mascarar falhas	`EnumerableSet.add` retorna `false` se já existia. Ignorar o retorno pode esconder inconsistências lógicas. Boa prática: checar o `bool` e, se fizer sentido, `require(ok, "already queued")`.

Retorno ignorado (remove)	Em `_completeQueuedWithdrawal(...)`, o retorno de `_stakerQueuedWithdrawalRoots[withdrawal.staker].remove(withdrawalRoot)` é ignorado.	Potencialmente	Se `remove` falhar (elemento não existe) e mesmo assim `delete` em estruturas relacionadas ocorrer, pode haver divergência entre `set` e `mappings`. Sugestão: checar o retorno e tratar o caso `false`.
Diretivas de pragma diferentes	Projeto usa múltiplas versões/constraints de Solidity (ex.: `^0.8.0`, `^0.8.1`, `^0.8.2`, `^0.8.4`, `^0.8.8`, `^0.8.27`, `>=0.5.0`).	Em geral, não	Em monorepo grande é comum. Não é vulnerabilidade direta, mas padronizar (ou compilar tudo com `solc = "0.8.27"` no Foundry) reduz ruído e possíveis diferenças sutis de compilador.
Operação custosa em loop (1)	Em `_completeQueuedWithdrawal(...)`, há `delete _queuedWithdrawals[withdrawalRoot]` dentro de laço chamado por `completeQueuedWithdrawals(...)`.	Potencial DoS por gás	Laços que deletam estruturas grandes podem estourar gás em lotes grandes controlados pelo usuário. Mitigue limitando tamanho do lote, usando processamento paginado ou verificando o

			consumo de gás.
Operação custosa em loop (2)	Em `_completeQueuedWithdrawal(...)`, há `delete pendingWithdrawals[withdrawalRoot]` dentro do mesmo laço.	Potencial DoS por gás	Mesmo raciocínio do item anterior. Avalie dividir a operação em etapas (ex.: uma transação por retirada) ou impor limites de comprimento.

Contrato CToken.sol do protocolo da JustLend

Mythril encontrou somente alertas de outros contratos que são importados para o contrato CToken.sol e não encontrou nenhum resultado do CToken.sol:

Alerta (SWC)	Severidade (Mythril)	Falso positivo?	Resumo rápido do problema detectado	Somente em `CToken.sol`?
Dependence on predictable environment variable (SWC-120)	Low	Tende a FP (contextual)	Uso de `block.number` em `accrueInterest()` para decidir se acumula juros dentro do mesmo bloco. Isso **não é** fonte de aleatoriedade; é um padrão do Compound/JustLend para calcular `blockDelta` e evitar dupla contabilização no mesmo bloco. Seguro se não houver outra lógica sensível dependente de miner timing. Evidência em `CToken.sol:395`.	**Não.** O mesmo alerta aparece na trilha passando por `CarefulMath.sol` (linhas 31 e 53 nos trechos do relatório), pois o fluxo de controle alcança utilitários matemáticos após a checagem de `block.number`.
Multiple Calls in a Single Transaction (SWC-113)	Low	Geralmente FP (design)	`borrowRatePerBlock()` e `supplyRatePerBlock()` fazem múltiplas chamadas externas (ex.: `getCashPrior()` e `interestRateModel.get*Rate(...)`) no mesmo tx. O Mythril avisa que um call prévio pode falhar e impedir os demais, mas neste desenho as chamadas são a **contratos de confiança** (modelo de juros/underlying), e a falha já reverteria a transação. Revisar apenas se algum callee pode ser **substituído por endereço não confiável**. Evidências em `CToken.sol:241` e `CToken.sol:249`.	**Sim** (neste relatório). Os achados listados estão nas funções do `CToken`.

São somente do `CToken.sol`?

- **Não.** Embora o núcleo esteja em `CToken.sol` (especialmente `accrueInterest`, `borrowRatePerBlock`, `supplyRatePerBlock`), o Mythril também mostrou o **SWC-120** em `CarefulMath.sol` por causa do caminho de execução que parte da checagem de `block.number` e segue para as rotinas matemáticas.

Slither encontrou esses resultados:

Alerta (categoria)	Severidade (aprox.)	Falso positivo?	Resumo rápido	Onde aparece
ERC20 incorreto (EIP20NonStandardInterface)	Baixa / Info	Sim (intencional)	Interface “non-standard” (transfer/transferFrom sem retornar `bool`); é um padrão deliberado para tokens antigos.	Em `EIP20NonStandardInterface.sol`, **não** em `CToken.sol`.
Dangerous strict equalities	Baixa	Geralmente sim	Igualdades estritas em `require/assert` (ex.: comparar código de erro ou `a==0`). Em padrões do Compound isso é comum e seguro quando os valores são internos/enum.	Diversas em `CToken.sol` e libs (ex.: `accrueInterest`, `mintFresh`, `repayBorrowFresh`, `CarefulMath`, `Exponential`).
Reentrancy (cross-function / state escrito após chamadas externas)	Média (contexto-dependente)	Parcial	Escrituras em estado após chamadas a `comptroller` / `cTokenCollateral`. Em Compound, esses contratos são “de confiança”; ainda assim revisar ordem *checks-effects-interactions* e se há risco de o `Comptroller` poder ser trocado por malicioso.	Vários fluxos de `CToken.sol`: `borrowFresh`, `mintFresh`, `redeemFresh`, `repayBorrowFresh`, `liquidateBorrowFresh`, `seizeInternal`, etc.
Variáveis locais não inicializadas	Baixa	Provável	Estruturas locais `vars` / `actualAddAmount` são declaradas sem valor “default”, mas os campos são atribuídos antes do uso. O Slither marca por precaução.	Em `CToken.sol` (ex.: `mintFresh`, `redeemFresh`, `repayBorrowFresh`, `_addReservesFresh`).
Sombreamento de variável local	Baixa	Não	`fraction` em `Exponential` sombreia função homônima — polui leitura, mas não é bug de segurança.	Em `Exponential.sol`.

Falta de eventos para mudanças críticas (initialize)	Baixa	Não (melhoria)	Recomenda emitir eventos ao setar `initialExchangeRateMantissa` e `reserveFactorMantissa` no `initialize`.	Em `CToken.sol`.
Falta de validação de endereço zero	Média	Não	`pendingAdmin` e `reserveAdmin` podem ser configurados como `address(0)`. Ideal validar `!= 0`.	Em `CToken.sol` (`_setPendingAdmin`, `_setReserveAdmin`).
Reentrancy (eventos/logs após chamadas externas; variantes)	Média (contexto-dependente)	Parcial	Versões “2/3” do mesmo tema: eventos/leitura após chamadas a `comptroller` / outros cTokens. Exige revisão do fluxo, mas normalmente é aceito no desenho Compound.	Múltiplas rotas de `CToken.sol`.
Código morto / funções nunca usadas	Info	Frequentemente sim	Em bases/contratos delegáveis, funções “não usadas” podem ser chamadas por herdeiros/implementações (ex.: delegator/delegatee).	Em `CToken.sol` e `Exponential.sol` (diversas).
Versão do Solidity desatualizada (^0.5.12) com bugs conhecidos	Alta (manutenibilidade/risco de toolchain)	Não	Pragma 0.5.12 lista issues conhecidos; recomendável migrar (≥0.8.x) quando possível.	Em vários arquivos, incluindo `CToken.sol`.
Conformidade de nomes (naming)	Info	Não (estilo)	Funções não `mixedCase` e constantes fora de `UPPER_CASE`.	Em `CToken.sol`, `CTokenInterfaces.sol`, `Exponential.sol`.
Funções não implementadas (contrato base/abstrato)	Info	Parcial (intencional)	`CToken` deixa ganchos abstratos (`getCashPrior`, `doTransferIn/out`) para implementações específicas.	Em `CToken.sol`.
Public que poderia ser external / data location calldata	Baixa (gás/estilo)	Não	Sugerir `external` e `calldata` para economia de gás/claridade.	Em `CToken.sol` e `CTokenInterfaces.sol`.

Alertas que são só do `CToken.sol`:

- Dangerous strict equalities (diversas funções: `accrueInterest`, `mintFresh`, `repayBorrowFresh`, `borrowBalanceStored`, `transfer`, etc.)
- Reentrancy (funções: `borrowFresh`, `mintFresh`, `redeemFresh`, `repayBorrowFresh`, `seizeInternal`, `liquidateBorrowFresh`, etc.)
- Variáveis locais não inicializadas (`vars` e `actualAddAmount` em várias funções do CToken)
- Falta de eventos em `initialize`
- Falta de validação de endereço zero (`_setPendingAdmin`, `_setReserveAdmin`)
- Dead code (várias funções internas nunca chamadas: `mintFresh`, `borrowFresh`, `repayBorrowFresh`, `redeemFresh`, etc.)
- Naming (funções como `_setPendingAdmin`, `_acceptAdmin`, `_setReserveFactor`, etc.)
- Funções não implementadas (`getCashPrior`, `doTransferIn`, `doTransferOut`)
- Funções que poderiam ser `external` (`initialize`)
- Versão do Solidity (`^0.5.12` usada no `CToken.sol`)

Alertas que vêm de outros contratos auxiliares (não apenas `CToken.sol`):

- `EIP20NonStandardInterface`: interface ERC20 não padrão em `EIP20NonStandardInterface.sol`
- `CarefulMath`: igualdade estrita em divisões/multiplicações (ex.: `b == 0`, `a == 0`)
- `Exponential`: `mulExp`, `divScalarByExpTruncate`, etc., com `assert`s e *shadowing* de variável `fraction`
- `ErrorReporter` e `ComptrollerErrorReporter`: funções `fail`, `failOpaque` nunca usadas
- `CTokenInterfaces.sol`: constantes/nomeação fora de convenção, funções de interface não em *mixedCase*

Resumindo: nem todos os alertas são do `CToken.sol`, vários vêm de arquivos auxiliares (`EIP20NonStandardInterface.sol`, `Exponential.sol`, `CarefulMath.sol`, `ErrorReporter.sol`, `CTokenInterfaces.sol`).
Mas os alertas mais críticos (reentrancy, zero-check, initialize sem eventos, variáveis não inicializadas) aparecem especificamente em `CToken.sol`.

Contrato PendleMarketV3.sol do protocolo da Pendle

Mythril encontrou esses resultados:

Alerta (SWC)	Severidade	Falso positivo?	Resumo rápido
Requirement violation (SWC-123) em `fallback`	Média	Provável FP	O Mythril simulou uma chamada para um "fallback" e viu um `revert` em chamada aninhada. O contrato não declara `fallback`, então isso tende a ser artefato do analisador/dispatcher e não uma falha real do `PendleMarketV3`.
Dependência de variável de ambiente previsível (SWC-116)	Baixa	Benigno	Uso de `block.timestamp` para decisões de fluxo (ex.: `mint(...)` e rotas internas). No `PendleMarketV3` ele é usado para "timestamp" observações/estado de mercado, não como fonte de aleatoriedade; é um padrão aceitável em AMMs/índices de preço.
Acesso a estado após chamada externa (SWC-107) — leituras/escritas após `external call`	Baixa	Tende a FP (mitigado)	O relatório marcou múltiplos PCs em uma função "desconhecida" (`@0x37d45e3a`). No `PendleMarketV3`, chamadas externas (ex.: `safeTransfer`) ocorrem e em seguida o contrato grava estado — mas as funções externas são protegidas por `nonReentrant` e usam `SafeERC20`, reduzindo o risco prático. Mantém-se uma boa prática, mas o alerta aqui é mais "sinal de cuidado" do que bug explorável.
Integer arithmetic bugs (SWC-101) em `name()` / `symbol()`	Alta	FP	O Mythril acusou underflow em `name()` / `symbol()` (selectors padrão ERC-20). Em Solidity ≥0.8 aritmética tem checagem por padrão e essas funções não fazem contas; isso normalmente é um falso positivo em análises simbólicas.

Isso é só do `PendleMarketV3.sol`?

- **Não exclusivamente.**
    - O alerta de `fallback` não aponta para uma função definida no arquivo; é efeito da simulação do Mythril sobre o dispatcher ("MAIN").

- Os de `name()` / `symbol()` (SWC-101) vêm da interface ERC-20 herdada (ex.: `PendleERC20`), não de lógica própria do `PendleMarketV3`.

- Os de `block.timestamp` (SWC-116) ocorrem em `mint(...)` (que está neste contrato) e também em rotas internas/auxiliares indicadas por seletores anônimos; no `PendleMarketV3`, `timestamp` é usado para observações/âncoras de taxa (comportamento esperado para oráculos/AMMs).

- Os de **acesso a estado após chamada externa** (SWC-107) aparecem várias vezes em uma função identificada só pelo seletor; no arquivo, os pontos típicos seriam transferências via `SafeERC20` seguidos de `_writeState`. Com `nonReentrant` isso tende a ser ruído (a ferramenta não "enxerga" o guard).

Slither encontrou esses resultados:

Categoria/alerta	Detecção do slither	Realmente é um problema?	Observação
Arbitrary `from` em `transferFrom`	`TokenHelper._transferIn/_transferFrom` usam `safeTransferFrom(from, ...)` com `from` arbitrário	Depende do contexto (geralmente OK)	Não é vulnerabilidade por si só: exige aprovação prévia do `from`. Verifique se quem chama garante que `from` seja esperado e se não há bypass de autorização. Boa prática: validar remetente esperado quando aplicável.
Sombreamento de variável de estado	`PendleMarketV3.SY` "sombra" `PendleGauge.SY`	Normal/Benigno	O aviso aponta nomes iguais em contratos distintos. Se não há herança direta que cause colisão de storage, é apenas ruído de nomenclatura. Mantenha consistência para evitar confusão.

“Dangerous strict equalities”	Comparações exatas em `RewardManagerAbstract` (ex.: `tokens.length == 0`, `userIndex == index`)	—	`index == 0`)
Retorno ignorado	`PendleGauge._redeemExternalReward()` ignora retorno de `IStandardizedYield(SY).claimRewards(...)`	Provável benigno	Muitos `claimRewards` têm efeitos via estado/eventos; retornar valores é opcional. Ignorar retorno é aceitável se o efeito colateral é o objetivo. Confirme a assinatura de `claimRewards` na interface.
Chamadas externas dentro de loop (`balanceOf`)	`TokenHelper._selfBalance` chama `IERC20(token).balanceOf(this)` em caminhos que podem ser iterados	Potencial (gás/DoS)	Risco principal é custo de gás/performance; não é bug de segurança direto. Se loops puderem ficar grandes, pode haver risco de DoS por out-of-gas. Otimize agregações/limite o tamanho do loop.
Chamadas externas dentro de loop (envio de ETH)	`_transferOut` usa `to.call{value: amount}()` num loop (redeem de rewards)	Potencial (reentrância/DoS)	Envio de ETH em loop pode permitir griefing (falha interrompe iteração) ou reentrância se o estado não estiver protegido. Garanta padrão **checks-effects-interactions**, possíveis proteções (reentrancy guard) e tratamento de falhas (p.ex. acumular e sacar individualmente).
Complexidade ciclomática alta	`LogExpMath._ln` com CC=13	Não	Sinaliza complexidade/testabilidade; não indica vulnerabilidade. Garanta boa cobertura de testes e revisões.
Versão do Solidity com “known issues”	`^0.8.17` listado com avisos de bugs conhecidos	Em geral Não (by design)	É um alerta genérico do Slither. Use uma versão do `solc` estável e alinhada ao projeto (com otimizador). Muitos projetos embora utilizem 0.8.x têm problemas conhecidos no seu escopo.

Contrato Sky.sol do protocolo da Sky

Mythril encontrou esses resultados:

Vulnerabilidade detectada	Severidade	Observação
Uso de `block.timestamp` para controle de fluxo (SWC-116)	Low	Também considerado falso positivo nesse contexto

Slither achou esses resultados:

Achado Slither	Tipo / Severidade	É falso positivo?	Observação
`block.timestamp` em `permit`	Uso de variável previsível	Sim	Apenas verifica expiração de assinatura (EIP-2612), não gera risco real.
Uso de `assembly` em `_isValidSignature`	Assembly inline	Sim	Necessário para decompor a assinatura; padrão seguro.
Low-level call em `_isValidSignature`	`staticcall`	Sim	Necessário para ERC1271; ferramenta alerta por precaução.
Versão Solidity `^0.8.21`	Alertas de bug conhecidos	Parcial	Apenas indica que a versão tem bugs conhecidos; não é vulnerabilidade no contrato.
Naming conventions (`_DOMAIN_SEPARATOR`, `DOMAIN_SEPARATOR()`)	Estilo / convenção	Sim	Apenas aviso de nomenclatura, não impacta segurança.

Contrato AuctionManager.sol do protocolo da Ether.fi

Mythril não encontrou nenhum resultado.

Slither encontrou esses resultados:

Alerta (Slither)	Severidade	Falso positivo?	Resumo rápido
Functions that send Ether to arbitrary destinations (em `transferAccumulatedRevenue`)	Informativo/Low	Depende	Usa `call{value:…}` para enviar ETH ao `membershipManagerContractAddress`. Se esse endereço for estritamente controlado por admin e confiável, tende a ser aceitável; se puder ser influenciado por usuários, há risco (DoS/reentrância).
Low-level calls (em `processAuctionFeeTransfer`, `transferAccumulatedRevenue`)	Informativo/Low	Depende	Uso de `address.call{value:…}()` em vez de primitivas seguras. Não é bug por si só, mas exige checagem de retorno e padrão CEI/`nonReentrant`.
Reentrancy vulnerabilities – `createBid(uint256,uint256)`	Medium	Depende	Dentro de um loop há chamada externa a `nodeOperatorManager.fetchNextKeyIndex(msg.sender)` **antes** de gravar estado. Se `nodeOperatorManager` for 100% confiável e não-chamável de volta, é ruído; caso contrário, há vetor real de reentrância. Ideal: mover writes antes da call, usar guardas.
External calls inside a loop – `createBid`	Informativo/Medium	Não	Chamada externa em cada iteração pode causar DoS por gas/rollback em lote. Melhor acumular e chamar fora do loop ou limitar o tamanho.
Reentrancy vulnerabilities – `_cancelBid(uint256)`	Medium	Não	Envia ETH ao `msg.sender` via `call` e só depois emite evento (e possivelmente outras ações). Mesmo com CEI parcial, a call externa em função pública pode permitir reentrância se invariantes não estiverem travados. Recomenda-se `nonReentrant`/CEI estrito.
External calls inside a loop – `_cancelBid`	Informativo/Medium	Não	Reembolso dentro de loop: mesma preocupação de DoS se algum recebedor revertir; melhor usar pull model / processar em lotes pequenos.

Missing events – `setStakingManagerContractAddress`	Informativo/Low	Não	Mudança sensível de configuração sem evento. Melhora auditabilidade emitindo evento.
Missing events – `updateWhitelistMinBidAmount`	Informativo/Low	Não	Atualização de parâmetro sem evento; afeta transparência/on-chain monitoring.
Costly operations inside a loop – `_cancelBid` (`numberOfActiveBids--`)	Informativo/Low	Sim (prático)	O decremento em si não é “caro”; o problema real é a **call externa** no loop (já coberta). Esse alerta específico costuma ser ruído.
Conformance to naming conventions (parâmetros/variáveis em snake case, `DEPRECATED_*`)	Informativo/Style	Sim	Apenas estilo; não impacta segurança/funcionalidade.

Contrato MainnetController.sol do protocolo da Spark

Mythril não encontrou nenhum resultado.

Slither encontrou esses resultados:

Alerta	Onde	Classificação	Por quê	Ação sugerida
Ignora valor de retorno em várias chamadas via `proxy.doCall(...)` (`mintUSDS`, `burnUSDS`, `transferAsset`, `redeemERC4626`, etc.)	Exemplos: `mintUSDS` (linhas **235–244**), `burnUSDS` (**252–261**), `transferAsset` (**275–278**), `redeemERC4626` (**336–339**)	**Provável falso positivo** (condicional)	Se `IALMProxy.doCall` reverte em falha, descartar o retorno não cria comportamento silencioso — o fluxo já falha.	(a) Confirmar que `doCall` reverte em erro; (b) opcional: capturar e checar retorno para "acalmar" o Slither.
Ignora retorno em outras funções do controller (`cooldownAssetsSUSDe`, `redeemSuperstate`, `swapUSDSToDAI/USDS`, `transferTokenLayerZero`, `depositToFarm`, `withdrawFromFarm`, helpers de rate limit)	Amostra: `cooldownAssetsSUSDe` (**628–631**), `redeemSuperstate` (**709–718, 711**), `swapUSDSToDAI` (**733–736**), `transferTokenLayerZero` (**827, 832–836**), `depositToFarm` (**872–875**)	**Provável falso positivo** (condicional)	Mesmo raciocínio: se calls reverterem internamente, ignorar retorno não mascara erro.	Igual acima. Se alguma dependência **não** reverte, usar `require(success)` / decodificar retorno.
Low-level call em `_approve` (usa `address(proxy).call(...)`)	`_approve` (**904–920**)	**Geralmente FP/estilo**	O padrão é deliberado: tenta `approve(0)`/re-approve e faz `require` no final; o risco principal (silenciar falhas) está mitigado.	Manter como está ou trocar por interface forte se viável.

“Should inherit from IEthenaMinterLike”	Sinalização de que `MainnetController` deveria herdar a interface	**Falso positivo / opinião de design**	O controller apenas encaminha chamadas (via proxy) para o minter; herdar a interface não é requisito funcional/segurança.	Nenhuma ação necessária. Documentar o motivo do design.
Naming (mixedCase)	Inclui `IATokenWithPool.POOL()` em `MainnetController.sol` (linha **32**)	**Ruído/estilo**	Conformidade de nomes não implica bug; muitas entradas são de libs.	Opcional: padronizar nomes se quiser relatório “limpo”.
Pragmas diferentes/“versões com issues conhecidas”	Reporta que `^0.8.21` é usado no controller (linha **2**)	**Ruído informativo**	É um heads-up genérico do Slither; não aponta falha concreta no controller	

Contrato UniswapV3Pool.sol do protocolo da Uniswap

Mythril não encontrou nenhum resultado.

Slither encontrou esses resultados:

Categoria/alerta	Detecção do slither	Realmente é um problema?	Observação
Incorrect exponentiation (FullMath)	Uso de ^ em vez de ** em FullMath.mulDiv (linha: inv = (3 * denominator) ^ 2)	Provável falso positivo / esperado	O operador ^ (XOR) é parte da implementação de seed para o inverso modular no FullMath; é esperado/aceito no contexto e o código da v3 é amplamente auditado e testado.
Divide-before-multiply	Várias ocorrências em FullMath, Oracle, Tick, TickMath	Não (benigno/esperado)	Padrão comum em aritmética de ponto fixo para evitar overflow e ajustar precisão; implementações da Uniswap v3 usam isso de forma intencional.
Dangerous strict equalities	Requires/igualdades rígidas em initialize() e checks em swap()	Não (invariante intencional)	Checks como `slot0.sqrtPriceX96 == 0` e comparações de preço no loop de swap são invariantes de protocolo e fazem parte da lógica de segurança do pool.
Reentrancy (v1/v2/v3)	collectProtocol, swap, flash, collect, mint com chamadas externas e gravações/emit após	Improvável (mitigado)	O pool usa `lock`/`slot0.unlocked` (no reentrancy guard) e checks-effects-interactions; chamadas externas são controladas. Risco residual é baixo; avaliar tokens não

			conformes caso de uso específico.
Uninitialized local variables	step, balance0Before, balance1Before, flippedLower/Upper, variáveis em swap()	Não (sem leitura antes de atribuir)	Em Solidity variáveis locais são zero-inicializadas; no código em questão são atribuídas antes do uso relevante. Alerta conservador do Slither.
Unused return value	observe() ignora retorno de observations.observe(...)	Não (intencional)	A função visa atualizar cumulativos/estado a partir do storage; ignorar o retorno é parte da API e não afeta a correção do estado do pool.
Block timestamp (comparisons)	Marca comparações em initialize() e swap() como “timestamp for comparisons”	Falso positivo	As comparações listadas não usam timestamp diretamente; o projeto usa um observation oracle. O rótulo do detector não se aplica às linhas citadas.
Assembly usage	Inline assembly em FullMath, TickMath, UnsafeMath	Não (intencional)	Uso necessário para desempenho/precisão em aritmética 512-bit. Códigos amplamente auditados na v3.
Different pragma directives	Múltiplas pragmas (>=0.5.0, =0.7.6 etc.)	Não (compatibilidade)	Projeto v3 fixa contratos core em =0.7.6 e bibliotecas/interfaces com faixas mais amplas; coerente com o repositório oficial.
Cyclomatic complexity	swap() com CC alta; SwapMath/TickMath também	Não (sinal de complexidade)	Indica necessidade de maior cobertura de testes e revisão, mas não é vulnerabilidade por si só.

Incorrect Solidity version	solc 0.7.6 “outdated” e listas de bugs conhecidas	Não (by design do v3)	Uniswap v3 é concebido para 0.7.6; atualizações exigiriam re-engenharia. Atenção apenas ao compilar com toolchains modernas.
Low-level calls	staticcall/call para balanceOf/transfer em balance0(), balance1(), TransferHelper.safeTransfer	Não (padrão compatibilidade ERC20)	TransferHelper checa sucesso e tamanho do retorno para lidar com tokens não conformes; padrão amplamente usado pelo ecossistema.
Too many digits (literals)	Constantes grandes em BitMath/FixedPoint*/TickMath	Não (estilo)	São constantes matemáticas (Q96/Q128, máscaras) necessárias para a aritmética; sem impacto de segurança.
Contract code size > 24,576 bytes (EIP-170)	Aviso de tamanho no UniswapV3Pool.sol	Falso positivo prático	Pools v3 são implantados em mainnet; builds de produção usam otimizador/links de biblioteca e ficam abaixo do limite. O aviso surge em compilações simples sem otimização.