SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared For: Hook

Prepared By: Sherlock

Lead Security Experts: WatchPug, 0xRajeev

Security Researchers: Secureum Bootcamp participants

Introduction

“Hook is an oracle-free, on-chain option protocol for non-fungible tokens (NFTs). Unlike many popular approaches to NFT DeFi, Hook does not sacrifice the non-fungible nature of NFTs by requiring that they are converted into fungible tokens. NFTs deposited into the Hook protocol only contain unique artistic images and do not contain, reference, represent the price, rate or level of any security, commodity, or financial instrument.”

This report is a CASEfile for Hook Protocol that was prepared by Sherlock Watsons WatchPug and 0xRajeev, with assistance from Secureum participants, in the context of Secureum CASE (Collaborative Assessment & Security Evaluation) during 17-30 June, 2022.

Scope

Branch: Master (https://github.com/hookart/protocol)
Tag: https://github.com/hookart/protocol/tree/v0.0.3
Contracts:

- /src/HookProtocol.sol
- /src/HookBeaconProxy.sol
- /src/HookUpgradeableBeacon.sol
- /src/HookERC721VaultFactory.sol
- /src/HookCoveredCallFactory.sol
- /src/HookERC721MultiVaultImplV1.sol
- /src/HookERC721VaultImplV1.sol
- /src/HookCoveredCallImplV1.sol
- /src/lib/BeaconSalts.sol
- /src/lib/Entitlements.sol
- /src/lib/HookStrings.sol
- /src/lib/Signatures.sol
- /src/mixin/EIP712.sol
- /src/mixin/HookInstrumentERC721.sol
- /src/mixin/PermissionConstants.sol
- /src/interfaces/*.sol

Final Commit Hash (post fix review):
c49350a455760ac9cab2fc942dd2c2f62355e0d1

Protocol Info

Language: Solidity

Blockchain: Ethereum
L2s: None

Tokens used: Ether, WETH

Findings

Each issue has an assigned severity:

- Informational issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgement as to whether to address such issues.
- Low issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Total Issues

Informational	Low	Medium	High
10	9	7	5

Issue H-01

Call and Vault implementations can be arbitrarily changed by anyone.

Summary
Hook protocol uses a beacon proxy pattern for Call and Vault implementations where the proxy gets the implementation address for each call from an upgradeable beacon. However, the beacon initialization function can be called by anyone repeatedly to re-initialize with a malicious beacon and return arbitrary implementations for Calls and Vaults.

Severity
High

Vulnerability Detail
Hook protocol uses a modified version of OpenZeppelin's beacon proxy pattern for Call and Vault implementations where the proxy gets the implementation address for each call from an upgradeable beacon. Instead of using a constructor to set the beacon address, the proxy uses an initialization function initializeBeacon to make this pattern usable with Create2 proxy deployments from a factory.

However, the beacon initialization function has public visibility without an initializer modifier protection (allowing initialization only once) which means that it can be called by anyone repeatedly to re-initialize with a malicious beacon and return arbitrary implementations for Calls and Vaults.

Impact
initializeBeacon for Call and Vaults can be called by anyone repeatedly to re-initialize with a malicious beacon and return arbitrary implementations at any point subverting the entire protocol.

Code Snippet
initializeBeacon makeCallInstrument makeMultiVault makeSoloVault OpenZeppelin BeaconProxy

Tool used
Manual Review

Recommendation
Add support for using the initializer modifier on initializeBeacon so it can be called only once during proxy creation for Calls and Vaults.

Hook Comment
Fixes H-01 issue uncovered in the audit by making the HookBeaconProxy contract initializable. This change simply utilizes the OpenZeppelin Initializable library already in use in the protocol.

SHERLOCK

https://github.com/hookart/protocol/pull/47

Follow up to remove Initializable import at L7:

https://github.com/hookart/protocol/pull/69

Sherlock Comment
Verified fix.

Issue H-02

Attacker can front-run mintWithVault with a replayed signature to steal option NFT.

Summary

An attacker can front-run a writer's call to mintWithVault with another transaction replaying the signature and other parameters as-is. Unlike mintWithERC721, mintWithVault does not perform access control to check if the option is being indeed minted by the token owner or operator. When it calls _mintOptionWithVault, it approves the msg.sender i.e. attacker for the new option NFT being minted which gives the attacker the right to transfer that to itself and pocket any spread later when the option gets settled in the future.

Severity

High

Vulnerability Detail

The call to mintWithVault fails the assumption made in the callee _mintOptionWithVault as described in its code comment: "*If msg.sender and tokenOwner are different accounts, approve the msg.sender to transfer the option NFT as it already had the right to transfer the underlying NFT*" because the msg.sender is never checked against token owner or operator in mintWithVault as is done in mintWithERC721. This allows an attacker to front-run a writer's call to mintWithVault with another transaction replaying the signature and other parameters as-is.

The writer's original mintWithVault will fail because an active entitlement already exists for that asset from the attacker's transaction. The front-running attacker becomes the option holder and can pocket the spread when the option gets settled in future.

Exploit scenario:

1. Alice deposits a BAYC NFT, currently valued by the market at 100 ETH, into a Hook vault.
2. Alice or her trusted relayer executes a mintWithVault corresponding to (1) with an entitlement signature and a strike price of 120 ETH.
3. Mallory observes Alice's mintWithVault transaction in the mempool and front-runs it with the same parameters which is successful but Alice's fails because hasActiveEntitlement reverts given the already active entitlement from Mallory's transaction.
4. Mallory transfers the option NFT to herself and becomes the holder. There is nothing Alice can do to force reclaim the option NFT from Mallory.
5. Bidding happens until the option expires and is settled for e.g. say 130 ETH. The spread of (130 - 120) = 10 ETH is sent to Mallory. Alice gets the strike price of 120 ETH. The highest bidder gets the BAYC NFT for 130 ETH.

6. Alice effectively suffers a loss of at least 10 ETH (the spread) which could have been hers if the option NFT had been minted to her as she expected. She could have pocketed the spread of 10 ETH or sold the option NFT earlier off-protocol for perhaps an even higher price.

Impact
The NFT owner and option writer suffers a loss of value at least equivalent to the spread which was captured by the front-running attacker.

Code Snippet
mintWithVault mintWithErc721 _mintOptionWithVault

Tool used
Manual Review

Recommendation
Enforce access control in mintWithVault to ensure that the caller is the token owner (i.e. writer) or operator (as is done in mintWithERC721).

Hook Comment
Add erc721-style approve and getApproved. Require that msg.sender is beneficial owner or approved for imposeEntitlement, mintWithVault, mintWithEntitledVault

https://github.com/hookart/protocol/pull/46

Sherlock Comment
Rajeev: This adds a new access control approval mechanism on top of ERC721 which is confusing. Seems to work but wonder if there are interaction risks from inconsistent ERC721 owner/spender/operator roles.

WatchPug: We can not use ERC721's approval mechanism directly as the NFT is now held by the vault and using ERC721's approval means the operator can transfer the NFT out.

Maybe consider changing the name to `getApprovedOperator`, `_approveOperator` to avoid misunderstanding?

Hook Comment
I've updated the naming in this PR. The goal here is to ensure that approvals CANNOT be set on tokens in the vault as an invariant.

https://github.com/hookart/protocol/pull/71

Sherlock Comment
Verified fix.

Issue H-03

Attacker can front-run mintWithVault to modify strike price.

Summary

An attacker can front-run the writer's call to mintWithVault with another transaction replaying the signature and other parameters as-is with a very low strike price (way below the market price) for the underlying NFT asset. Unlike mintWithERC721, mintWithVault does not perform access control to check if the option is being indeed minted by the token owner or operator. This gives the attacker the right to transfer the option NFT holder to itself, while the writer's mintWithVault will fail because an active entitlement already exists for that asset from the attacker's transaction. Assuming the auction bids come in at the market price for that NFT, the attacker who has become the option holder will pocket the significant spread while the option writer will get the low strike price and effectively lose considerable funds related to the market value of the underlying NFT asset.

Severity

High

Vulnerability Detail

The call to mintWithVault fails the assumption made in the callee _mintOptionWithVault as described in its code comment: "*If msg.sender and tokenOwner are different accounts, approve the msg.sender to transfer the option NFT as it already had the right to transfer the underlying NFT*" because the msg.sender is never checked against token owner or operator as is done in mintWithERC721.

Exploit scenario:

1. Alice deposits a BAYC NFT, currently valued by the market at 100 ETH, into a Hook vault.
2. Alice executes a mintWithVault corresponding to (1) a strike price of 120 ETH.
3. Mallory observes Alice's mintWithVault transaction in the mempool and front-runs it, replaying the same parameters except replacing the strike price with 1 ETH. Mallory's mintWithVault is successful but Alice's fails because hasActiveEntitlement reverts given the already active entitlement from Mallory's transaction.
4. Mallory transfers the option NFT to herself and becomes the holder. There is nothing Alice can do to force reclaim the option NFT from Mallory.
5. Bidders can come in at any price above the option's strike price of 1 ETH. Let's assume that they come in at a market discounted price of 90 ETH.
6. The option expires and is settled. The spread of (90 - 1) == 89 ETH is sent to Mallory. Alice only gets the strike price of 1 ETH. The highest bidder gets the BAYC NFT at a discounted price of 90 ETH.

7. Alice effectively suffers a loss of at least 99 ETH, assuming market price of the BAYC NFT is still 100 ETH, and perhaps more considering the market cost of the option NFT.

Impact
The NFT owner and option writer suffers a significant loss of value having been forced to sell the NFT at a heavily discounted price with the spread going to the front-running attacker.

Code Snippet
mintWithVault mintWithErc721 _mintOptionWithVault

Tool used
Manual Review

Recommendation
Enforce access control in mintWithVault to ensure that the caller is the token owner (i.e. writer) or operator (as is done in mintWithERC721). Additionally, include strike price in the entitlement signature to enforce checks on it so that it cannot be modified by a man-in-the-middle attack.

Hook Comment
Add erc721-style approve and getApproved. Require that msg.sender is beneficial owner or approved for imposeEntitlement, mintWithVault, mintWithEntitledVault

https://github.com/hookart/protocol/pull/46

Sherlock Comment
Rajeev: This adds a new access control approval mechanism on top of ERC721 which is confusing. Seems to work but wonder if there are interaction risks from inconsistent ERC721 owner/spender/operator roles.

WatchPug: We can not use ERC721's approval mechanism directly as the NFT is now held by the vault and using ERC721's approval means the operator can transfer the NFT out.

Maybe consider changing the name to `getApprovedOperator`, `_approveOperator` to avoid misunderstanding?

Hook Comment
I've updated the naming in this PR. The goal here is to ensure that approvals CANNOT be set on tokens in the vault as an invariant.

https://github.com/hookart/protocol/pull/71

Sherlock Comment
Verified fix.

SHERLOCK

Issue H-04

Anyone can call mintWithEntitledVault with an arbitrary strike price.

Summary

An attacker can call mintWithEntitledVault on an entitled contract for a vaulted asset with an expected expiration time but with any arbitrary strike price e.g. way below the market price for the underlying NFT asset. Unlike mintWithERC721, mintWithEntitledVault does not perform access control to check if the option is being indeed minted by the token owner or operator. This gives the attacker the right to transfer the option NFT holder to itself. Assuming the auction bids come in at the market price for that NFT, the attacker who has become the option holder will pocket the significant spread while the option writer will get the low strike price and effectively lose considerable funds related to the market value of the underlying NFT asset.

Severity

High

Vulnerability Detail

The call to mintWithEntitledVault fails the assumption made in the callee _mintOptionWithVault as described in its code comment: "*If msg.sender and tokenOwner are different accounts, approve the msg.sender to transfer the option NFT as it already had the right to transfer the underlying NFT*" because the msg.sender is never checked against token owner or operator as is done in mintWithERC721.

Exploit scenario:

1. Alice executes a mintWithERC721 for a BAYC NFT, currently valued by the market at 100 ETH, at a strike price of 120 ETH. Let's call this Option1.
2. Mallory observes Alice's transaction and executes a mintWithEntitledVault with the same parameters except replacing the strike price with 1 ETH. Mallory's mintWithEntitledVault is successful and creates a new call option, say Option2, covered by the same underlying BAYC NFT belonging to Alice.
3. Mallory transfers the Option2 NFT to herself and becomes the holder.
4. Bidders can come in at any price above the Option2 strike price of 1 ETH. Let's assume that they come in at a market discounted price of 90 ETH. Mallory herself could be the highest bidder.
5. Options 1 & 2 expire at the same time. Let's assume that Option1 had no bidders given the higher strike price of 120 but Option2 had the highest bidder at 90 ETH. At settlement of Option2, the spread of (90 - 1) == 89 ETH is sent to Mallory. Alice only gets the strike price of 1 ETH.
6. The highest bidder gets the BAYC NFT at a discounted price of 90 ETH but only if they manage to withdraw the asset for Option2 before Alice does for Option1 because both options are covered by the same underlying asset.

7. If Mallory is the highest bidder of Option2 then she knows that she has to front-run Alice's withdrawal to get the NFT asset. If someone else is the highest bidder of Option2 and doesn't realize this aspect, Alice may withdraw her asset given her failed auction for Option1 and the highest bidder for Option2 will not get the asset despite winning the bid.
8. If Alice loses the asset to the highest bidder of Option2, then she effectively suffers a loss of at least 99 ETH, assuming market price of the BAYC NFT is still 100 ETH.

Impact
The NFT owner and option writer suffers a significant loss of value having been forced to sell the NFT at a heavily discounted price with the spread going to the attacker who created a duplicate call option covered by the same NFT asset.

Code Snippet
mintWithVault mintWithErc721 _mintOptionWithVault

Tool used
Manual Review

Recommendation
Enforce access control in mintWithEntitledVault to ensure that the caller is the token owner (i.e. writer) or operator (as is done in mintWithERC721).

Hook Comment
Require that msg.sender of mintWithEntitledVault is the beneficial owner of the underlying asset

https://github.com/hookart/protocol/pull/45

Sherlock Comment
Same changes as for H-02 & H-03 except it's not only the owner but also the approved address who can call this.

Issue H-05

Multiple call options can be created that are covered by the same underlying NFT asset.

Summary

An attacker can call mintWithEntitledVault on an entitled contract for a vaulted asset multiple times to mint an arbitrary number of call option NFTs covered by the same underlying NFT asset. Upon expiration and settlement, only the first highest bidder of all of such option auctions will be able to withdraw their winning NFT while others would lose their winning bid without owning the won NFT.

Severity

High

Vulnerability Detail

There is no accounting in the protocol to enforce that an underlying NFT can cover only one call option instrument. This may be abused by anyone, including a malicious writer, to generate an arbitrary number of call options using mintWithEntitledVault all of which are covered by the same vaulted NFT asset. Upon option expiry and settlement, the strike price of all the options will go to the NFT owner and option writer while all the spreads will go to the respective option NFT holders. However, only one of the auction winners i.e. whoever is the first to successfully execute withdrawalAsset will be able to withdraw the covered NFT asset while all the other option auction winners will end up losing their winning bids without getting the underlying asset.

Exploit scenario:

1. Mallory executes a mintWithERC721 for a BAYC NFT, currently valued by the market at 100 ETH, at a strike price of 90 ETH. Let's call this Option1.
2. Malicious Mallory then creates another option Option2 with mintWithEntitledVault on the same NFT from (1) again with a strike price of 90 ETH.
3. Let's assume that bidders come in for both options at the market price of 100 ETH with Alice winning Option1 and Bob winning Option2.
4. Upon expiry and settlement, the strike price of 90 ETH and spread of 10 ETH from both options are sent to Mallory who nets a total of 200 ETH.
5. When Alice and Bob both try to withdrawalAsset their BAYC NFT, only one of them succeeds, say Alice. Bob ends up losing his 100 ETH without getting the BAYC NFT.

Impact

Winners of all auctions except one lose their winning bids without getting the underlying NFT asset. The strike prices and spreads of multiple options benefit the option writer and holders.

Code Snippet
mintWithEntitledVault _mintOptionWithVault

Tool used
Manual Review

Recommendation
Enforce a 1:1 mapping between an option instrument and the underlying NFT asset. Consider adding a mapping(vaultAddress ⇒ mapping(assetId ⇒ CallOption)) to ensure one NFT can only have one call option active at a time.

Hook Comment
Change first option id to be 1 so we can assume 0 to be a “null” or nonexistent option. Added mapping from (IHookVault) address to a mapping of asset id to call option id.

https://github.com/hookart/protocol/pull/44

Sherlock Comment
Checks that while minting, if optionID != 0 then it is settled, thereby enforcing only one outstanding option on any assetId.

Issue M-01

Incorrect asset withdrawal address emitted.

Summary
The AssetWithdrawn event emitted in clearEntitlementAndDistribute uses an incorrect withdrawal address.

Severity
Medium

Vulnerability Detail
The AssetWithdrawn event in clearEntitlementAndDistribute is emitted when an asset is withdrawn from the vault and its second parameter indicates the address to which the asset is sent upon withdrawal.

clearEntitlementAndDistribute takes a receiver parameter which indicates the intended receiver of the asset to which safeTransferFrom sends the asset. However, the AssetWithdrawn event emitted uses msg.sender (operator) instead of the receiver address (beneficialOwner) for the 'to' address argument, when msg.sender need not be equal to the receiver address.

Impact
This may mislead protocol user interfaces and offchain monitoring systems to misinterpret the recipient (operator vs beneficial owner) of asset distribution to cause confusion, flagging of alerts or DoS.

Code Snippet
AssetWithdrawn clearEntitlementAndDistribute

Tool used
Manual Review

Recommendation
Change emit AssetWithdrawn(assetId, msg.sender, assets[assetId].beneficialOwner);
To emit AssetWithdrawn(assetId, receiver, assets[assetId].beneficialOwner);

Hook Comment
Use receiver instead of msg.sender in AssetWithdrawn event.

https://github.com/hookart/protocol/pull/50

Sherlock Comment
Ok.

Issue M-02

Incorrect asset owner address may be emitted.

Summary
The AssetReceived event emitted in onERC721Received may use an incorrect owner address if additional data is sent.

Severity
Medium

Vulnerability Detail
The AssetReceived event in onERC721Received is emitted when an asset is received in the vault and its first parameter indicates the address which will be the beneficial owner.

However, the event emission always uses the from parameter of onERC721Received as the beneficial owner. This may not be the case when additional data is sent with a beneficialOwner different from the from parameter for entitlement registration.

Impact
This may mislead protocol user interfaces and offchain monitoring systems to misinterpret the actual beneficial owner and cause confusion, flagging of alerts or DoS.

Code Snippet
HookERC721MultiVaultImplV1 .onERC721Received HookERC721VaultImplV1 .onERC721Received

Tool used
Manual Review

Recommendation
Use a different owner variable in the event emit which is set to from address or the decoded beneficial owner (when additional data is sent) accordingly.

Hook Comment
Use getBeneficialOwner to specify the operator

https://github.com/hookart/protocol/pull/54

Sherlock Comment
The fix is applied to the incorrect emit argument. It should be applied to the first `from` argument and not to the second one.

SHERLOCK

Hook Comment
https://github.com/hookart/protocol/pull/73

Sherlock Comment
Verified fix.

SHERLOCK

Issue M-03

CallOption NFT holder may lose funds on a custodial marketplace.

Summary
If the CallOption NFT holder lists the NFT on a custodial marketplace then they may lose the spread sent on settlement.

Severity
Medium

Vulnerability Detail

If the CallOption NFT holder sends it to a custodial marketplace for listing where it's not sold until settlement then the spread amount will be transferred to the address of the custodial marketplace which may not be claimable by the CallOption NFT holder.

Impact
CallOption NFT holder loses access to the spread amount they are entitled to on expiry and settlement of option.

Code Snippet
settleOption

Tool used
Manual Review

Recommendation
Consider changing from the push model to a pull model for transferring the spread and strike price amounts ie. allowing the owner of the CallOption NFT to redeem the spread and then burn the nft.

Hook Comment
Added mapping to track eth amounts to be claimed by option owner and claimOptionProceeds external function to access the claimable eth

https://github.com/hookart/protocol/pull/53

Sherlock Comment
Ok. Claimable by owner if settled by someone else.

claimOptionProceeds() has a reentrancy. Add nonReentrant modifier and also follow CEI pattern by caching optionClaims[optionId] in a local variable and deleting it before making the _safeTransferETHWithFallback call.

Hook Comment
Thanks Rajeev, this is resolved here:
https://github.com/hookart/protocol/pull/72?no-redirect=1

Sherlock Comment
Verified fix.

Issue M-04

Option writer may lose airdropped tokens.

Summary

A NFT owner who creates an option in a Hook multivault will lose any NFT's/ERC20's airdropped to the multivault under certain scenarios.

Severity

Medium

Vulnerability Detail

Hook multivault implementation is not designed to handle NFT's/ERC20's airdropped to owners of the vaulted NFTs. Unlike the solo vault which has the execTransaction function to transfer out any airdropped assets, the multivault does not have support for ownership accounting and transferring of such airdropped assets because it can hold multiple tokens with different owners.

The current approach is to:

1. Configure multivaults to reject airdrops which leads to loss of airdropped tokens
2. Rely on airdrops which can be claimed via flashloans instead of being pushed/distributed to owners
3. Document the lack of support for pushed/distributed airdrops for multivault NFTs and expect to add support in a future contract upgrade

Scenario of Concern:

1. Alice owns a BAYC NFT which she puts into a Hook multivault and writes an option with a strike price of 100 ETH and an expiry of 30 days
2. Alice then sells the CallOption NFT to Mallory for 1 ETH
3. BAYC announces an airdrop of MAYC NFT to all BAYC holders in 2 days. MAYC is expected to be valued at a floor price of 10 ETH on launch
4. Alice's BAYC is stuck in Hook multivault
5. If Alice is unaware of the airdrop, it is transferred to the Hook multivault which either reverts if airdrops are configured disabled or accepts it but is locked in the contract until it can be upgraded to allow transfer to the actual NFT owners
6. If Alice is aware of the airdrop, she can attempt to reclaim the NFT out of the vault but may have to convince Mallory to sell the CallOption NFT back to her. Mallory may charge a premium if she is aware of the MAYC airdrop and its expected floor price

Impact

In any scenario of the current approach, Alice will effectively lose all/some of the funds due to the airdrop because of writing a Hook option on a multivault instead of using a solo vault or not using Hook at all.

Code Snippet
HookERC721MultiVaultImplV1 .onERC721Received

Tool used
Manual Review

Recommendation
Consider adding airdrop support in multivaults with ownership accounting and transferring of such assets. At a minimum, document this missing feature and make users aware of potential loss of airdropped assets under the scenarios of concern.

Hook Comment
"NFT holders often get airdrops from the projects. These airdrops are typically implemented either as a claim from another contract that checks that the msg.sender is the ownerOf the nft OR by directly sending an erc-20 to all of the ownerOf using something like disperse.app. In the second case, a user cannot flashloan their asset, and if the asset is vaulted they will not have access to the erc-20s. The execTransaction method would allow them to call the transferTo method on an airdropped token."

"I think we'd intend to disable the airdrops using the config when deploying a multivault to stop this issue. storing the correct beneficial owner for an asset in a multivault context was not able to be determined. The reason for keeping the codepath is that if a major project were to announce this sort of airdrop, we'd at least have a route to accept it and then potentially perform a contract upgrade to make the assets recoverable (if there was sufficient economic value)"

"The other issue is that there are some erc721 contracts in existence that limit gas when calling erc721recieved on the receiver."

"Also note: we believe that the majority of economically value able airdrops today would be claimed via the flashloan vs. True "airdrops" to the wallet, if nothing else to save fees for the sender when distributing"

"It's much more common to pull (ie ens and apes, or using an autoglyph to mint a meebit). Projects like Loot saw a lot of the push direction."

Hook Comment
Change multivault implementation to block airdrops by default.

https://github.com/hookart/protocol/pull/52

Sherlock Comment
Ok.

Issue M-05

NFTs may get locked in MultiVaults leading to loss of user tokens.

Summary

NFTs with tokenId > type(uint32).max will cause corresponding NFTs with colliding tokenIds to get locked in MultiVaults with their original beneficial owners unable to withdraw them anymore.

Severity

Medium

Vulnerability Detail

Hook project documentation says: "*The multi vault does not support any collection with token ids that overflow uint32. Uint32 is used to reduce the number of storage slots that must be allocated to add an asset to the vault; very few NFT projects based on ERC-721s actually utilize tokenIds outside this range*" with the assumption that authorized ALLOWLISTER_ROLE will never list a MultiVault for an NFT project with tokenIDs in that range.

However, instead of checking received NFTs with tokenId > type(uint32).max and reverting, the current implementation forces a downcast in onERC721Received via uint32(tokenId). If tokenId > type(uint32).max then the forced downcasting will make this tokenId collide with the corresponding NFT whose tokenId is within the type(uint32).max range, if it were to have been vaulted earlier.

ENS uses hash(name) as its uint64 tokenId which is greater than type(uint32).max. Another NFT project Artblocks uses tokenId = (projectNumber * 1000000) + mintNumber, which could be greater than type(uint32).max. NFT projects may also get creative after the initial mint to issue new ones with tokenIds outside the initially planned type(uint32).max.

Example scenario:

1. ALLOWLISTER_ROLE lists BAYC NFT series in a MultiVault assuming that its tokenId will never exceed type(uint32).max
2. BAYC approves a change to mint many more BAYC NFTs with randomized tokenIds that now can exceed type(uint32).max
3. Alice creates an option for her BAYC with tokenId 0
4. Mallory creates an option for her BAYC with tokenId 0×000000001 which when forcibly downcasted to uint32 becomes 0×0 and collides with Alice's BAYC
5. Mallory is made the beneficial owner for Alice's NFT and depending on the option status of Alice there could be other unexpected behavior

6. Alice loses access to her BAYC which gets locked in the MultiVault

Impact
The worst case scenario is that NFTs with colliding tokenIds may get locked in MultiVaults leading to loss of such user tokens. There could be other unexpected behavior depending on different scenarios.

Code Snippet
_setBeneficialOwner _registerEntitlement

Tool used
Manual Review

Recommendation
Use SafeCast.toUint32(tokenId) instead of uint32(tokenId).

Hook Comment
"*The intention would be to only deploy these vaults on contracts where the tokenId is limited, but it is also true that sometimes (i.e. artblocks) its hard to know the eventual limit. So we'd be open to using SafeCast.toUint32 or similar checks.*"

Hook Comment
Revert if token id is greater than uint32's max value.

https://github.com/hookart/protocol/pull/48

Sherlock Comment
Ok.

Issue M-06

Protocol admin is a single point of failure.

Summary

Protocol admin can arbitrarily and unilaterally list vaults/options, update vault/call factory addresses, configure the market/collection parameters and pause/unpause the markets/protocol. This presents a critical single point of failure.

Severity

Medium

Vulnerability Detail

Hook protocol uses OpenZeppelin's AccessControl.sol to implement role-based access control (RBAC) with seven different roles: ADMIN_ROLE, ALLOWLISTER_ROLE, PAUSER_ROLE, VAULT_UPGRADER, CALL_UPGRADER, MARKET_CONF and COLLECTION_CONF.

Protocol documentation comments that: "*This contract does not implement any specific timelocks or other safety measures. The roles are granted with the principal of least privilege. As the protocol matures, these additional measures can be layered by granting these roles to other contracts. In the extreme, the upgrade and other roles can be burned, which would effectively make the protocol static and non-upgradeable.*"

However, the initial configuration assigns the admin address to all the seven roles thereby defeating the motivation behind separation of roles. If a protocol admin becomes malicious or compromised, the entire protocol is immediately at risk for all existing/future markets/participants. While the updation of many critical parameters emit events, that only lets market participants react after the fact because these changes are not time-delayed.

Impact

A scenario that affects future markets/participants is the updation of vault/call factory addresses to malicious ones. Neither of these actions emit events and will go unnoticed by option writers/holders/bidders.

A DoS scenario, for existing markets/participants, is the protocol admin, in the MARKET_CONF role, setting the minimumOptionDuration, newSettlementStartOffset or minBidIncrementBips to unreasonable values which prevents the expected functioning of covered call option markets. The admin in the COLLECTION_CONF role can disable airdrops, flash loans and execTransaction capability of solo vaults which may cause DoS for certain assets and actions.

Code Snippet

HookProtocol.constructor setVaultFactory setCoveredCallFactory onlyMarketController

setMinOptionDuration setSettlementAuctionStartOffset setBidIncrement Security Pitfalls & Best Practices 201

Tool used
Manual Review

Recommendation
While it has been alluded in the comments that the current RBAC roles and configuration is part of the guarded launch strategy, the protocol admin should nevertheless be a reasonable threshold multisig (e.g. 4/7, 5/9) with diverse owners and (cold/hardware) wallets until it is backed by token-holder governance, i.e., it should certainly never be an EOA. The highest possible operational security measures should be taken for all multisig owners and wallets.

The assignment of roles to and management by different addresses should be enforced at the earliest in the spirit of the Principle of Least Privilege and Principle of Separation of Privilege.

Hook Comment
Use the separate address in the protocol constructor for ALLOWLISTER, PAUSER, VAULT_UPGRADER, CALL_UPGRADER, MARKET_CONF, and COLLECTION_CONF instead of a single admin address.

https://github.com/hookart/protocol/pull/55

Sherlock Comment
Ok. To enforce that addresses are really different, we can add a != check in the constructor on the seven input parameters.

Issue M-07

Reclaiming asset will fail when the option writer is also the highest bidder.

Summary

Hook allows for the option writer to reclaim i.e. withdraw the underlying asset from an active option if certain conditions are met. However, this will fail if the option writer is also the highest bidder.

Severity

Medium

Vulnerability Detail

Protocol documentation states that: "*If the writer wishes to withdraw the underlying asset from an active option, they first must obtain the option instrument NFT and transfer it to the writer account. Then, they may call the reclaim function, which will release the entitlement on the vault, return any active bids to the bidder, and burn the option NFT. They may then either withdrawal the asset from the vault or mint a new option with it.*"

However, the implementation does not account for the scenario where the option writer is also the highest bidder at the time of reclaiming. _returnBidToPreviousBidder and settleOption handle this scenario by subtracting the call strike amount during _safeTransferETHWithFallback to the option writer. This is because the option writer is allowed to bid by only paying the difference between their bid and strike price.

Therefore, if call.writer is also the call.highBidder at the time of attempting to reclaim then _safeTransferETHWithFallback(call.highBidder, call.bid) will revert because the option contract will only have ETH amount equal to the spread but not the strike part of call.bid.

Impact

Option writer is unable to reclaim the underlying asset in the scenario when the writer is also the highest bidder.

Code Snippet

bid reclaimAsset _returnBidToPreviousBidder settleOption Reclaim active auction

Tool used

Manual Review

Recommendation

Check if the highest bidder is the option writer during reclaiming and pay back only the spread amount in that scenario instead of call.bid which includes both the strike and spread parts.

Hook Comment
Return the correct amount of eth (high bid - strike) to high bidder during reclaim when high bidder is call writer.

https://github.com/hookart/protocol/pull/64

Sherlock Comment
Ok.

Issue L-01

Missing zero-address validation.

Summary
Lack of zero-address validation on address parameters will lead to reverts and may force contract redeployments in the protocol.

Severity
Low

Vulnerability Detail
Many functions externally accessible by users and owners lack zero-address validation on address parameters.

Impact
This may lead to transaction reverts, waste gas, require resubmission of transactions and may even force contract redeployments in certain cases within the protocol.

Code Snippet
HookProtocol.constructor HookERC721VaultFactory.constructor HookCoveredCallFactory.constructor

Tool used
Manual Review

Recommendation
Add explicit zero-address validation on input parameters of address type.

Hook Comment
Add zero-address validation, and validation to ensure that code is deployed at an address such that the incorrect addresses are less likely to be set (including the zero address)

https://github.com/hookart/protocol/pull/59

Sherlock Comment
Ok.

Issue L-02

Time-delayed change of critical parameters is absent.

Summary
Change of critical parameters should be enforced only after a time delay.

Severity
Low

Vulnerability Detail

When critical parameters of systems need to be changed, it is required to broadcast the change via event emission and recommended to enforce the changes after a time-delay. This is to allow system users to be aware of such critical changes and give them an opportunity to exit or adjust their engagement with the system accordingly.

None of the access-controlled privileged (e.g. hasRole(), adminOnly, onlyOwner, onlyMarketController) functions that create/change critical protocol addresses/parameters have a timelock for a time-delayed change to alert: (1) users and give them a chance to engage/exit protocol if they are not agreeable to the changes (2) team in case of compromised owner(s) and give them a chance to perform incident response.

Impact
Users may be surprised when critical market configuration parameters or factory/vault addresses are changed/created without notice. Furthermore, it can erode users' trust since they can't be sure the protocol rules won't be changed later on.

Compromised owner keys may be used to change protocol addresses/parameters to benefit attackers. Without a time-delay, authorised owners have no time for any planned incident response.

Code Snippet
setMinOptionDuration setBidIncrement setSettlementAuctionStartOffset upgradeTo setCoveredCallFactory setVaultFactory setCollectionConfig

Tool used
Manual Review

Recommendation
All access-controlled functions that set/change critical addresses/parameters in these contracts should apply a timelock. Consider evaluating the use of OpenZeppelin's TimelockController.

Hook Comment

Won’t fix.

The protocol is designed to allow the roles which control each of these parameters to be transferred, including transferred to other contracts such as the OpenZeppelin timelock contracts. As a part of progressively upgrading the governance of the system, the Hook team plans first to hold these critical roles using MultiSigs and then, as the system becomes more proven, upgrade the holders of these roles to be timelocks gated by multisigs. It is important to note that because vaulting assets for periods of time is a core feature of the system, and because these periods of time could be quite long, timelocks only provide additional transparency to users in cases that their options happen to expire before the timelock ending. In the case of a protocol upgrade, the timelocks also provide attackers with more time to carry out an exploit when attacking the system, and allow them time to see the fix deployed for a period of time when an exploit is possible in the wild.

Sherlock Comment
Ok.

Issue L-03

Missing sanity/threshold checks.

Summary
Sanity/threshold checks (depending on their types) on input parameters are missing.

Severity
Low

Vulnerability Detail

Functions that update critical protocol parameters are missing sanity/threshold validation on some parameters.

Examples include:

- Missing isContract sanity check on implementation address in HookUpgradeableBeacon constructor as is done in the setter _setImplementation.
- Missing sanity check to ensure this upgraderRole is either CALL_UPGRADER or VAULT_UPGRADER in HookUpgradeableBeacon constructor because those are the only roles authorized to upgrade the Call and Vault implementations.
- Missing sanity/threshold checks on newMinDuration, newBidIncrement and newSettlementStartOffset values. For e.g. there should be an equivalent check on newMinDuration to ensure it is greater than settlementAuctionStartOffset similar to the check in setSettlementAuctionStartOffset.

Impact
Parameters may accidentally be initialized with invalid values in the context of the protocol or in relation to other parameters. This may lead to incorrect accounting and protocol malfunction.

Code Snippet
HookUpgradeableBeacon.constructor _setImplementation
HookUpgradeableBeacon.constructor setMinOptionDuration setBidIncrement
setSettlementAuctionStartOffset

Tool used
Manual Review

Recommendation
Add explicit sanity/threshold validation to check that input parameters fall within range or have values as expected in the context of the protocol or in relation to other parameters.

Hook Comment
Added more sanity checks on the input parameters, including on call option parameters that interact with each other.

https://github.com/hookart/protocol/pull/60

Sherlock Comment
Ok.

Issue L-04

Missing equivalence checks in setters.

Summary
Setter functions are missing checks to validate if the new value being set is the same as the current value already set in the contract.

Severity
Low

Vulnerability Detail
Setter functions are missing checks to validate if the new value being set is the same as the current value already set in the contract. Such checks will showcase mismatches between on-chain and off-chain states.

Impact
This may hinder detecting discrepancies between on-chain and off-chain states leading to flawed assumptions of on-chain state and protocol behavior.

Code Snippet
pause unpause setMarketPaused

Tool used
Manual Review

Recommendation
Add equivalence checks to validate (and revert) if the new value being set is the same as the current value already set in the contract.

Hook Comment
Add checks to verify that a paused contract cannot be paused again.

https://github.com/hookart/protocol/pull/58

Sherlock Comment
Ok.

Issue L-05

Missing validation and events in init functions.

Summary
None of the init functions perform zero-address validation on address parameters or emit events.

Severity
Low

Vulnerability Detail
None of the init functions perform zero-address validation on address parameters or emit init-specific events. They all however have the initializer modifier (from Initializable) so that they can be called only once.

Impact
Accidental use of incorrect parameters will require contract redeployment. Off-chain monitoring of calls to these critical functions is not possible.

Code Snippet
HookCoveredCallImplV1.initialize HookERC721MultiVaultImplV1.initialize HookERC721VaultImplV1.initialize

Tool used
Manual Review

Recommendation
It is recommended to perform validation of input parameters and emit events.

Hook Comment
Added zero address checking and event emissions. NOTE: the vault will emit two addresses, not just one.

https://github.com/hookart/protocol/pull/66

Sherlock Comment
Ok.

Issue L-06

Missing nonReentrant modifier may lead to unexpected behavior.

Summary
withdrawalAsset and all three mint functions are missing the nonReentrant modifier.

Severity
Low

Vulnerability Detail
Function withdrawalAsset transfers the vault asset out of the contract, can be called by any beneficialOwner and allows the recipient contract of safeTransferFrom to trigger a callback at onERC721Received. However, this is missing the nonReentrant modifier unlike other similar functions in the codebase.

All three mint functions call _safeMint later in _mintOptionWithVault which could re-enter but are missing the nonReentrant modifier.

Impact
An exploitable reentrancy is likely not possible given the use of the CEI pattern.

Code Snippet
withdrawalAsset mintWithVault mintWithEntitledVault mintWithErc721

Tool used
Manual Review

Recommendation
Add the nonReentrant modifier for defensive programming (despite the use of the CEI pattern) and to be consistent with other similar functions in the codebase.

Hook Comment
Added the missing nonReentrant modifies and updated the relevant tests

https://github.com/hookart/protocol/pull/61

Sherlock Comment
Ok.

Issue L-07

Unsafe default value for minBidIncrementBips.

Summary
minBidIncrementBips is set to a default value of 0 assuming that the market controller will call setBidIncrement to reset this to a reasonable value, failing which an attacker can back-run a winning bid to claim the underlying asset without outbidding.

Severity
Low

Vulnerability Detail
Any new bid is expected to outbid the previous highest bidder by a value of minBidIncrementBips. However, minBidIncrementBips is set to a default value of 0 in initialize assuming that the market controller will subsequently call setBidIncrement to reset this to a reasonable value. If this does not happen and minBidIncrementBips remains at 0 then there can be an arbitrary number of bids made just above the strike price.

Impact
The last bidder before call expiration and settlement can claim the underlying asset without outbidding the previous highest bidder i.e. at the same price as the previous bid amount.

Code Snippet
bid initialize setBidIncrement

Tool used
Manual Review

Recommendation
Change the initialization of minBidIncrementBips in initialize to a reasonable non-zero value.

Hook Comment
Set minBidIncrementBips to 50bps by default on initialization.

https://github.com/hookart/protocol/pull/49

Sherlock Comment
Ok.

Issue L-08

uint32 timestamp may not be future proof enough.

Summary
Asset.expiry which is represented using uint32 is future proof only for 83.6 years.

Severity
Low

Vulnerability Detail
Asset.expiry which is represented using uint32 will overflow after 83.6 years.

Impact
block.timestamp value will be larger than what can be represented by a uint32 and so asset expiry cannot be accurately compared to block.timestamp leading to protocol malfunction.

Code Snippet
Asset.expiry CallOption

Tool used
Manual Review

Recommendation
Asset expiry can be changed to uint64 to make it more future proof while still benefiting from storage packing.

Hook Comment
Will note in relevant documentation.

Sherlock Comment
Ok.

Issue L-09

Unnecessary operator privilege requirements.

Summary
mintWithErc721 requires isApprovedForAll operator privileges by token owner for msg.sender and call option contract, which is unnecessarily excessive.

Severity
Low

Vulnerability Detail
mintWithErc721 requires token owner to have approved msg.sender and call option contract on all the tokens for the specific NFT address. This is unnecessarily excessive because the call option is only being minted on one tokenID. Also, the following safeTransferFrom checks ownership/approval for the specific tokenId being minted for the call option making the earlier check on msg.sender redundant.

Impact
A malicious msg.sender relayer or a compromised call option contract may steal all the tokens belonging to the option writer besides the one being used for the call option.

Code Snippet
isApprovedForAll safeTransferFrom

Tool used
Manual Review

Recommendation
Evaluate replacing the operator approval with a token approval for the token being minted into a call option to make appropriate changes in approval checks and address used in safeTransferFrom. This is a best-practice for enforcing the Principle of Least Privilege.

Hook Comment
Check for both single and all token approval in mintWithErc721.

https://github.com/hookart/protocol/pull/57

Sherlock Comment
Ok.

Issue I-01

Redundant event emitted.

Summary
Hook protocol pause and unpause functions emit PausedUpdated which is redundant.

Severity
Informational

Vulnerability Detail
Hook protocol pause and unpause functions emit PausedUpdated which is redundant because similar events Paused and Unpaused are emitted by OpenZeppelin's functions called by them.

Impact
Redundant events waste gas and could be confusing.

Code Snippet
pause and unpause OpenZeppelin Pausable

Tool used
Manual Review

Recommendation
Consider removing the redundant event PausedUpdated

Hook Comment
Removed the hook protocol events to depend on the OZ events instead.

https://github.com/hookart/protocol/pull/63

Sherlock Comment
Ok.

Issue I-02

Misleading protocol terminology for call option.

Summary
Hook protocol claims to be an option protocol for NFTs similar to a European call option but it instead implements an English auction to enforce an obligation on the highest bidder to buy the NFT at the bid price.

Severity
Informational

Vulnerability Detail

Hook protocol claims to be an option protocol for NFTs similar to a European call option. However it implements an English auction to enforce an obligation on the highest bidder to buy the NFT at the bid price. The seller/writer specified strike price part of the highest bid is sent to the option writer while the spread i.e. (highest bid - strike price) is sent to the CallOption NFT holder. This deviates from the commonly understood aspect of option, i.e. the right to buy the covered asset at a specific price.

1. Alice executes mintWithErc721 for a BAYC NFT with a strikePrice of 50 ETH and the expiry time set to 10 days later for which she receives a CallOption NFT
2. Bob buys the CallOption NFT from Alice for 1 ETH because Bob likes the BAYC NFT and is willing to pay 50 ETH 10 days later to get it
3. Nine days later the auctions start and Bob places a bid at 50 ETH
4. At the very last block before the expiry time, someone else (even Alice) submits a bid at 50 ETH + 1 Wei and wins the auction to get the BAYC NFT
5. Alice nets a profit of about 1 ETH from the sale of the call option to Bob besides getting the strike price of 50 ETH from the highest bidder
6. Bob receives only 1 Wei of spread and loses 1 ETH paid for the CallOption NFT without being able to execute the right to buy the BAYC NFT as he had assumed holding the CallOption NFT would entitle.

Impact
Users of the Hook protocol who buy the CallOption NFTs assuming it gives them the right to buy the underlying asset at strike price on expiry will be surprised and may claim a loss of funds related to the buying of the CallOption NFT because it only entitles them to the spread and not the covered asset.

Code Snippet
Documentation reclaimAsset

Tool used

SHERLOCK

Manual Review

Recommendation

Consider allowing the CallOption NFT holder to reclaimAsset instead of only the writer. At a minimum, consider changing the terminology and documentation to notify users that holding the CallOption NFT only entitles them to the spread and not the option of buying the underlying NFT at the strike price on expiry.

Hook Comment

Won't fix.

Call options are not necessarily physically settled, and covered call options in particular are most often cash settled in other markets.

Sherlock Comment

Ok.

Issue I-03

Unused or inconsistently used named returns

Summary
Functions use a mix of explicit returns and named return variables.

Severity
Informational

Vulnerability Detail
Functions use a mix of explicit returns and named return variables. Some named returns are declared but unused with functions favoring explicit returns. This affects readability.

Impact
Reduced code comprehension and auditability.

Code Snippet
expiry holdsAsset

Tool used
Manual review

Recommendation
Consistently use explicit returns or named returns. Remove unused named returns. Favor explicit returns over implicit named returns.

Hook Comment
Utilize explicit returns instead of named returns.

https://github.com/hookart/protocol/pull/51

Sherlock Comment
Ok.

Issue I-04

Missing indexed event parameters

Summary
Events that do not use indexed parameters make it harder and inefficient for off-chain tools to analyse them. None of the event parameters are indexed.

Severity
Informational

Vulnerability Detail
Indexed parameters (“topics”) are searchable event parameters. They are stored separately from unindexed event parameters in an efficient manner to allow for faster access. This is useful for efficient off-chain analysis, but it is also more costly gas-wise.

Impact
Makes it harder and inefficient for off-chain tools to analyse them.

Code Snippet
IHookCoveredCall IHookERC721VaultFactory and others

Tool used
Manual review

Recommendation
Consider which event parameters could be particularly useful for off-chain tools and index those.

Hook Comment
Won’t fix.

Indexing events results in higher gas costs.

Sherlock Comment
Ok.

Issue I-05

Invalid optionID 0 is not handled.

Summary
Option identifiers start with 1 but the invalid identifier 0 is unhandled across functions.

Severity
Informational

Vulnerability Detail
OpenZeppelin Counter which is used for _optionIds is incremented before being used for newOptionId in _mintOptionWithVault. Therefore option identifiers start with 1. However, the invalid 0 option identifier is not checked across functions.

Impact
Accidentally using 0 for option identifier may return confusing 0 values for getters, cause unexpected reverts or behavior in other functions.

Code Snippet
_mintOptionWithVault biddingEnabled getExpiration getStrikePrice getAssetId getVaultAddress and others

Tool used
Manual review

Recommendation
Consider starting option Identifiers at 0 or checking for invalid 0 identifier.

Hook Comment
Fixed in the resolution of H-05, the 0 value now has a meaning (no option exists for a specific asset)

https://github.com/hookart/protocol/pull/44

Sherlock Comment
Ok.

Issue I-06

Push payment deviates from recommended best-practice.

Summary
Hook uses a push model for sending ETH to recipients which deviates from recommended best-practice.

Severity
Informational

Vulnerability Detail
During option bidding, settlement and reclaiming, the protocol sends back ETH to the appropriate recipients instead of expecting them to claim it afterwards. This push model has drawbacks such as having to assume malicious contracts and mitigating that risk.

The protocol recognizes such risks and implements mitigations such as transferring only 30K gas in the external call to reduce risk of reentrancies and falling back to WETH if the ETH transfer fails accidentally or maliciously (intentionally reverting to attempt a DoS during bidding or settlement).

Impact
Push model increases design complexity to mitigate risks from malicious recipients and may leave some latent risks due to assumptions on gas prices, reentrancy possibilities and DoS vectors.

Code Snippet
_safeTransferETHWithFallback _safeTransferETH _returnBidToPreviousBidder settleOption reclaimAsset ConsenSys Diligence Best-practice

Tool used
Manual review

Recommendation
Consider replacing the push model with a pull model.

Hook Comment
Partial fix - we now implement pull payments on settlement for the option NFT holder. Beyond this fix, pull payments are too confusing to users to be utilized in the protocol.

Sherlock Comment
Ok.

Issue I-07

Inconsistent pausing mechanism.

Summary
reclaimAsset and burnExpiredOption have the whenNotPaused modifier which is not consistent with the documentation and other functions.

Severity
Informational

Vulnerability Detail
Protocol documentation for Pausing says: "*The protocol can be paused in the event that an incident occurs. When the protocol is paused, no new options can be minted, no new assets can be deposited into any vault. However, in order to preserve the economic implications even in the event of a protocol pause, the settlement auctions may still occur. This is because options are time-sensitive; i.e. the value of the option is determined at a very specific time. If that time can be arbitrarily changed, it is difficult to assess the value of the option asset.*"

reclaimAsset and burnExpiredOption functions are related to the settlement auction (not related to minting or depositing) but have the whenNotPaused modifier which is not consistent with the documentation and functions bid and settleOption which do not have the whenNotPaused modifier.

Impact
Paused markets/protocol will prevent option writers from reclaiming assets and anyone from burning expired options. The latter may allow MARKET_CONF or PAUSER_ROLE to cause DoS to option writers by pausing the market.

Code Snippet
reclaimAsset burnExpiredOption Pausing Documentation

Tool used
Manual review

Recommendation
Evaluate removing whenNotPaused modifier on reclaimAsset and burnExpiredOption functions.

Hook Comment
Removed the whenNotPaused on the reclaimAsset and burnExpiredOption

https://github.com/hookart/protocol/pull/56

Sherlock Comment
burnExpiredOption still has this modifier:

https://github.com/hookart/protocol/blob/7f69cfa11c58c0c90727df3ef60ea3f3c0a44cc4/src/HookCoveredCallImplV1.sol#L658-L662

Issue I-08

Code, comments & documentation.

Summary

Discrepancies in/between or inaccuracies/deficiencies in code, comment and documentation can be misleading and could indicate the presence of inaccurate implementation or documentation.

Severity

Informational

Vulnerability Detail

1. Typo: Error string should say HookUpgradeableBeacon, not UpgradeableBeacon (copy-paste error from OZ)
2. Missing NatSpec for setCollectionConfig
3. Misleading error strings in pause and unpause functions says admin instead of PAUSER_ROLE
4. Incomplete NatSpec for HookCoveredCallImplV1.initialize is missing preApprovedMarketplace param
5. Typo: ERC712 should be ERC721
6. Typo: Error strings in mintWithEntitledVault function should say mintWithEntitledVault, not mintWithVault
7. Incorrect error string in _mintOptionWithVault assumes the default value of minimumOptionDuration == 1 day, but which can be changed by market controller
8. makeSoloVault is missing from IHookERC721VaultFactory
9. Typo: Error strings in _registerEntitlement function should say _registerEntitlement, not _verifyAndRegisterEntitlement

Impact

Reduced code comprehension, auditability and maintainability.

Code Snippet

UpgradeableBeacon setCollectionConfig pause HookCoveredCallImplV1.initialize ERC712 mintWithVault _mintOptionWithVault makeSoloVault _registerEntitlement

Tool used

Manual Review

Recommendation

SHERLOCK

Code, comments and documentation should all be complete, accurate and consistent before security review.

Hook Comment
Resolve the issues highlighted.

https://github.com/hookart/protocol/pull/62

Sherlock Comment
Ok.

Issue I-09

Code structure deviates from best-practice.

Summary
It is a best practice to order different contract constructs in a certain widely-used layout for better readability and auditability.

Severity
Informational

Vulnerability Detail
The best-practice layout for a contract should follow the following order: state variables, events, modifiers, constructor and functions.

Function ordering helps readers identify which functions they can call and find constructor and fallback functions easier. Functions should be grouped according to their visibility and ordered as: constructor, receive function (if exists), fallback function (if exists), external, public, internal, private.

Some constructs deviate from this recommended best-practice: Modifiers are in the middle of contracts. External/public functions are mixed with internal/private ones. Few events are declared in contracts while most others are in corresponding interfaces.

Impact
Reduced readability, auditability and maintainability.

Code Snippet
Modifiers Private and External Functions Events

Tool used
Manual review

Recommendation
Consider adopting recommended best-practice for code structure and layout.

Hook Comment
Won't fix.

Refactor creates too much post-audit complexity, too much risk of introducing bugs that may be missed.

Sherlock Comment
Ok.

Issue I-10

Potential gas optimizations

Summary
There are some opportunities to optimize gas usage.

Severity
Informational

Vulnerability Detail
There are categories of gas optimizations possible across contracts:

1. Caching of variables in memory instead of using storage variables
2. Using memory equivalents of storage variables
3. Use of immutables
4. Redundant calls
5. Checking for zero transfer amounts
6. Unnecessary initialization to default value of type

Impact
Increased gas usage.

Code Snippet
Examples

1. Caching: _nftContract allowedUnderlyingAddress call assets[assetId].beneficialOwner
2. Memory instead of storage: Use address(bp) instead of getCallInstrument[assetAddress]
3. Immutables: getWETHAddress _beacon _protocol _preApprovedMarketplace _hookProtocol _beacon _multiBeacon
4. Redundant call to setAddressForEipDomain in initialize and super.initialize: setAddressForEipDomain setAddressForEipDomain
5. unNormalizedHighBid will be zero for first bid: unNormalizedHighBid
6. Unnecessary initialization: _preApprovedMarketplace

Tool used
Manual review

Recommendation
Consider optimizing for gas where possible.

Hook Comment
Fixed some in the context of other bugs.

Fixed max contract size warnings (i.e. cannot deploy big bytecode).

https://github.com/hookart/protocol/pull/68

Sherlock Comment
Ok.