ENIGMA DARK

Securing the Shadows

Security Review

Juicebox Protocol v4

April, 2024

Contents

Summary

Enigma Dark
Enigma Dark is a web3 security firm leveraging the best talent in the space to secure all kinds of blockchain protocols and decentralized apps. Our team comprises experts who have honed their skills at some of the best auditing companies in the industry. With a proven track record as highly skilled white-hats, they bring a wealth of experience and a deep understanding of the technology and the ecosystem.

Learn more about us at enigmadark.com

Juicebox v4
Juicebox v4 is a protocol designed to manage token-backed programmable treasuries for individuals and projects.

Engagement Overview

Over the course of 4 weeks starting April 3rd 2024, the Enigma Dark team conducted a security review of the Juicebox v4 project. The review was performed by two Lead Security Researchers, vnmrtz.eth & 0xWeiss.

The following repositories were reviewed at the specified commits:

Repository	Commit
Bananapus/nana-core	`040d6ee4054e01fd4913f362d8c917c988803011`
Bananapus/nana-buyback-hook	`f4dc05a95237008e984ff554b09f909ec25a47b7`
Bananapus/nana-swap-terminal	`4d9c9730381a0919574518ebad4691e7b6755282`
mejango/bannyverse-core	`397d8bede346e9ff60047c6fc91ccc7bfc544620`

Bananapus/nana-721-hook	`b0414d6aafe0003deef69de3f798c251a92b5b1f`
mejango/croptop-core	`be8876d5b2851e2622f13ece4071bc36df6888fb`
rev-net/revnet-core	`ec8b7bb50f6b539e1d594b1092b78b13a517c7a3`
Bananapus/nana-project-handles	`616764a1dea9ae9741480a7b3731dbf7f00960d5`
Bananapus/nana-address-registry	`32f2229cce52b2b211a85f9f32a2283783cbfb72`
Bananapus/nana-ownable	`45140fabc0b335e061adcac47a75e793908e45c5`
Bananapus/nana-suckers	`ba0e4053c63288e59926a91ca80f840e24e30dcc`

Risk Classification

Severity	Description
Critical	Vulnerabilities that lead to a loss of a significant portion of funds of the system.
High	Exploitable, causing loss or manipulation of assets or data.
Medium	Risk of future exploits that may or may not impact the smart contract execution.
Low	Minor code errors that may or may not impact the smart contract execution.

Vulnerability Summary

Severity	Count	Fixed	Acknowledged
Critical	0	0	0
High	5	5	0
Medium	16	16	0
Low	8	7	1

Informational	6	4	2

Findings

Index	Issue Title	Status
H-01	Hook swap execution is completely exposed to MEV when no quote is provided	Fixed
H-02	Missing `transferFrom` on afterPayRecordedWith will make hook DOS for all ERC20 tokens	Fixed
H-03	`JBSwapTerminal` Incorrectly Uses `sqrtPriceX96` for Slippage Protection Calculation	Fixed
H-04	Anyone can change the outfits of another users banny	Fixed
H-05	A `worldId` that you don't own can be attached to your Banny or update someone elses	Fixed
M-01	Partial DOS on PERMIT2, permit flow	Fixed
M-02	Controller cannot call `setPrimaryTerminalOf`	Fixed
M-03	`setTerminalsOf` is able to remove the primary terminal from the `_terminalsOf` list, rendering `isTerminalOf` to return `address(0)`	Fixed
M-04	metadata wrong rounding direction on offset increment	Fixed
M-05	`createMetadata` can overflow offset variable by one extra item	Fixed
M-06	Price Feed staleness is not validated in `JBChainlinkV3PriceFeed`	Fixed
M-07	Usage of deprecated `answeredInRound` parameter.	Fixed
M-08	ROOT permission is DOS'd in the whole codebase	Fixed
M-09	Missing sequencer check in `JBChainlinkV3PriceFeed` when deploying to L2s	Fixed

M-10	Project's owner can set a project's pool to an arbitrary contract	Fixed
M-11	`uniswapV3SwapCallback` basic structure and checks differs from Uniswap v3 `SwapRouter` implementation	Fixed
M-12	Low Liquidity in Uniswap V3 Pool can lead to tokens being locked up in `JBSwapTerminal` contract	Fixed
M-13	Default outfits are not attached to Bannys with a smaller category than `_FACE_CATEGORY` on their Banny	Fixed
M-14	Broken URIs for `_NAKED_CATEGORY` and `_ONESIE_CATEGORY`	Fixed
M-15	Tiers with the flag `noNewTiersWithReserves` can still have reserve beneficiaries	Fixed
M-16	Unsafe casting allows to set `tiersToAdd` for the wrong category post	Fixed
L-01	`_addTerminalIfNeeded` is allowed to add a terminal if interface is not supported	Fixed
L-02	Reverse selector clashing can be encountered when setting the `currency`	Fixed
L-03	A pool related to a project can be changed to a pool with a different fee	Fixed
L-04	Consider extracting `amountToSwapWith > totalPaid` check out of the if clause	Fixed
L-05	Partial DOS on `PERMIT2`, `permit` flow	Fixed
L-06	If `OUT_IS_NATIVE_TOKEN` is set to true, swap terminal will increase unnecesarily the approval of WETH for the `nextTerminal`	Fixed
L-07	Use `safeTransfer` instead of transfer	Fixed
L-08	Sucker deployers can't be disallowed	Acknowledged
I-01	Missing parameter on natspec	Fixed

I-02	`_swap`, `zeroForOne` unnecessary calculation	Fixed
I-03	Missing indexing in some key events	Acknowledged
I-04	Quality assurance issues	Open
I-05	Quality assurance issues	Fixed
I-06	`peerChainID` uses testnet chainIds	Acknowledged
G-01	Consider reusing cached storage variable	Fixed

Detailed Findings

High Risk

H-01 - nana-buyback-hook: Hook swap execution is completely exposed to MEV when no quote is provided

Severity: High Risk

Technical Details: In line 312, the hook is intended to check for slippage: `if (quoteExists && exactSwapAmountOut < minimumSwapAmountOut) revert SpecifiedSlippageExceeded();`

However, if the payer/client has not specified a minimum amount, this check will not be triggered. This renders the TWAP (Time-Weighted Average Price) quote useless and exposes the trade to MEV (Miner Extractable Value).

Impact: High. Users who do not specify a minimum quote will be exposed to MEV, despite a TWAP minimum amount quote being calculated.

Recommendation: Consider extracting the `quoteExists` from the check so it can be used both for specified minimum quote trades and TWAP quotes.

Developer Response:

Fixed at: https://github.com/Bananapus/nana-buyback-hook/pull/23

H-02 - nana-buyback-hook: Missing `transferFrom` on afterPayRecordedWith will make hook DOS for all ERC20 tokens

Severity: High Risk

Technical Details: The function `afterPayRecordedWith` can process swaps using either native tokens or ERC20 tokens. For native tokens, `MultiTerminal` sends the corresponding amount as a value when calling `afterPayRecordedWith`. However, for ERC20 tokens, neither transfer nor `transferFrom` is called, resulting in no tokens being transferred to the hook. This prevents the swap from transferring tokens to the Uni V3 pool, rendering the hook inoperable for ERC20 tokens.

Impact: High. The hook cannot function for ERC20 tokens.

Recommendation: Implement a `transferFrom` call to move the tokens from the terminal to the hook when dealing with ERC20 tokens. Consider adding e2e tests to the hook repository, avoiding mocking other contracts of the protocol.

Developer Response:

Fixed at: https://github.com/Bananapus/nana-buyback-hook/pull/23

H-03 - nana-swap-terminal: `JBSwapTerminal` Incorrectly Uses `sqrtPriceX96` for Slippage Protection Calculation

Severity: High Risk

Technical Details: `JBSwapTerminal` uses the Uniswap v3 TWAP oracle to calculate `minAmountOut` in order to prevent the swap from incurring excessive slippage. Usually, this calculation is performed by converting the `arithmeticMeanTick` into a human-readable quote using the `OracleLibrary` function `getQuoteAtTick`.

```
// Keep a reference to the TWAP tick.
(int24 arithmeticMeanTick,) = OracleLibrary.consult(address(pool),
twapWindow);

// Get a quote based on this TWAP tick.
amountOut = OracleLibrary.getQuoteAtTick({
    tick: arithmeticMeanTick,
    baseAmount: uint128(amountIn),
    baseToken: terminalToken,
    quoteToken: address(projectToken)
});
```

However, in the case of the swap terminal, the `sqrtPriceX96` is being calculated instead and used as a quote, resulting in an incorrect slippage-adjusted `minAmountOut`. This can lead to two possible scenarios: either a Denial of Service (DOS) or the user being vulnerable to Miner Extractable Value (MEV) attacks.

```
(int24 arithmeticMeanTick,) = OracleLibrary.consult(address(pool),
secondsAgo);

// Get a quote based on that TWAP tick.
uint160 sqrtPriceX96 = TickMath.getSqrtRatioAtTick(arithmeticMeanTick);

// Return the lowest acceptable price for the swap based on the TWAP and
slippage tolerance.
return zeroForOne
    ? sqrtPriceX96 - (sqrtPriceX96 * slippageTolerance) /
SLIPPAGE_DENOMINATOR
    : sqrtPriceX96 + (sqrtPriceX96 * slippageTolerance) /
SLIPPAGE_DENOMINATOR;
```

Impact: High. The slippage protection is not only incorrectly calculated but it may also DOS the functionality in some cases.

Recommendation: Use the `OracleLibrary` function `getQuoteAtTick` to calculate the `minAmountOut` from the `arithmeticMeanTick`.

Developer Response:

Fixed at: https://github.com/Bananapus/nana-swap-terminal/pull/28

H-04 - bannyverse-core: Anyone can change the outfits of another users banny

Severity: High Risk

Technical Details:

On the `decorateBannyWith` function you can basically dress your Naked Banny with outfits.

The function expects a hook, which is where all the tiers of the NFT are stored, the Id of your bunny, the outfits and the `worldId`, which is the ID of the world that'll be associated with the specified banny:

```
 function decorateBannyWith(
        address hook,
        uint256 nakedBannyId,
        uint256 worldId,
        uint256[] calldata outfitIds
    )
        external
    {
```

The attack of changing another users outfits is performed similarly to the [C-1] issue where you can use a phantom contract to claim ownership of an NFT is not yours.

Therefore with custom returns you are able to bypass the following 4 checks:

```
 if (IERC721(hook).ownerOf(outfitId) != _msgSender()) revert
UNAUTHORIZED_OUTFIT();
            // Get the outfit's tier.
            outfitTier = IJB721TiersHook(hook).STORE().tierOfTokenId(hook,
outfitId, false);

            // Tier must exist
            if (outfitTier.id == 0) revert UNRECOGNIZED_OUTFIT();

            // The tier's category must be a known category.
            if (outfitTier.category < _BACKSIDE_CATEGORY ||
outfitTier.category > _TOPPING_CATEGORY) { //ok makes sense
                revert UNRECOGNIZED_CATEGORY();
            }
```

and update the outfits of another user:

`_attachedOutfitIdsOf[nakedBannyId] = outfitIds;`

Impact:

First impact: Anyone can attach whatever `outfitId` they want of whatever Id and rarity to their Banny without owning it.

Second impact: Anyone can change the `outfitId` of any user by setting themselves as owners in the phantom contract.

Recommendation:

Do have a proper whitelist for hooks do you wan't "fake" contracts.

Developer Response:

Fixed at: https://github.com/mejango/bannyverse-core/pull/13

H-05 - bannyverse-core: A `worldId` that you don't own can be attached to your Banny or update someone elses

Severity: High Risk

Technical Details:

On the `decorateBannyWith` function you can basically dress your Naked Banny with outfits.

The function expects a hook, which is where all the tiers of the NFT are stored, the Id of your bunny, the outfits and the `worldId`, which is the ID of the world that'll be associated with the specified banny:

```
function decorateBannyWith(
        address hook,
        uint256 nakedBannyId,
        uint256 worldId,
        uint256[] calldata outfitIds
    )
        external
    {
```

The attack of adding a `worldId` that you don't own to your banny is achieved with "Phantom contracts" which is basically creating a contract that has the interface expected and custom returns so that you can minupulate results.

In this case, `hook` is not validated, therefore an attacker is able to create a "hook" contract that adheres to the following interfaces `IERC721` and `IJB721TiersHook` to mimic the behaviour of the actual real hook contract and send custom returns to bypass checks:

the ownership check can be bypassed by on the `ownerOf` function on your phantom contract return yourself as the owner to any Id sent:

`if (IERC721(hook).ownerOf(worldId) != _msgSender()) revert UNAUTHORIZED_WORLD();`

And the last check can be bypassed by creating another phantom contract with the same interface as the `STORE` contract that returns a `worldTier.id` that is not 0

```
 JB721Tier memory worldTier =
IJB721TiersHook(hook).STORE().tierOfTokenId(hook, worldId, false);

 if (worldTier.id == 0) revert UNRECOGNIZED_WORLD();
```

After bypassing both checks, the attacker will have added whatever worldId they wanted to their Banny:

`_attachedWorldIdOf[nakedBannyId] = worldId;`

Impact:

First impact: Anyone can attach whatever `worldId` they want of whatever Id and rarity to their Banny without owning it.

Second impact: Anyone can change the `worldId` of any user by setting themselves as owners in the phantom contract.

Recommendation:

Do have a whitelist of hooks and check against it. There is a core problem across bannyverse of invalidated parameters. Would be wise to create a whitelist check against addresses like the hooks.

Developer Response:

Fixed at: https://github.com/mejango/bannyverse-core/pull/13

Medium Risk

M-01 - nana-core: Partial DOS on PERMIT2, permit flow

Severity: Medium Risk

Technical Details: `JBMultiTerminal` uses `PERMIT2` allowances to streamline user approvals and flows. However, due to the nature of this contract, if permit is not invoked within a try/catch block, it becomes vulnerable to a Denial of Service (DoS) attack. An attacker could front-run the transaction and activate the allowance on the `PERMIT2` contract, causing subsequent calls to permit with the same signature to fail.

Impact: Low, the `pay` and `addToBalanceOf` functions may be subject to DoS attacks if the actions described above are exploited.

Recommendation: Call permit within a try/catch block. If the call fails, check for an already available PERMIT2 allowance.

Developer Response:

Fixed at: https://github.com/Bananapus/nana-core/pull/149

M-02 - nana-core: Controller cannot call `setPrimaryTerminalOf`

Severity: Medium Risk

Technical Details: The function `setPrimaryTerminalOf` calls `_addTerminalIfNeeded` where the specifications state that if the ruleset's `allowSetTerminals` is not activated, only the project's controller can execute the function.

However the `JBController` does not implement any logic to perform this operation.

Impact: Low, the controller cannot set the primary terminal of a project causing a DOS.

Recommendation: Add the functionality for the controller to perform this operation.

Developer Response: Fixed at: https://github.com/Bananapus/nana-core/pull/162

M-03 - nana-core: `setTerminalsOf` is able to remove the primary terminal from the `_terminalsOf` list, rendering `isTerminalOf` to return `address(0)`

Severity: Medium Risk

Technical Details: The `setTerminalsOf` function can remove all the terminals of a project by passing an empty terminals array. This action will remove the primary terminal from the `_terminalsOf` list, resulting in `primaryTerminalOf` returning an empty address.

Impact: Medium, an address with the `SET_TERMINALS` permission is capable of removing a primary terminal without needing the `SET_PRIMARY_TERMINAL` permission. This

Recommendation: If this behaviour is not desired, make sure after calling `setTerminalsOf` that the primary terminal address is still a registered terminal of the project.

Developer Response:

Fixed at https://github.com/Bananapus/nana-permission-ids/pull/7

M-04 - nana-core: metadata wrong rounding direction on offset increment

Severity: Medium Risk

Technical Details: In the `createMetadata` function at line `226`, the following comment states that the offset should be increased by the data length rounded up:

```
// increment the offset by the data length (rounded up)
```

However, on line when performing the offset increment:

```
_offset += _datas[_i].length / JBMetadataResolver.WORD_SIZE;
```

We can see that it gets truncated (rounded down) making parts of data overlap in case the length of the current data item is not a perfect multiple of `WORD_SIZE`.

Impact: Medium, since metadata will be corrupted.

Recommendation: In the case `data[i].length` is not a multiple of `WORD_SIZE`, round up.

Developer Response:

Fixed at: https://github.com/Bananapus/nana-core/pull/164

M-05 - nana-core: `createMetadata` can overflow offset variable by one extra item

Severity: Medium Risk

Technical Details: In the `createMetadata` function, the `_offset` variable tracks the next available offset. This variable is stored as a `uint8`, which can hold a maximum value of 255 (2^8 - 1). However, on line 233, the following check is performed:

```
if (_offset > 2 ** 8) revert METADATA_TOO_LONG();
```

This condition allows `_offset` to overflow by one unit, potentially leading to metadata corruption.

Impact: Allowing the `_offset` variable to exceed its maximum value by one can lead to an overflow, resulting in the offset of the last item being truncated hence part of the metadata corrupted.

Recommendation: Change the check into the following in order to properly fit the max offset into a `uint8`:

```
if (_offset > 2 ** 8 - 1) revert METADATA_TOO_LONG();
```

Developer Response:

Fixed at: https://github.com/Bananapus/nana-core/pull/144/files

M-06 - nana-core: Price Feed staleness is not validated in `JBChainlinkV3PriceFeed`

Severity: Medium Risk

Technical Details

Price Feed staleness is not validated in `JBChainlinkV3PriceFeed` allowing the oracle to return stale prices and be accepted by the protocol

Impact

The oracle returns stale prices which will accepted by the protocol.

Recommendation

Add the following line:

```
function currentUnitPrice(uint256 decimals) external view override returns
(uint256) {
        // Get the latest round information from the feed.
        // slither-disable-next-line unused-return
        (uint80 roundId, int256 price,, uint256 updatedAt, uint80
answeredInRound) = FEED.latestRoundData();

        // Make sure the price isn't stale.
        if (answeredInRound < roundId) revert STALE_PRICE();

        // Make sure the round is finished.
        if (updatedAt == 0) revert INCOMPLETE_ROUND();

        // Make sure the price is positive.
        if (price < 0) revert NEGATIVE_PRICE();

+     if (block.timestamp - updatedAt > TIMEOUT) revert STALE_PRICE();

        // Get a reference to the number of decimals the feed uses.
        uint256 feedDecimals = FEED.decimals();

        // Return the price, adjusted to the specified number of decimals.
        return uint256(price).adjustDecimals({decimals: feedDecimals,
targetDecimals: decimals});
    }
```

Developer Response

Fixed at: https://github.com/Bananapus/nana-core/pull/146/files

M-07 - nana-core: Usage of deprecated `answeredInRound` parameter.

Severity: Medium Risk

Technical Details

Usage of deprecated `answeredInRound` parameter in `JBChainlinkV3PriceFeed`.

In the past, it was recommended to perform a staleness check by using the answeredInRound parameter. However, this parameter is now deprecated, and it is no longer necessary to check for staleness.

You can verify this by examining the OffchainAggregator.sol contract, specifically at line 810 where the latestRoundData() function is defined.

Here, you will notice that answeredInRound is always equal to roundId.
https://etherscan.deth.net/address/0x780f1bD91a5a22Ede36d4B2b2c0EcCB9b1726a28#c

Impact

Usage of deprecated `answeredInRound` parameter.

Recommendation

Remove the following:

```
 function currentUnitPrice(uint256 decimals) external view override returns
(uint256) {
        // Get the latest round information from the feed.
        // slither-disable-next-line unused-return
        (uint80 roundId, int256 price,, uint256 updatedAt, uint80
answeredInRound) = FEED.latestRoundData();

        // Make sure the price isn't stale.
-        if (answeredInRound < roundId) revert STALE_PRICE();
+       if(roundId = 0) revert STALE_PRICE();
```

Developer Response

Fixed at: https://github.com/Bananapus/nana-core/pull/146/files

M-08 - nana-core: ROOT permission is DOS'd in the whole codebase

Severity: Medium Risk

Technical Details

The whole architecture has a very clear permission and owership pattern, where `ROOT` is the main permission that is allowed to execute any call:

```
uint256 internal constant ROOT = 1; // All permissions across every contract. Very dangerous. BE CAREFUL!
```

The issue is that this permission is not checked against in the `_requirePermission()` functions.

```
function _requirePermissionFrom(address account, uint256 projectId, uint256 permissionId) internal view {
        address sender = _msgSender();
        if (
            sender != account && !PERMISSIONS.hasPermission(sender, account, projectId, permissionId)
                && !PERMISSIONS.hasPermission(sender, account, 0, permissionId)
        ) revert UNAUTHORIZED();
    }
```

Impact

ROOT permission is DOS'd in the whole codebase

Recommendation

Include the `ROOT` permission on the function:

```
function _requirePermissionFrom(address account, uint256 projectId, uint256 permissionId) internal view {
        address sender = _msgSender();
        if (
            sender != account && !PERMISSIONS.hasPermission(sender, account, projectId, permissionId)
-                && !PERMISSIONS.hasPermission(sender, account, 0, permissionId)
+                && !PERMISSIONS.hasPermission(sender, account, 0, permissionId) && !PERMISSIONS.hasPermission(sender, account, projectId, 1)
        ) revert UNAUTHORIZED();
    }
```

Developer Response

Fixed at: https://github.com/Bananapus/nana-core/pull/147

M-09 - nana-core: Missing sequencer check in `JBChainlinkV3PriceFeed` when deploying to L2s

Severity: Medium Risk

Technical Details

Using Chainlink in L2 chains such as Arbitrum requires to check if the sequencer is down to avoid prices from looking like they are fresh although they are not.

The bug could be leveraged by malicious actors to take advantage of the sequencer downtime in any action that fetches `currentUnitPrice`.

Impact

Missing sequencer check in `JBChainlinkV3PriceFeed` when deploying to L2s could be leveraged by malicious actors to take advantage of the sequencer downtime.

Recommendation

Follow chainlinks recommendation to check for the state of the sequencer:

https://docs.chain.link/data-feeds/l2-sequencer-feeds#example-code

To make it cleaner perhap have 2 different contracts one for mainnet and the other one for L2s with sequencers.

Developer Response

Fixed at: https://github.com/Bananapus/nana-core/pull/146/files

M-10 - nana-swap-terminal: Project's owner can set a project's pool to an arbitrary contract

Severity: Medium Risk

Technical Details: The function `addDefaultPool` allows either a project owner or a permissioned address to assign a pool for a `projectId` and a `token`. Currently, this function lacks validation ensuring that the specified pool address is a Uniswap V3 pool and relates to both tokens involved in the swap.

```
function addDefaultPool(uint256 projectId, address token, IUniswapV3Pool
pool) external {
    // Only the project owner can set the default pool for a token, only the
project owner can set the
    // pool for its project.
    if (!(projectId == DEFAULT_PROJECT_ID && msg.sender == owner())) {
        _requirePermissionFrom(PROJECTS.ownerOf(projectId), projectId,
JBPermissionIds.ADD_SWAP_TERMINAL_POOL);
    }

    // Update the project's default pool for the token.
    _poolFor[projectId][token] = PoolConfig({pool: pool, zeroForOne: token <
TOKEN_OUT});

    // Update the project's accounting context for the token.
    _accountingContextFor[projectId][token] = JBAccountingContext({
        token: token,
        decimals: IERC20Metadata(token).decimals(),
        currency: uint32(uint160(token))
    });

    _tokensWithAContext[projectId].push(token);
}
```

While this oversight is a "Centralisation Risk" issue, it's noteworthy that this risk can be entirely mitigated by computing Uniswap V3 pool addresses based on the involved tokens. Therefore, eliminating this additional centralization vector makes practical sense.

Impact: Medium. A project owner can designate a malicious or arbitrary contract instead of a legitimate Uniswap pool.

Recommendation: Calculate the pool address from the tokens or utilize the Uniswap factory to retrieve the corresponding pool for a token pair.

Developer Response:

Fixed at https://github.com/Bananapus/nana-swap-terminal/pull/34

M-11 - nana-swap-terminal: `uniswapV3SwapCallback` basic structure and checks differs from Uniswap v3 `SwapRouter` implementation

Severity: Medium Risk

Technical Details: The `SwapRouter` in Uniswap v3 incorporates the `uniswapV3SwapCallback` interface, featuring a series of validations and strategic considerations. Conversely, `JBSwapTerminal` lacks this implementation.

- Absence of `verifyCallback` validation: Despite being outlined in the natspec, the absence of this validation leaves potential vulnerabilities unaddressed. Restricting the callback function solely to Uniswap v3 pools could mitigate several potential attack vectors.
- Adjustment in `amountToSendToPool` calculation: Instead of employing a strict comparison, the calculation should account for values equal to or less than 0.

Impact: Considered low, these adjustments primarily align with industry best practices and heuristic guidelines.

Recommendation: It is advisable to correctly implement both validations.

Developer Response:

Fixed at https://github.com/Bananapus/nana-swap-terminal/pull/34

M-12 - nana-swap-terminal: Low Liquidity in Uniswap V3 Pool can lead to tokens being locked up in `JBSwapTerminal` contract

Severity: Medium Risk

Technical Details: `JBSwapTerminal` contract uses Uniswap v3 pools to exchange the tokens it received for tokens that another one of its project's terminals can accept. The operation provides the `sqrtPriceLimitX96` to the lowest possible price, and the slippage is checked at the callback.

However, if the Uniswap V3 pool lacks sufficient liquidity or being manipulated before the transaction is executed, the swap will halt once the pool's price reaches the `sqrtPriceLimitX96` value. Consequently, not all the tokens sent to the contract will be utilized, resulting in the remaining tokens becoming permanently locked within the contract.

PoC:

The `_swap` function interacts with the Uniswap V3 pool. It sets `sqrtPriceLimitX96` to the minimum or maximum feasible value to ensure that the swap attempts to use all available liquidity in the pool.

```
(int256 amount0, int256 amount1) = swapConfig.pool.swap({
    recipient: address(this), // Send output tokens to this terminal.
    zeroForOne: zeroForOne, // The direction of the swap.
    amountSpecified: int256(swapConfig.amountIn), // The amount of input
tokens to swap.
    sqrtPriceLimitX96: zeroForOne ? TickMath.MIN_SQRT_RATIO + 1 :
TickMath.MAX_SQRT_RATIO - 1, // The price //
        // limit for the swap.
    data: abi.encode(tokenIn, swapConfig.inIsNativeToken) // Additional data
which will be forwarded to the
        // callback.
});
```

In the Uniswap V3 pool, this check stops the loop if the price limit is reached or the entire input has been used. If the pool does not have enough liquidity, it will still do the swap until the price reaches the minimum/maximum price.

```
// continue swapping as long as we haven't used the entire input/output and
haven't reached the price limit
while (state.amountSpecifiedRemaining != 0 && state.sqrtPriceX96 !=
sqrtPriceLimitX96) {
    StepComputations memory step;

    step.sqrtPriceStartX96 = state.sqrtPriceX96;

    (step.tickNext, step.initialized) =
tickBitmap.nextInitializedTickWithinOneWord(
        state.tick,
        tickSpacing,
        zeroForOne
    );
```

Finally, the `uniswapV3SwapCallback` function uses the input from the pool callback to transfer the inToken amount to the pool. So, if `amountToSendToPool` < `amountIn`, the unused token amount is locked in the contract.

```
function uniswapV3SwapCallback(int256 amount0Delta, int256 amount1Delta,
bytes calldata data) external override {
    // Unpack the data from the original swap config (forwarded through
`_swap(...)`).
    (address tokenIn, bool shouldWrap) = abi.decode(data, (address, bool));

    // Keep a reference to the amount of tokens that should be sent to
fulfill the swap (the positive delta).
    uint256 amountToSendToPool = amount0Delta < 0 ? uint256(amount1Delta) :
uint256(amount0Delta);
    // Wrap native tokens if needed.
    if (shouldWrap) WETH.deposit{value: amountToSendToPool}();

    // Transfer the tokens to the pool.
    // This terminal should NEVER keep a token balance.
    IERC20(tokenIn).transfer(msg.sender, amountToSendToPool);
}
```

Impact: Medium, user funds would get stuck into the contract.

Recommendation: Consider implementing a check for unused funds in the contract and returning them to the user after the swap.

Developer Response:

Fixed at https://github.com/Bananapus/nana-swap-terminal/pull/35

M-13 - bannyverse-core: Default outfits are not attached to Bannys with a smaller category than `_FACE_CATEGORY` on their Banny

Severity: Medium Risk

Technical Details:

When fetching the SVG contents for a list of outfit IDs in the, `_outfitContentsFor()` function, if your Banny does not have the basics, which are a Neckclace, a face, eyes and a mouth, they will return default items for those categories such as `_DEFAULT_MOUTH` for example.

The way they do it is checking that the category equal one of the basic categories such as `_FACE_CATEGORY` and if not, it checks that if it is bigger than `_FACE_CATEGORY` and you do not have one attached, it will return the default one: `else contents = string.concat(contents, _DEFAULT_STANDARD_EYES);`

As you can see in the following snippet, if you only have 1 outfit like a Necklace (this outfit should be smaller than `_FACE_CATEGORY`), then the default items will not be included in the URI: `string memory contents`

```
if (category == _NECKLACE_CATEGORY) {
          hasNecklace = true;
     } else if (category > _NECKLACE_CATEGORY && !hasNecklace) {
          contents = string.concat(contents, _DEFAULT_NECKLACE);
          hasNecklace = true;
          }
     if (category == _FACE_CATEGORY) {
          hasFace = true;
     } else if (category > _FACE_CATEGORY && !hasFace) {
```

This can also happen if the category is smaller than `_NECKLACE_CATEGORY`, but less probable.

Impact:

Bannys will be under-dressed and return incorrect SVGs

Recommendation:

Include cases for when the user only has one outfit and it is smaller than `_FACE_CATEGORY` or `_NECKLACE_CATEGORY`, so they can still get the default outfit returned

Developer Response:

Fixed at: https://github.com/mejango/bannyverse-core/blob/1c785f6f61af0e2d452c446555286404ad6f1127

M-14 - bannyverse-core: Broken URIs for `_NAKED_CATEGORY` and `_ONESIE_CATEGORY`

Severity: Medium Risk

Technical Details:

The function `_nameOf` which is used to get the name of each TokenId depending on their `category` checks through all the possible categories:

```
if (category == _WORLD_CATEGORY) {
        return string.concat("World: ", name);
    } else if (category == _BACKSIDE_CATEGORY) {
        return string.concat("Backside: ", name);
    } else if (category == _SHOE_CATEGORY) {
        return string.concat("Shoe: ", name);
    } else if (category == _NECKLACE_CATEGORY) {
        return string.concat("Necklace: ", name);
    } else if (category == _FACE_CATEGORY) {
        return string.concat("Face: ", name);
    } else if (category == _FACE_EYES_CATEGORY) {
        return string.concat("Eyes: ", name);
    } else if (category == _FACE_MOUTH_CATEGORY) {
        return string.concat("Mouth: ", name);
    } else if (category == _HEADGEAR_CATEGORY) {
        return string.concat("Hair: ", name);
    } else if (category == _HEAD_CATEGORY) {
        return string.concat("Head: ", name);
    } else if (category == _SUIT_CATEGORY) {
        return string.concat("Suit: ", name);
    } else if (category == _SUIT_TOP_CATEGORY) {
        return string.concat("Suit top: ", name);
    } else if (category == _SUIT_BOTTOM_CATEGORY) {
        return string.concat("Suit bottom: ", name);
    } else if (category == _FIST_CATEGORY) {
        return string.concat("Fist: ", name);
    } else if (category == _TOPPING_CATEGORY) {
        return string.concat("Topping: ", name);
    }
    return "";
```

Though the name will be completly empty for 2 categories that are missing on the `if` cases, which are `_NAKED_CATEGORY` and `_ONESIE_CATEGORY`.

The `_nameOf()` function is called after from `tokenUriOf()` to get the correct URI of an specified NFT, which for the two previous mentioned categories, the URI will be broken.

Impact:

Broken URI for any NFTs with the `_NAKED_CATEGORY` and `_ONESIE_CATEGORY` categories.

Recommendation:

Add a case for those categories:

```
  if (category == _WORLD_CATEGORY) {
          return string.concat("World: ", name);
      } else if (category == _BACKSIDE_CATEGORY) {
          return string.concat("Backside: ", name);
      } else if (category == _SHOE_CATEGORY) {
          return string.concat("Shoe: ", name);
      } else if (category == _NECKLACE_CATEGORY) {
          return string.concat("Necklace: ", name);
      } else if (category == _FACE_CATEGORY) {
          return string.concat("Face: ", name);
      } else if (category == _FACE_EYES_CATEGORY) {
          return string.concat("Eyes: ", name);
      } else if (category == _FACE_MOUTH_CATEGORY) {
          return string.concat("Mouth: ", name);
      } else if (category == _HEADGEAR_CATEGORY) {
          return string.concat("Hair: ", name);
      } else if (category == _HEAD_CATEGORY) {
          return string.concat("Head: ", name);
      } else if (category == _SUIT_CATEGORY) {
          return string.concat("Suit: ", name);
      } else if (category == _SUIT_TOP_CATEGORY) {
          return string.concat("Suit top: ", name);
      } else if (category == _SUIT_BOTTOM_CATEGORY) {
          return string.concat("Suit bottom: ", name);
      } else if (category == _FIST_CATEGORY) {
          return string.concat("Fist: ", name);
      } else if (category == _TOPPING_CATEGORY) {
          return string.concat("Topping: ", name);
+     }else if (category == _NAKED_CATEGORY) {
+         return string.concat("Naked: ", name);
+     }else if (category == _ONESIE_CATEGORY) {
+         return string.concat("Onesie: ", name);
+     }

      return "";
```

Developer Response:

Fixed at https://github.com/mejango/bannyverse-core/blob/1c785f6f61af0e2d452c446555286404ad6f1127

M-15 - nana-721-hook: Tiers with the flag `noNewTiersWithReserves` can still have reserve beneficiaries

Severity: Medium Risk

Technical Details

On the `recordAddTiers` function, you can specify a group of flags, including `bool noNewTiersWithReserves;` , which means that a new tier with any sort of reserves is not wanted.

On the check for it, it misses to check for the atcual `reserveBeneficiary` which allows for it being set even when the `noNewTiersWithReserves` would be true:

```
 if ((flags.noNewTiersWithReserves || tierToAdd.allowOwnerMint) &&
tierToAdd.reserveFrequency != 0) {
                revert RESERVE_FREQUENCY_NOT_ALLOWED();
}
```

At the end of the function, the `noNewTiersWithReserves` flag is not checked and it is directly updated with the `reserveBeneficiary` :

```
// Set the reserve beneficiary if needed.
if (tierToAdd.reserveBeneficiary != address(0)) {
    if (tierToAdd.useReserveBeneficiaryAsDefault) {
        if (defaultReserveBeneficiaryOf[msg.sender] !=
tierToAdd.reserveBeneficiary) {
            defaultReserveBeneficiaryOf[msg.sender] =
tierToAdd.reserveBeneficiary;
        }
    } else {
        _reserveBeneficiaryOf[msg.sender][tierId] =
tierToAdd.reserveBeneficiary;
    }
}
```

Impact
Tiers with the flag `noNewTiersWithReserves` can still have reserve beneficiaries

Recommendation

Add the following:

```
-    if ((flags.noNewTiersWithReserves || tierToAdd.allowOwnerMint) &&
tierToAdd.reserveFrequency != 0) {
+.   if ((flags.noNewTiersWithReserves || tierToAdd.allowOwnerMint) &&
tierToAdd.reserveFrequency != 0 && tierToAdd.reserveBeneficiary !=
address(0)) {
-     revert RESERVE_FREQUENCY_NOT_ALLOWED();
+     revert RESERVE_NOT_ALLOWED();

}
```

Developer Response

Fixed at: https://github.com/Bananapus/nana-721-hook/pull/26

M-16 - croptop-core: Unsafe casting allows to set `tiersToAdd` for the wrong category post

Severity: Medium Risk

Technical Details:

Currently, when calling `_setupPosts` , an array of `CTPost[] memory posts` will be passed as an argument:

```
function _setupPosts(
     uint256 projectId,
     address nft,
     CTPost[] memory posts
   )


struct CTPost {
      bytes32 encodedIPFSUri;
      uint32 totalSupply;
      uint88 price;
      uint16 category;
}
```

This array uses both a `uint16 category;` and a `uint88 price;` values which notice that they are uint16 and uint88 respectively.

At the end, this 2 values are stored with their casting, therefore a `category = 257` will be stored as `category = 1` . Same with price (though this should not happen as then you have to pay the price as msg.value).

```
// Set the tier.
tiersToAdd[numberOfTiersBeingAdded] = JB721TierConfig({
      price: uint80(post.price),
      initialSupply: post.totalSupply,
      votingUnits: 0,
      reserveFrequency: 0,
      reserveBeneficiary: address(0),
      encodedIPFSUri: post.encodedIPFSUri,
      category: uint8(post.category),
```

Impact:

Category and price can overflow and be stored with the incorrect values.

Recommendation:

Cap those 2 values to their correct castings from the start in the struct or do use safeCast from OZ.

```
struct CTPost {
     bytes32 encodedIPFSUri;
     uint32 totalSupply;
-    uint88 price;
-    uint16 category;
+    uint80 price;
+    uint8 category;
}
```

Developer Response:

Fixed.

Low Risk

L-01 - nana-core: `_addTerminalIfNeeded` is allowed to add a terminal if interface is not supported

Severity: Low Risk

Technical Details: Following the `_addTerminalIfNeeded` specs, "Unless the caller is the project's controller, the project's ruleset must have `allowSetTerminals` set to `true`"

Any address that is not the controller should only be able to add a terminal if the controller allows it. However on line 300 in case a controller does not support the IJBDirectoryAccessControl any arbitrary terminal is still able to be added to the project.

```
bool allowSetTerminals =
!controller.supportsInterface(type(IJBDirectoryAccessControl).interfaceId)
|| IJBDirectoryAccessControl(address(controller))
   .setTerminalsAllowed(projectId);
```

Impact: Incorrect behaviour of the contracts

Recommendation: If this is not desired functionality remove the first part of the or clause.

Developer Response:

Fixed at: https://github.com/Bananapus/nana-core/pull/162

L-02 - nana-core: Reverse selector clashing can be encountered when setting the `currency`

Severity: Low Risk

Technical Details

When calling `addAccountingContextsFor` on the terminal, you set the `JBAccountingContext` which is composed of a token and the 4 last bytes of the token that are converted to `uint32()`:

```
accountingContext.token = token;
accountingContext.decimals = token == JBConstants.NATIVE_TOKEN ? 18 :
IERC20Metadata(token).decimals();
// Use the last 4 bytes of the address as the currency.
accountingContext.currency = uint32(uint160(token));
```

This last 4 bytes can collide or brute force to collide, therefore adding a different token could have the same currency `accountingContext.currency`

This will affect the `uint256 weightRatio = amount.currency == ruleset.baseCurrency()` calculation when calling `recordPaymentFrom()`

Impact

This last 4 bytes can collide or brute force to collide, therefore adding a different token could have the currency `accountingContext.currency`

Recommendation

Either use a bigger casting for the currency, or modify the currency term to include the address too directly.

```
-    accountingContext.currency = uint32(uint160(token));
+    accountingContext.currency = uint64(uint160(token));
```

Developer Response

Fixed at: https://github.com/Bananapus/nana-core/pull/160/files.

L-03 - nana-buyback-hook: A pool related to a project can be changed to a pool with a different fee

Severity: Low Risk

Technical Details: Inside the buyback hook, the pool should only be set once, following the comment on line 462:

```
// Make sure this pool hasn't already been set in this hook.
```

This is correct for pools using the same fee configuration. However, since Uni V3 pool addresses also depend on the fee parameter, a different pool could be generated for the same inputToken and projectId parameters. This would render the following check useless:

```
if (poolOf[projectId][terminalToken] == newPool) revert PoolAlreadySet();
```

Since a project owner could change the pool from one fee configuration to another (e.g., 0.05%, 0.30%, and 1%).

Impact: Medium. Pools that shouldn't be changed once set can be switched to a different pool with different fees, including non-existent pools.

Recommendation: Consider changing the pool check line to:

```
if (poolOf[projectId][terminalToken] != address(0)) revert PoolAlreadySet();
```

Developer Response:

Fixed at: https://github.com/Bananapus/nana-buyback-hook/pull/26

L-04 - nana-buyback-hook: Consider extracting `amountToSwapWith > totalPaid` check out of the if clause

Severity: Low Risk

Technical Details: Following the early revert best practice consider extracting the check on line 198:

```
if (amountToSwapWith > totalPaid) revert InsufficientPayAmount();
```

Outside of the if clause so in case an invalid quote is sent on the metadata it reverts across all possible cases.

Impact: Low

Recommendation: Consider extracting `amountToSwapWith > totalPaid` check our of the if clause.

Developer Response:

Fixed at: https://github.com/Bananapus/nana-buyback-hook/pull/23

L-05 - nana-swap-terminal: Partial DOS on `PERMIT2`, `permit` flow

Severity: Low Risk

Technical Details: `JBSwapTerminal` uses PERMIT2 allowances to streamline user approvals and flows. However, due to the nature of this contract, if `permit` is not invoked within a try/catch block, it becomes vulnerable to a Denial of Service (DoS) attack. An attacker could front-run the transaction and activate the allowance on the PERMIT2 contract, causing subsequent calls to `permit` with the same signature to fail.

Impact: Low: The `pay` and `addToBalanceOf` functions may be subject to DoS attacks if the actions described above are exploited.

Recommendation: Call `permit` within a try/catch block. If the call fails, check for an already available PERMIT2 allowance.

Developer Response:

Fixed at https://github.com/Bananapus/nana-swap-terminal/pull/35

L-06 - nana-swap-terminal: If `OUT_IS_NATIVE_TOKEN` is set to true, swap terminal will increase unnecesarily the approval of WETH for the `nextTerminal`

Severity: Low Risk

Technical Details: During the execution of `_handleTokenTransfersAndSwap`, `_beforeTransferFor` is called, which is supposed to approve the `nextTerminal` with `amountToSend` so the swapped tokens can be forwarded.

In the case of the terminal being configured to use native tokens as output, this should not happen. However, since in this condition `TOKEN_OUT` is set to WETH, the check on line 705 can never be evaluated to true:

```
if (token == JBConstants.NATIVE_TOKEN) return;
```

Therefore, the WETH allowance of `nextTerminal` is increased.

Impact: Low, even though an approval is being created, under normal conditions the swap terminal is not supposed to hold tokens between transactions.

Recommendation:

Change the previous check to the following:

```
if (OUT_IS_NATIVE_TOKEN) return;
```

Developer Response:

Fixed at https://github.com/Bananapus/nana-swap-terminal/pull/35

L-07 - nana-swap-terminal: Use `safeTransfer` instead of transfer

Severity: Low Risk

Technical Details: Tokens not compliant with the ERC20 specification could return false from the transfer function call to indicate the transfer fails, while the calling contract would not notice the failure if the return value is not checked. Checking the return value is a requirement, as written in the EIP-20 specification.

Impact: LOW

Recommendation: Use the SafeERC20 library implementation from OpenZeppelin and call `safeTransfer` or `safeTransferFrom` when transferring ERC20 tokens.

Developer Response:

Fixed at https://github.com/Bananapus/nana-swap-terminal/pull/35

L-08 - nana-suckers: Sucker deployers can't be disallowed

Severity: Low Risk

Technical Details:

The function `allowSuckerDeployer` adds a suckers deployer to the allowlist of deployers.

```
function allowSuckerDeployer(address deployer) public override onlyOwner {
      suckerDeployerIsAllowed[deployer] = true;
      emit SuckerDeployerAllowed(deployer);
}
```

While this is an Ownable function, if for whatever reason the sucker is malicious of the private keys are lost, there is no function to remove the allowance to the sucker

Impact:

Sucker deployers can't be disallowed

Recommendation:

Add a disallowed sucker function.

```
+   function disallowSuckerDeployer(address deployer) public override
onlyOwner {
+      suckerDeployerIsAllowed[deployer] = false;
+      emit SuckerDeployerNotAllowed(deployer);
+     }
```

Developer Response:
Acknowledged

Informational

I-01 - nana-core: Missing parameter on natspec

Severity: Informational

Technical Details: The function `deployERC20For` is missing the salt parameter description on the natspec.

Recommendation: Add the NATSPEC accordingly

Developer Response: Fixed at: https://github.com/Bananapus/nana-core/pull/163

I-02 - nana-swap-terminal: `_swap`, `zeroForOne` unnecessary calculation

Severity: Informational

Technical Details: Line `651` of the `_swap` function calculates `zeroForOne` order of the tokens in the pool.

`bool zeroForOne = tokenIn < TOKEN_OUT;`

The problem is that this boolean has already been calculated at `_pickPoolAndQuote`.

Recommendation: Remove the line and use the variable in memory `swapConfig.zeroForOne`.

Developer Response:

Fixed at https://github.com/Bananapus/nana-swap-terminal/pull/35

I-03 - bannyverse-core: Missing indexing in some key events

Severity: Informational

Technical Details:

In solidity you can index up to 3 items per event:

The following events should have the `caller` address indexed

Recommendation:

Add the indexed keyword to the following events:

```
-    event DecorateBanny(
-        address indexed hook, uint256 indexed nakenBannyId, uint256
worldId, uint256[] outfitIds, address caller
-    );
-    event SetSvgContents(uint256[] indexed tierId, string[] svgContents,
address caller);
-    event SetSvgHashes(uint256[] indexed tierIds, bytes32[] indexed
svgHashs, address caller);
-    event SetSvgBaseUri(string baseUri, address caller);
-    event SetTierNames(uint256[] indexed tierIds, string[] names, address
caller);

+    event DecorateBanny(
+        address indexed hook, uint256 indexed nakenBannyId, uint256
worldId, uint256[] outfitIds, address indexed caller
+    );
+    event SetSvgContents(uint256[] indexed tierId, string[] svgContents,
address indexed caller);
+    event SetSvgHashes(uint256[] indexed tierIds, bytes32[] indexed
svgHashs, address indexed caller);
+    event SetSvgBaseUri(string baseUri, address indexed caller);
+    event SetTierNames(uint256[] indexed tierIds, string[] names, address
indexed caller);
```

Developer Response:

Acknowledged. Throughout jb, we emit a trailing caller address in events. they're not indexed anywhere else, i think it's not needed here either, despite us having the ability to add 3.

I-04 - croptop-core: Quality assurance issues

Severity: Informational

Technical Details:

- `CTPublisher public PUBLISHER;` in `CTDeployer` should be immutable.
- The `error HOOK_NOT_PROVIDED()` error in `CTPublisher` is un-used, remove it
- The following comment in `CTPublisher` repeats `reference` twice:

```
// Keep a reference a reference to the fee.
uint256 fee;
```

- The following comment is incorrect, as it repeats `// Make sure there is a minimum supply.` twice:

```
// Make sure there is a minimum supply.
if (allowedPost.minimumTotalSupply == 0) {
    revert TOTAL_SUPPY_MUST_BE_POSITIVE();
}

// Make sure there is a minimum supply.
if (allowedPost.minimumTotalSupply > allowedPost.maximumTotalSupply) {
    revert MAX_TOTAL_SUPPLY_LESS_THAN_MIN();
}
```

Impact:
Quality assurance issues

Recommendation:

Do change all the issues above with the given recommendations.

Additionally, for the incorrect comment issue:

```
      // Make sure there is a minimum supply.
      if (allowedPost.minimumTotalSupply == 0) {
          revert TOTAL_SUPPY_MUST_BE_POSITIVE();
      }

-      // Make sure there is a minimum supply.
+      // Make sure the minimum supply does not surpass the maximum supply.
      if (allowedPost.minimumTotalSupply > allowedPost.maximumTotalSupply) {
          revert MAX_TOTAL_SUPPLY_LESS_THAN_MIN();
      }
```

Developer Response:

great!!

I-05 - nana-suckers: Quality assurance issues

Severity: Informational

Technical Details:

- The `ONLY_SUCKERS` error in `BPOptimismSuckerDeployer` is not used, remove it
- The `event SuckingToRemote(address token, uint64 nonce);` event in `BPOptimismSucker` is un-used, remove it

Impact:

Dead code.

Recommendation:

Remove the code above

Developer Response:
Fixed at commit: `8028bbc518640eef9d44c84b9a03f061140398e0` & `88f4e836c6a325ba9d3a9fe8bb933a1b59eaae99`.

I-06 - nana-suckers: `peerChainID` uses testnet chainIds

Severity: Informational

`peerChainID` uses testnet chainIds

Technical Details:

The `peerChainID` function uses testnet chainIds for testing purposes:

```
 function peerChainID() external view virtual override returns (uint256
chainId) {
      uint256 _localChainId = block.chainid;
      if (_localChainId == 1) return 10;
      if (_localChainId == 10) return 1;
      if (_localChainId == 11155111) return 11155420;
      if (_localChainId == 11155420) return 11155111;
    }
```

Contracts that will be deployed should not reference testnet or testing values.

Impact:

Deployment commit references testnet chain Ids

Recommendation:

Remove them:

```
 function peerChainID() external view virtual override returns (uint256
chainId) {
        uint256 _localChainId = block.chainid;
        if (_localChainId == 1) return 10;
        if (_localChainId == 10) return 1;
-       if (_localChainId == 11155111) return 11155420;
-       if (_localChainId == 11155420) return 11155111;
    }
```

Developer Response:
Acknowledged

Gas Optimization

G-01 - nana-core: Consider reusing cached storage variable

Technical Details: In the contract `JBDirectory` line `304`, `controllerOf[projectId]` reads storage for the second time instead of using the `controller` variable cached at line `297`.

Recommendation: Consider using the local variable instead of reading from storage in order to save one `SLOAD` operation.

Developer Response:

Fixed at: https://github.com/Bananapus/nana-core/pull/161

Disclaimer

This report does not endorse or critique any specific project or team. It does not assess the economic value or viability of any product or asset developed by parties engaging Enigma Dark for security assessments. We do not provide warranties regarding the bug-free nature of analyzed technology or make judgments on its business model, proprietors, or legal compliance.

This report is not intended for investment decisions or project participation guidance. Enigma Dark aims to improve code quality and mitigate risks associated with blockchain technology and cryptographic tokens through rigorous assessments.

Blockchain technology and cryptographic assets inherently involve significant risks. Each entity is responsible for conducting their own due diligence and maintaining security measures. Our assessments aim to reduce vulnerabilities but do not guarantee the security or functionality of the technologies analyzed.

This security engagement does not guarantee against a hack. It is a review of the codebase at a during a specific period of time. Enigma Dark makes no warranties regarding the security of the code and does not warrant that the code is free from defects. By deploying or using the code, the project and users of the contracts agree to use the code at their own risk. Any modifications to the code will require a new security review.