SHERLOCK

SHERLOCK SECURITY REVIEW FOR

SHERLOCK

Prepared for:	NFTPort
Prepared by:	Sherlock
Lead Security Expert:	GimelSec
Dates Audited:	October 19 - October 26, 2022
Prepared on:	November 7, 2022

Introduction

Bring your NFT application to market in hours instead of months. We take care of the NFT infrastructure so you can focus on your application. Built by developers, for developers.

Scope

```
Factory.sol
Base64.sol
ERC2981.sol
Config.sol
GranularRoles.sol
NFTCollection.sol
ERC721NFTProduct.sol
ERC1155NFTProduct.sol
```

Findings

Each issue has an assigned severity:

- 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

Medium	High
9	0

Security experts who found valid issues

obront
bin2chen
rvierdiiev
ElKu
GimelSec
cccz
0xSmartContract
keccak123
ignacio
ctf_sec
JohnSmith
ak1
minhquanym
8olidity
Lambda
JohnnyTime
pashov
joestakey
Dravee

Issue M-1: `abi.encodePacked` Allows Hash Collision

Source: https://github.com/sherlock-audit/2022-10-nftport-judging/issues/118

Found by

ignacio, keccak123

Summary

From the solidity documentation: https://docs.soliditylang.org/en/v0.8.17/abi-spec.html?highlight=collisions#non-standard-packed-mode > If you use `keccak256(abi.encodePacked(a,b))` and both `a` and `b` are dynamic types, it is easy to craft collisions in the hash value by moving parts of `a` into `b` and vice-versa. More specifically, `abi.encodePacked("a","bc")==abi.encodePacked("ab","c")`.

This issue exists in the Factory contract can results in hash collisions, bypassing the `signedOnly` modifier.

Vulnerability Detail

The issue is in these lines of code: https://github.com/sherlock-audit/2022-10-nftport/blob/main/evm-minting-master/contracts/Factory.sol#L171 https://github.com/sherlock-audit/2022-10-nftport/blob/main/evm-minting-master/contracts/Factory.sol#L195 https://github.com/sherlock-audit/2022-10-nftport/blob/main/evm-minting-master/contracts/Factory.sol#L222

As the solidity docs describe, two or more dynamic types are passed to `abi.encodePacked`. Moreover, these dynamic values are user-specified function arguments in external functions, meaning anyone can directly specify the value of these arguments when calling the function. The `signedOnly` modifier is supposed to protect functions to permit only function arguments that have been properly signed to be passed to the function logic, but because a collision can be created, the modifier can be bypassed for certain select inputs that result in the same `encodePacked` value.

Impact

The `signedOnly` modifier (line 537) is not effective because the modifier can be bypassed by different function arguments that result in the same signature when the values are `encodePacked` together. This can result in the submission of values that were not actually signed.

Code Snippet

All instances of `abi.encodePacked` in the contract pass multiple dynamic type arguments https://github.com/sherlock-audit/2022-10-nftport/blob/main/evm-minting-m aster/contracts/Factory.sol#L171 https://github.com/sherlock-audit/2022-10-nftport/blob/main/evm-minting-master/contracts/Factory.sol#L195 https://github.com/sherlock-audit/2022-10-nftport/blob/main/evm-minting-master/contracts/Factory.sol#L222

Tool used

Manual Review

Recommendation

Instead of writing functions to accept several arguments that are hashed inside the function, consider rewriting the function to take the hashed value as a function argument directly so that the hashing process happens off-chain. This approach would solve the issue and save gas.

Discussion

Evert0x

Downgrading to medium severity, fails to show an exploit pattern.

Issue M-2: Factory.sol : Issue with arbitrary data as signature in signature based call and deploy methods.

Source: https://github.com/sherlock-audit/2022-10-nftport-judging/issues/106

Found by

Lambda, ak1, obront, bin2chen, minhquanym, JohnSmith, ctf_sec, 8olidity

Summary

In Factory contract, deploy and call methods are using the signature based approach for deployment. This is not safe when we look at the way the signature is comes from user.

Vulnerability Detail

https://github.com/sherlock-audit/2022-10-nftport/blob/main/evm-minting-master/contracts/Factory.sol#L147-L225

In above line of codes, user is allowed for deployment by using the signature data. This could have multiple impacts as explained in Impact section.

Impact

1. Signature replay attack.
2. Signature reuse across different NFT Port projects if it is to be launched in multiple chains. Because the chain ID is not included in the data, all signatures are also valid when the project is launched on a chain with another chain ID. For instance, let's say it is also launched on Polygon. An attacker can now use all of the Ethereum signatures there. Because the Polygon addresses of user's (and potentially contracts, when the nonces for creating are the same) are often identical, there can be situations where the payload is meaningful on both chains.
3. Signature without domain , nonces are not safe along with the standard specified in EIP 712.
4. Signature reuse from different Ethereum projects & phishing Because the signature is very generic, there might be situations where a user has already signed data with the same format for a completely different Ethereum application. Furthermore, an attacker could set up a DApp that uses the same format and trick someone into signing the data. Even a very security-conscious owner that has audited the contract of this DApp (that does not have any vulnerabilities and is not malicious, it simply consumes signatures that happen to have the same

format) might be willing to sign data for this DApp, as he does not anticipate that this puts his NFT Port project in danger.

Code Snippet

https://github.com/sherlock-audit/2022-10-nftport/blob/main/evm-minting-master/contracts/Factory.sol#L147-L225

Tool used

Manual Review

Recommendation

I strongly recommend to follow EIP-712 and not implement your own standard / solution. While this also improves the user experience, this topic is very complex and not easy to get right, so it is recommended to use a battle-tested approach that people have thought in detail about. All of the mentioned attacks are not possible with EIP-712: 1.) There is always a domain separator that includes the contract address. 2.) The chain ID is included in the domain separator 5.) There is a type hash (of the function name / parameters) 6.) The domain separator does not allow reuse across different projects, phishing with an innocent DApp is no longer possible (it would be shown to the user that he is signing data for Rigor, which he would off course not do on a different site)

Issue M-3: The supply of NFT for each tokenID in ERC1155NFTProc cannot be modified after the first minting

Source: https://github.com/sherlock-audit/2022-10-nftport-judging/issues/95

Found by

cccz, GimelSec

Summary

The supply of NFT for each tokenID in ERC1155NFTProduct cannot be modified after the first minting

Vulnerability Detail

When minting NFT in the ERC1155NFTProduct contract, it will check whether the tokenID already exists, that is, as long as tokenSupply[tokenId] > 0, the ERC1155 NFT of this tokenId will not be able to be minted again. This will lead to

1. The supply of NFT for each tokenID is determined after the first mint
2. Unable to mint ERC1155 NFT with the same tokenId for different users

This will greatly limit the application scenarios of ERC1155NFTProduct

Impact

1. The supply of NFT for each tokenID is determined after the first mint
2. Unable to mint ERC1155 NFT with the same tokenId for different users

Code Snippet

https://github.com/sherlock-audit/2022-10-nftport/blob/main/evm-minting-master/contracts/templates/ERC1155NFTProduct.sol#L279-L286 https://github.com/sherlock-audit/2022-10-nftport/blob/main/evm-minting-master/contracts/templates/ERC1155NFTProduct.sol#L259-L265 https://github.com/sherlock-audit/2022-10-nftport/blob/main/evm-minting-master/contracts/templates/ERC1155NFTProduct.sol#L388-L390

Tool used

Manual Review

Recommendation

Consider removing the _exists check when minting ERC1155 NFT in the ERC1155NFTProduct contract

Discussion

hyperspacebunny

We've decided to not change the behaviour at this point and revisit it once we have requests from users. It's currently working as designed.

Issue M-4: Attackers can bypass `tokensPerMint` and mint lots of tokens in a transaction

Source: https://github.com/sherlock-audit/2022-10-nftport-judging/issues/85

Found by

0xSmartContract, GimelSec

Summary

Attackers can bypass `tokensPerMint`, allowing them to mint over `tokensPerMint` per transaction.

Vulnerability Detail

NFTCollection has `tokensPerMint` that enforces the maximum number of tokens the user can mint per transaction. But attackers can bypass `tokensPerMint` by reentrancy attack to call `mint()` or `presaleMint()` multiple times in a transaction.

For example, we assume that `tokensPerMint` is `5` and we have a large amount of `availableSupply()`:

1. Alice (attacker) first creates an `AttackerContract` to call `mint(5)`, then L318 will check that `amount<=_deploymentConfig.tokensPerMint` and call `_safeMint()` on L321.
2. But `_safeMint()` has a callback that it will call `onERC721Received()` if `to` address is a contract.
3. Now `_safeMint()` calls `onERC721Received()` on the `AttackerContract`, and the `AttackerContract` re-enter `mint(5)` again. `mint()` function will pass the check of `tokensPerMint` on L318 again. Finally, Alice will mint 5+5 tokens in a transaction.

Impact

Attackers can re-enter `mint()` function again and again to bypass the check of `tokensPerMint`, and mint lots of tokens in a transaction.

Code Snippet

https://github.com/sherlock-audit/2022-10-nftport/blob/main/evm-minting-master/contracts/templates/NFTCollection.sol#L317-L322

Tool used

Manual Review

Recommendation

If the protocol wants to check `to` address, keep `_safeMint()` and add ReentrancyGuard.sol on `mint()` and `presaleMint()` to prevent reentrancy attack.

Or just replace `_safeMint()` with `_mint()`.

Discussion

hyperspacebunny

I believe this only applies to free mints since `msg.value` will be 0 when called from the attacker's contract and `mint()` will revert due to the `paymentProvided` modifier.

Issue M-5: Template implementations doesn't validate configurations properly

Source: https://github.com/sherlock-audit/2022-10-nftport-judging/issues/83

Found by

rvierdiiev, ak1, ElKu, obront, GimelSec, pashov, cccz, Dravee, joestakey, JohnSmith, ctf_sec, JohnnyTime

Summary

In past audits, we have seen contract admins claim that invalidated configuration setters are fine since "admins are trustworthy". However, cases such as Nomad got drained for over and Misconfiguration in the Acala stablecoin project allows attacker to steal 1.2 billion aUSD have shown again and again that even trustable entities can make mistakes. Thus any fields that might potentially result in insolvency of protocol should be thoroughly checked.

NftPort template implementations often ignore checks for config fields. For the rest of the issue, we take `royalty` related fields as an example to illustrate potential consequences of misconfigurations. Notably, lack of check is not limited to `royalty`, but exists among most config fields.

Admins are allowed to set a wrong `royaltiesBps` which is higher than `ROYALTIES_BASIS`. `royaltyInfo()` will accept this invalid `royaltiesBps` and users will pay a large amount of royalty.

Vulnerability Detail

EIP-2981 (NFT Royalty Standard) defines `royaltyInfo()` function that specifies how much to pay for a given sale price. In general, royalty should not be higher than 100%. NFTCollection.sol checks that admins can't set royalties to more than 100%:

```
/// Validate a runtime configuration change
function _validateRuntimeConfig(RuntimeConfig calldata config)
    internal
    view
{
    // Can't set royalties to more than 100%
    require(config.royaltiesBps <= ROYALTIES_BASIS, "Royalties too high");

    ...
```

But `NFTCollection` only check `royaltiesBps` when admins call `updateConfig()`, it doesn't check `royaltiesBps` in `initialize()` function, leading to admins could set an invalid `royaltiesBps` (higher than 100%) when initializing contracts.

The same problem exists in ERC721NFTProduct and ERC1155NFTProduct. Both ERC721NFTProdu and ERC1155NFTProduct don't check `royaltiesBasisPoints` in `initialize()` function. Furthermore, these contracts also don't check `royaltiesBasisPoints` when admins call `update()` function. It means that admins could set an invalid `royaltiesBasisPoints` which may be higher than 100% in any time.

Impact

EIP-2981 only defines `royaltyInfo()` that it should return royalty amount rather than royalty percentage. It means that if the contract has an invalid royalty percentage which is higher than 100%, `royaltyInfo()` doesn't revert and users will pay a large amount of royalty.

Code Snippet

https://github.com/sherlock-audit/2022-10-nftport/blob/main/evm-minting-master/contracts/templates/NFTCollection.sol#L348 https://github.com/sherlock-audit/2022-10-nftport/blob/main/evm-minting-master/contracts/templates/NFTCollection.sol#L153 https://github.com/sherlock-audit/2022-10-nftport/blob/main/evm-minting-master/contracts/templates/ERC721NFTProduct.sol#L91 https://github.com/sherlock-audit/2022-10-nftport/blob/main/evm-minting-master/contracts/templates/ERC721NFTProduct.sol#L201 https://github.com/sherlock-audit/2022-10-nftport/blob/main/evm-minting-master/contracts/templates/ERC1155NFTProduct.sol#L96 https://github.com/sherlock-audit/2022-10-nftport/blob/main/evm-minting-master/contracts/templates/ERC1155NFTProduct.sol#L238

Tool used

Manual Review

Recommendation

Check `royaltiesBps<=ROYALTIES_BASIS` both in `initialize()` and `update()` functions.

Issue M-6: registerTemplate() can't handle properly when ITemplate version is 0

Source: https://github.com/sherlock-audit/2022-10-nftport-judging/issues/80

Found by

bin2chen

Summary

Factory.sol when register one template , and template ' s version is 0, the latestImplementation[templateName] will be address(0) and add other version, "_templateNames" will duplicate

Vulnerability Detail

When version is equal 0 latestImplementation[templateName] don't set

```
    function _setTemplate(
         string memory templateName,
         uint256 templateVersion,
         address implementationAddress
     ) internal {
...

         if (latestImplementation[templateName] == address(0)) { /****add other
↪    version, _templateNames will duplicate ****/
             _templateNames.push(templateName);
         }

         if (templateVersion > latestVersion[templateName]) {
             latestVersion[templateName] = templateVersion;
             latestImplementation[templateName] = implementationAddress;
↪    /****templateVersion==0 ,  don't set ****/
         }

     }
```

Impact

latestImplementation[templateName] and _templateNames will error. external contracts may think there is no setup, resulting in duplicate setups that keep failing

Code Snippet

https://github.com/sherlock-audit/2022-10-nftport/blob/main/evm-minting-master/contracts/Factory.sol#L415

Tool used

Manual Review

Recommendation

```
   function _setTemplate(
       string memory templateName,
       uint256 templateVersion,
       address implementationAddress
   ) internal {

-      if (templateVersion > latestVersion[templateName]) {
+      if (templateVersion > = latestVersion[templateName]) {
           latestVersion[templateName] = templateVersion;
           latestImplementation[templateName] = implementationAddress;
       }
```

Issue M-7: _validateDeploymentConfig function in NFTCollection.sol doesn't check all conditions

Source: https://github.com/sherlock-audit/2022-10-nftport-judging/issues/51

Found by

ElKu

Summary

Due to not thoroughly checking all conditions of the deployment config, users might not be able to mint certain amount of tokens.

Vulnerability Detail

In `NFTCollection.sol`, there is a function called _validateDeploymentConfig, which checks whether the `deploymentConfig` input to initialize function is valid.

The function checks whether the `maxSupply` and `tokensPerMint` are greater than zero. But it never checks if the `tokensPerMint` is less than or equal to `maxSupply`.

Suppose `maxSupply` < `tokensPerMint`. Then if the user calls a function which in-turn calls the _mintTokens function with an `amount` equal to `tokensPerMint`, it will revert. Even though he is technically trying to mint as per the `rules`. This is because the `availableSupply()` won't be greater than `amount`.

Impact

1. Dissatisfaction and Frustration of the user along with loss of his gas fees.
2. The contract would need redeployment as deployment config cannot be changed once its set.

Code Snippet

```
function _mintTokens(address to, uint256 amount) internal {
    require(amount <= _deploymentConfig.tokensPerMint, "Amount too large");
    require(amount <= availableSupply(), "Not enough tokens left");

    _safeMint(to, amount);
}
```

Tool used

Manual Review, VSCode

Recommendation

Add a require statement in the `_validateDeploymentConfig` function:

```
require(config.tokensPerMint <= config.maxSupply, "Tokens per mint must be lte
↪  Maximum supply");
```

Issue M-8: Factory uses signature that do not have expiration

Source: https://github.com/sherlock-audit/2022-10-nftport-judging/issues/46

Found by

rvierdiiev

Summary

NftPort can't remove license from user, once the signature was provided to it, without changing `SIGNER_ROLE` address.

Vulnerability Detail

In Factory contract there are few methods that are called when signed by trusted signer.

This is how the signature is checked

```
signedOnly(abi.encodePacked(msg.sender, instance, data), signature)
```

As you can see there is no any expiration time. That means that once, the signer has signed the signature for the user it can use it for the end of life. It's like lifetime license. The only option to remove the license from user is to revoke `SIGNER_ROLE` and set it to another account. But it's possible that the NFTPort will have a need to do that with current signer.

Impact

License can't be removed.

Code Snippet

https://github.com/sherlock-audit/2022-10-nftport/blob/main/evm-minting-master/contracts/Factory.sol#L222

Tool used

Manual Review

Recommendation

Add expiration param to the signature.

Issue M-9: Missing check for equal length arrays in transferByOwnerBatch and mintByOwnerBatch

Source: https://github.com/sherlock-audit/2022-10-nftport-judging/issues/33

Found by

obront

Summary

The `transferByOwnerBatch()` and `mintByOwnerBatch()` functions in the ERC1155 implementation does not check whether the lengths of the arrays submitted are equal. This can lead to unexpected results.

Vulnerability Detail

In the `transferByOwnerBatch()` function, the user submits three arrays (addresses, ids, and amounts), while in the `mintByOwnerBatch()` function, the user submits four arrays (addresses, ids, amounts, uris).

The expectation is that the user submitting the function will ensure that the indexes of the arrays correspond to the correct values in the other arrays, and thus that the lengths will be the same.

Common practice in such a situation is to verify that the lengths are equal to ensure the user hasn't made an error. In other functions like `burnBatch()`, this verification is done by the underlying ERC1155 function.

However, in these two functions, we simply iterate through the `ids` array without performing this check, and then call out to the underlying ERC1155 `_safeTransferFrom()` or `_mint()` functions for each id separately.

Impact

If the `ids` array is a shorter length than the other arrays, the additional values in the other arrays will be ignored. This could lead to transfers with unexpected results, which would be better served by reverting.

Code Snippet

https://github.com/sherlock-audit/2022-10-nftport/blob/main/evm-minting-master/contracts/templates/ERC1155NFTProduct.sol#L206-L215

https://github.com/sherlock-audit/2022-10-nftport/blob/main/evm-minting-master/contracts/templates/ERC1155NFTProduct.sol#L279-L299

Tool used

Manual Review

Recommendation

Add a check to the `transferByOwnerBatch()` function that confirms that to, ids, and amounts are all equal length.

```
require(ids.length == to.length, "mismatched array lengths");
require(ids.length == amounts.length, "mismatched array lengths");
```

Add a check to the `mintByOwnerBatch()` function that confirms that to, ids, amounts, and uris are all equal length.

```
require(ids.length == to.length, "mismatched array lengths");
require(ids.length == amounts.length, "mismatched array lengths");
require(ids.length == uris.length, "mismatched array lengths");
```