Quantstamp

NiftyApes - Seller Financing

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	NFT Marketplace
Timeline	2023-04-20 through 2023-04-20
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Source Code	• NiftyApes/sellerFinancing #338e829
Auditors	• Ibrahim Abouzied Auditing Engineer • Jonathan Mevs Auditing Engineer • Michael Boyle Auditing Engineer • Hytham Farah Auditing Engineer

Documentation quality	High
Test quality	High
Total Findings	20 Fixed: 11 Acknowledged: 8 Mitigated: 1
High severity findings ⓘ	4 Fixed: 4
Medium severity findings ⓘ	1 Fixed: 1
Low severity findings ⓘ	7 Fixed: 4 Acknowledged: 2 Mitigated: 1
Undetermined severity findings ⓘ	0
Informational findings ⓘ	8 Fixed: 2 Acknowledged: 6

Summary of Findings

NiftyApes Seller Financing is a protocol that allows sellers to offer financing options to purchase their NFTs. Offers are signed by sellers off-chain for interested buyers to execute on-chain. During a loan, the `NiftyApesSellerFinancing` contract holds the NFT in escrow and delegates it to the buyer through `delegate.cash`. Both the seller and the buyer are given ticket vouchers representing their end of the loan, which they are free to transfer to other addresses. A seller can seize back their NFT if the loan has defaulted, whilst optionally providing a grace period for late payments. The contract includes integration with Seaport, allowing buyers to sell the loaned NFT to Seaport bids that cover the remainder of their loan. The protocol prohibits interactions by addresses on the OFAC sanctions list.

Over the course of the audit, we prioritized validating that the NFTs and their loan ticket vouchers maintained proper access control. In addition to standard security practices, we also looked for vulnerabilities surrounding faulty accounting, drifts in NFT delegations, and the potential for locked funds with Seaport integration or sanctioned users.

We were able to uncover a few important issues. `delegate.cash` integration for collection offers breaks when the buyer ticket is transferred (NFTY-1). The current Seaport integration results in locked funds for some bids and is unable to complete others (NFTY-2 & NFTY-5). The contract is also missing some access control for prohibiting sanctioned users (NFTY-3 & NFTY-4).

The protocol has a strong testing suite and thorough documentation. The NiftyApes team was able to answer all of our questions throughout the course of the audit.

Fix Review: The NiftyApes team addressed all of the issues found in the audit. Many of the informational severity issues served to highlight the contract's behavior rather than point to security concerns and were addressed by updating the documentation. We encourage users to thoroughly read through the project's documentation before interacting with the contract.

ID	DESCRIPTION	SEVERITY	STATUS
NFTY-1	Underlying NFTs are Assigned Incorrect `nftId` for Collections	• High ⓘ	Fixed
NFTY-2	Locked Funds when Fulfilling Seaport Orders	• High ⓘ	Fixed
NFTY-3	Sanctioned Users Can Transfer Their Tickets	• High ⓘ	Fixed

ID	DESCRIPTION	SEVERITY	STATUS
NFTY-4	Funds May Be Sent to Sanctioned Address	High	Fixed
NFTY-5	Unable to Fulfill Seaport Bids with Multiple Considerations	Medium	Fixed
NFTY-6	Signatures Cannot Be Revoked when the Contract Is Paused	Low	Fixed
NFTY-7	Users Can Buy An NFT that They Cannot Receive	Low	Fixed
NFTY-8	Privileged Roles and Ownership	Low	Fixed
NFTY-9	Application Monitoring Can Be Improved by Emitting More Events	Low	Acknowledged
NFTY-10	Anyone Can Call Initialize	Low	Fixed
NFTY-11	Missing Input Validation	Low	Mitigated
NFTY-12	Loss of Precision Due to Division before Multiplication	Low	Acknowledged
NFTY-13	Expiration Date for Protocol Support	Informational	Acknowledged
NFTY-14	`withdrawOfferSignature()` Vulnerable to Front-Running	Informational	Acknowledged
NFTY-15	It Is Possible to Transfer Buyer Tickets for Defaulted Loans	Informational	Acknowledged
NFTY-16	A Buyer May Be Able to Make Free Payments	Informational	Acknowledged
NFTY-17	Seller Cannot Make Same Offer Twice	Informational	Acknowledged
NFTY-18	Missing Call to Parent Contract	Informational	Fixed
NFTY-19	Incorrect Version of Solidity	Informational	Fixed
NFTY-20	Excessive Ether Funds are Refunded to the `buyer` Rather than `msg.sender`	Informational	Acknowledged

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Disclaimer
Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power

- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

Files Included

- `src/SellerFinancing.sol`
- `src/marketplaceIntegration/MarketplaceIntegration.sol`

Findings

NFTY-1 Underlying NFTs are Assigned Incorrect `nftId` for Collections

● High ⓘ Fixed

Update
Marked as "Fixed" by the client. Addressed in: `cf0c09a0c59961622a93ea527023538e91df8358` . The client provided the following explanation: Specify nftId for underlying nft in `_createLoan()` .

File(s) affected: `SellerFinancing.sol`

Description: `_createLoan()` incorrectly assigns the `nftId` of both the `buyerUnderlyingNFT` and `sellerUnderlyingNFT` when financing collection offers. Collection offers have the `offer.nftId` value assigned as `~uint256(0)` , which is used to denote that the offer exists for the entire collection instead of an individual NFT. When buying from a collection offer, the buyer specifies the individual NFT they are purchasing in `nftId` field in `buyWithFinancing()` . However, the individual `nftId` is not assigned to the buyer and seller's underlying NFT, rather the `offer.nftId` is, which would be the maximum value for a `uint256` .

This incorrect assignment would break the delegate.cash functionality if a buyer transfers their ticket, as delegate.cash would be attempting to delegate an NFT that does not exist.

Recommendation: Ensure that the underlying NFT is assigned the `nftId` of the NFT that is being purchased.

NFTY-2 Locked Funds when Fulfilling Seaport Orders

● High ⓘ Fixed

Update
Marked as "Fixed" by the client. Addressed in: `9640068555ab08295853e53a1bf1e4d47e21227a` . The client provided the following explanation: Add seaport order validation for only one `order.offer` item.

File(s) affected: `SellerFinancing.sol`

Description: Any offerings that are not WETH are locked forever after fulfilling a Seaport order. `_validateSaleOrder()` does not ensure that the length of `order.parameters.offer[]` is 1, meaning that there could be offers accepted that receive tokens outside of WETH. Since only WETH is transferred to the buyer, any additional funds would be locked in the contract without a method to withdraw them.

Recommendation: If offers are to be limited to WETH, ensure that the length of `order.parameters.offer` is 1 as an additional check in `_validateSaleOrder()` .

NFTY-3 Sanctioned Users Can Transfer Their Tickets

● High ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `b4ba7964209c99e878fdb10b52a9bdd5713a8f86`. The client provided the following explanation: Add `_requireIsNotSanctioned(msg.sender)` to `_transfer()`.

File(s) affected: `SellerFinancing.sol`

Description: Sanctioned users are allowed to transfer their buyer and seller tickets, as there is no check in `_transfer` to ensure the sending party is not sanctioned. A sanctioned user can easily transfer their ticket to a secondary address.

Recommendation: Ensure sanctioned users cannot transfer buyer and seller tickets. Consider overriding `_beforeTokenTransfer()` rather than the `_transfer()` method.

NFTY-4 Funds May Be Sent to Sanctioned Address

● High ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `9594cb328b26a37c15582d737e3e06ef452b6eec`. The client provided the following explanation: Add return of funds to buyer if seller is sanctioned.

File(s) affected: `SellerFinancing.sol`

Description: `_makePayment()` currently allows payments to a sanctioned seller. If a buyer refuses (or is under legal obligation not) to send funds to the sanctioned seller, once enough time has passed, the sanctioned seller can transfer the ticket to a non-sanctioned address and seize the NFT. If a sanctioned seller does this near the end of the payment plan, then they will have effectively stolen all the funds from the buyer.

Recommendation: Update `_conditionalSendValue()` to refund the buyer when the `to` address is sanctioned. This will allow them to close out their loan without paying any more funds (aside from potential royalties). Resolving NFTY-3 can help mitigate this issue.

NFTY-5 Unable to Fulfill Seaport Bids with Multiple Considerations

● Medium ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `90bad0b68bf4c3266c7dc78dd7cc354cea21e571`. The client provided the following explanation: Add += to totalConsiderationAmount calculation for more dynamic payout scenarios.

File(s) affected: `SellerFinancing.sol`

Description: In `sellAsset()`, the total consideration is calculated as follows:

```
uint256 totalConsiderationAmount;
for (uint256 i = 1; i < order.parameters.totalOriginalConsiderationItems; i++) {
    totalConsiderationAmount = order.parameters.consideration[i].endAmount;
}
```

This is equivalent to setting the total consideration to the final element of `order.parameters.consideration[]` rather than the sum of the array. The contract will be unable to fulfill any Seaport orders that have multiple WETH considerations because it hasn't approved enough WETH to send to the Seaport contract.

Recommendation: When calculating the `totalConsiderationAmount` do `+=` instead of `=`.

NFTY-6 Signatures Cannot Be Revoked when the Contract Is Paused

● Low ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `510afbe1453cd5283a76f6d72f920adc491ef11d`. The client provided the following explanation: `whenNotPaused` modifier removed from `withdrawOfferSignature()`.

File(s) affected: `SellerFinancing.sol`

Description: The `withdrawOfferSignature()` is restricted by the `whenNotPaused` modifier. Sellers should have the freedom to revoke their signatures at any time.

Recommendation: Remove the `whenNotPaused` modifier from the `withdrawOfferSignature()` function.

NFTY-7 Users Can Buy An NFT that They Cannot Receive

• Low ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `09aa45320d87df560c0a47848c0c24cdbcdeca34`. The client provided the following explanation: Add protocol warnings section and Making payments and Receiving NFTs as a Contract to README. This finding has been addressed via documentation.

File(s) affected: `SellerFinancing.sol`

Description: If a contract that has not correctly implemented the `onERC721Receive()` function acquires a buyer ticket, then they will not be able to receive the NFT at the end of their payment plan. Each time they go to make the final payment, the `makePayment()` function will revert when calling `_transferNft()`. At some point the loan will expire and the seller will be able to seize the NFT.

Despite this, the buyer will be allowed to make payments and may not be aware of this issue until the end of their payment plan.

This could happen in two scenarios:

1. The buyer ticket has been transferred to the contract without calling `safeTransferFrom()`.
2. A contract calls `buyWithFinancing()` from its constructor. This is possible because the `isContract()` in `safeMint()` will return `false` and the check will be skipped.

A clever seller may trick a buyer into purchasing an NFT in this way as they stand to benefit greatly.

Recommendation: Ensure that a buyer is aware of this possibility. Perhaps include a check that the buyer can receive ERC721 tokens after the first payment is made to catch the problem early.

NFTY-8 Privileged Roles and Ownership

• Low ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `f3ff981ec6e74831e352c4ed9c1e3a0008294207`. The client provided the following explanation: Add Privileged Roles and Ownership section to readme. This finding has been addressed via documentation.

File(s) affected: `SellerFinancing.sol`, `MarketplaceIntegration.sol`

Description: All privileged permissions should be clearly documented for users. The owner has the ability to arbitrarily set `royaltiesEngineContractAddress`. A custom version of this contract could redirect 100% of payments made to the owner.

Additionally, these functions can all be invoked by the owner and modify core state addresses of `SellerFinancing`:

- `updateRoyaltiesEngineContractAddress()`
- `updateDelegateRegistryContractAddress()`
- `updateSeaportContractAddress()`
- `updateWethContractAddress()`
- `pause()`, `unpause()`
- `pauseSanctions()`, `unpauseSanctions()`

As well as these functions in `MarketplaceIntegration`:

- `updateSellerFinancingContractAddress()`
- `updateMarketplaceFeeRecipient()`
- `updateMarketplaceFeeBps()`
- `pause()`, `unpause()`
- `pauseSanctions()`, `unpauseSanctions()`

Exploit Scenario:

1. Owner calls `updateRoyaltiesEngineContractAddress()` with an address to a custom implementation of `IRoyalyEngineV1`.
2. Subsequent calls to `getRoyaltyView()` for any NFT could return an address owned by the owner.
3. Owner and/or their addresses are paid a portion or all of each payment made using `makePayment()`.

Recommendation: Ensure that these roles and privileges are clearly documented to the user.

NFTY-9 Application Monitoring Can Be Improved by Emitting More Events

• Low ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation: We have chosen not to address QSP-9 as a way to reduce compile time contract size and as these events would only be emitted on functions only callable by `owner()` we find them to be of lower impact.

File(s) affected: `SellerFinancing.sol` , `MarketPlaceIntegration.sol`

Description: In order to validate the proper deployment and initialization of the contracts, it is a good practice to emit events. Also, any important state transitions can be logged, which is beneficial for monitoring the contract, and also tracking eventual bugs or hacks. Below we present a non-exhaustive list of events that could be emitted to improve application management:

1. Contract address changes:
    - `SellerFinancing` :
        - `updateRoyaltiesEngineContractAddress()`
        - `updateDelegateRegistryContractAddress()`
        - `updateSeaportContractAddress()`
        - `updateWethContractAddress()`
    - `MarketplaceIntegration` :
        - `updateSellerFinancingContractAddress()`
        - `updateMarketplaceFeeRecipient()`
2. Pausing or unpausing of features:
    - `SellerFinancing` and `MarketplaceIntegration` :
        - `pause()`
        - `unpause()`
        - `pauseSanctions()`
        - `unpauseSanctions()`
3. Changes to the fee structure:
    - `MarketPlaceIntegration` :
        - `updateMarketplaceFeeBps()`

Recommendation: Consider emitting the events.

NFTY-10 Anyone Can Call Initialize

• Low ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `d9de3991673eb3245d83f585c587d538dd651ad4` . The client provided the following explanation: Add empty constructor with initializer.

File(s) affected: `SellerFinancing.sol`

Description: `SellerFinancing` is intended to be the implementation contract for an upgradeable proxy, as it contains an `initialize()` function. Currently, anyone can call the `initialize()` method and write state variables to this contract. This could lead to an attacker taking control of the implementation contract.

Recommendation: Add a constructor with an `initializer` modifier.

NFTY-11 Missing Input Validation

• Low ⓘ Mitigated

Update

Marked as "Fixed" by the client. Addressed in: `5ba9e24e8cb504b66e040e7a4d10858f53c83ac7` . The client provided the following explanation: There are multiple sub points in NFTY-11.

1. Since these validations would only be called once on a function that would only be called behind an owned proxy we have chosen to save compile time contract size by not adding validation to these values.
2. We do not enforce a maximum basis points of interest so have renamed MAX_BPS to BASE_BPS.
3. This point appears to be a misunderstanding by the auditing team. There are conduits provided on either side of a seaport order, one is from the offer creater encoded in the order struct and which can be anything with no validation possible or needed. The second is from the offer accepter, i.e. NiftyApes, and we have hardcoded it to bytes32(0) which is mapped to the seaport address. So, no input validation is required for this point.
4. We do not enforce a maximum basis points of interest so have renamed MAX_BPS to BASE_BPS.

File(s) affected: `SellerFinancing.sol` , `MarketplaceIntegration.sol`

Related Issue(s): SWC-123

Description: It is important to validate inputs, even if they only come from trusted addresses, to avoid human error. The following is a non-exhaustive list of places in the code that could benefit from extra input validation.

- `SellerFinancing.initialize()`:
  - Validate `royaltiesEngineContractAddress` is not equal to the zero address.
  - Validate `delegateRegistryContractAddress` is not equal to the zero address.
  - Validate `seaportContractAddress` is not equal to the zero address.
  - Validate `wethContractAddress` is not equal to the zero address.
- `SellerFinancing.buyWithFinancing()`: Validate that the `offer.periodInterestRateBps` does not exceed `MAX_BPS`.
- `SellerFinancing._validateSaleOrder()`: Validate that the order doesn't use a conduit.
- `MarketPlaceIntegration.updateMarketplaceFeeBps()`: Validate that the `newMarketplaceFeeBps` does not exceed `MAX_BPS`, or 100%.

Recommendation: We recommend adding the relevant checks.

NFTY-12 Loss of Precision Due to Division before Multiplication

• Low ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation: The multiplication after division is intended in this case. We calculate the period interest in its floor value (and lose the precision there) and then we multiply it to number of periods passed to make sure that the final number is always the multiple of one period interest.

File(s) affected: `SellerFinancing.sol`

Description: Division before multiplication may result in a loss of precision when the operations are carried over integer numbers. `SellerFinancing.calculateMinimumPayment()` has a multiplication done after division when calculating `periodInterest`.

Recommendation: Rewrite equations so that division happens after multiplication.

NFTY-13 Expiration Date for Protocol Support

• Informational ⓘ Acknowledged

Update

Marked as "Fixed" by the client. Addressed in: `c05dfbcb8844f6a5a8cc7a41e31445c5a6b040b8`. The client provided the following explanation: Add Limited Protocol and Loan Time Horizon Section to README. This finding has been addressed via documentation.

File(s) affected: `SellerFinancing.sol`

Description: This implementation casts all timestamps to `uint32`, which will support UNIX timestamps up until January 19, 2038 at 3:14:07 UTC. After that, this protocol will no longer support these timestamps.

Recommendation: Consider the implications of this and modify the contracts accordingly.

NFTY-14 `withdrawOfferSignature()` Vulnerable to Front-Running

• Informational ⓘ Acknowledged

Update

Marked as "Fixed" by the client. Addressed in: `03c6cabf98dc399effed4210d539f8d6ec5843cb`. The client provided the following explanation: Add `withdrawOfferSignature()` Vulnerable to Front-Running to README. This finding has been addressed via documentation.

File(s) affected: `SellerFinancing.sol`

Description: `withdrawOfferSignature()` is vulnerable to front-running. If a seller submits a transaction calling `withdrawOfferSignature()` to the mempool on an undesirable existing offer, a savvy witnesser could see this and buy the offer prior to the cancellation. This can result in sellers being forced to finance NFTs on Offers that they intend to keep or are no longer desirable due to market conditions.

Recommendation: Clearly document this behavior to users.

NFTY-15 It Is Possible to Transfer Buyer Tickets for Defaulted Loans

• Informational ⓘ Acknowledged

Update
Marked as "Fixed" by the client. Addressed in: `631a59de5016c57831d92383aad827736f2b1a15`. The client provided the following explanation: Add Purchase of Defaulted Buyer Tickets to readme. This finding has been addressed via documentation.

File(s) affected: `SellerFinancing.sol`

Description: Ticket transfers are still possible after a loan has defaulted. Buyer tickets for defaulted loans hold no value as their NFT's seized at any time.

Recommendation: Users should be weary of purchasing Buyer tickets that correspond to defaulted loans. Consider disabling transfers of Buyer tickets for defaulted/hard-defaulted loans.

NFTY-16 A Buyer May Be Able to Make Free Payments

● Informational ⓘ Acknowledged

Update
Marked as "Fixed" by the client. Addressed in: `27c78f19475de50475788fc87585c0ddecc892a5`. The client provided the following explanation: Add Accepting Payments As A Contract Section to the Readme. This finding has been addressed via documentation.

File(s) affected: `SellerFinancing.sol`

Description: Payouts to the seller in `_makePayment()` are done through a call to `_conditionalSendValue()`, which refunds the buyer if the seller cannot receive ETH payments. In this scenario, `_makePayments()` considers the principal and interest payments fulfilled regardless of whether the seller received any compensation.

A buyer could exploit this situation and continuously send payments (which will be sent back to them) until they have completed the payment plan and essentially receive the NFT cost-free.

Recommendation: For cases like this, it is generally recommended to implement a withdrawal pattern, however, as the comment above `_conditionalSendValue()` states, "we do not want ETH hanging in contract". In this case, it is recommended to make another vault contract that will hold the funds until the seller is able to retrieve them.

If there is a principled reason for not wanting to hold ether in any capacity, then these risks must be disclosed to the seller in public-facing documentation.

NFTY-17 Seller Cannot Make Same Offer Twice

● Informational ⓘ Acknowledged

Update
Marked as "Fixed" by the client. Addressed in: `5a0b90d7059321eb18a76e8df3fdf63f42120a35`. The client provided the following explanation: Add Seller cannot make the exact same offer twice to readme. This finding has been addressed via documentation.

File(s) affected: `SellerFinancing.sol`

Description: If an offer's signature gets used once or has been previously canceled, the seller will be unable to make the same offer again, even if it is intentional.

The impact is low since they can make a minor change (e.g. increase `expiration` by a single second) and have it be different.

Recommendation: Include a nonce in the offer, so the same offer can be made twice with different signatures.

NFTY-18 Missing Call to Parent Contract

● Informational ⓘ Fixed

Update
Marked as "Fixed" by the client. Addressed in: `57b296fc11cafca50a88b65fa71a3055c66b5804`. The client provided the following explanation: Add call to __ERC721URIStorage_init() in initializer.

File(s) affected: `SellerFinancing.sol`

Description: `__ERC721URIStorageUpgradeable_init()` is not called on construction. Although it does nothing in this version of the Open Zeppelin contracts, it is still a best practice to include this to support future upgradeability.

Recommendation: Include the call to `__ERC721URIStorageUpgradeable_init()`.

NFTY-19 Incorrect Version of Solidity

● Informational ⓘ Fixed

Update
Marked as "Fixed" by the client. Addressed in: `186f6c1645b9700d2cf0f60abee4bf45a1852742` . The client provided the following explanation: Solidity version changed from 0.8.13 to 0.8.18.

File(s) affected: `SellerFinancing.sol` , `MarketplaceIntegration.sol`

Description: The contracts currently use Solidity version 0.8.13. This version is not on the list of recommended versions.

Recommendation: Consider using Solidity version 0.8.18 instead and refer to the list of recommended versions for up-to-date suggestions.

NFTY-20 Excessive Ether Funds are Refunded to the `buyer` Rather than `msg.sender`

● Informational ⓘ Acknowledged

Update
Marked as "Fixed" by the client. Addressed in: `ebf209a4d61287f7927cc21b23026f216e23ec1d` . The client provided the following explanation: Add Excessive Ether Funds are Refunded to the Buyer Rather than `msg.sender` to README. This finding has been addressed via documentation.

File(s) affected: `SellerFinancing.sol`

Description: In `buyWithFinancing()` , if `msg.value` exceeds `offer.downPaymentAmount()` , the refund is issued to the `buyer` rather than the `msg.sender` . According to the NiftyApes team, this is done so that buyers receive their refund for purchases made through the `MarketplaceIntegration` contract, which collects a fee before making a call to `SellerFinancing.buyWithFinancing()` .

This use case works because the `MarketplaceIntegration` contract acts as an intermediary contract between the user and `SellerFinancing` , and passes along the full `msg.value` . Users interacting directly with the `SellerFinancing` contract under a different paradigm should be aware that excessive payments are sent directly to the `buyer` . For example, if a different user makes an excess payment on behalf of the `buyer` , the excess funds are sent to the `buyer` rather than user.

Recommendation: Make this behavior clear to users in public-facing documentation.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Adherence to Best Practices

1. The event `OfferSignatureUsed()` is used a single time with the following parameters: `OfferSignatureUsed(offer.nftContractAddress, offer.nftId, offer, signature)` . The first two arguments are contained in the third and are therefore redundant.
2. Sometimes the internal function `_requireNonZeroAddress()` is used to validate against the zero address and other times the validation is written out explicitly. For consistency, ensure all validations are done using the internal function. Furthermore, consider making this function a modifier.

3. Consider turning validation functions whose parameters are known before the function is called into modifiers. For example, `_requireNonZeroAddress()`.
4. There are several functions that are used in both the `MarketplaceIntegration` and `SellerFinancing` contracts. Consider making a single library that consists of these common functions and letting both contracts inherit from this library. This may avoid a scenario where one function is updated or changed, but the other one is forgotten.
5. Consider allowing a seller to immediately seize an asset from a sanctioned buyer rather than waiting for a default.
6. Function parameters that are not manipulated in the function can be marked as `calldata` to save gas. E.g., `offer` in `buyWithFinancing()` is not modified and can be declared as `calldata`.
7. Avoid shadowing state variables with memory variables or function parameters. `owner` in `_require721Owner()` shadows the variable for the owner of the contract.

Adherence to Specification

In `SellerFinancing.buyWithFinancing()#L256` there is a comment stating `requireMinimumPrincipalLessThanTotalPrincipal`. However, the check below passes if the minimum principle is less than or equal to the total principal. Change the check name to `requireMinimumPrincipalLessThanOrEqualToTotalPrincipal`.

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

- `c60...ef8 ./src/SellerFinancing.sol`
- `577...b4b ./src/interfaces/Ownership.sol`
- `2cc...ca8 ./src/interfaces/delegateCash/IDelegationRegistry.sol`
- `4e9...ad3 ./src/interfaces/royaltyRegistry/IRoyaltyEngineV1.sol`
- `929...804 ./src/interfaces/sanctions/SanctionsList.sol`
- `f95...b93 ./src/interfaces/sellerFinancing/ISellerFinancingStructs.sol`
- `6f8...b28 ./src/interfaces/sellerFinancing/ISellerFinancingErrors.sol`
- `91c...f3d ./src/interfaces/sellerFinancing/ISellerFinancing.sol`
- `e7b...4be ./src/interfaces/sellerFinancing/ISellerFinancingEvents.sol`
- `7f0...f8b ./src/interfaces/sellerFinancing/ISellerFinancingAdmin.sol`
- `aaf...ea1 ./src/interfaces/seaport/ISeaport.sol`
- `76c...253 ./src/lib/ECDSABridge.sol`
- `065...b3c ./src/marketplaceIntegration/MarketplaceIntegration.sol`

Tests

- `7d7...739 ./test/common/BaseTest.sol`
- `1c2...7e6 ./test/common/Hevm.sol`
- `960...27c ./test/common/Console.sol`
- `d9a...72f ./test/utils/fixtures/NFTFixtures.sol`
- `f68...307 ./test/utils/fixtures/OffersLoansFixtures.sol`
- `b24...225 ./test/utils/fixtures/SellerFinancingDeployment.sol`
- `c6c...1f4 ./test/utils/fixtures/UsersFixtures.sol`
- `705...cc9 ./test/unit/pauseSanctions.t.sol`
- `083...76f ./test/unit/seizeAsset.t.sol`
- `5b8...e0c ./test/unit/makePayment.t.sol`
- `1e8...f1c ./test/unit/updateDelegateRegistryContractAddress.t.sol`
- `0fe...bdf ./test/unit/getOfferHash.t.sol`
- `7b3...5a8 ./test/unit/safeTransferFrom.t.sol`
- `af6...c8f ./test/unit/pause.t.sol`
- `d33...81b ./test/unit/withdrawOfferSignature.sol`
- `e7b...5ba ./test/unit/getOfferSignatureStatus.t.sol`
- `c04...e4c ./test/unit/instantSell.t.sol`

- `ffb...152 ./test/unit/unpauseSanctions.t.sol`
- `cca...9dc ./test/unit/initialize.t.sol`
- `d63...b18 ./test/unit/unpause.t.sol`
- `7ee...4ff ./test/unit/buyWithFinancing.t.sol`
- `627...8cf ./test/unit/getOfferSigner.t.sol`
- `e6a...6be ./test/unit/updateRoyaltiesEngineContractAddress.t.sol`
- `0ad...f5a ./test/unit/getUnderlyingNft.t.sol`
- `6cb...9ef ./test/unit/updateSeaportContractAddress.t.sol`
- `01e...3fe ./test/unit/updateWethContractAddress.t.sol`
- `672...65a ./test/unit/marketplaceIntegration/pauseSanctions.t.sol`
- `a86...f2d ./test/unit/marketplaceIntegration/pause.t.sol`
- `78e...e07 ./test/unit/marketplaceIntegration/unpauseSanctions.t.sol`
- `24d...5c9 ./test/unit/marketplaceIntegration/updateMarketplaceFeeRecipient.sol`
- `53c...7d9 ./test/unit/marketplaceIntegration/unpause.t.sol`
- `737...82c ./test/unit/marketplaceIntegration/buyWithFinancingSuperrare.sol`
- `3d8...20e ./test/unit/marketplaceIntegration/updateSellerFinancingContractAddress.sol`
- `7d1...a4f ./test/unit/marketplaceIntegration/updateMarketplaceFeeBps.sol`

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.9.3

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither src/SellerFinancing.sol --solc-remaps @openzeppelin/=lib/openzeppelin-contracts-upgradeable/ --checklist --exclude-informational --exclude-low --exclude-dependencies --exclude-optimization > slither.md`

Automated Analysis

Slither

Slither analyzed 33 contracts with 71 detectors, and 209 results were found. Nearly all were false positives, but one instance of the loss of precision due to a division before a multiplication was found and included in the report.

Test Suite Results

The test suite was run by calling `forge test --optimize --fork-url https://eth-mainnet.g.alchemy.com/v2/jxUUn2DsYODlc68SEU_7eNGCn2hQ7b63`.

```
Running 1 test for
test/unit/marketplaceIntegration/updateMarketplaceFeeBps.sol:TestUpdateMarketplaceFeeBps
[PASS] test_unit_marketPlace_updateMarketplaceFeeBps_nonZeroAddress() (gas: 18459)
Test result: ok. 1 passed; 0 failed; finished in 1.04s

Running 1 test for test/unit/initialize.t.sol:TestSellerFinancingInitialize
[PASS] test_unit_SellerFinancing_initialize() (gas: 820380)
Test result: ok. 1 passed; 0 failed; finished in 1.04s

Running 2 tests for test/unit/updateSeaportContractAddress.t.sol:TestUpdateSeaportContractAddress
[PASS] test_unit_updateSeaportContractAddress_nonZeroAddress() (gas: 27111)
[PASS] test_unit_updateSeaportContractAddress_reverts_if_zeroAddress() (gas: 20466)
Test result: ok. 2 passed; 0 failed; finished in 1.04s

Running 1 test for test/unit/getOfferSigner.t.sol:TestGetOfferSigner
[PASS] test_unit_getOfferSigner() (gas: 41620)
```

```
Test result: ok. 1 passed; 0 failed; finished in 1.04s

Running 3 tests for test/unit/getOfferSignatureStatus.t.sol:TestGetOfferSignatureStatus
[PASS] test_unit_getOfferSignature_returnsFalse_whenNotWithdrawnOrUsed() (gas: 32967)
[PASS] test_unit_getOfferSignature_returnsTrue_whenUsed() (gas: 887725)
[PASS] test_unit_getOfferSignature_returnsTrue_whenWithdrawn() (gas: 77341)
Test result: ok. 3 passed; 0 failed; finished in 1.06s

Running 2 tests for
test/unit/marketplaceIntegration/updateSellerFinancingContractAddress.sol:TestUpdateSellerFinancingContra
ctAddress
[PASS] test_unit_marketPlace_updateSellerFinancingContractAddress_nonZeroAddress() (gas: 18911)
[PASS] test_unit_marketPlace_updateSellerFinancingContractAddress_reverts_if_zeroAddress() (gas: 13172)
Test result: ok. 2 passed; 0 failed; finished in 8.67ms

Running 2 tests for
test/unit/marketplaceIntegration/updateMarketplaceFeeRecipient.sol:TestUpdateMarketplaceFeeRecipient
[PASS] test_unit_marketPlace_updateMarketplaceFeeRecipient_nonZeroAddress() (gas: 18912)
[PASS] test_unit_marketPlace_updateMarketplaceFeeRecipient_reverts_if_zeroAddress() (gas: 13216)
Test result: ok. 2 passed; 0 failed; finished in 9.34ms

Running 2 tests for
test/unit/updateRoyaltiesEngineContractAddress.t.sol:TestUpdateRoyaltiesEngineContractAddress
[PASS] test_unit_updateRoyaltiesEngineContractAddress_nonZeroAddress() (gas: 27035)
[PASS] test_unit_updateRoyaltiesEngineContractAddress_reverts_if_zeroAddress() (gas: 20402)
Test result: ok. 2 passed; 0 failed; finished in 9.53ms

Running 3 tests for test/unit/safeTransferFrom.t.sol:TestSafeTransferFrom
[PASS] test_unit_safeTranferFromData_updates_delagates_for_buyer_ticket() (gas: 978899)
[PASS] test_unit_safeTranferFrom_updates_delagates_for_buyer_ticket() (gas: 978348)
[PASS] test_unit_tranferFrom_updates_delagates_for_buyer_ticket() (gas: 976054)
Test result: ok. 3 passed; 0 failed; finished in 1.07s

Running 2 tests for test/unit/updateWethContractAddress.t.sol:TestUpdateWethContractAddress
[PASS] test_unit_updateWethContractAddress_nonZeroAddress() (gas: 27067)
[PASS] test_unit_updateWethContractAddress_reverts_if_zeroAddress() (gas: 20553)
Test result: ok. 2 passed; 0 failed; finished in 7.81ms

Running 1 test for test/unit/getOfferHash.t.sol:TestGetOfferHash
[PASS] test_unit_getOfferHash() (gas: 22484)
Test result: ok. 1 passed; 0 failed; finished in 19.68ms

Running 2 tests for test/unit/getUnderlyingNft.t.sol:TestGetUnderlyingNft
[PASS] test_unit_getUnderlyingNft_returns_Zeros_whenLoanClosed() (gas: 874992)
[PASS] test_unit_getUnderlyingNft_returns_underylingNftDetails_whenLoanActive() (gas: 893665)
Test result: ok. 2 passed; 0 failed; finished in 36.71ms

Running 1 test for test/unit/pause.t.sol:TestPause
[PASS] test_unit_pause_simple_case() (gas: 144860)
Test result: ok. 1 passed; 0 failed; finished in 10.97ms

Running 2 tests for test/unit/withdrawOfferSignature.sol:TestWithdrawOfferSignature
[PASS] test_unit_cannot_withdrawOfferSignature_not_signer() (gas: 52511)
[PASS] test_unit_withdrawOfferSignature_works() (gas: 74080)
Test result: ok. 2 passed; 0 failed; finished in 18.49ms

Running 1 test for test/unit/marketplaceIntegration/pause.t.sol:TestPauseMarketplace
[PASS] test_unit_pause_Marketplace_simple_case() (gas: 106873)
Test result: ok. 1 passed; 0 failed; finished in 6.85ms

Running 1 test for test/unit/pauseSanctions.t.sol:TestPauseSanctions
[PASS] test_unit_pauseSanctions_simple_case() (gas: 918855)
Test result: ok. 1 passed; 0 failed; finished in 13.83ms

Running 1 test for test/unit/marketplaceIntegration/unpause.t.sol:TestUnpauseMarketplace
[PASS] test_unit_unpause_Marketplace_simple_case() (gas: 964522)
Test result: ok. 1 passed; 0 failed; finished in 9.04ms

Running 1 test for test/unit/marketplaceIntegration/pauseSanctions.t.sol:TestPauseSanctionsMarketplace
[PASS] test_unit_pauseSanctions_Marketplace_simple_case() (gas: 940706)
Test result: ok. 1 passed; 0 failed; finished in 9.62ms
```

```
Running 1 test for test/unit/unpauseSanctions.t.sol:TestUnpauseSanctions
[PASS] test_unit_unpauseSanctions_simple_case() (gas: 947210)
Test result: ok. 1 passed; 0 failed; finished in 10.25ms

Running 1 test for
test/unit/marketplaceIntegration/unpauseSanctions.t.sol:TestUnpauseSanctionsMarketplace
[PASS] test_unit_unpauseSanctions_Marketplace_simple_case() (gas: 957425)
Test result: ok. 1 passed; 0 failed; finished in 13.55ms

Running 2 tests for
test/unit/updateDelegateRegistryContractAddress.t.sol:TestUpdateDelegateRegistryContractAddress
[PASS] test_unit_updateDelegateRegistryContractAddress_nonZeroAddress() (gas: 27037)
[PASS] test_unit_updateDelegateRegistryContractAddress_reverts_if_zeroAddress() (gas: 20514)
Test result: ok. 2 passed; 0 failed; finished in 4.84ms

Running 4 tests for
test/unit/marketplaceIntegration/buyWithFinancingSuperrare.sol:TestBuyWithFinancingMarketplace
[PASS]
test_fuzz_buyWithFinancingMarketplace_reverts_ifValueSentLessThanDownpaymentPlusMarketFee((uint128,uint12
8,uint128,uint32,uint32,uint32)) (runs: 128, μ: 113580, ~: 113580)
[PASS]
test_fuzz_buyWithFinancingMarketplace_simplest_case((uint128,uint128,uint128,uint32,uint32,uint32))
(runs: 128, μ: 948178, ~: 948178)
[PASS] test_unit_buyWithFinancingMarketplace_reverts_ifValueSentLessThanDownpaymentPlusMarketFee() (gas:
113022)
[PASS] test_unit_buyWithFinancingMarketplace_simplest_case() (gas: 947730)
Test result: ok. 4 passed; 0 failed; finished in 67.27s

Running 1 test for test/unit/unpause.t.sol:TestUnpause
[PASS] test_unit_unpause_simple_case() (gas: 946744)
Test result: ok. 1 passed; 0 failed; finished in 8.45ms

Running 10 tests for test/unit/seizeAsset.t.sol:TestSeizeAsset
[PASS] test_fuzz_seizeAsset_reverts_ifCallerNotSeller((uint128,uint128,uint128,uint32,uint32,uint32))
(runs: 128, μ: 922252, ~: 922252)
[PASS] test_fuzz_seizeAsset_reverts_ifCallerSanctioned((uint128,uint128,uint128,uint32,uint32,uint32))
(runs: 128, μ: 935419, ~: 935419)
[PASS] test_fuzz_seizeAsset_reverts_if_loanClosed((uint128,uint128,uint128,uint32,uint32,uint32)) (runs:
128, μ: 905893, ~: 905980)
[PASS] test_fuzz_seizeAsset_reverts_if_not_expired((uint128,uint128,uint128,uint32,uint32,uint32)) (runs:
128, μ: 914820, ~: 914820)
[PASS] test_fuzz_seizeAsset_simplest_case((uint128,uint128,uint128,uint32,uint32,uint32)) (runs: 128, μ:
813591, ~: 813592)
[PASS] test_unit_seizeAsset_reverts_ifCallerNotSeller() (gas: 921693)
[PASS] test_unit_seizeAsset_reverts_ifCallerSanctioned() (gas: 934860)
[PASS] test_unit_seizeAsset_reverts_if_loanClosed() (gas: 905742)
[PASS] test_unit_seizeAsset_reverts_if_not_expired() (gas: 914327)
[PASS] test_unit_seizeAsset_simplest_case() (gas: 813368)
Test result: ok. 10 passed; 0 failed; finished in 360.56s

Running 18 tests for test/unit/makePayment.t.sol:TestMakePayment
[PASS] test_fuzz_makePayment_fullRepayment_in_gracePeriod((uint128,uint128,uint128,uint32,uint32,uint32))
(runs: 128, μ: 904055, ~: 904180)
[PASS] test_fuzz_makePayment_fullRepayment_simplest_case((uint128,uint128,uint128,uint32,uint32,uint32))
(runs: 128, μ: 946650, ~: 946762)
[PASS]
test_fuzz_makePayment_partialRepayment_in_grace_period((uint128,uint128,uint128,uint32,uint32,uint32))
(runs: 128, μ: 1009757, ~: 1009958)
[PASS]
test_fuzz_makePayment_partialRepayment_simplest_case((uint128,uint128,uint128,uint32,uint32,uint32))
(runs: 128, μ: 1009325, ~: 1009432)
[PASS]
test_fuzz_makePayment_returns_anyExtraAmountNotReqToCloseTheLoan((uint128,uint128,uint128,uint32,uint32,u
int32)) (runs: 128, μ: 906576, ~: 906668)
[PASS]
test_fuzz_makePayment_reverts_ifAmountReceivedLessThanReqMinPayment((uint128,uint128,uint128,uint32,uint3
2,uint32)) (runs: 128, μ: 934786, ~: 934893)
[PASS] test_fuzz_makePayment_reverts_ifCallerSanctioned((uint128,uint128,uint128,uint32,uint32,uint32))
(runs: 128, μ: 933618, ~: 933677)
[PASS] test_fuzz_makePayment_reverts_ifLoanAlreadyClosed((uint128,uint128,uint128,uint32,uint32,uint32))
(runs: 128, μ: 911404, ~: 911516)
[PASS] test_fuzz_makePayment_reverts_if_post_grace_period((uint128,uint128,uint128,uint32,uint32,uint32))
```

```
(runs: 128, μ: 936269, ~: 936370)
[PASS] test_unit_makePayment_fullRepayment_in_gracePeriod() (gas: 903890)
[PASS] test_unit_makePayment_fullRepayment_simplest_case() (gas: 946580)
[PASS] test_unit_makePayment_partialRepayment_in_grace_period() (gas: 1009422)
[PASS] test_unit_makePayment_partialRepayment_simplest_case() (gas: 1008874)
[PASS] test_unit_makePayment_returns_anyExtraAmountNotReqToCloseTheLoan() (gas: 906396)
[PASS] test_unit_makePayment_reverts_ifAmountReceivedLessThanReqMinPayment() (gas: 934311)
[PASS] test_unit_makePayment_reverts_ifCallerSanctioned() (gas: 933096)
[PASS] test_unit_makePayment_reverts_ifLoanAlreadyClosed() (gas: 911244)
[PASS] test_unit_makePayment_reverts_if_post_grace_period() (gas: 935788)
Test result: ok. 18 passed; 0 failed; finished in 374.92s

Running 28 tests for test/unit/instantSell.t.sol:TestInstantSell
[PASS] test_fuzz_instantSell_loanClosed_simplest_case((uint128,uint128,uint128,uint32,uint32,uint32))
(runs: 128, μ: 1156183, ~: 1156296)
[PASS] test_fuzz_instantSell_loanClosed_withoutSeaportFee((uint128,uint128,uint128,uint32,uint32,uint32))
(runs: 128, μ: 1119599, ~: 1119712)
[PASS] test_fuzz_instantSell_reverts_ifCallerIsNotBuyer((uint128,uint128,uint128,uint32,uint32,uint32))
(runs: 128, μ: 938536, ~: 938611)
[PASS] test_fuzz_instantSell_reverts_ifCallerSanctioned((uint128,uint128,uint128,uint32,uint32,uint32))
(runs: 128, μ: 989421, ~: 989480)
[PASS]
test_fuzz_instantSell_reverts_ifInvalidOrderConsideration1Token((uint128,uint128,uint128,uint32,uint32,ui
nt32)) (runs: 128, μ: 950532, ~: 950634)
[PASS]
test_fuzz_instantSell_reverts_ifInvalidOrderOffer0ItemType((uint128,uint128,uint128,uint32,uint32,uint32)
) (runs: 128, μ: 949739, ~: 949889)
[PASS]
test_fuzz_instantSell_reverts_ifInvalidOrderOffer0Token((uint128,uint128,uint128,uint32,uint32,uint32))
(runs: 128, μ: 949983, ~: 950106)
[PASS] test_fuzz_instantSell_reverts_ifLoanInHardDefault((uint128,uint128,uint128,uint32,uint32,uint32))
(runs: 128, μ: 936595, ~: 936673)
[PASS]
test_fuzz_instantSell_reverts_ifOrderConsideration0NotERC721Type((uint128,uint128,uint128,uint32,uint32,u
int32)) (runs: 128, μ: 949665, ~: 949767)
[PASS]
test_fuzz_instantSell_reverts_ifOrderConsideration1NotERC20Type((uint128,uint128,uint128,uint32,uint32,ui
nt32)) (runs: 128, μ: 950290, ~: 950424)
[PASS]
test_fuzz_instantSell_reverts_ifOrderNftAddressNotEqualToLoanNftAddress((uint128,uint128,uint128,uint32,u
int32,uint32)) (runs: 128, μ: 949454, ~: 949604)
[PASS]
test_fuzz_instantSell_reverts_ifOrderNftIdNotEqualToLoanNftId((uint128,uint128,uint128,uint32,uint32,uint
32)) (runs: 128, μ: 949376, ~: 949505)
[PASS]
test_fuzz_instantSell_reverts_ifSaleAmountLessThanMinSaleAmountRequested((uint128,uint128,uint128,uint32,
uint32,uint32)) (runs: 128, μ: 1262335, ~: 1262453)
[PASS] test_fuzz_instantSell_reverts_post_grace_period((uint128,uint128,uint128,uint32,uint32,uint32))
(runs: 128, μ: 1046632, ~: 1046679)
[PASS] test_unit_instantSell_loanClosed_simplest_case() (gas: 1156112)
[PASS] test_unit_instantSell_loanClosed_withoutSeaportFee() (gas: 1119476)
[PASS] test_unit_instantSell_reverts_ifCallerIsNotBuyer() (gas: 938008)
[PASS] test_unit_instantSell_reverts_ifCallerSanctioned() (gas: 989260)
[PASS] test_unit_instantSell_reverts_ifInvalidOrderConsideration1Token() (gas: 950054)
[PASS] test_unit_instantSell_reverts_ifInvalidOrderOffer0ItemType() (gas: 949374)
[PASS] test_unit_instantSell_reverts_ifInvalidOrderOffer0Token() (gas: 949569)
[PASS] test_unit_instantSell_reverts_ifLoanInHardDefault() (gas: 936180)
[PASS] test_unit_instantSell_reverts_ifOrderConsideration0NotERC721Type() (gas: 949118)
[PASS] test_unit_instantSell_reverts_ifOrderConsideration1NotERC20Type() (gas: 949866)
[PASS] test_unit_instantSell_reverts_ifOrderNftAddressNotEqualToLoanNftAddress() (gas: 949001)
[PASS] test_unit_instantSell_reverts_ifOrderNftIdNotEqualToLoanNftId() (gas: 949012)
[PASS] test_unit_instantSell_reverts_ifSaleAmountLessThanMinSaleAmountRequested() (gas: 1261895)
[PASS] test_unit_instantSell_reverts_post_grace_period() (gas: 1046121)
Test result: ok. 28 passed; 0 failed; finished in 374.98s

Running 32 tests for test/unit/buyWithFinancing.t.sol:TestBuyWithFinancing
[PASS] test_fuzz_buyWithFinancing_collection_offer((uint128,uint128,uint128,uint32,uint32,uint32)) (runs:
128, μ: 953116, ~: 953116)
[PASS]
test_fuzz_buyWithFinancing_collection_offer_reverts_if_limitReached((uint128,uint128,uint128,uint32,uint3
2,uint32)) (runs: 128, μ: 933722, ~: 933722)
[PASS] test_fuzz_buyWithFinancing_emitsExpectedEvents((uint128,uint128,uint128,uint32,uint32,uint32))
```

```
(runs: 128, μ: 902118, ~: 902118)
[PASS]
test_fuzz_buyWithFinancing_returnsExtraAmountMoreThanDownpayment((uint128,uint128,uint128,uint32,uint32,u
int32)) (runs: 128, μ: 895425, ~: 895425)
[PASS]
test_fuzz_buyWithFinancing_reverts_if_buyerSanctioned((uint128,uint128,uint128,uint32,uint32,uint32))
(runs: 128, μ: 158165, ~: 158165)
[PASS]
test_fuzz_buyWithFinancing_reverts_if_callerSanctioned((uint128,uint128,uint128,uint32,uint32,uint32))
(runs: 128, μ: 161802, ~: 161802)
[PASS]
test_fuzz_buyWithFinancing_reverts_if_invalidDownpaymentValue((uint128,uint128,uint128,uint32,uint32,uint
32)) (runs: 128, μ: 160563, ~: 160563)
[PASS]
test_fuzz_buyWithFinancing_reverts_if_invalidMinPrincipalPerPeriod((uint128,uint128,uint128,uint32,uint32
,uint32)) (runs: 128, μ: 161101, ~: 161101)
[PASS]
test_fuzz_buyWithFinancing_reverts_if_invalidPeriodDuration((uint128,uint128,uint128,uint32,uint32,uint32
)) (runs: 128, μ: 159573, ~: 159573)
[PASS]
test_fuzz_buyWithFinancing_reverts_if_nftIdNotEqualToOfferNftId_for_nonCollectionOffer((uint128,uint128,u
int128,uint32,uint32,uint32)) (runs: 128, μ: 61643, ~: 61643)
[PASS] test_fuzz_buyWithFinancing_reverts_if_offerExpired((uint128,uint128,uint128,uint32,uint32,uint32))
(runs: 128, μ: 160538, ~: 160538)
[PASS]
test_fuzz_buyWithFinancing_reverts_if_offerForSellerFinancingTicket((uint128,uint128,uint128,uint32,uint3
2,uint32)) (runs: 128, μ: 958465, ~: 958465)
[PASS]
test_fuzz_buyWithFinancing_reverts_if_offerPriceLessThanDownpayment((uint128,uint128,uint128,uint32,uint3
2,uint32)) (runs: 128, μ: 160640, ~: 160640)
[PASS]
test_fuzz_buyWithFinancing_reverts_if_offerSignerNotOwner((uint128,uint128,uint128,uint32,uint32,uint32))
(runs: 128, μ: 213502, ~: 213502)
[PASS]
test_fuzz_buyWithFinancing_reverts_if_signatureAlreadyUsed((uint128,uint128,uint128,uint32,uint32,uint32)
) (runs: 128, μ: 822009, ~: 822008)
[PASS] test_fuzz_buyWithFinancing_simplest_case((uint128,uint128,uint128,uint32,uint32,uint32)) (runs:
128, μ: 959778, ~: 959778)
[PASS] test_unit_buyWithFinancing_collection_offer() (gas: 952557)
[PASS] test_unit_buyWithFinancing_collection_offer_reverts_if_limitReached() (gas: 933096)
[PASS] test_unit_buyWithFinancing_emitsExpectedEvents() (gas: 901560)
[PASS] test_unit_buyWithFinancing_returnsExtraAmountMoreThanDownpayment() (gas: 894909)
[PASS] test_unit_buyWithFinancing_reverts_if_buyerSanctioned() (gas: 157694)
[PASS] test_unit_buyWithFinancing_reverts_if_callerSanctioned() (gas: 161197)
[PASS] test_unit_buyWithFinancing_reverts_if_invalidDownpaymentValue() (gas: 160094)
[PASS] test_unit_buyWithFinancing_reverts_if_invalidMinPrincipalPerPeriod() (gas: 160542)
[PASS] test_unit_buyWithFinancing_reverts_if_invalidPeriodDuration() (gas: 159081)
[PASS] test_unit_buyWithFinancing_reverts_if_nftIdNotEqualToOfferNftId_for_nonCollectionOffer() (gas:
61085)
[PASS] test_unit_buyWithFinancing_reverts_if_offerExpired() (gas: 159936)
[PASS] test_unit_buyWithFinancing_reverts_if_offerForSellerFinancingTicket() (gas: 957906)
[PASS] test_unit_buyWithFinancing_reverts_if_offerPriceLessThanDownpayment() (gas: 160036)
[PASS] test_unit_buyWithFinancing_reverts_if_offerSignerNotOwner() (gas: 212944)
[PASS] test_unit_buyWithFinancing_reverts_if_signatureAlreadyUsed() (gas: 821768)
[PASS] test_unit_buyWithFinancing_simplest_case() (gas: 959264)
Test result: ok. 32 passed; 0 failed; finished in 376.04s
```

Code Coverage

Coverage was gathered by running `forge coverage --optimize --fork-url https://eth-mainnet.g.alchemy.com/v2/jxUUn2DsYODlc68SEU_7eNGCn2hQ7b63`. The contracts show strong code coverage.

File	% Lines	% Statements	% Branches	% Funcs
src/SellerFinancing.sol	97.01% (**227/**234)	96.69% (**263/**272)	80.00% (**72/**90)	93.02% (**40/**43)

File	% Lines	% Statements	% Branches	% Funcs
src/marketplaceIntegration/ MarketplaceIntegration.sol	100.00% (**23/**23)	96.00% (**24/**25)	100.00% (**8/**8)	100.00% (**10/**10)

Changelog

- 2023-04-20 - Initial report
- 2023-05-04 - Fix Review

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:
- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT