

EWA Contracts Audit Report

Table of Contents

1. Executive Summary

About PACT EWA Contracts

Audit Summary

Overall Security Posture

Before Fix Review Summary

Launch Recommendations

Audit Scope

2. Assumptions and Considerations

Audit Assumptions

Centralization Risks

Trust Assumptions

3. Severity Definitions

Impact

Likelihood

Severity Classification Matrix

4. Enhancement Opportunities

E01: Adding a post-wiring assertion will improve createLoanBook deployment consistency

E02: Validate registered factory implementations before proxy deployment

E03: Correcting stale salt documentation will improve ProtocolFactory integration reliability

E04: Validate nonzero identity registry addresses before activation

E05: Make Vehicle fallback behavior observable

E06: Verify Vehicle settlement delivery after receivePayment

E07: Removing unused ProtocolFactory interface fields will improve API clarity

E08: Implementing consistent behaviour for non-existent Draw will prevent confusion in EwaCollection view functions

About Us.....20

 About Adevar Labs 20

 Audit Methodology 20

 Confidentiality Notice 22

 Legal Disclaimer 22

Appendix A: Operational Security for Privileged EWA Roles.....23

Appendix B: EWA Monitoring, Reconciliation, and Data Hygiene.....26

1. Executive Summary

About PACT EWA Contracts

PACT's Earned Wage Access (EWA) contracts manage short-term wage advances issued to employees and repaid through payroll deductions. Each draw is represented by a receivable NFT. Whoever owns that NFT receives the settlement proceeds, while the EWA collection tracks the principal lifecycle across issued, expected, settled, and marked-down amounts.

The core contract is **EwaCollection**. It handles draw issuance, payroll settlement, markdowns, and funding custody. **ReceivableCollectionNft** represents draw ownership and can gate secondary transfers through an identity registry. **EwaDrawIncentive** pays employee rewards using reward terms snapshotted by block. The EWA path in **ProtocolFactory** creates and wires an **EwaCollection** with its receivable NFT.

Audit Summary

No critical, high, medium, or low severity vulnerabilities were identified in the reviewed EWA scope. The useful output from the review is therefore an enhancement set, not a vulnerability list. These recommendations focus on factory deployment safety, Vehicle settlement integration, identity registry configuration, and API/documentation clarity.

Risk Level	Count	Fixed	Acknowledged
Critical	0	0	0
High	0	0	0
Medium	0	0	0
Low	0	0	0

Enhancement opportunities: 8.

Overall Security Posture

The review did not identify any critical, high, medium, or low severity vulnerabilities in the EWA scope. The accounting model is compact and the reviewed state transitions for issuance, settlement, and markdown were coherent.

The main residual risk is integration and operations around privileged configuration: factory implementation registration, receiver behavior during Vehicle settlement, and identity registry changes.

Before Fix Review Summary

The before-fix review did not find a missed accounting update or durable double-payment issue in the Vehicle settlement path. The findings are non-blocking enhancements around factory deployment checks, implementation validation, CREATE2 documentation, Vehicle settlement observability, identity registry validation, interface cleanup, and consistent draw view behavior.

Launch Recommendations

Before launch, the team should either implement or explicitly acknowledge the enhancement items. The highest-value checks are factory deployment smoke tests, validation of registered implementations, receiver-type tests for Vehicle settlement, and validation of nonzero identity registry addresses.

Audit Scope

- **Repository:** <https://github.com/Lucid-Fi/pact-evm-contracts/tree/ewa-audit>
- **Before-Fix Review Commit Hash:** [b4ff31e28c673926e17e3c4311db398c223907d4](#)
- **Files/Modules in Scope:**
 - Core implementations: [src/EwaCollection.sol](#), [src/EwaDrawIncentive.sol](#), [src/ReceivableCollectionNft.sol](#), [src/ProtocolFactory.sol](#)
 - Interfaces: [src/interfaces/IEwaCollection.sol](#), [src/interfaces/IEwaDrawIncentive.sol](#), [src/interfaces/IReceivableCollectionNft.sol](#), [src/interfaces/IReceivableCollection.sol](#)
 - Types: [src/types/Types.sol](#)

2. Assumptions and Considerations

Audit Assumptions

The review relied on the scope, trust model, and known design decisions documented in the project README. Where the implementation matched those documented assumptions, we did not repeat them as findings.

In particular, the review treated the following as accepted design constraints: the collection owner is trusted and operationally protected; payment tokens are standard, non-rebasing, non-fee ERC-20s; receivable NFTs are intentionally non-burnable; zero-value settlements are rejected; NFT minting to the issuer intentionally bypasses identity-registry checks; incentive claims are independent of draw settlement status; the identity registry can be disabled; and permissionless incentive claims always pay the employee address stored on the draw.

If any of these assumptions change in production, the related flows should be re-reviewed. For example, supporting fee-on-transfer tokens, changing who receives minted NFTs, or making settlement permissionless would materially change the risk model.

Centralization Risks

The EWA system intentionally uses privileged ownership rather than fully permissionless operation. That is a reasonable product choice, but it means the privileged accounts are production security boundaries.

The EWA collection owner can issue draws, settle draws, batch settle, mark down shortfalls, withdraw funding, and authorize upgrades. The receivable NFT owner can change the identity registry after a one-day timelock, including disabling it. The incentive owner can change future reward terms, withdraw incentive funding, and authorize upgrades. The factory owner can change implementations used by future deployments.

The practical risk is not only that these roles exist. It is also how they are operated. If several multisig signers share the same devices, password manager, cloud account, or recovery process, the real-world threshold may be lower than the on-chain threshold suggests.

Trust Assumptions

Several dependencies sit outside the reviewed smart contract logic but still matter for production safety. Payroll systems determine the draw and settlement inputs. Token contracts determine whether transfers behave as expected. Identity registries determine whether receivable NFTs can move. Vehicle contracts determine whether Vehicle-routed settlement proceeds are actually pulled into the expected account.

The system also depends on operational tooling: multisig signers, hardware wallets, proposal builders, transaction decoders, deployment scripts, indexers, monitors, and reconciliation jobs. A strong on-chain design can still fail operationally if these pieces are weak or if alerts do not reach people who can act.

3. Severity Definitions

Each issue identified in this report is assigned a severity level based on two dimensions: **Impact** and **Likelihood**. These dimensions help project our team's understanding of both the potential consequences of a vulnerability and how likely a vulnerability is to be discovered and exploited in the real world.

Note: Enhancements represent non-blocking improvements-typically usability, observability, or defense-in-depth tweaks that do not pose an immediate asset risk but would improve the product's reliability and/or user experience if implemented.

Impact

Impact reflects the potential consequences of the issue-particularly on **project funds, user funds**, and the **availability or integrity** of the protocol.

- **High Impact:** Successful exploitation could result in a complete loss of user or protocol funds, disruption of core protocol functionality, or permanent loss of control over critical components.
- **Medium Impact:** Exploitation could cause significant disruption or partial loss of funds, but not a total compromise. May impact some users or non-core functionality.
- **Low Impact:** The issue has minor or negligible consequences. It may affect edge cases, expose metadata, or degrade performance slightly without putting funds or core logic at serious risk.

Likelihood

Likelihood reflects how easy a vulnerability is to discover and exploit by an attacker, as well as how economically attractive the exploit is to an attacker.

- **High Likelihood:** The vulnerability is trivially exploitable. This means it can be exploited by a wide range of actors without privileged access rights, with minimal capital requirements and low financial risks.
- **Medium Likelihood:** This type of vulnerability can be found and exploited with moderate effort. It might require a significant capital investment, but with manageable financial risk.
- **Low Likelihood:** Exploitation of these vulnerabilities is often technically unfeasible or requires highly specialized conditions. They may require extraordinary effort or a significant financial risk for an attacker, with a high chance of failure and minimal potential return.

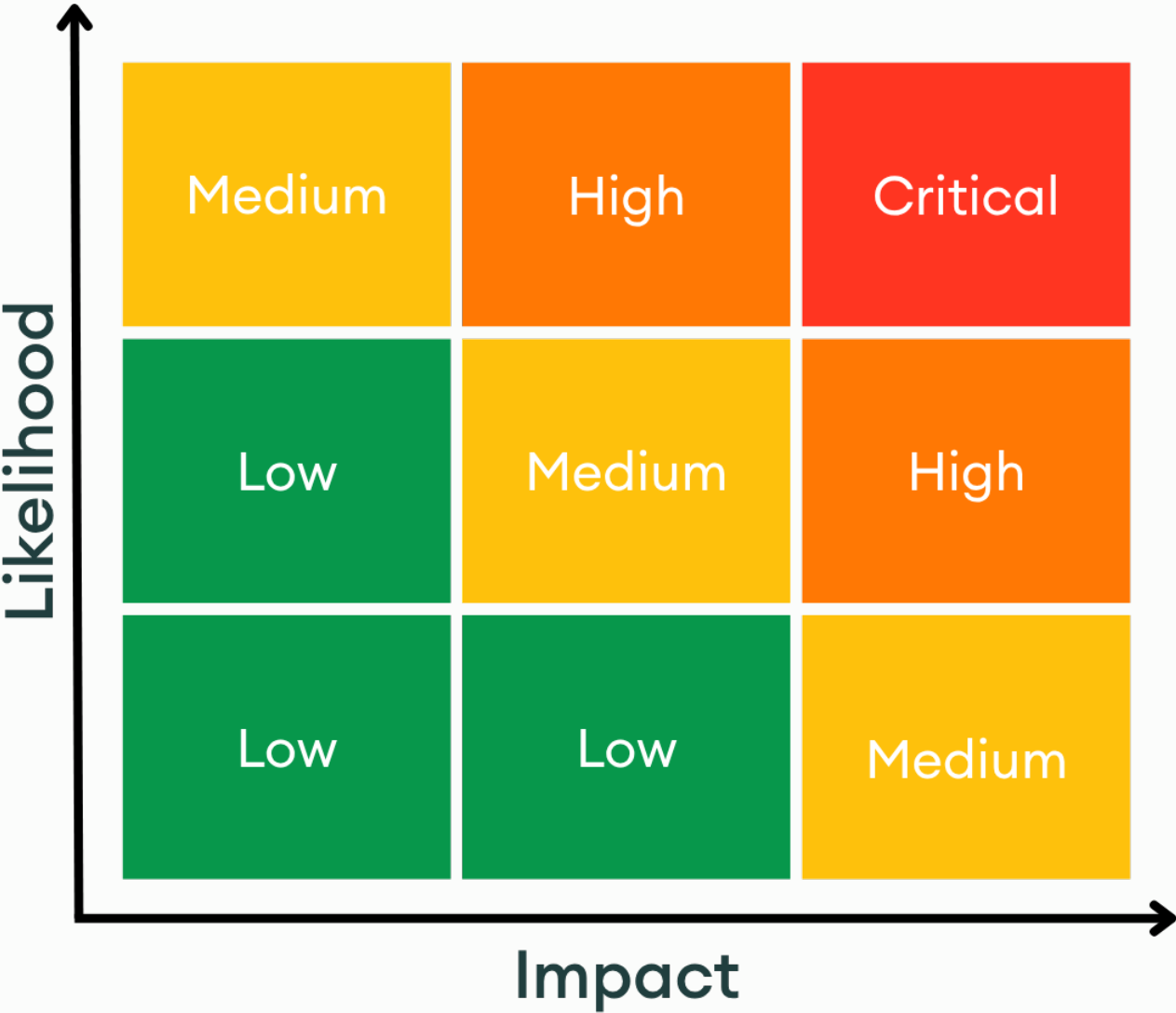
Severity Classification Matrix

By combining **Impact** and **Likelihood**, we assign a severity level using the matrix below:

- **Critical:** High impact + high likelihood (e.g. a bug that could allow anyone to drain a substantial amount of protocol funds with minimal effort)
- **High:** High impact with medium likelihood, or medium impact with high likelihood
- **Medium:** Moderate impact and/or discoverability

- **Low:** Minimal impact or unlikely to be exploited

This structured approach helps teams prioritize fixes and mitigate the most dangerous threats first.



Severity Matrix

4. Enhancement Opportunities

E01: Adding a post-wiring assertion will improve `createLoanBook` deployment consistency

LOCATION

pact-evm-contracts/b4ff31e2/src/ProtocolFactory.sol:L104 [↗](#)

RELEVANT SNIPPET

>_ ProtocolFactory.sol		SOLIDITY
102:		
103:	/// @inheritdoc IProtocolFactory	
104:	function createLoanBook(CreateLoanBookParams calldata params) external returns (address loanBook, address nft) {	
105:	// 1. Deploy NFT with factory as temporary owner (so we can call setCollection)	
106:	bytes32 nftSalt = keccak256(abi.encodePacked(params.salt, "LoanBookNft"));	

DESCRIPTION

`createLoanBook()` deploys a `ReceivableCollectionNft`, deploys a `LoanBook`, calls `nft.setCollection(loanBook)`, and transfers NFT ownership to the `admin`. This mirrors the paired deployment flow used by `createEwaCollection()`.

However, `createEwaCollection()` includes an explicit post-wiring assertion that verifies both sides of the relationship:

		SOLIDITY
if (		
ReceivableCollectionNft(collectionNft).collection() != collection		
EwaCollection(collection).nft() != collectionNft		
) revert EwaNftLinkMismatch();		

`createLoanBook()` does not include the equivalent consistency check. The enhancement is defensive: it would catch future refactors, wrong implementation registration, or accidental initializer drift at deployment time.

POTENTIAL BENEFIT

This adds one post-deployment invariant check to the `LoanBook` pair flow, bringing it to parity with the existing EWA deployment hardening. It can prevent deployment of a miswired `LoanBook`/NFT pair and turns a latent configuration bug into an immediate revert.

RECOMMENDATION

Add a `LoanBook`-specific mismatch error and assertion after ownership transfer or immediately after `setCollection()`.

DIFF

```
ReceivableCollectionNft(nft).setCollection(loanBook);
OwnableUpgradeable(nft).transferOwnership(params.admin);

+ if (
+   ReceivableCollectionNft(nft).collection() != loanBook
+   || LoanBook(loanBook).nft() != nft
+ ) revert LoanBookNftLinkMismatch();
```

E02: Validate registered factory implementations before proxy deployment

LOCATION

[pact-evm-contracts/b4ff31e2/src/ProtocolFactory.sol:L69-L74](#) 

RELEVANT SNIPPET

> _ProtocolFactory.sol	SOLIDITY
67:	
68:	/// @inheritdoc IProtocolFactory
69:	function setImplementation(bytes32 contractType, address impl) external onlyOwner {
70:	if (impl == address(0)) revert ZeroAddress();
71:	if (impl.code.length == 0) revert NoCode();
72:	_implementations[contractType] = impl;
73:	emit ImplementationSet(contractType, impl);
74:	}
75:	
76:	/// @inheritdoc IProtocolFactory

OTHER LOCATIONS

- [pact-evm-contracts/b4ff31e2/src/ProtocolFactory.sol:L585-L591](#) 

DESCRIPTION

ProtocolFactory deploys ERC-1967 proxies that point to implementation addresses registered by the factory owner. **setImplementation** rejects the zero address and addresses with no code, but it does not validate that the registered implementation is UUPS-compatible or that it matches the expected contract type.

This is owner-gated, so it is not an external attacker path by itself. The practical risk is operational: a wrong implementation address can make future factory deployments revert, deploy proxies with incompatible initialization behavior, or deploy proxies that are not upgradeable through the expected UUPS path.

POTENTIAL BENEFIT

- Catches implementation registration mistakes before they affect future deployments.
- Gives deployment operators stronger assurance that factory-created proxies use the expected implementation class.

RECOMMENDATION

Add implementation validation before accepting a registered implementation, or enforce it in deployment scripts before calling **setImplementation**. At minimum, check UUPS compatibility through **proxiableUUID()** and require the ERC-1967 implementation slot. For EWA-specific types, also consider type-level validation, such as checking expected ERC-165 support or running a deployment-script smoke test that deploys a proxy with the intended initializer and verifies the expected public interface.

E03: Correcting stale salt documentation will improve ProtocolFactory integration reliability

LOCATION

pact-evm-contracts/b4ff31e2/src/interfaces/IProtocolFactory.sol:L44 [↗](#)

RELEVANT SNIPPET

>_ IProtocolFactory.sol		SOLIDITY
42:	/// `paymentToken` and `originator` are event-only metadata – they are NOT passed to	
43:	/// LoanBook.initialize(). The LoanBook itself is token-agnostic; each loan specifies	
44:	/// its own payment token at origination. The NFT salt is derived as keccak256(salt)	
45:	struct CreateLoanBookParams {	
46:	string name;	

OTHER LOCATIONS

- pact-evm-contracts/b4ff31e2/src/interfaces/IProtocolFactory.sol:L149 [↗](#)

DESCRIPTION

The interface contains stale **CREATE2** salt documentation.

For **CreateLoanBookParams**, the interface states that the NFT salt is derived as **keccak256(salt)**, but the implementation derives it as:

keccak256(abi.encodePacked(params.salt, "LoanBookNft"))

For **CreateEwaCollectionParams**, the interface tells callers to include **msg.sender** in the salt to prevent front-running or salt griefing. This guidance is outdated because **_deployProxy()** already derives the effective CREATE2 salt as:

keccak256(abi.encodePacked(salt, msg.sender))

As a result, callers may mis-document deterministic address derivation or implement off-chain address computation incorrectly.

POTENTIAL BENEFIT

Fixing the comments removes 2 misleading documentation paths affecting the two paired deployment flows.

RECOMMENDATION

Update the comments to match the implementation:

1. The NFT salt is derived as **keccak256(abi.encodePacked(salt, "LoanBookNft"))**.
2. The NFT salt is derived as **keccak256(abi.encodePacked(salt, "EwaCollectionNft"))** and the factory additionally namespaces all **CREATE2** salts by **msg.sender** internally.

E04: Validate nonzero identity registry addresses before activation

LOCATION

[pact-evm-contracts/b4ff31e2/src/ReceivableCollectionNft.sol:L93-L117](#) 

RELEVANT SNIPPET

> _ReceivableCollectionNft.sol	SOLIDITY
<pre>91: 92: /// @inheritdoc IReceivableCollectionNft 93: function proposeIdentityRegistry(address registry) external onlyOwner { 94: if (_hasProposedIdentityRegistry) revert ProposalPending(); 95: uint64 effectiveAfter = SafeCast.toUint64(block.timestamp) + _IDENTITY_REGISTRY_TTL; 96: _proposedIdentityRegistry = registry; 97: _identityRegistryEffectiveAfter = effectiveAfter; 98: _hasProposedIdentityRegistry = true; 99: emit IdentityRegistryProposed(registry, effectiveAfter); 100: if (registry == address(0)) emit IdentityRegistryDisableProposed(effectiveAfter); 101: } 102: 103: /// @inheritdoc IReceivableCollectionNft 104: function acceptIdentityRegistry() external onlyOwner { 105: if (!_hasProposedIdentityRegistry) revert NoPendingProposal(); 106: if (block.timestamp < _identityRegistryEffectiveAfter) { 107: revert TimelockNotElapsed(_identityRegistryEffectiveAfter, block.timestamp); 108: } 109: 110: address oldRegistry = _identityRegistry; 111: _identityRegistry = _proposedIdentityRegistry; 112: _proposedIdentityRegistry = address(0); 113: _identityRegistryEffectiveAfter = 0; 114: _hasProposedIdentityRegistry = false; 115: 116: emit IdentityRegistryChanged(oldRegistry, _identityRegistry); 117: } 118: 119: /// @inheritdoc IReceivableCollectionNft</pre>	

OTHER LOCATIONS

- [pact-evm-contracts/b4ff31e2/src/ReceivableCollectionNft.sol:L141-L146](#) 

DESCRIPTION

The NFT owner can propose any address as the identity registry. This includes EOAs, contracts with no **isVerified(address)** implementation, and contracts that always revert. Once accepted, an invalid registry can break intended secondary transfers because **_update** calls **isVerified** for non-mint transfers.

The one-day timelock and owner-only control reduce severity, but a simple validation step would catch configuration mistakes before they become operational incidents.

POTENTIAL BENEFIT

- Catches bad registry configuration before it affects receivable transfers.
- Keeps **address(0)** as an intentional disabled state while reducing accidental freezes.

RECOMMENDATION

For nonzero registry addresses, require deployed code and perform an interface or call-shape check before accepting the new registry. Keep **address(0)** available as an explicit disabled state, since that is an intentional design decision

E05: Make Vehicle fallback behavior observable

LOCATION

pact-evm-contracts/b4ff31e2/src/EwaCollection.sol:L486-L502 [↗](#)

RELEVANT SNIPPET

> _EwaCollection.sol	SOLIDITY
<pre>484: } 485: 486: /// @dev Routes payment to owner. If owner is a Vehicle (IVehicle via ERC-165), calls 487: /// receivePayment() to credit the collection account. Otherwise, direct transfer 488: /// The ERC-165 probe never reverts for EOAs/non-ERC165 contracts, and receivePay 489: /// ment is wrapped in try/catch so a reverting Vehicle cannot permanently block settleme 490: nt. 490: function _transferToOwner(IERC20 token, address owner_, uint128 amount) internal { 491: if (owner_.safeSupportsInterface(type(IVehicle).interfaceId)) { 492: token.forceApprove(owner_, amount); 493: try IVehicle(owner_).receivePayment(address(token), amount, DistributionType.P 494: rincipal) { 495: token.forceApprove(owner_, 0); 496: return; 497: } catch { 498: token.forceApprove(owner_, 0); 499: } 500: } // Fallback: direct transfer (EOA, non-Vehicle contract, or failed receivePayment) 501: token.safeTransfer(owner_, amount); 502: } 503: 504: /// @dev Pulls `amount` from `msg.sender` into this contract and reverts if the contra 505: ct's</pre>	

DESCRIPTION

If a Vehicle receiver reverts during **receivePayment**, **_transferToOwner** silently clears the allowance and falls back to a direct token transfer. That behavior is useful because it prevents a reverting receiver from permanently blocking settlement. The tradeoff is that operators do not get an explicit signal that a Vehicle integration failed and settlement was delivered through the fallback path.

Full gas forwarding to **receivePayment** is also worth making explicit in integration docs. A malicious or broken receiver can waste gas and cause the owner-initiated settlement transaction to fail. This does not let the receiver steal more than its own settlement amount, but it can create operational friction.

POTENTIAL BENEFIT

- Gives operators a clear signal when settlement uses the fallback path.
- Makes broken Vehicle integrations easier to detect before they become reconciliation issues.

RECOMMENDATION

Emit an event when the fallback path is used, including the token, receiver, amount, and whether the receiver advertised **IVehicle**. If gas grieving becomes a real operational concern, consider using a bounded low-level call for Vehicle notification and falling back to direct transfer on failure.

E06: Verify Vehicle settlement delivery after **receivePayment**

LOCATION

pact-evm-contracts/b4ff31e2/src/EwaCollection.sol:L486-L502 [↗](#)

RELEVANT SNIPPET

>_ EwaCollection.sol	SOLIDITY
484:	}
485:	
486:	/// @dev Routes payment to owner. If owner is a Vehicle (IVehicle via ERC-165), calls
487:	/// receivePayment() to credit the collection account. Otherwise, direct transfer
488:	/// The ERC-165 probe never reverts for EOAs/non-ERC165 contracts, and receivePay
489:	ment is wrapped in try/catch so a reverting Vehicle cannot permanently block settleme
490:	nt.
490:	function _transferToOwner(IERC20 token, address owner_, uint128 amount) internal {
491:	if (owner_.safeSupportsInterface(type(IVehicle).interfaceId)) {
492:	token.forceApprove(owner_, amount);
493:	try IVehicle(owner_).receivePayment(address(token), amount, DistributionType.P
494:	incipal) {
495:	token.forceApprove(owner_, 0);
496:	return;
497:	} catch {
498:	token.forceApprove(owner_, 0);
499:	}
500:	// Fallback: direct transfer (EOA, non-Vehicle contract, or failed receivePayment)
501:	token.safeTransfer(owner_, amount);
502:	}
503:	
504:	/// @dev Pulls `amount` from `msg.sender` into this contract and reverts if the contra
	ct's

DESCRIPTION

When the current NFT owner advertises **IVehicle**, **_transferToOwner** approves the Vehicle for **amount** and calls **receivePayment**. If the call returns successfully, the function clears the allowance and returns without performing a direct transfer.

This is safe for canonical Vehicles that pull the approved amount. However, a Vehicle-like contract can advertise the interface, return success, and not pull the tokens. In that case, EWA accounting records the settlement as paid while the actual tokens remain in the EWA collection. This is mostly self-harm for the receiver, but it can confuse reconciliation and make settlement delivery harder to reason about.

This is not a double-payment issue. If **receivePayment** pulls funds and then reverts, the nested token movement is rolled back and the fallback direct transfer sends the funds only once.

POTENTIAL BENEFIT

- Makes settlement delivery easier to reconcile.
- Reduces ambiguity when non-canonical Vehicle-like receivers are used.

RECOMMENDATION

Either restrict the Vehicle branch to known/canonical Vehicle contracts, or verify that the EWA collection's token balance decreased by exactly **amount** after a successful **receivePayment**. If the expected decrease did not happen, clear the allowance and use the direct-transfer fallback or revert with a clear error.

E07: Removing unused **ProtocolFactory** interface fields will improve API clarity

LOCATION

[pact-evm-contracts/b4ff31e2/src/interfaces/IProtocolFactory.sol:L19-L20](#) 

RELEVANT SNIPPET

	>_ IProtocolFactory.sol	SOLIDITY
17:	<code>bytes32 constant TYPE_FUNDING_MANAGER = keccak256("FundingManager");</code>	
18:	<code>bytes32 constant TYPE_EWA_COLLECTION = keccak256("EwaCollection");</code>	
19:	<code>bytes32 constant TYPE_COLLECTION_CONTRACT = keccak256("CollectionContract");</code>	
20:	<code>bytes32 constant TYPE_CROSS_CHAIN_HUB = keccak256("CrossChainHub");</code>	
21:		
22:	<code>/// @title IProtocolFactory</code>	

OTHER LOCATIONS

- [pact-evm-contracts/b4ff31e2/src/interfaces/IProtocolFactory.sol:L49](#) 
- [pact-evm-contracts/b4ff31e2/src/interfaces/IProtocolFactory.sol:L63](#) 
- [pact-evm-contracts/b4ff31e2/src/interfaces/IProtocolFactory.sol:L115](#) 

DESCRIPTION

The factory interface exposes several constants and struct fields that are not used by **ProtocolFactory**.

TYPE_COLLECTION_CONTRACT and **TYPE_CROSS_CHAIN_HUB** are documented as implementation-registry constants, but the cross-chain contracts are deployed as raw immutable contracts, not registered proxy implementations.

Similarly, **originator** in both **LoanBook** deployment parameter structs is not passed into **LoanBook.initialize()** and is not emitted in **LoanBookCreated.CreateVaultParams.initialNavPerShare** is also not passed into **DeFiVault.initialize()**; the vault hardcodes its initial stored NAV to 1e18.

This is not a security issue, but it creates integration ambiguity: callers may assume these values influence deployment behavior or deterministic address computation when they do not.

POTENTIAL BENEFIT

Removing these fields eliminates 5 dead interface declarations. The three unused struct fields each add 32 bytes of calldata to affected create / compute calls, with no behavioral effect. It also reduces the chance of incorrect SDK, frontend, or deployment-script assumptions around deterministic address calculation.

RECOMMENDATION

Remove unused constants and struct fields.

About Us

About Adevar Labs

Adevar Labs is a boutique blockchain security firm specializing in web3 audits.

Built by a mix of experienced professionals in traditional enterprise and crypto natives who have contributed to some of the most critical projects in blockchain infrastructure.

Our team's background spans companies like Bitdefender, Asymmetric Research, Quantstamp, Chainproof, and Juicebox, and includes experience securing smart contracts, bridges, and L1 and L2 protocols across ecosystems like Solana, Ethereum, Polkadot, Cosmos, and MultiversX.

With over 100 audits completed and a portfolio that includes custom fuzzers, exploit modeling, and runtime testing frameworks, Adevar Labs brings both depth and precision to every engagement.

Our auditors have discovered critical vulnerabilities, built high-impact tooling, and placed in top positions in premier audit competitions including Code4rena and Sherlock.

Team members hold distinctions such as PhDs in software protection, and key roles at Fortune 500 companies, and leadership of flagship conferences like ETH Bucharest.

With team members having publications with over 1,100 academic citations and trusted by projects securing over \$500M in on-chain value, Adevar Labs blends elite technical rigor with real-world security impact.

We also collaborate with some of the best independent security researchers in the web3 space.

Projects may optionally request specific contributors to be part of their audit.

Audit Methodology

Our audit methodology is specialized to provide thorough security assessments of Ethereum smart contracts. We focus explicitly on rigorous manual analysis, detailed threat modeling, and careful validation of implemented fixes to ensure your smart contracts operate securely and as intended.

1. Smart Contract Context and Architecture Analysis

Our auditors begin by deeply examining your Ethereum smart contract's documentation and intended functionality. We meticulously map out contract interactions, transaction processing flows, and state management logic. Special attention is given to understanding how your smart contract interfaces with critical EVM standards, such as ERC-20, ERC-721, or ERC-4626 tokens, and governance/timelock modules. Last but not least we check external integrations with other DeFi protocols and verify if the inputs and return values are handled properly.

2. Threat Modeling

We conduct targeted threat modeling tailored specifically for the EVM execution environment. We carefully define attacker capabilities and identify potential vulnerabilities that may arise from EVM-specific issues, including but not limited to:

- Unauthorized state variable modification
- Improper Access Control, role checks (**onlyOwner**, **require(hasRole)**) or **msg.sender** verification
- Misuse of low-level calls (**call**, **delegatecall**) or proxy/upgradeability patterns (e.g., storage collisions)
- Incorrect use of external calls, including reentrancy vectors
- Risks stemming from proxy initialization logic or denial-of-service related to gas limitations

3. In-depth Manual Security Review

Our experienced auditors perform an extensive manual security review of your Solidity-based Ethereum smart contracts. This involves a comprehensive line-by-line inspection of source code, focusing on common EVM vulnerabilities including, but not limited to:

- Missing or insufficient Access Control and Role Checks
- Inadequate **msg.sender** and transaction origin checks
- Incorrect handling of external calls and invocation privileges, particularly potential reentrancy risks
- Arithmetic and integer overflow or underflow errors
- Unsafe use of ABI Encoding/Decoding or low-level assembly
- Improper ERC-20/Token Handling (e.g., approval logic, flash loan vectors, token locking)
- Logic flaws in financial operations or state transitions
- Edge-case handling in function input validation
- Potential denial-of-service vectors related to gas usage and transaction execution

During this stage, we clearly document any discovered vulnerabilities, including detailed descriptions, precise severity ratings, and recommendations for secure implementation. In situations where it is not clear how the vulnerability might be exploited we may also include a detailed proof-of-concept exploit including code snippets and instructions on how the exploit could be performed.

4. Detailed Fix Review and Validation

After the initial audit and your team's subsequent remediation efforts, we perform a comprehensive fix review to ensure the vulnerabilities identified have been effectively resolved.

We verify each fix individually, confirming that:

- Corrections effectively eliminate the security risks
- Changes do not inadvertently introduce new vulnerabilities or regressions
- The fixes align closely with EVM best practices and secure coding guidelines

Our detailed validation ensures that security improvements are robust, complete, and aligned with best practices specific to the EVM development ecosystem.

Confidentiality Notice

This report, including its content, data, and underlying methodologies, is subject to the confidentiality and feedback provisions in your agreement with Adevar Labs. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Adevar Labs.

Legal Disclaimer

The review and this report are provided by Adevar Labs on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Adevar Labs disclaims all warranties, expressed or implied, in connection with this report, its content, and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

You agree that access to and/or use of the report and other results of the review, including any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided.

You acknowledge that blockchain technology remains under development and is subject to unknown risks and flaws. Adevar Labs does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open-source or third-party software, code, libraries, materials, or information accessible through the report. As with the purchase or use of a product or service in any environment, you should use your best judgment and exercise caution where appropriate.

Appendix A: Operational Security for Privileged EWA Roles

These recommendations are not contract vulnerabilities. They are controls for safely operating the privileged roles that the EWA system intentionally relies on.

1. Multisig signer model

The collection owner, NFT owner, incentive owner, factory owner, and proxy upgrade owners should be controlled by multisigs or equivalent governance accounts. The threshold only has value if the signers are genuinely independent. In practice, that means hardware-backed production wallets, offline seed storage, separate devices, separate password managers, and a clear process for signer rotation after departures, device loss, suspected compromise, or long inactivity.

Signer independence should be reviewed as part of launch readiness. A 3-of-5 Safe controlled by five people using the same laptop fleet, email provider, browser wallet pattern, and recovery process is weaker than it looks on-chain.

2. Proposal verification ceremony

Every privileged transaction should go through a small, repeatable review ceremony before signatures are collected. The reviewer should confirm the chain ID, target address, function selector, decoded calldata, token address, token decimals, amount, recipient, expected balance movement, and any owner, module, guard, registry, or implementation change.

For EWA-specific actions, the review should also cover draw IDs, external IDs, employee addresses, payment tokens, settlement amounts, and expected payroll dates. For upgrades, the reviewer should check the implementation address, bytecode hash, storage layout diff, initializer or reinitializer calldata, and post-upgrade smoke test plan.

For high-risk proposals, one person should prepare the transaction and a different person should independently decode it from raw calldata. Signers should compare the final transaction hash in their wallet before signing.

3. Offline transaction review tool

PACT would benefit from a small offline or read-only transaction review tool for privileged proposals. The tool should not hold keys and should not broadcast transactions. Its job is to make blind signing harder.

The useful version of this tool is not complicated. It takes a Safe transaction JSON, chain ID, nonce, target address, calldata, and value. It resolves the target address against a pinned address book, decodes calldata using pinned ABIs from the reviewed source version, and prints the intended action in plain language. For known contracts, it should show token movements, owner changes, registry changes, upgrade calls, module changes, guard changes, and the final transaction hash that signers should compare in their wallet.

For higher-risk operations, the tool should simulate the proposal on a fork and show the relevant state diff. It should fail closed when the ABI, chain ID, address label, or expected implementation hash is missing.

4. Upgrade and deployment runbooks

The reviewed contracts use UUPS upgrades. Upgrades are normal for this system, but they should be treated like production releases, not ad hoc admin transactions.

Each upgrade runbook should include the source commit or release tag, compiler version, optimizer settings, EVM target, implementation bytecode hash, proxy address, current implementation address, storage layout diff, initializer calldata if any, named approvers, post-upgrade smoke tests, and a rollback or forward-fix plan.

Factory implementation registration should follow the same discipline. A wrong factory implementation affects future deployments, even if it does not mutate existing proxies.

5. Safe modules and guards

Safe modules and guards can change the effective security model of a multisig. Adding or changing one should be treated as seriously as a protocol upgrade.

The production Safe should stay as simple as possible. If a module or guard is needed, it should be reviewed, simulated, monitored, and paired with a tested recovery path. Safe's documentation warns that a broken guard can block Safe transaction execution, so the recovery path should not be theoretical.

6. Production monitoring

Monitoring should cover both protocol events and privileged-account events. At minimum, alerts should cover ownership transfers, UUPS upgrades, implementation changes, **FundingWithdrawn**, **IncentiveFundingWithdrawn**, large draw issuance, large settlement, markdown activity, incentive ratio changes, incentive cap changes, identity registry proposals, registry acceptance, registry disable proposals, unexpected ERC-20 balances left in EWA collections, Safe owner changes, threshold changes, module changes, and guard changes.

The point is not to create a dashboard that someone checks occasionally. Alerts should route to people who can act, with enough decoded context to decide whether the event is expected.

7. Incident response

PACT should keep short runbooks for the incidents that matter most: compromised signer, wrong identity registry accepted, bad upgrade deployed, incentive ratio misconfigured, wrong employee or amount used during draw issuance, settlement routed to an unexpected NFT holder, and Vehicle integration failure.

Each runbook should name the responsible team, expected response time, safe actions, dangerous actions, and communications path. Because the EWA collection has no pause function, the practical response options are ownership rotation, upgrade, funding withdrawal, registry correction, and an operational halt outside the contract.

8. Treasury exposure limits

The collection owner can withdraw contract-held funding and issue draws, so the collection should not hold materially more working capital than needed. Collections should be funded in planned tranches, reconciled daily during active operation, and monitored for balances above expected working capital. Old incentive contracts should not be left with unused reward funding.

Owner-account allowances to the EWA collection are also worth reviewing. Large stale allowances are not a protocol bug, but they are a common operational footgun.

9. Tooling note

If the team uses OpenZeppelin Defender as part of its operational process, it should confirm a migration plan. OpenZeppelin's current documentation says Defender remains operational until July 1, 2026, with migration paths toward open-source Monitor and Relayer tooling.

References

Safe guard reference: <https://docs.safe.global/reference-smart-account/guards/setGuard>

Safe module overview: <https://docs.safefoundation.org/smart-account/modules>

OpenZeppelin access control overview: <https://docs.openzeppelin.com/contracts/5.x/access-control>

OpenZeppelin Defender overview and migration notice: <https://docs.openzeppelin.com/defender>

Appendix B: EWA Monitoring, Reconciliation, and Data Hygiene

This appendix is EWA-specific. It is meant to help the team operate the system without relying on manual memory or post-incident chain archaeology.

1. Reconcile payroll records to on-chain draws

Each issued draw should be tied to a complete off-chain record containing the on-chain **drawId**, opaque **externalId**, employer address, employee payout address, payment token, draw amount, expected payroll date, issuance transaction hash, and current NFT owner if the receivable has moved.

Each settlement should be reconciled against the payroll deduction amount, on-chain amount applied, cumulative paid amount, remaining principal, settlement recipient, and token balance movement. The important operational check is simple: payroll records and on-chain state should agree on what is still expected, what has been recovered, and what has been formally marked down.

2. Monitor accounting aggregates

For each payment token, operators should monitor **totalDisbursed**, **totalExpected**, **totalSettled**, **totalMarkedDown**, and the EWA collection's token balance. The intended aggregate relationship is:

totalDisbursed == totalSettled + totalExpected + totalMarkedDown

This relationship should hold through issue, settlement, and markdown. A separate business metric is still useful: overdue-but-unmarked principal. Those amounts remain in **totalExpected** until **markDown** is called, even if they are already overdue from an operational point of view.

3. Keep public metadata opaque

The chain permanently exposes draw amount, employer address, employee address, payment token, expected payroll date, settlement timing, NFT owner, and the raw **externalId** emitted in **DrawIssued**.

The safest policy is to use random opaque IDs on-chain and keep the mapping to payroll records in a restricted off-chain system. Names, payroll IDs, invoice IDs, customer IDs, emails, phone numbers, and predictable references should not appear in **externalId**. The off-chain mapping should be encrypted, access-controlled, and treated as sensitive payroll metadata.

4. Receiver-type integration tests

Before production deployment, settlement behavior should be tested against the receiver types the system expects to see: EOAs, Safe or other smart contract wallets, canonical Vehicles that pull funds correctly, Vehicles that revert, and Vehicle-like contracts that return success without pulling funds.

Those tests should confirm token movement, event emission, allowance cleanup, and reconciliation behavior. They are especially useful because the receiver path is where on-chain settlement meets external account design.

5. Batch settlement runbook

Batch settlement is efficient, but it concentrates many accounting updates into one transaction. Operators should treat large batches like small releases.

The batch should be built from a deterministic export, checked for a single payment token, checked for nonzero amounts, checked against remaining principal, simulated before execution, and reconciled after execution. The input file, simulation result, transaction hash, and reconciliation report should be saved together. If gas, receiver behavior, or review clarity becomes uncomfortable, the batch should be split.

Large settlement batches should also account for receiver-level failure modes. If a batch includes a Vehicle receiver, settlement may call **receivePayment** on that receiver. A reverting or gas-consuming receiver can cause the whole batch transaction to fail, even if the other draws in the batch are valid. This does not let the receiver steal additional funds, but it can delay settlement processing.

Operationally, new, recently upgraded, or previously failing Vehicle receivers should be processed in smaller batches or standalone transactions first. This keeps one bad receiver from delaying an otherwise clean settlement batch.

6. Incentive funding controls

The incentive contract pays the employee recorded in the EWA draw. Claims are permissionless, so funding and reward terms should be monitored instead of assumed.

Operators should track incentive funding runway against outstanding unclaimed rewards, alert when the incentive balance falls below expected claims, review ratio and max cap changes before signing, and reconcile **IncentiveClaimed** events against employee reward records. Incentive tokens should follow the same standard-token expectation unless the contract is changed to enforce received amounts.

7. Identity registry operations

Identity registry changes affect whether receivable NFTs can move. The one-day timelock is useful only if someone watches it.

A registry proposal should trigger immediate review of the proposed address and expected **isVerified** behavior before the timelock expires. Registry disable proposals should be treated as high-signal events, and the team should keep a documented process for cancelling bad proposals before acceptance.