



July 10, 2023

SMART CONTRACT AUDIT REPORT

Sablier PRB
Proxy



omniscia.io



info@omniscia.io



Online report: [sablier-prb-proxy](https://sablier-prb-proxy.sablier.tech)

PRB Proxy Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
6fbb6df62d	June 7th 2023	395bafc4c4
82d14b9d0b	July 6th 2023	36e3ade808
27594d50f0	July 10th 2023	fea4aaae24

Audit Overview

We were tasked with performing an audit of the Sablier codebase and in particular their `PRBProxy` system permitting "smart wallets" to be created that act on behalf of EOAs akin to the `DSPProxy` implementation of the Maker protocol.

Over the course of the audit, we identified multiple vulnerabilities of significant severity that arise from on-chain race conditions as well as insufficient protection measures in place by the `PRBProxy`.

We advise the Sablier team to closely evaluate all minor-and-above findings identified in the report and promptly remediate them as well as consider all optimizational exhibits identified in the report.

Post-Audit Conclusion

The Sablier team iterated through all findings within the report and provided us with a revised commit hash to evaluate all exhibits on.

We evaluated all alleviations performed by Sablier and have identified that all exhibits have been adequately dealt with no outstanding issues remaining in the report.

Contracts Assessed

Files in Scope	Repository	Commit(s)
PRBProxy.sol (PRB)	prb-proxy	6fbb6df62d, 82d14b9d0b, 27594d50f0
PRBProxyAnnex.sol (PRP)	prb-proxy	6fbb6df62d, 82d14b9d0b, 27594d50f0
PRBProxyPlugin.sol (PRN)	prb-proxy	6fbb6df62d, 82d14b9d0b, 27594d50f0
PRBProxyStorage.sol (PRS)	prb-proxy	6fbb6df62d, 82d14b9d0b, 27594d50f0
PRBProxyRegistry.sol (PRR)	prb-proxy	6fbb6df62d, 82d14b9d0b, 27594d50f0

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
<div><div></div>Unknown</div>	0	0	0	0
<div><div></div>Informational</div>	7	7	0	0
<div><div></div>Minor</div>	1	0	0	1
<div><div></div>Medium</div>	1	1	0	0
<div><div></div>Major</div>	4	4	0	0

During the audit, we filtered and validated a total of **2 findings utilizing static analysis** tools as well as identified a total of **11 findings during the manual review** of the codebase. We strongly recommend that any minor severity or higher findings are dealt with promptly prior to the project's launch as they can introduce potential misbehaviours of the system as well as exploits.

Compilation

The project utilizes `foundry` as its development pipeline tool, containing an array of tests and scripts coded in Solidity.

To compile the project, the `build` command needs to be issued via the `forge` CLI tool:

```
BASH
```

```
forge build
```

The `forge` tool automatically selects Solidity version `0.8.19` based on the version specified within the `foundry.toml` file.

The project contains discrepancies with regards to the Solidity version used as the `pragma` statements of the contracts are open-ended (`>=0.8.18`).

We advise them to be locked to `0.8.19` (`=0.8.19`), the same version utilized for our static analysis as well as optimizational review of the codebase.

During compilation with the `foundry` pipeline, no errors were identified that relate to the syntax or bytecode size of the contracts.

Static Analysis

The execution of our static analysis toolkit identified **17 potential issues** within the codebase of which **13 were ruled out to be false positives** or negligible findings.

The remaining **4 issues** were validated and grouped and formalized into the **2 exhibits** that follow:

ID	Severity	Addressed	Title
PRB-01S	<div><div></div>Informational</div>	<div><div></div>Nullified</div>	Illegible Numeric Value Representation
PRR-01S	<div><div></div>Minor</div>	<div><div></div>Acknowledged</div>	Inexistent Sanitization of Input Addresses

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Sablier's special-purpose proxy system.

As the project at hand implements an account abstraction protocol, intricate care was put into ensuring that the **access control flow within the system conforms to the specifications and restrictions** laid forth within the protocol's specification.

We validated that **all state transitions of the system occur within sane criteria** and that all rudimentary formulas within the system execute as expected. We **pinpointed multiple race-condition vulnerabilities** within the system which could have had **severe ramifications** to its overall operation as well as **a significant denial-of-service attack** arising from the system's gas stipend management.

Additionally, the system was investigated for any other commonly present attack vectors such as re-entrancy attacks, mathematical truncations, logical flaws and **ERC / EIP** standard inconsistencies. The documentation of the project was satisfactory to an exemplary extent, containing comprehensive documentation in the `interface` declaration of each contract within the scope of the audit.

A total of **11 findings** were identified over the course of the manual review of which **6 findings** concerned the behaviour and security of the system. The non-security related findings, such as optimizations, are included in the separate **Code Style** chapter.

The finding table below enumerates all these security / behavioural findings:

ID	Severity	Addressed	Title
PRB-01M	Major	Yes	Insufficient Protection Against Hostile Takeover
PRB-02M	Major	Yes	Proxy Denial of Service Attack
PRP-01M	Informational	Nullified	Suboptimal Error Handling
PRR-01M	Medium	Yes	Inexistent Verification of Acceptance
PRR-02M	Major	Yes	Insecure Deployment Methodology
PRR-03M	Major	Yes	Insecure Post-Execution Transfer

Code Style

During the manual portion of the audit, we identified **5 optimizations** that can be applied to the codebase that will decrease the operational cost associated with the execution of a particular function and generally ensure that the project complies with the latest best practices and standards in Solidity.

Additionally, this section of the audit contains any opinionated adjustments we believe the code should make to make it more legible as well as truer to its purpose.

These optimizations are enumerated below:

ID	Severity	Addressed	Title
PRB-01C	<div><div></div>Informational</div>	<div><div></div>Nullified</div>	Redundant Function Declaration
PRP-01C	<div><div></div>Informational</div>	<div><div></div>Nullified</div>	Inefficient Event Emission
PRR-01C	<div><div></div>Informational</div>	<div><div></div>Nullified</div>	Non-Uniform Invocation Style
PRR-02C	<div><div></div>Informational</div>	<div><div></div>Yes</div>	Redundant Code Duplication
PRR-03C	<div><div></div>Informational</div>	<div><div></div>Nullified</div>	Redundant Usage of <code>bytes32</code> Type

PRBProxy Static Analysis Findings

PRB-01S: Illegible Numeric Value Representation

Type	Severity	Location
Code Style	<div><div></div>Informational</div>	PRBProxy.sol:L40

Description:

The linked representation of a numeric literal is sub-optimally represented decreasing the legibility of the codebase.

Example:

```
src/PRBProxy.sol
SOL
40 minGasReserve = 5000;
```

Recommendation:

To properly illustrate the value's purpose, we advise the following guidelines to be followed. For values meant to depict fractions with a base of `1e18`, we advise fractions to be utilized directly (i.e. `1e17` becomes `0.1e18`) as they are supported. For values meant to represent a percentage base, we advise each value to utilize the underscore (`_`) separator to discern the percentage decimal (i.e. `10000` becomes `100_00`, `300` becomes `3_00` and so on). Finally, for large numeric values we simply advise the underscore character to be utilized again to represent them (i.e. `1000000` becomes `1_000_000`).

Alleviation (82d14b9d0b0f6be3df43268c41cffec4701368f8):

The referenced numeric literal is no longer present in the codebase rendering this exhibit no longer applicable.

As an added note, the codebase was utilizing standard formatting rules of the `forge` compilation tool meaning that the exhibit would have been nullified in any case as the `number_threshold` rule deliberately did not introduce an underscore character to the highlighted literal.

PRBProxyRegistry Static Analysis Findings

PRR-01S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	<div><div></div>Minor</div>	PRBProxyRegistry.sol:L72, L91, L104

Description:

The linked function(s) accept `address` arguments yet do not properly sanitize them.

Impact:

The presence of zero-value addresses, especially in `constructor` implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

Example:

```
src/PRBProxyRegistry.sol
```

```
104 function transferOwnership(address newOwner) external override noProxy(newOwner) {
105     // Check that the caller has a proxy.
106     IPRBProxy proxy = proxies[msg.sender];
107     if (address(proxy) == address(0)) {
108         revert PRBProxyRegistry_OwnerDoesNotHaveProxy({ owner: msg.sender });
109     }
110
111     // Delete the proxy for the caller.
112     delete proxies[msg.sender];
113
114     // Set the proxy for the new owner.
115     proxies[newOwner] = proxy;
116
117     // Transfer the proxy.
118     proxy.transferOwnership(newOwner);
119
120     // Log the transfer of the proxy ownership.
121     emit TransferOwnership({ proxy: proxy, oldOwner: msg.sender, newOwner: newOwner })
122 }
```

Recommendation:

We advise some basic sanitization to be put in place by ensuring that each `address` specified is non-zero.

Alleviation (82d14b9d0b0f6be3df43268c41cffec4701368f8):

All but the `PRBProxyRegistry::deployFor` functions have been removed from the code as part of alleviations for other exhibits outlined in the audit report.

The Sablier team evaluated the remaining function and proceeded to retain the current behaviour in place as they wish to minimize the execution cost of the function and see no foul with a proxy deployed for the zero address.

As a result, we consider this exhibit acknowledged.

PRBProxy Manual Review Findings

PRB-01M: Insufficient Protection Against Hostile Takeover

Type	Severity	Location
Logical Fault	<div><div></div>Major</div>	PRBProxy.sol:L148

Description:

The `PRBProxy::_safeDelegateCall` function will attempt to protect the proxy against a hostile takeover by ensuring that the `owner` (and indirectly the `registry` given that it is `immutable`) remain the same after the execution of the `delegatecall` function. The issue with this mechanism is that it does not protect the `permissions` mapping which can be arbitrarily updated and thus cause undetected entries within it to permit a malicious user to take over the contract by performing arbitrary actions.

Impact:

As a "silent" contract can be `delegatecall`-ed to that updates `permissions` entries without emitting an event, it would be practically impossible for every user to track their proxy's storage changes and eliminate malicious `permissions` entries. As such, we consider the severity of this exhibit to be "major".

Example:

```
src/PRBProxy.sol
```



```
135 /// @notice Performs a DELEGATECALL to the provided address with the provided data.
136 /// @dev Shared logic between the {execute} and the {fallback} functions.
137 function _safeDelegateCall(address to, bytes memory data) internal returns (bool success) {
138     // Save the owner address in memory so that this variable cannot be modified during the call.
139     address owner_ = owner;
140
141     // Reserve some gas to ensure that the contract call will not run out of gas.
142     uint256 stipend = gasleft() - minGasReserve;
143
144     // Delegate call to the provided contract.
145     (success, response) = to.delegatecall{ gas: stipend }(data);
146
147     // Check that the owner has not been changed.
148     if (owner_ != owner) {
149         revert PRBProxy_OwnerChanged(owner_, owner);
150     }
151 }
```

Recommendation:

We advise the `storage` space of the `PRBProxy` to instead lie on a separate contract. In such a case, it is possible to mark the separate contract as "non-entrant" and thus prevent any `PRBProxy::_safeDelegateCall` from affecting the storage variables of the `PRBProxy` instance.

To allow configurability of the `PRBProxy`, the special `PRBProxyAnnex` contract could be marked as allowed to alter the storage space of the secondary contract.

These security measures would significantly increase the security of the `PRBProxy` while not sacrificing its flexibility.

Alleviation (82d14b9d0b0f6be3df43268c41cffe4701368f8):

The `owner` of a proxy is no longer adjustable as it is exposed via an `immutable` variable data entry. As an added security measure, the `permissions` mapping was relocated to the `PRBProxyRegistry` contract and is solely adjustable by the owner of a proxy.

As a result of these actions, `delegatecall` instructions can no longer result in a hostile takeover as they cannot affect any of the permission-related data entries thus alleviating this exhibit in full.

As a final point, the `PRBProxy::_safeDelegateCall` implementation was rendered redundant by these adjustments optimizing the gas cost of `PRBProxy::_execute` calls given that they now perform a direct `delegatecall` instruction.

PRB-02M: Proxy Denial of Service Attack

Type	Severity	Location
Logical Fault	<div><div></div>Major</div>	PRBProxy.sol:L142

Description:

The `PRBProxyAnnex::setMinGasReserve` function will not enforce a maximum value for the `newMinGasReserve` value, permitting it to be arbitrarily set. This value is utilized within `PRBProxy::_safeDelegateCall` invocations to ensure a safe gas stipend remains in the contract after the `delegatecall` instruction.

By setting this value abnormally high, the `gasleft() - minGasReserve` subtraction can be forced to underflow and thus cause all invocations of `PRBProxy::_safeDelegateCall` to fail effectively disabling the proxy as it will not be able to execute any more `delegatecall` instructions.

Coupled with the fact that the `PRBProxyRegistry` permits a proxy to be deployed for someone else and to execute an arbitrary payload, it is possible to create proxies on behalf of other users that will be unusable.

To note, simply updating the `PRBProxyAnnex::setMinGasReserve` function is insufficient as the `PRBProxyRegistry::deployAndExecuteFor` function permits an arbitrary contract to be executed.

Impact:

The combination of functions mentioned in the description of this exhibit arise the severity of this exhibit to major as it is actively exploitable.

Example:

```
src/PRBProxy.sol
```

```
135 /// @notice Performs a DELEGATECALL to the provided address with the provided data.
136 /// @dev Shared logic between the {execute} and the {fallback} functions.
137 function _safeDelegateCall(address to, bytes memory data) internal returns (bool success) {
138     // Save the owner address in memory so that this variable cannot be modified during the call.
139     address owner_ = owner;
140
141     // Reserve some gas to ensure that the contract call will not run out of gas.
142     uint256 stipend = gasleft() - minGasReserve;
143
144     // Delegate call to the provided contract.
145     (success, response) = to.delegatecall{ gas: stipend }(data);
146
147     // Check that the owner has not been changed.
148     if (owner_ != owner) {
149         revert PRBProxy_OwnerChanged(owner_, owner);
150     }
151 }
```

Recommendation:

As the `minGasReserve` variable lies within the state, any contract that is `delegatecall`-ed to can update it. Based on this, the only solution to this issue is to perform the subtraction opportunistically and to use a default gas stipend value if the subtraction would fail.

Alleviation (82d14b9d0b0f6be3df43268c41cffe4701368f8):

The Sablier team evaluated this exhibit and opted to eliminate the `minGasReserve` concept entirely. As such, the vulnerability described is no longer feasible addressing the exhibit in full.

PRBProxyAnnex Manual Review Findings

PRP-01M: Suboptimal Error Handling

Type	Severity	Location
Input Sanitization	<div><div></div>Informational</div>	PRBProxyAnnex.sol:L46, L50-L52

Description:

The `PRBProxyAnnex::installPlugin` function is meant to yield an informative error in case the `plugin` specified in the function has no entries in its `PRBProxyPlugin::methodList`, however, the function does not gracefully handle the case whereby a `plugin` with no `PRBProxyPlugin::methodList` function is supplied.

Impact:

While the `PRBProxyAnnex::installPlugin` function contains special code to handle the case whereby a plugin has no methods defined, it does not properly handle the case whereby a `plugin` with no `PRBProxyPlugin::methodList` function is provided.

Given that a scenario of an empty method list is less likely than an unimplemented function, we advise the Sablier team to integrate this exhibit's recommendation to the code.

Example:

```
src/PRBProxyAnnex.sol
SOL
44 function installPlugin(IPRBProxyPlugin plugin) external override {
45     // Get the method list to install.
46     bytes4[] memory methodList = plugin.methodList();
47
48     // The plugin must have at least one listed method.
49     uint256 length = methodList.length;
50     if (length == 0) {
51         revert PRBProxy_NoPluginMethods(plugin);
52     }
```

Recommendation:

We advise such a case to be handled properly by enclosing the `PRBProxyPlugin::methodList` function invocation in a `try-catch` clause, depicting an informative error message in the `catch` clause introduced.

Alleviation (82d14b9d0b0f6be3df43268c41cffec4701368f8):

The `PRBProxyAnnex` contract is no longer part of the codebase rendering exhibits in relation to it no longer applicable.

PRBProxyRegistry Manual Review Findings

PRR-01M: Inexistent Verification of Acceptance

Type	Severity	Location
Logical Fault	<div><div></div>Medium</div>	PRBProxyRegistry.sol:L103-L122

Description:

The `PRBProxyRegistry::transferOwnership` function works in a "push" pattern manner whereby a user can "push" their proxy to another user without their consent.

A side-effect of this is that it is possible for a user to create a proxy, setup malicious `permissions` entries, and proceed to transfer it to an unsuspecting user.

Impact:

As the `PRBProxyRegistry` permits a single proxy to be created per user, an on-chain race condition manifests whenever a proxy creation transaction is created whereby another user can simply transfer the ownership of their proxy to the user attempting to create a new one, causing their original transaction to fail.

This process can be repeated whenever the targeted user attempts to transfer their proxy elsewhere (to create a new, proper one) by creating proxies to the intended recipients of these transfers. As such, it is possible to effectively force a user to retain a single proxy they never wished to create via these on-chain race conditions.

Example:

```
src/PRBProxyRegistry.sol
```



```
103 /// @inheritdoc IPRBProxyRegistry
104 function transferOwnership(address newOwner) external override noProxy(newOwner) {
105     // Check that the caller has a proxy.
106     IPRBProxy proxy = proxies[msg.sender];
107     if (address(proxy) == address(0)) {
108         revert PRBProxyRegistry_OwnerDoesNotHaveProxy({ owner: msg.sender });
109     }
110
111     // Delete the proxy for the caller.
112     delete proxies[msg.sender];
113
114     // Set the proxy for the new owner.
115     proxies[newOwner] = proxy;
116
117     // Transfer the proxy.
118     proxy.transferOwnership(newOwner);
119
120     // Log the transfer of the proxy ownership.
121     emit TransferOwnership({ proxy: proxy, oldOwner: msg.sender, newOwner: newOwner })
122 }
```

Recommendation:

We advise the code to use a "pull" pattern instead whereby a transfer is initiated by the existing `owner` and accepted by the `newOwner`.

The acceptance mechanism would need to apply the `PRBProxyRegistry::noProxy` modifier and as an added feature the code could also permit a direct `PRBProxyRegistry::transferOwnership` to happen by validating an ECDSA signature by the `newOwner`.

Alleviation (82d14b9d0b0f6be3df43268c41cffe4701368f8):

To alleviate this exhibit, the Sablier team assessed that removal of ownership transferability is the most suitable choice.

Ownership transferability was consequently removed from the `PRBProxyRegistry` eliminating this vulnerability.

PRR-02M: Insecure Deployment Methodology

Type	Severity	Location
Logical Fault	<div><div></div>Major</div>	PRBProxyRegistry.sol:L72-L74, L90-L101

Description:

The `PRBProxyRegistry::deployAndExecuteFor` function permits a proxy to be deployed for another user and an arbitrary `delegatecall` operation to be performed.

This combination is dangerous and should simply not be exposed by the system as it enables an on-chain race condition whereby a user's original proxy deployment transaction is detected and a malicious user deploys a proxy on their behalf with an arbitrary payload that can even set malicious `permissions` entries, allowing a hostile takeover of the proxy to be performed at a later date if undetected.

Impact:

It is possible for an on-chain race condition to manifest whereby a proxy deployment transaction is detected and it is superseded by a malicious proxy creation transaction with malicious storage adjustments, such as `permissions`, `plugins`, `minGasReserve` etc. values.

Example:

```
src/PRBProxyRegistry.sol
```

```

66  /// @inheritdoc IPRBProxyRegistry
67  function deploy() external override noProxy(msg.sender) returns (IPRBProxy proxy) {
68      proxy = _deploy({ owner: msg.sender });
69  }
70
71  /// @inheritdoc IPRBProxyRegistry
72  function deployFor(address owner) public override noProxy(owner) returns (IPRBProxy pr
73      proxy = _deploy(owner);
74  }
75
76  /// @inheritdoc IPRBProxyRegistry
77  function deployAndExecute(
78      address target,
79      bytes calldata data
80  )
81      external
82      override
83      noProxy(msg.sender)
84      returns (IPRBProxy proxy, bytes memory response)
85  {
86      (proxy, response) = _deployAndExecute({ owner: msg.sender, target: target, data: d
87  }
88
89  /// @inheritdoc IPRBProxyRegistry
90  function deployAndExecuteFor(
91      address owner,
92      address target,
93      bytes calldata data
94  )
95      public
96      override
97      noProxy(owner)
98      returns (IPRBProxy proxy, bytes memory response)
99  {
100      (proxy, response) = _deployAndExecute(owner, target, data);
101  }

```

Recommendation:

We advise the `PRBProxyRegistry::deployAndExecuteFor` methodology to validate an ECDSA signature of the `owner` that authorizes the caller to perform the deployment and arbitrary execution, guaranteeing that the `owner` is aware of the proxy being deployed and has authorized the payload to be executed.

The same authorization mechanism should be employed for `PRBProxyRegistry::deployFor` invocations although these are not vulnerable to the same degree as `PRBProxyRegistry::deployAndExecuteFor` invocations.

Alleviation (82d14b9d0b0f6be3df43268c41cffe4701368f8):

The Sablier team evaluated this exhibit and removed the `PRBProxyRegistry::deployAndExecuteFor` function.

The `PRBProxyRegistry::deployFor` function has remained in the codebase as-is given that introducing an ECDSA recovery mechanism would defeat the function's intended use-case per the Sablier team's statements.

Based on the fact that the `PRBProxyRegistry::deployFor` function has a minimal and not directly exploitable attack surface, we consider this exhibit sufficiently alleviated as the `PRBProxyRegistry::deployAndExecuteFor` function was properly removed from the codebase.

PRR-03M: Insecure Post-Execution Transfer

Type	Severity	Location
Logical Fault	<div><div></div>Major</div>	PRBProxyRegistry.sol:L192, L195

Description:

The `PRBProxyRegistry::_deployAndExecute` function will insecurely transfer the ownership of the proxy after an arbitrary `delegatecall` instruction has been performed.

Given that no reentrancy protection measures are in place, it is possible for a `PRBProxyRegistry::_deployAndExecute` call to re-enter the `PRBProxyRegistry` contract and execute a `PRBProxyRegistry::transferOwnership` operation.

The end result would be a `proxies` entry of the `newOwner` pointing to the `proxy` that was just created whilst the `proxy.owner()` value would be the original deployer, invalidating the `proxies[owner] == proxy.owner()` logical assumption the system relies on.

Impact:

It is possible to create a proxy on behalf of another user while retaining ownership rights of it.

Example:

```
src/PRBProxyRegistry.sol
```

SOL

```
182 // Set the proxy for the owner.
183 proxies[owner] = proxy;
184
185 // Increment the seed.
186 // We're using unchecked arithmetic here because this cannot realistically overflow, e
187 unchecked {
188     nextSeeds[tx.origin] = bytes32(uint256(seed) + 1);
189 }
190
191 // Delegate call to the target contract.
192 response = proxy.execute(target, data);
193
194 // Transfer the ownership to the specified owner.
195 proxy.transferOwnership(owner);
```

Recommendation:

We advise the code of `PRBProxyRegistry::_deployAndExecute` to ensure that the `proxies[owner]` still points to the `proxy` after the execution of `PRBProxy::execute`, alleviating the vulnerability described by this exhibit.

Alleviation (82d14b9d0b0f6be3df43268c41cffec4701368f8):

The execution logic during a proxy's instantiation has been relocated to the proxy itself which evaluates the `constructorParams` present in its deployer (the `PRBProxyRegistry` contract) and utilizes them to perform the call.

The actual alleviation of this exhibit is the removal of the `nextSeeds` concept. In the latest reviewed implementation, the deployment `salt` of a proxy is the `owner` itself rendering re-deployments for the same `owner` impossible even if the `_proxies` data entry has not been written yet within the `PRBProxyRegistry::_deploy` function.

As a result of these changes, we consider this exhibit no longer exploitable.

PRBProxy Code Style Findings

PRB-01C: Redundant Function Declaration

Type	Severity	Location
Gas Optimization	<div><div></div>Informational</div>	PRBProxy.sol:L80

Description:

The `PRBProxy::receive` function is declared alongside a `PRBProxy::fallback` declaration that is payable .

Example:

```
src/PRBProxy.sol
```

```
49 /// @notice Fallback function used to run plugins.
50 /// @dev WARNING: anyone can call this function and thus run any installed plugin.
51 fallback(bytes calldata data) external payable returns (bytes memory response) {
52     // Check if the function signature exists in the installed plugins mapping.
53     IPRBProxyPlugin plugin = plugins[msg.sig];
54     if (address(plugin) == address(0)) {
55         revert PRBProxy_PluginNotInstalledForMethod({ caller: msg.sender, selector: ms
56     }
57
58     // Delegate call to the plugin.
59     bool success;
60     (success, response) = _safeDelegateCall(address(plugin), data);
61
62     // Log the plugin run.
63     emit RunPlugin(plugin, data, response);
64
65     // Check if the call was successful or not.
66     if (!success) {
67         // If there is return data, the call reverted with a reason or a custom error.
68         if (response.length > 0) {
69             assembly {
70                 let returndata_size := mload(response)
71                 revert(add(32, response), returndata_size)
72             }
73         } else {
74             revert PRBProxy_PluginReverted(plugin);
75         }
76     }
77 }
78
79 /// @dev Called when `msg.value` is not zero and the call data is empty.
80 receive() external payable { }
```

Recommendation:

We advise the `PRBProxy::fallback` implementation to be solely utilized, potentially with a simple conditional at the beginning that evaluates whether `msg.sig` is empty in which case the `PRBProxy::receive` flow would have been executed.

Alleviation (82d14b9d0b0f6be3df43268c41cffe4701368f8):

The Sablier team evaluated this exhibit and decided to retain the current split of logic in place so as to minimize logic clutter and ensure that each declaration is responsible for its dedicated purpose.

We agree with Sablier's sentiment and given that this exhibit would in fact lead to an increase in execution cost (with a miniscule decrease in deployment cost), we consider it in err and thus nullified.

PRBProxyAnnex Code Style Findings

PRP-01C: Inefficient Event Emission

Type	Severity	Location
Gas Optimization	<div><div></div>Informational</div>	PRBProxyAnnex.sol:L75

Description:

The `PRBProxyAnnex::setMinGasReserve` function will store a storage variable to a local variable, overwrite the storage variable, and ultimately emit an event of the old and new storage values from memory.

Example:

src/PRBProxyAnnex.sol

SOL

```
66  /// @inheritdoc IPRBProxyAnnex
67  function setMinGasReserve(uint256 newMinGasReserve) external override {
68      // Load the current minimum gas reserve.
69      uint256 oldMinGasReserve = minGasReserve;
70
71      // Update the minimum gas reserve.
72      minGasReserve = newMinGasReserve;
73
74      // Log the minimum gas reserve update.
75      emit SetMinGasReserve(oldMinGasReserve, newMinGasReserve);
76  }
```

Recommendation:

We advise the code to `emit` the `SetMinGasReserve` event directly before overwriting it, permitting the `minGasReserve` variable to be utilized directly and the `oldMinGasReserve` variable to be rendered redundant, optimizing the function's gas cost.

Alleviation (82d14b9d0b0f6be3df43268c41cffec4701368f8):

The `PRBProxyAnnex` contract is no longer part of the codebase rendering exhibits in relation to it no longer applicable.

PRBProxyRegistry Code Style Findings

PRR-01C: Non-Uniform Invocation Style

Type	Severity	Location
Code Style	<div><div></div>Informational</div>	PRBProxyRegistry.sol:L68, L73, L86, L100

Description:

The codebase of `PRBProxyRegistry` will invoke functions using both the index-based and the key-value-based argument invocation methods.

Example:

src/PRBProxyRegistry.sol

SOL

```
66  /// @inheritdoc IPRBProxyRegistry
67  function deploy() external override noProxy(msg.sender) returns (IPRBProxy proxy) {
68      proxy = _deploy({ owner: msg.sender });
69  }
70
71  /// @inheritdoc IPRBProxyRegistry
72  function deployFor(address owner) public override noProxy(owner) returns (IPRBProxy pr
73      proxy = _deploy(owner);
74  }
```

Recommendation:

We advise either of the two styles to be adopted throughout the codebase ensuring consistency in the code's style.

Alleviation (82d14b9d0b0f6be3df43268c41cffe4701368f8):

As part of remediations for other exhibits outlined in the audit report, the non-uniform invocation styles are no longer present in the codebase meaning this exhibit is no longer applicable.

PRR-02C: Redundant Code Duplication

Type	Severity	Location
Gas Optimization	<div><div></div>Informational</div>	PRBProxyRegistry.sol:L130-L147, L171-L189

Description:

The referenced code blocks are identical with the sole difference being the `transientProxyOwner` being set to `address(this)` for the `PRBProxy::execute` call to succeed.

Example:

```
src/PRBProxyRegistry.sol
```



```

128 /// @dev See the documentation for the public functions that call this internal functi
129 function _deploy(address owner) internal returns (IPRBProxy proxy) {
130     // Load the next seed.
131     bytes32 seed = nextSeeds[tx.origin];
132
133     // Prevent front-running the salt by hashing the concatenation of "tx.origin" and
134     bytes32 salt = keccak256(abi.encode(tx.origin, seed));
135
136     // Deploy the proxy with CREATE2.
137     transientProxyOwner = owner;
138     proxy = new PRBProxy{ salt: salt }();
139     delete transientProxyOwner;
140
141     // Set the proxy for the owner.
142     proxies[owner] = proxy;
143
144     // Increment the seed.
145     // We're using unchecked arithmetic here because this cannot realistically overflow
146     unchecked {
147         nextSeeds[tx.origin] = bytes32(uint256(seed) + 1);
148     }
149
150     // Log the proxy via an event.
151     // forgefmt: disable-next-line
152     emit DeployProxy({
153         origin: tx.origin,
154         operator: msg.sender,
155         owner: owner,
156         seed: seed,
157         salt: salt,
158         proxy: proxy
159     });
160 }
161
162 /// @dev See the documentation for the public functions that call this internal functi
163 function _deployAndExecute(
164     address owner,
165     address target,
166     bytes calldata data
167 )
168     internal
169     returns (IPRBProxy proxy, bytes memory response)
170 {
171     // Load the next seed.
172     bytes32 seed = nextSeeds[tx.origin];
173
174     // We're using unchecked arithmetic here because this cannot realistically overflow

```

173

```
174 // Prevent front-running the salt by hashing the concatenation of "tx.origin" and
175 bytes32 salt = keccak256(abi.encode(tx.origin, seed));
176
177 // Deploy the proxy with CREATE2. The registry will temporarily be the owner of the
178 transientProxyOwner = address(this);
179 proxy = new PRBProxy{ salt: salt }();
180 delete transientProxyOwner;
181
182 // Set the proxy for the owner.
183 proxies[owner] = proxy;
184
185 // Increment the seed.
186 // We're using unchecked arithmetic here because this cannot realistically overflow
187 unchecked {
188     nextSeeds[tx.origin] = bytes32(uint256(seed) + 1);
189 }
```

Recommendation:


We advise the code to refactor both code blocks into an `internal` / `private` function that is invoked by both instances, minimizing code duplication and decreasing the bytecode signature of the `PRBProxyRegistry`.

Alleviation (82d14b9d0b0f6be3df43268c41cffec4701368f8):

The codebase was significantly refactored as part of the proxy ownership overhaul, minimizing code duplication and ensuring that proxies are deployed via a single `PRBProxyRegistry::_deploy` function.

To this end, we consider this exhibit alleviated as the referenced code segments were merged into a single logic block.

PRR-03C: Redundant Usage of `bytes32` Type

Type	Severity	Location
Gas Optimization	 Informational	PRBProxyRegistry.sol:L131, L147

Description:

The `nextSeed` mapping values are constantly cast to a `uint256` to be incremented prior to being re-cast to a `bytes32` value to be stored in the `nextSeeds` mapping.

Example:

src/PRBProxyRegistry.sol

SOL

```
171 // Load the next seed.
172 bytes32 seed = nextSeeds[tx.origin];
173
174 // Prevent front-running the salt by hashing the concatenation of "tx.origin" and the
175 bytes32 salt = keccak256(abi.encode(tx.origin, seed));
176
177 // Deploy the proxy with CREATE2. The registry will temporarily be the owner of the pr
178 transientProxyOwner = address(this);
179 proxy = new PRBProxy{ salt: salt }();
180 delete transientProxyOwner;
181
182 // Set the proxy for the owner.
183 proxies[owner] = proxy;
184
185 // Increment the seed.
186 // We're using unchecked arithmetic here because this cannot realistically overflow, e
187 unchecked {
188     nextSeeds[tx.origin] = bytes32(uint256(seed) + 1);
189 }
```

Recommendation:

We advise the code to store them in their `uint256` format as the `abi.encode` mechanism will generate unique values regardless of whether the `seed` is of a `bytes32` or a `uint256` type.

Alleviation (82d14b9d0b0f6be3df43268c41cffec4701368f8):

The `CREATE2` deployment system was redesigned, no longer requiring the usage of a "seed" in conjunction with the `tx.origin` member.

The `nextSeeds` concept was omitted from the codebase entirely as a result, rendering this exhibit no longer applicable.

Finding Types

A description of each finding type included in the report can be found below and is linked by each respective finding. A full list of finding types Omniscia has defined will be viewable at the central audit methodology we will publish soon.

External Call Validation

Many contracts that interact with DeFi contain a set of complex external call executions that need to happen in a particular sequence and whose execution is usually taken for granted whereby it is not always the case. External calls should always be validated, either in the form of `require` checks imposed at the contract-level or via more intricate mechanisms such as invoking an external getter-variable and ensuring that it has been properly updated.

Input Sanitization

As there are no inherent guarantees to the inputs a function accepts, a set of guards should always be in place to sanitize the values passed in to a particular function.

Indeterminate Code

These types of issues arise when a linked code segment may not behave as expected, either due to mistyped code, convoluted `if` blocks, overlapping functions / variable names and other ambiguous statements.

Language Specific

Language specific issues arise from certain peculiarities that the Solidity language boasts that discerns it from other conventional programming languages. For example, the EVM is a 256-bit machine meaning that operations on less-than-256-bit types are more costly for the EVM in terms of gas costs, meaning that loops utilizing a `uint8` variable because their limit will never exceed the 8-bit range actually cost more than redundantly using a `uint256` variable.

Code Style

An official Solidity style guide exists that is constantly under development and is adjusted on each new Solidity release, designating how the overall look and feel of a codebase should be. In these types of findings, we identify whether a project conforms to a particular naming convention and whether that convention is consistent within the codebase and legible. In case of inconsistencies, we point them out under this category. Additionally, variable shadowing falls under this category as well which is identified when a

local-level variable contains the same name as a contract-level variable that is present in the inheritance chain of the local execution level's context.

Gas Optimization

Gas optimization findings relate to ways the codebase can be optimized to reduce the gas cost involved with interacting with it to various degrees. These types of findings are completely optional and are pointed out for the benefit of the project's developers.

Standard Conformity

These types of findings relate to incompatibility between a particular standard's implementation and the project's implementation, oftentimes causing significant issues in the usability of the contracts.

Mathematical Operations

In Solidity, math generally behaves differently than other programming languages due to the constraints of the EVM. A prime example of this difference is the truncation of values during a division which in turn leads to loss of precision and can cause systems to behave incorrectly when dealing with percentages and proportion calculations.

Logical Fault

This category is a bit broad and is meant to cover implementations that contain flaws in the way they are implemented, either due to unimplemented functionality, unaccounted-for edge cases or similar extraordinary scenarios.

Centralization Concern

This category covers all findings that relate to a significant degree of centralization present in the project and as such the potential of a Single-Point-of-Failure (SPoF) for the project that we urge them to re-consider and potentially omit.

Reentrant Call

This category relates to findings that arise from re-entrant external calls (such as EIP-721 minting operations) and revolve around the inapplicacy of the Checks-Effects-Interactions (CEI) pattern, a pattern that dictates checks (`require` statements etc.) should occur before effects (local storage updates) and interactions (external calls) should be performed last.

Disclaimer

The following disclaimer applies to all versions of the audit report produced (preliminary / public / private) and is in effect for all past, current, and future audit reports that are produced and hosted under Omniscia:

IMPORTANT TERMS & CONDITIONS REGARDING OUR SECURITY AUDITS/REVIEWS/REPORTS AND ALL PUBLIC/PRIVATE CONTENT/DELIVERABLES

Omniscia ("Omniscia") has conducted an independent security review to verify the integrity of and highlight any vulnerabilities, bugs or errors, intentional or unintentional, that may be present in the codebase that were provided for the scope of this Engagement.

Blockchain technology and the cryptographic assets it supports are nascent technologies. This makes them extremely volatile assets. Any assessment report obtained on such volatile and nascent assets may include unpredictable results which may lead to positive or negative outcomes.

In some cases, services provided may be reliant on a variety of third parties. This security review does not constitute endorsement, agreement or acceptance for the Project and technology that was reviewed. Users relying on this security review should not consider this as having any merit for financial advice or technological due diligence in any shape, form or nature.

The veracity and accuracy of the findings presented in this report relate solely to the proficiency, competence, aptitude and discretion of our auditors. Omniscia and its employees make no guarantees, nor assurance that the contracts are free of exploits, bugs, vulnerabilities, deprecation of technologies or any system / economical / mathematical malfunction.

This audit report shall not be printed, saved, disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Omniscia.

All the information/opinions/suggestions provided in this report does not constitute financial or investment advice, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report.

Information in this report is provided 'as is'. Omniscia is under no covenant to the completeness, accuracy or solidity of the contracts reviewed. Omniscia's goal is to help reduce the attack vectors/surface and the high level of variance associated with utilizing new and consistently changing technologies.

Omniscia in no way claims any guarantee, warranty or assurance of security or functionality of the technology that was in scope for this security review.

In no event will Omniscia, its partners, employees, agents or any parties related to the design/creation of this security review be ever liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this security review.

Cryptocurrencies and all other technologies directly or indirectly related to cryptocurrencies are not standardized, highly prone to malfunction and extremely speculative by nature. No due diligence and/or safeguards may be insufficient and users should exercise maximum caution when participating and/or investing in this nascent industry.

The preparation of this security review has made all reasonable attempts to provide clear and actionable recommendations to the Project team (the "client") with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts in scope for this engagement.

It is the sole responsibility of the Project team to provide adequate levels of test and perform the necessary checks to ensure that the contracts are functioning as intended, and more specifically to ensure that the functions contained within the contracts in scope have the desired intended effects, functionalities and outcomes, as documented by the Project team.

All services, the security reports, discussions, work product, attack vectors description or any other materials, products or results of this security review engagement is provided "as is" and "as available" and with all faults, uncertainty and defects without warranty or guarantee of any kind.

Omniscia will assume no liability or responsibility for delays, errors, mistakes, or any inaccuracies of content, suggestions, materials or for any loss, delay, damage of any kind which arose as a result of this engagement/security review.

Omniscia will assume no liability or responsibility for any personal injury, property damage, of any kind whatsoever that resulted in this engagement and the customer having access to or use of the products, engineers, services, security report, or any other other materials.

For avoidance of doubt, this 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 any other type of advice.