PRBProxy Audit

Executive Summary

Auditor	Iaroslav Mazur, Smart Contracts Engineer
Timeline	21.06.2023 – 03.07.2023
Language	Solidity
Audit Methods	Architecture Review, Static Code Analysis, Automated Testing (Unit, Fuzz)
Source Code	PRBProxy (82d14b9)

Overall Assessment

PRBProxy is an on-chain Proxy solution that extends the capabilities of Externally Owned Accounts (EOA), serving as a smart wallet for the latter. It positions itself as a new “building block” in the decentralized world of blockchain, meant to be useful both by itself - and, also, as a foundation for other projects to be built on top of, enabling an even bigger advancement opportunity for the Ethereum ecosystem.

The architecture of PRBProxy offers the best of both worlds: it’s simple, yet sophisticated enough to be a very valuable tool for its users. The functionality of the solution is straightforward, but, also, very versatile, making it profoundly user-oriented.

The codebase of the project is of a high quality, exhibiting the team’s focus on the clean code practices and robust security. PRBProxy has got a rigorous testing suite and a 100% code coverage.

The team has also been very open to suggestions and findings during the audit, and to discussions surrounding the technical and general aspects of the project.

During the audit, 1 Medium, 4 Gas Optimisation and 15 QA findings (according to **Code4Arena Severity Categorization**) have been identified.

Aside from the findings listed in this report, there have, also, been a number of smaller improvements/optimizations proposed for integration into the PRBProxy codebase directly via the Pull Requests **#145** and **#146** on GitHub. The fixes for a part of the findings from this report have also been included in the Pull Requests.

Update: The findings from this report and the associated Pull Requests have been addressed as of commit **972afc9**.

Scope

The reference point for the codebase analysed during this audit was the **82d14b9** commit from the **PRBProxy** repository. More specifically, the following has been reviewed:

- All the code under "src" and "test";
- The deployment scripts under "script".

Assumptions

Here are the assumptions that were kept in mind while analysing the project codebase:

- Proxies are always deployed via the registry; never directly.
- Users trust the `target` contracts they are delegate-calling to.
- Users trust the `plugin` contracts they are installing.
- The potential "feature clash" with Account Abstraction (if and when it gets implemented in Ethereum) has been taken into consideration.

Findings

Outlined below for your review are the findings identified during the audit, categorized by their severity and impact. Where appropriate, a short guidance on a solution for the finding is also provided. For clarity, each finding is prepended (where applicable) with a link to the file(-s) which it applies to.

Medium

1. PRBProxyRegistry.sol (line 203): a plugin passed to the `uninstallPlugin()` function may be completely unrelated to any of the the currently-installed plugins, but “pretending” to be one of them - by returning the selectors for their methods, when queried. This means that a malicious plugin could cause the methods of the other installed plugins to be uninstalled.

Gas Optimisation

1. PRBProxyRegistry.sol (line 62): the initialization of a local `IPRBProxy` variable could be ditched for the direct access of the Proxy contract from `storage`. While the invalid calls would, indeed, need to access the `storage` twice in this case, the valid ones would become more efficient.
2. PRBProxyRegistry.sol (line 264): it should be more efficient to declare `method` outside the `for` loop. Possibly, even with `--via-ir`.
3. IPRBProxy.sol (line 15) : the `PRBProxy_CallerNotRegistry` error is never thrown.
4. IPRBProxy.sol (line 33) : the `PRBProxy_TargetRegistry` error is never thrown.

Quality Assurance

1. PRBProxyRegistry.sol (line 182): the `deployAndInstallPlugin()` function is missing the `noProxy()` modifier.
2. IPRBProxyRegistry.sol (line 105): `getPermissionByOwner()` could be renamed to something like `isEnvoyPermissionedByOwner()` to make it more suggestive (i.e. by calling the function, you’re not getting a “permission” (as its name suggests), but, rather, a yes/no flag).
   Similarly, `getPermissionByProxy()` could become something like
   `` `isEnvoyPermissionedByProxysOwner()` `` (Proxies don’t permission anyone - their owners do).

3. PRBProxyRegistry.sol (line 61): `noProxy` would be better named to `onlyNonOwner` or `onlyNonProxyOwner`, while its argument would better be named just `user`: if the passed-in argument is known to be an owner when the `modifier` is called, there is no reason to call the `modifier`, at all.
4. PRBProxyRegistry.sol (line 82): `getMethodsByOwnerAndPlugin()` looks like a more suggestive name for the function, given its dependency on both of the arguments for the methods retrieval.
   Also applies for the other methods-retrieval function, `getMethodsByProxy()`.
5. IPRBProxyRegistry.sol (line 178): the `Execute` event is also emitted by the function.
6. IPRBProxyRegistry.sol (line 185): the other 2 function arguments are not documented.
7. IPRBProxyRegistry.sol (line 187): the function would, probably, be better named `deployWithExecutionAndInstallPlugin`, to not have the repetitive "and" conjunctions.
   If this gets implemented, functions like `deployAndExecute` should also be renamed accordingly.
8. runPlugin.tree (lines 5-7): the "owner change during the delegate call" branch mentioned in the `.tree` file isn't tested. Given that the owner of a `PRBProxy` can't be changed (being kept in an `immutable` variable), should the `.tree` file be cleaned of that branch?
9. execute.tree (lines 11-13): the "target is the registry" branch isn't tested.
10. getPermissionByOwner.t.sol (line 14): `getPermissionByOwner()` isn't, actually, tested in this contract - `getPermissionByProxy` is.
11. installPlugin.t.sol (line 14), uninstallPlugin.t.sol (line 14) and setPermission.t.sol (line 13): these "change pranks" are redundant. Quote from the Foundry Book: "After calling `expectRevert`, calls to other cheatcodes before the reverting call are ignored."
12. deployAndExecuteAndInstallPlugin.t.sol (line 61), deployAndExecuteAndInstallPlugin.t.sol (line 71), deployAndInstallPlugin.t.sol(line 51) and deployAndInstallPlugin.t.sol (line 61): the fuzzing in these 4 functions isn't, actually, happening (due to the missing argument(-s)).
13. deployAndInstallPlugin.t.sol (line 20): `registry.deployAndInstallPlugin()` should, rather, be called here, as that is the `PRBProxyRegistry` function that's being tested in this contract.
14. runPlugin.t.sol (line 51), runPlugin.t.sol (line 58), runPlugin.t.sol (line 64): wrong arguments are passed to `vm.expectrevert()` in these 3 places.

15. installPlugin.t.sol (line 14), setPermission.t.sol (line 13) and uninstallPlugin.t.sol (line 14): changing the prank to `users.bob` isn't necessary, as `users.alice` doesn't own a `PRBProxy` at this point either.

Additional Feedback / Things to Consider

1. Maybe, use the term "delegate" instead of "envoy"? This would, also, fit well with `DELEGATECALL`-ing from inside `PRBProxy`.
2. PRBProxy doesn't currently offer any "official" way to recover any Ether (`msg.value`) sent to the contract via the `fallback()` or the `receive()` functions.
3. The contract of an installed plugin could "selfdestruct", potentially causing `PRBProxyRegistry` to not be able to uninstall it from its storage.
4. runPlugin.t.sol: because of an issue identified in this file, the underlying contract functionality can't be properly tested. The issue is that, for some reason, the `vm.expectRevert()` in the "test_RevertWhen_Error_…" functions doesn't seem to care what argument is being passed to it - resulting in the respective tests succeeding anyways. For example, `IPRBProxy.PRBProxy_PluginReverted.selector` can be passed to it in `test_RevertWhen_Error_Require()` or no argument, at all - in `test_RevertWhen_Error_ReasonString()`.