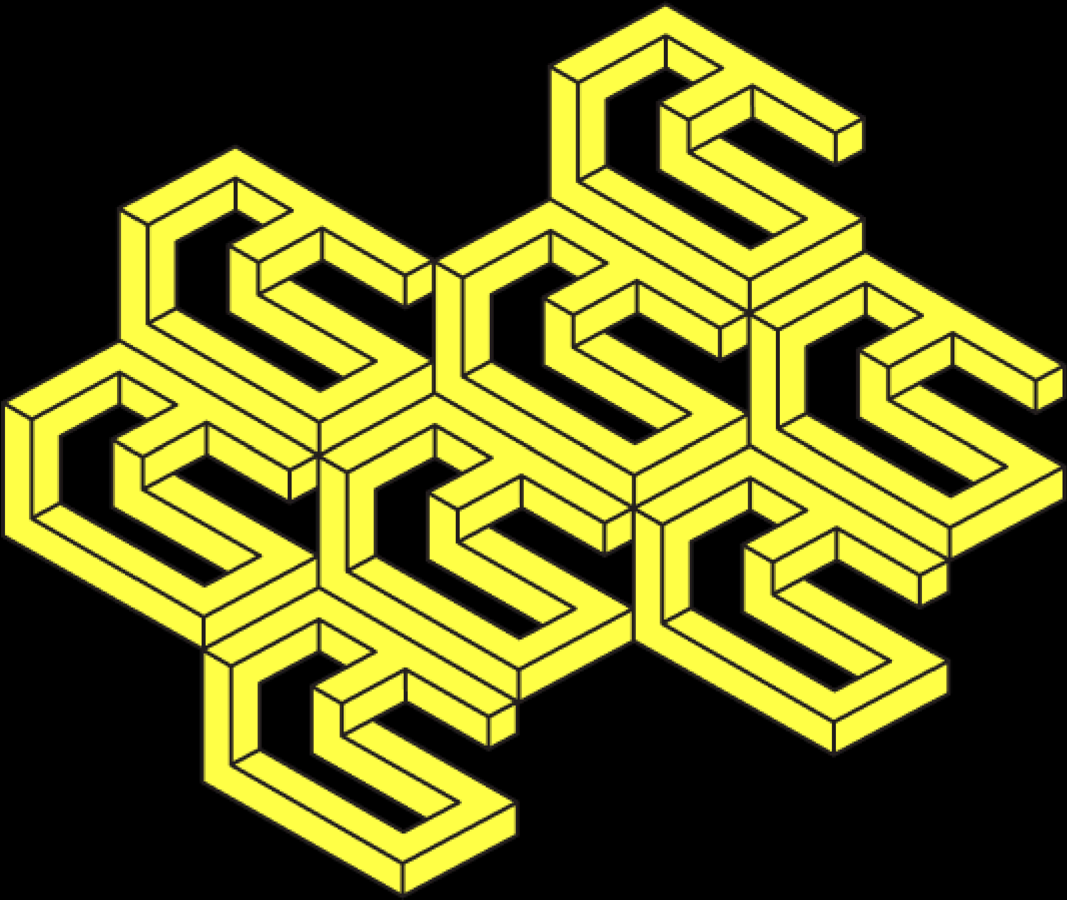

B3 Points Service Smart Contracts Review | August 2024

by ChainSafe Systems | August 2024

Table of contents

1. Introduction

Date	Auditor
August 2024	Oleksii Matiiasevych

B3 Team requested **ChainSafe Systems** to perform a review of the B3 Points Service smart contracts. The contracts can be identified by the following git commit hashes:

`404374416182ea41358abc3b4c2946ccc4418791 b3st-points`
`8f18ca86f79bb6954df63e63aec9c16791adf455 ens-offchain-resolver-contracts`

There are 6 contracts, interfaces and libraries in scope.

After the initial review, B3 Team applied a number of updates which can be identified by the following git commit hashes:

`fbb41f062ab95cb2720cc34b8a49410fe09dae59 b3st-points`
`043270c50477119dc347a8505a7824e4bae11226 ens-offchain-resolver-contracts`

Additional verification was performed after that.

Defining Severity

Each finding is assigned a severity level.

Note	Notes are informational in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
Optimization	Optimizations are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
Minor	Minor issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
Major	Major issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
Critical	Critical issues are directly exploitable security vulnerabilities that need to be fixed.

Referencing updated code

Resolved	The finding has been acknowledged and the team has since updated the code.
Rejected	The team dismissed this finding and no changes will be made.

Disclaimer

The review makes no statements or warranties about the utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about the fitness of the contracts for any specific purpose, or their bug free status.

2. Executive Summary

There were zero minor or above severity issues identified, and most of the optimizational and note-level findings were fixed and not present in the final version of the contracts.

There are **no** known compiler bugs for the specified compiler version (0.8.24), that might affect the contracts’ logic.

There were **0 critical**, **0 major**, **0 minor**, 12 informational/optimization issues identified in the initial version of the contracts.

We are looking forward to our continuous engagement with the B3 Team.

3. Critical Bugs and Vulnerabilities

No critical issues were identified in the contracts.

4. Line-by-line review

src/AppRegistry.sol

L102 Optimization Resolved

```
AppRecord storage app = _apps[id];
if (app.operator != msg.sender) {
    revert NotOperator(id, msg.sender);
}

address oldOperator = app.operator;
```

The `updateOperator()` function reads `app.operator` storage variable twice.

L138 Optimization Resolved

```
if (l1Resolver == resolver_) {
    revert InvalidResolver(resolver_);
}

address oldResolver = l1Resolver;
```

The `_updateResolver()` function reads `l1Resolver` storage variable twice.

L149 Optimization Resolved

```
if (b3Resolver == resolver_) {
    revert InvalidResolver(resolver_);
}

address oldResolver = b3Resolver;
```

The `_updateResolver()` function reads `b3Resolver` storage variable twice.

src/IBPS.sol

L18 Optimization Rejected

```
struct Transfer {
    bytes32 uid; // A unique identifier of the transfer.
    ...
}
```

The `Transfer` struct has an excessive `uid` field, which is already used as a storage key.

L19 Optimization Rejected

```
struct Transfer {
    bytes32 uid; // A unique identifier of the transfer.
    uint256 session; // The session when the transfer is created.
    uint256 appId; // The app ID that is transferring the point.
    address recipient; // The user address that is receiving the point.
    uint256 point; // The point that user receives.
    uint64 createdAt; // The time when the transfer is created.
    uint64 canceledAt; // The time when the transfer is canceled.
}
```

The `Transfer` struct `session` field could be shrank to fit in the same storage slot with timestamps.

L20 Optimization Rejected

```
struct Transfer {
    bytes32 uid; // A unique identifier of the transfer.
    uint256 session; // The session when the transfer is created.
    uint256 appId; // The app ID that is transferring the point.
    address recipient; // The user address that is receiving the point.
    uint256 point; // The point that user receives.
    uint64 createdAt; // The time when the transfer is created.
    uint64 canceledAt; // The time when the transfer is canceled.
}
```

The `Transfer` struct `appId` field could be shrank to fit in the same storage slot with timestamps.

src/B3PointService.sol

L91 Optimization Resolved

```
if (_availablePoints[currentSession][appId] < point) {
    revert InsufficientBalance(currentSession, appId, recipient);
}
```

The `transferPoints()` function reads `currentSession` variable from storage multiple times.

L122 Optimization Rejected

```
for (uint256 i = 0; i < requests.length; i++) {
    ...
    _availablePoints[currentSession][appId] -= point;
    ...
}
```

The `transferPoints()` function updates `_availablePoints` variable in storage multiple times.

L152 Optimization Resolved

```
if (transfer.session != currentSession) {
    revert TransferAlreadyFinalized(uid);
}
...
_availablePoints[transfer.session][appId] += transfer.point;
```

The `cancelTransfer()` function reads `transfer.session` variable from storage twice.

src/SignatureVerifier.sol

L15 Note Resolved

```
/**
 * @dev Generates a hash for signing/verifying.
 * @param target: The address the signature is for.
 * @param request: The original request that was sent.
 * @param result: The `result` field of the response (not including the
signature part).
 */
function makeSignatureHash(address target, uint64 expires, bytes memory
request, bytes memory result)
```

The `makeSignatureHash()` function description is missing `expires` parameter.

L31 Note Resolved

```
function verify(bytes calldata request, bytes calldata response) internal
view returns (address, bytes memory) {
```

The `verify()` function `request` parameter should be named `extraData` for naming consistency.

L33 Note Resolved

```
(bytes memory extraData, address sender) = abi.decode(request, (bytes,
address));
```

The `verify()` function `extraData` local variable should be named `callData` for naming consistency.