Urbit Constitution

Security Audit

June 10th 2018

Version 1.0.0

Prepared by

Bloctrax

Table of Contents

Introduction

This document includes the results of the preliminary security audit for Urbit's smart contract code as found in the section titled 'Source Code'. The security audit was performed by the Bloctrax team from May 18th 2018 to June 10th 2018.

The purpose of this audit is to review Urbit's source code, and provide feedback on the design, architecture, and quality of the source code with an emphasis on validating the correctness and security of the software in its entirety.

Overall Assessment

Based on our review of Urbit's source code it's clear that the Urbit engineering team has put a lot of thought and care into their design and architecture.

We identified a small number of issues of medium severity that should be addressed before the project's deployment to the main net, but they primarily exist in the higher level contracts that are built on top of the "core" contracts, and thus are not as severe as if they were found in the storage contracts (which cannot be upgraded).

We did not identify any issues of high severity that would substantially compromise the integrity of the project.

Specification

Our understanding of the specification was based on the following sources:

- Urbit Whitepaper
- Comments in the contract source code / README
- Various urbit blog posts
- Direct discussions with the Urbit team

Source Code

The following source code was reviewed during the audit:

Repository	Commit
Github	70f9302d36fdb8b1def9a0dbed43f1cb09f26666

Specifically, we audited the following contracts:

Contract	Sha256
Censures.sol	23256cd92eab1b575ec1959f420ebd1a9a7d879c cd60e2905b7b4e8b4fc812d8
Claims.sol	4b0c377808c820d11544a39e624da6578be9e61 700154877706742a0be5603b2
ConditionalStarRelease.sol	7b042250956f32aaf82d62eccda1d5d840cbe7db fb9c359514e92f438baa436f
Constitution.sol	9e476849e2f53b3f6305112f9133142f75c377f1 71f49c091b902a576aa6393
ConstitutionBase.sol	8958f8f6ad09efe452d851d98c2b0ad5becbb4b9 3a50d5c219642941632b7104
DelegatedSending.sol	1bcb89903ba0ac417641b3d76119ca9c2a18424 a07944f1913609a9d8ed1447a
ENSRegistry.sol	f8ab3216b8f27d1cc9d50579cbac93096f50d2df1 18a456268e3e8f26576cd2b
LinearStarRelease.sol	d1f2dadf4330d0ada814b8835f18bb2ac159af43 62b51493e529b4fe88a4bdaf
PlanetSale.sol	7b263a417e38189272d645672ebadf75b9fa2e1e 2d88db6418861e89f76ba232
Polls.sol	f51d29f868587e6d5c4d91dd5c5c9282fd7e920b 8d551318338d8254c8bae5e9
Pool.sol	30584863dc6247988db3cf27dbf614b15f991c62 f823e54aa8e8a19d017a6de1
SafeMath8.sol	f60b661f42965834867aee279e35167030a8f8c5 7a520a73e868fd6e817b0f66

Ships.sol	**7817c01b3b6ca0d32813a51b788a19fecd9ab7b2c03bee9e224564c6c2a1e8be**

Note: This document contains an audit only of the code contained in the repository and contracts listed in this section above written in Solidity. Specifically, the audit pertains only to the contracts themselves, and does not pertain to any other programs or scripts, including deployment scripts.

Methodology

The audit was conducted in three steps.

First, we reviewed in detail all available documentation and specifications for the project, as described in the 'Specification' section above.

Second, we performed a thorough manual review of the code, checking that the code matched up with the specification, as well as the spirit of the contract (i.e. the intended behavior). During the manual review portion of the audit we also evaluated the code for best practices, quality, and potential security vulnerabilities.

Finally, we performed the automated portion of the review consisting of measuring test coverage (while also assessing the quality of the test suite) and running various symbolic execution tools against the code, such as Mythril and Oyente.

Severity Level Reference

Level	Description
High	The issue poses existential risk to the project, and the issue identified could lead to massive financial or reputational repercussions.
Medium	The potential risk is large, but there is some ambiguity surrounding whether or not the issue would practically manifest.
Low	The risk is small, unlikely, or not relevant to the project in a meaningful way.
Code Quality	The issue identified does not pose any obvious risk, but fixing it would improve overall code quality, conform to recommended best practices, and perhaps lead to fewer development issues in the future.

Issues Descriptions and Recommendations

DelegatedSending contract limitations can be bypassed by ships invited by ship zero

MEDIUM

The **DelegatedSending** contract uses the **getPool** function to determine which pool the ship performing the invitation belongs to, and subsequently, how many ships it is allowed to invite.

The **getPool** function performs the following lookup:

```
pool = fromPool[_ship];
```

For any ship that was invited by ship zero, this will return zero.

This will then trigger the following logic:

```
//  no pool explicitly registered means they have their own pool
   //
   if (0 == pool)
   {
     return _ship;
   }
```

As a result, ships invited by ship zero will always be treated as if they have their own pool even though they're not supposed to.

Owner of Polls contract can manipulate voting process

Medium **Resolved by: d7048eb91186ebfe03499742ae704176b5b0b79f**

One potential issue with the **Polls** contract is that the **pollDuration** and **pollCooldown** variables can be manipulated by the owner, even while there are outstanding polls. For example, imagine the following situation:

- ○ A poll currently has 65 yes votes and 64 no votes (out of a total of 256 galaxies), with one week of voting period remaining.
    - ■ If the poll were to end now, it would pass because it has a 50% +1 majority of all galaxies that have voted, AND it has achieved more than the 25% required total participation.
    - ■ But, there are still 127 galaxies that could vote no and fail the Poll.
- ○ The owner of the of the contract decides that they want the Poll to pass, and so they call **reconfigure** on the **Polls** contract to change the **pollDuration** such that the poll is now over, and then they can call **updateConcretePoll** or **updateAbstractPoll** to finalize it.

This is mitigated by the fact that the owner of the Polls contract will be the **Constitution** contract which has no formal way of calling **reconfigure** on the **Polls** contract, but that could change in future **Constitution** versions.

One potential solution to this issue is to store the **pollDuration** and **pollCooldown** as values that exists when the poll is started in the **Poll** struct itself.

Failure to reset escapeRequestedTo field in-between escape attempts

LOW **Resolved by: fe99c159f553016e9301157cefe64dda185668cb**

The **escapeRequestedTo** struct field in the **Ships** contract does not get reset in the **cancelEscape** and **doEscape** functions. Consider zeroing it out between escape attempts to prevent any bugs where the code accidentally relies on a previous value.

Failure to use SafeMath

LOW **Resolved by: 768a829d15dae13ff2d5195341ee2137abda7a1f**

The **Polls** contract imports the **SafeMath8** library, but does not use it for any operations. In addition, the **Polls** contract uses raw mathematical operations (**+ / -**) instead of safe operations. Consider using **SafeMath8** as well as OpenZeppelin's SafeMathLibrary (for **uint256**) to prevent common sources of vulnerabilities like integer overflow and underflow.

Similarly, the **ConditionalStarRelease** contract implements a manual overflow check in the **totalStars** function, as well as unsafe mathematical operations in the **withdrawLimit** function. Consider using a library for both of these use cases.

Finally, the **LinearStarRelease** contract should replace all raw mathematical operations with use of a safe math library, especially considering that **rate** and **rateUnit** are unbounded.

Lack of reasonable limits on value of polls duration and cooldown

LOW **Resolved by: 16bfbe664982f88a04902087f628f331ad4432df**

The **pollDuration** and **pollCooldown** state variables in the **Polls** contract can be set by the owner of the contract, but there are no limitations on their values. Consider setting reasonable minimums and maximums to prevent abuse. Specifically, so the owner can't set the values very low so that they can pass votes before everyone has had a chance to review them, or set the value so high that all votes are effectively blocked.

Use of uint8 to store indexes in array of unbounded length

LOW **Resolved by: e97843412539347443e5dd7a66d774f3abef7dd**

The **ConditionalStarRelease** contract makes frequent use of the **uint8** type for tranche indexing, but there is no limitation on the number of tranches. Consider switching to the **uint256** type to avoid integer overflow when dealing with large number of tranches, or even better, impose a reasonable limit on the number of tranches.

Pool contract permits easy destruction of Urbit address space

LOW

The **Pool** contract, by nature of inheriting from **BurnableToken,** has a public **burn** method that makes it easy to accidentally destroy Urbit address space, trapping ships in the **Pool** contract forever.

Consider not inheriting from the **BurnableToken** contract at all, as the functionality required from it can be implemented in a few lines of code.

Outdated solidity compiler version

~~CODE QUALITY~~ **Resolved by: 997bd514c75a093ba1885c7d70ad27f5baf017dc**

Consider upgrading to version **0.4.24** of the Solidity compiler and simultaneously making the following changes:

- Use the **emit** keyword when emitting all events.
- Replace the existing legacy constructors with the new **constructor** keywords syntax (applies to every contract).

Note: All instances where this is not done can be detected by simply running **truffle compile** with the latest version of the Solidity compiler.

shipOwner modifier is repeated multiple times

~~CODE QUALITY~~ **Resolved by: e07d2046000e3ddc5230577a6603217f9e7fe405 and 2f89b0c560065c4ed9e30abe0423253fb8710ac1**

Several contracts implement the **shipOwner** modifier. Consider creating a base contract (called **hasShipsStorage** or equivalent), which includes the ship's storage state variable, as well as the **shipOwner** modifier so that the logic can be reused to avoid unnecessary repetition. Also, consider renaming the **shipOwner** modifier to **onlyShipOwner**.

Lack of indexed fields in emitted events

~~CODE QUALITY~~ **Resolved by: ebb25a36c01e3ffc8d3158c374a18f1f9b4c8e0f**

The following contracts have zero indexed fields in any of their events:

- **Ships**
- **Polls**
- **Censures**
- **Claims**
- **ConditionalStarRelease**

It would be prudent to index some of the important fields in these events to make client development substantially easier.

Multiple functions with the same name, but different arguments

~~CODE QUALITY~~ **Resolved by: b2ca144c2b0287754d71cbc6c79beb79a319b781**

In the **Ships** contract there are two functions called **getOwnedShips** that have the same name but different arguments. This can be confusing for readers and makes auditing for correctness more difficult. Consider giving each function its own name.

Lack of consistency between function comments and function code

~~CODE QUALITY~~ **Resolved by: 36686ead1036639e834e5ceb502cf1df0c379604**

The comment above the reconfigure function in the Polls contract says: "change poll duration, cooldown, and vote requirements", but the function only changes the duration and cooldown.

Also, the **withdrawOverdue** function in the **ConditionalStarRelease** contract is designed to prevent loss of address space in the case where participants lose their keys. The comment at the top of the file states that the owner should be able to withdraw any remaining stars 10 years after the contract launch if the stars haven't been claimed yet. However, the actual implementation contrasts in that it requires that 10 years have elapsed since the first tranche is unlocked.

Use of custom assembly code instead of pre-audited library code

~~CODE QUALITY~~ **Resolved by: 77f1c6a2fe8c8426413e50012620eb18816fc12b**

The **safeTransferFrom** function in the **Constitution** contract could get rid of the in line assembly (which is always tricky to get right), as well as the manual calling code, by reusing OpenZeppelin's AddressUtils / ERC721BasicToken contracts.

Unhandled TODOs in source code

~~CODE QUALITY~~ **Resolved by: ebb25a36c01e3ffc8d3158c374a18f1f9b4c8e0f**

There is an unhandled TODO in the **Constitution** contract which states that some portions of the code need to be manually commented and uncommented for the purposes of deployment. Consider addressing this TODO such that the same contract can be compiled, as well as deployed without modifying the source code.

Constitution implementation does not follow ERC-721 specification exactly

~~CODE QUALITY~~ **Resolved by: 70f9302d36fdb8b1def9a0dbed43f1cb09f26666 and b676e433c5f34d8dfba6d4119d3c7822922cd2aa**

The ERC-721 interface specifies that **safeTransferFrom** and **transferFrom** should both throw if the **_to** address is zero, but it does not look like the existing **Constitution** contract implementation does this.

In addition, the ERC721 specification states that the **totalSupply()** function should return "A count of valid NFTs tracked by this contract, where each one of them has an assigned and queryable owner not equal to the zero address". However, the existing implementation will always return **2^32** even if the ships have not been activated and do not have owners.

Contracts inherit from contracts that they do not import

~~CODE QUALITY~~ **Resolved by: a9e2fc45b3d7888e743e2cb5fee395d7aaecf9ce**

The **ConditionalStarRelease** and **LinearStarRelease** contracts both inherit from the **Ownable** contract, but do no appear to import it.

Claims contract can be simplified

~~CODE QUALITY~~ **Resolved by: 70d628b2eb8c5680ecb51f1196b22356df486c1d**

The restriction to a maximum of 16 claims provides an opportunity to simplify the implementation of the **Claims** contract by changing:

```
mapping(uint16 => uint32[]) public claim;
```

to

```
mapping(uint16 => uint32[16]) public claim;
```

This will simplify the code because since the array will never be larger than 16, we can get rid of all the array deletion logic, and just use iteration.

This change would also enable deletion of the **indexes** state entirely as it could be replaced with a function that iterates through the (bounded) **censures** array and looks for a given address.

Test Coverage

Steps Taken

- Updated the require compiler version for each contract 0.4.24
- Ran **npm install web3-eth-abi**

Truffle Test Results

Output of running **truffle test** at the root of the project directory:

```
Contract: Censures
    ✓ censuring (484ms)
    ✓ forgiving (163ms)

  Contract: Claims
    ✓ claiming (498ms)
    ✓ disclaiming (151ms)
    ✓ clearing claims (1307ms)

  Contract: Conditional Star Release
    ✓ creation sanity check (39ms)
    ✓ analyzing tranches (4575ms)
    ✓ registering commitments (500ms)
    ✓ forfeiting early
    ✓ withdraw limit (4475ms)
    ✓ depositing stars (1854ms)
    ✓ withdrawing (819ms)
    ✓ transferring commitment (222ms)
    ✓ forfeiting and withdrawing (6537ms)

  Contract: Constitution
    ✓ setting dns domains (92ms)
    ✓ creating galaxies (343ms)
    ✓ spawning ships (502ms)
    ✓ setting spawn proxy (232ms)
    ✓ transfering ownership (405ms)
    ✓ allowing transfer of ownership (171ms)
    ✓ rekeying a ship (187ms)
    ✓ setting and canceling an escape (738ms)
    ✓ adopting or reject an escaping ship (236ms)
    ✓ voting on and updating abstract poll (4378ms)
    ✓ voting on concrete poll (727ms)
    ✓ updating concrete poll (4668ms)

  Contract: Delegated Sending
    ✓ configuring (263ms)
    ✓ sending (1023ms)
    ✓ resetting a pool (384ms)

  Contract: NFTokenMock
    ✓ correctly checks all the supported interfaces (116ms)
    ✓ returns correct balanceOf after mint (178ms)
    ✓ throws when trying to mint 2 NFTs with the same claim
(180ms)
    ✓ throws when trying to mint NFT to 0x0 address  (116ms)
    ✓ finds the correct amount of NFTs owned by account (260ms)
    ✓ throws when trying to get count of NFTs owned by 0x0
address (66ms)
    ✓ finds the correct owner of NFToken id (166ms)
    ✓ throws when trying to find owner od non-existing NFT id
(97ms)
    ✓ correctly approves account (231ms)
    ✓ correctly cancels approval of account[1] (225ms)
    ✓ throws when trying to get approval of non-existing NFT id
(57ms)
    ✓ throws when trying to approve NFT ID which it does not own
(238ms)
```

```
    ✓ throws when trying to approve NFT ID which it already owns
(237ms)
    ✓ correctly sets an operator (200ms)
    ✓ correctly sets then cancels an operator (260ms)
    ✓ throws when trying to set a zero address as operator
(75ms)
    ✓ correctly transfers NFT from owner (404ms)
    ✓ correctly transfers NFT from approved address (477ms)
    ✓ corectly transfers NFT as operator (393ms)
    ✓ throws when trying to transfer NFT as an address that is
not owner, approved or operator (141ms)
    ✓ throws when trying to transfer NFT to a zero address
(126ms)
    ✓ throws when trying to transfer a invalid NFT (140ms)
    ✓ correctly safe transfers NFT from owner (332ms)
    ✓ throws when trying to safe transfer NFT from owner to a
smart contract (376ms)
    ✓ corectly safe transfers NFT from owner to smart contract
that can recieve NFTs (333ms)

  Contract: NFTokenMetadataEnumerableMock
    ✓ correctly checks all the supported interfaces (94ms)
    ✓ returns the correct issuer name (71ms)
    ✓ returns the correct issuer symbol (56ms)
    ✓ returns the correct NFT id 2 url (148ms)
    ✓ throws when trying to get uri of none existant NFT id
(54ms)
    ✓ returns the correct total supply (56ms)
    ✓ returns the correct token by index (312ms)
    ✓ throws when trying to get token by unexistant index (65ms)
    ✓ returns the correct token of owner by index (264ms)
    ✓ throws when trying to get token of owner by unexistant
index (251ms)
```

```
Contract: Linear Star Release
  ✓ registering batches (196ms)
  ✓ withdraw limit (12072ms)
  ✓ depositing stars (1895ms)
  ✓ transferring batch (256ms)
  ✓ withdrawing (760ms)

Contract: Planet Sale
  ✓ configuring price (66ms)
  ✓ checking availability (130ms)
  ✓ purchasing (394ms)
  1) withdrawing
  > No events were emitted
  ✓ ending (536ms)

Contract: Polls
  ✓ configuring polls (201ms)
  ✓ concrete poll start & majority (1173ms)
  ✓ concrete poll minority & restart (13106ms)
  ✓ abstract poll start & majority (782ms)
  ✓ abstract poll minority & restart (13106ms)

Contract: Pool
  ✓ deposit star as galaxy owner (314ms)
  ✓ deposit star as star owner (329ms)
  ✓ withdraw a star (197ms)

Contract: Ships
  ✓ getting prefix parent (58ms)
  ✓ getting class (64ms)
  ✓ setting dns domain (146ms)
  ✓ getting and setting the ship owner (104ms)
  ✓ getting owned ships (252ms)
  ✓ activating and spawn count (224ms)
```

```
      ✓ setting, canceling, and doing escape (256ms)
      ✓ setting keys (163ms)
      ✓ setting spawn proxy (176ms)
      ✓ setting transfer proxy (121ms)


  91 passing (2m)
  1 failing

  1) Contract: Planet Sale
       withdrawing:
     AssertionError: expected 999647663000000000000 to equal
999647663000000010000
      at Context.<anonymous> (test/TestPlanetSale.js:62:12)
      at <anonymous>
      at process._tickCallback
(internal/process/next_tick.js:188:7)
```

Code Coverage

Code coverage was measured by running **solidity-coverage** at the root of the project.

File	%Stmts	%Branch	%Funcs	%Lines	Uncovered Lines
Censures.sol	100	100	100	100	
Claims.sol	100	90	100	100	
ConditionalStarRelease.sol	95.12	82.61	94.12	95.18	278, 283, 284, 289
Constitution.sol	100	87.88	100	100	
ConstitutionBase.sol	100	100	100	100	
DelegatedSending.sol	100	100	100	100	

LinearStarRelease.sol	89.36	83.33	91.67	89.13	..., 179, 183, 315
PlanetSale.sol	100	100	100	100	
Polls.sol	100	95	100	100	
Pool.sol	100	90	100	100	
Ships.sol	100	95.45	100	100	
All Files	98.01	88.99	98.68	98.22	

Evaluation

The code coverage for the Urbit repository is good, but it would be better if the Urbit team invested in bringing the overall branch coverage closer to 100%, especially in the **ConditionalStartRelease, Constitution, LinearStarRelease,** and **Pool** contracts.

Also, there is some flakiness associated with the test suite that should ideally be addressed. We were unable to achieve a single run in which all the tests passed, despite multiple attempts.

Automated analysis

Mythril

Mythril is a security analysis tool that uses concolic analysis, taint analysis, and control flow checking to detect a variety of security vulnerabilities.

In order to run Mythril against the codebase we performed the following steps:

- Run **myth --truffle**

Mythril identified several potential instances of integer underflow. As a result, consider replacing all these expressions with safe math operations as described in the 'Issues Descriptions and Recommendations' section above:

- **ConditionalStarRelease**
    - **Com.total - com.withdrawn**
    - **_com.stars.length -1**
- **LinearStarRelease**
    - **batch.amount - batch.withdrawn**
    - **block.timestamp - (start + batch.windup)**
    - **_batch.stars.length - 1**
- **PublicResolver**
    - **content-type - 1**
- **Claims**
    - **claims[_as][cur-1] = Claim(_protocol, _claim, _dossier);**
    - **uint256 last = clams.length - 1;**
    - We believe both of these to be false positives as the preceding code contains logic that guarantees subtracting that the values from which 1 is subtracted are equal to 1 or higher.

In addition, Mythril identified two instances of calling a function on a contract whose address was passed in as an argument in the **ConditionalStarRelease** and **LinearStarRelease** contracts. We believe they're both false positives.

Oyente

Oyente is a smart contract analysis tool that attempts to identify the following types of security vulnerabilities:

- Parity Multisig Bug 2
- Transaction-Ordering Dependence
- Call Stack Depth Attack
- Timestamp Dependency
- Reentrancy Vulnerability

In order to run Oyente against the codebase we performed the following steps:

- Run **truffle-flattener** on each contract

Oyente did not identify any security vulnerabilities in any of the contracts.

Appendix

Exhibit A - Dependency Graphs

Black, filled-in circles represent “inherits from” relationships and white circles represent “imports” relationships.

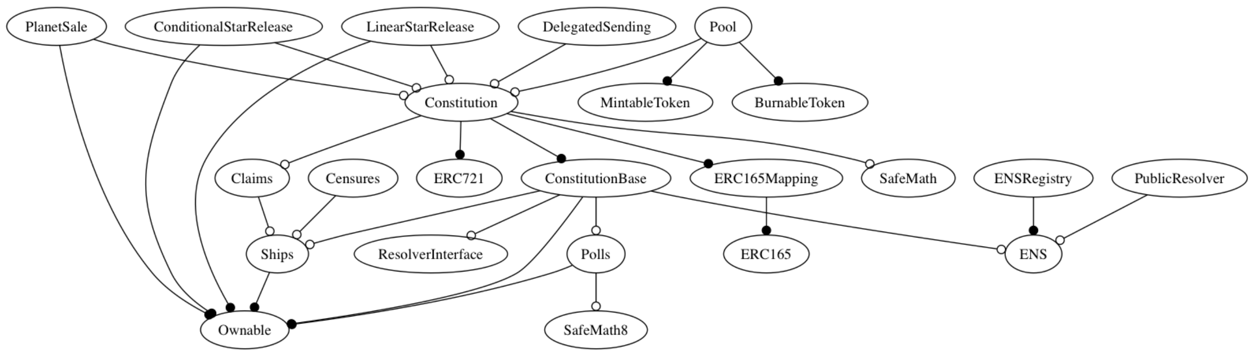

Exhibit B - Suggested Renamings and Refactors

Ships

Suggestions taken into consideration by: ac2de37750da6f51f9ccfed8c7fbc4f43235369e

- **getOwnedShipAtIndex** should either use the temporary variable **owned** in the return statement, or not create it at all and simply use **owners[_whose]** in the **require** statement.
- Consider renaming **owners** state variable to something more descriptive, such as **shipsOwnedBy**.
- Consider renaming the **Transferred** event to **OwnerSet** since the event does not represent a full transfer, as opposed to the transfer logic as found in the **Constitution** contract.
- Consider renaming **setActive** (and the associated **active** struct field) to something like **initializeShip**, which conveys more about the semantics of the function.
- Consider using more descriptive variable names in the **ChangedKeys** event. For example:
  - **crypt → encryptionKey**
  - **auth → authenticationKey**
  - **rev → revisionNumber**
- Consider renaming **getEscape** to **getEscapeDestination** or **getEscapeRequestedTo**.
- Consider renaming **isEscape** to **isRequestingEscapeTo**.
- Consider renaming **setEscape** to something similar to any of the following: **setEscapeDestination, setEscapeTarget, setEscapeRequestTo, inititateEscape** etc.
- Consider renaming **doEscape** to **performEscape** for consistency with the function's comment.

Polls

Suggestions taken into consideration by: 9206736c9f892dff461b6170d4d5b3d3e53c6e47

- Consider renaming all references to **concretePolls** in the code and documentation to **constitutionPolls** for clarity.
- Consider renaming all references to **abstractPolls** in the code and documentation to **documentPolls** for clarity.
- Consider renaming **concreteMajorityMap** and **abstractMajorityMap** to **concretePollHasEverAchievedMajority** and **abstractPollHasEverAchievedMajority** respectively.
- Consider defining a **MAXIMUM_VOTERS** (256) constant which can be reused by this function, as well as the declaration of the **voted** array in the **Poll** struct.
- Consider renaming the **_who** argument in **hasVotedOnConcretePoll**, **hasVotedOnAbstractPoll**, and **hasVoted** to **_galaxy**.
- Consider rewriting **hasVotedOnConcretePoll** and **hasVotedOnAbstractPoll** to just check their respective maps/arrays in-line instead of sharing the single line of logic in **hasVoted**. It looks like its implemented properly right now, but we prefer to avoid passing around storage type arguments when possible.
- Update **castConcreteVote** and **castAbstractVote** to be consistent such that both return a boolean, or neither does.
    - This will require updating **updateConcretePool** and **updateAbstractPool** to be consistent in the same way.
- consider adding an additional guard in the require statement in **processVote** such that **block.timestamp >= poll.start**. That assertion is guaranteed to be true due to the way that poll start is initialize to **block.timestamp** when creating a new poll, but an extra assertion will allow symbolic execution tools to detect if it's possible for the invariant to be violated.
- Adding **return majority;** to the end of **updateConcretePoll** would improve readability.
    - Same goes for **updateAbstractPoll**.
- On line 302 of **updateConcretePool**, specify that majority is of type **bool** to be consistent with **updateAbstractPool**.
- We understand why it's safe for **updateAbstractPoll** to be called by anyone, but is there a compelling reason to allow it? Seems safer to lock it behind **onlyOwner** the same way **updateConcretePool** does.

- **checkPollMajority** would be easier to read if each of the possible conditions was broken into intermediary variables. In particular, the calculation for “there aren’t enough remaining voters to tip the scale” is particularly difficult to read as is.

Censures

Suggestions taken into consideration by: 1e42ecff973d9003f6e2d8fd42fda50bc597d708

- The **require(whoClass >= asClass);** statement in the censure function is confusing because the casual reader would expect a Galaxy class to be higher than a star class, when in fact it’s the opposite. Consider adding a comment explaining this, or re-ordering the enums so that in fact a Galaxy class is higher than a star class, although this would require refactoring all other code that dependencies on this ordering.

Claims

Suggestions taken into consideration by: d27a65bd6a912a90419d4879be32a4e7f26e74ca

- Consider making **16** a constant called **MAX_CLAIMS**.
- The comment above indexes refers to **hashes**, but everywhere else in the contract they’re just called **IDs**. Consider using one or the other for consistency.
- Consider renaming the **claim** and **disclaim** functions (and associated events) to **addClaim** and **removeClaim** as **disclaim**. The current names sound as if it’s one ship disputing another ship’s claims, which is not the case.
- **clearClaims** function uses “**clams”** and “**clam”** to refer to **claims** and **claim** in order to avoid overriding existing variables. Consider renaming to **currClaims** and **currClaim** respectively as its less jarring than using the wrong word.
- Modify the clearing claims test case to add new claims after clearing the existing claims, and then asserting that the new claims are present.

ConstitutionBase

Suggestions taken into consideration by: 5fa636ec5fc112e115064b7f3ebc0a6eaf10ed47

- Consider making the **upgraded** function check if ownership of **Ships** and **Polls** has been transferred.

- The **upgraded** function name is confusing. The doc string states that the function is used to verify that the expected upgrade path has taken place, but the comment located above where the **upgraded** function is called in the **upgrade** function says that **upgraded** triggers the upgrade logic which implies it is responsible for more than just verification. Consider updating the doc string to clarify that the function can be overridden to include any logic that should be executed during the upgrade process, as long as super is still called.

Constitution

Suggestions taken into consideration by: 8cfbc427ed7030edad384940bed2c83975ef4442

- Consider renaming the **shipId** modifier to **isValidShipID** for clarity. Could also reuse the **totalSupply** constant in the **shipId** modifier instead of duplicating the hard coded value of 2^32.
- Consider moving the **setTransferProxy** function to directly follow the **approve** function.
- **tokenOfOwnersByIndex** will throw if the _owner address is zero because of **require(_index < owned.length);** in the Ships contract which is the desired behavior, but it would be nice to make this an explicit check in **tokenOfOwnersByIndex**.
- Consider adding a check in **spawn** that the prefix ship is active, just to be safe.
- Consider renaming the **_ship** argument in **setSpawnProxy** to **_prefix**.

ConditionalStarRelease

Suggestions taken into consideration by: 20ce9961de38726e53fbe9201c100e0a88c6db21

- Use of the word **tranche** is more confusing than helpful. Consider removing it entirely from both the comments and code in favor of the word condition. For example, the **tranches** field in the **Commitment** struct could be renamed to **numStarsByCondition**, which is easier to understand.
- Consider adding a check in the **register** function to make sure that the return result of **totalStars** is less than the theoretical maximum (2^16 - 256?)
- Consider adding a basic sanity check to the **deposit** function to reject galaxies (i.e. **require(_start > 255)**).

- Consider adding a temporary variable in the **deposit** function to store the prefix of the **_star** to make the first conditional statement more readable.
- Should the **deposit** function also take into account forfeited stars? For example, not allow the owner to deposit more stars than the participant is entitled to deposit, while taking into account that they may have forfeited some of their total stars already?
- Consider making the **approveCommitmentTransfer** function verify that **msg.sender** is in fact a participant.
- Consider adding a check in the constructor to make sure that the number of tranches is larger than zero. This will also prevent exceptions in the various parts of the contract directly accessing index 0 in the **timestamps** array.
- Consider renaming the **stars** field to one of the following (or equivalent): **remainingStars**, **availableStars**, **pendingStars,** etc.

LinearStarRelease

Suggestions taken into consideration by: 4376952a0b57544cc31f1ac176f7dab9b4e6aafe

- Should the **deposit** function also take into account forfeited stars? For example, not allow the owner to deposit more stars than the participant is entitled to deposit, while taking into account that they may have forfeited some of their total stars already?
- Replace **0 == ships.getKeyRevisionNumber(_star))** with **!ships.hasBeenBooted(_star)**.
- Consider renaming the **stars** field to one of the following (or equivalent): **remainingStars**, **availableStars**, **pendingStars**, etc.
- Consider making the **approveBatchTransfer** function verify that **msg.sender** is a participant.
- Consider adding test coverage for the batch transfer code paths which currently appear to be completely untested.

PlanetSale

Suggestions taken into consideration by: 152f8ce38766f3bdb3715fa61f9ff764c293b0b3

- Consider indexing the **planet** argument in the **PlanetSold** event.

- Consider adding **require(_target != 0)** to both the **withdraw** function and **close** function to prevent accidental loss of funds.
- Consider adding a check to make sure that the price is not zero in the constructor function, as well as in the **setPrice** function.
- Consider allowing the owner of **PlanetSale** to control which ships are available for sale, or at least limit the number of ships that can be sold without further approval.

DelegatedSending

- Consider indexing the **by** and **ship** arguments in the **Sent** event.
- Consider allowing star owners to limit the maximum number of planets they would like to gift out (regardless of the per-planet limit).
- Consider implementing a **withdraw** pattern in this contract such that instead of immediately spawning ships for any address that is invited, the ship is only spawned once the potential new owner “accepts” the ship. This will prevent loss of address space to users who never even interact with the contract.

Pool

- Barely any of the code from the **MintableToken** contract is being used. We think it might be preferable if the contract did not inherit from **MintableToken**, and instead using something like the following two lines to implement minting:
    - `totalSupply_ = totalSupply_.add(_amount);`
    - `balances[_to] = balances[_to].add(_amount);`
- The refactor above would also enable removing the following line of code which is a fairly bad code smell:
    - `address(this).call.gas(50000)(bytes4(keccak256("mint(address,uint256)")),msg.sender, oneStar);`

Exhibit C - Comment and Documentation Typos and Potential Improvements

Suggestions taken into consideration by various commits.

Ships

- Consider adding a comment above the **getSpawned** function clarifying that the functions is only useful for user interface purposes since Solidity does not yet support returning dynamically sized arrays.
- Consider adding the following information to the comment at the top of the file:
    - An operator is an address who is authorized to transfer all of a different addresses ships, while a transfer proxy is an address that is allowed to transfer a single ship for a specified address. It would also be helpful to explain how these concepts interact with ERC-721 (i.e. proxies enable **approve** functionality and operators enable **approveAll** functionality).
    - Also explain that a spawn proxy is an address that is allowed to spawn ships on behalf of another prefix (address).

Polls

- Update the comment above the state variable **concreteMajorityMap** from: “if we did not store this, we would have to look at old poll data to see whether or not a proposal has achieved majority. but since the results calculated from poll data rely on contract configuration that may not be accurate across time. by storing majority flags explicitly, we can always tell with certainty whether or not a majority was achieved.” to “If we did not store this, we would have to look at old poll data to see whether or not a proposal has ever achieved majority. Since the outcome of a poll is calculated based on contract configuration that may not be accurate across time, we need to store outcomes explicitly instead of re-calculating them, so that we can always tell with certainty whether or not a majority was achieved, regardless of the contract’s current configuration.”

- Consider improving the comment at the top of the file to provide more details about the **Polls** contract. Below is a non-exhaustive list of things the comment could cover:
    - Polls require at least **25%** (with integer math based rounding) to pass.
    - Polls require a strict **50% + 1** majority of galaxies to pass.
    - Galaxies may only vote for a given **Poll** once, and once a Galaxy has voted the vote cannot be changed, even when within the valid voting period.
    - Votes for a given **Poll** (concrete or abstract) can never be repeated once a majority has been achieved.
    - Once a poll has started, the **Poll** has **start + pollDuration** time to achieve a majority.
        - If the **Poll** does not achieve a majority within the specified time frame, **pollCooldown** must elapse before voting for the same **Poll** can begin again.
    - There is no “finalization” period for Polls, as soon as the final vote required to achieve majority is cast, the **Poll** is over and no further transactions are required to finalize it.

Censures

- The comments above **censures** and **indexes** should be updated to make it explicit that the first uint16 is the prefix for the ship doing the censure, not the ship being censured.
- Consider improving the comment at the top of the file by explicitly stating that censures can be forgiven.

Claims

- Consider improving the comment at the top of the file by adding the following:
    - Explicitly state that only **16** claims can be made per ship.
    - Explain that the owner of the **Ships** contract can clear all claims associated with a ship for the purposes of ship transfers.

Constitution

- The comment “prevent galaxies from spawning planets” in the **spawn** function should be made more explicit. In practice, the require statement prevents ships from spawning other ships that are not exactly one class below (or above based on how the enum is set up).

- The comment in **canEscapeTo** “We must escape to a sponsor of the same class, except in the special case where the escaping ship hasn't been born yet -- to support lightweight invitation chains.” seems inaccurate. Consider changing it to say “We must escape to a sponsor of a higher class, except in the special case where the escaping ship hasn't been born yet, in which case the escaping ship can escape to a ship of an identical class in order to support lightweight invitation chains.”

ConditionalStarRelease

- The comment at the top of the file refers to a **Votes** contract repeatedly, but we think it should be referring to the **Polls** contract.
- Consider improving the comment at the top of the file more useful by clarifying the following:
    - A **ConditionalStarRelease** contract represents a single set of conditions and milestones that must be achieved in order for participants to receive their allocated stars.
    - A single **ConditionalStarRelease** contract may have multiple participants who all share the same conditions and milestones, but may receive varying amounts of stars per condition at varying rates.
    - Stars are always given to participants starting from the lowest index.
    - Stars are always withdrawn (to the participant or back to the owner) starting at the highest index.
    - Keeping track of how many stars have already been claimed is done by incrementing the **withdrawn** field, and keeping track of which stars are forfeited is done by incrementing the **forfeited** field, which keeps track of how many remaining stars in the stars array are not available for withdrawal because they’ve been forfeited.
    - Once a star is withdrawn (to the participant, or back to the owner due to forfeiture) the **stars** array is shrunk.
- Add a comment above the **continue** in **withdrawLimit** explaining that a break can’t be used because the conditions can be met in basically any arbitrary order.
- Consider adding comments explaining that **verifyBalance**, **getTranches**, and **getRemainingStars** are for client use only.

LinearStarRelease

- Consider adding comments explaining that **verifyBalance** and **getRemainingStars** are for client use only.

PlanetSale

- Change the comment at the top of the file from “This contract is intended to be deployed by star owners that want to sell that planets on-chain.” to “This contract is intended to be deployed by star owners who want to sell their planets on-chain.”

DelegatedSending

- Explain in the comment at the top of the file that:
    - Stars that want to use this contract must make the **DelegatedSending** contract a spawn proxy.

Pool

- The comment above the deposit function is highly repetitive.

Exhibit D -Disclaimer

Bloctrax makes no warranties, either express, implied, statutory, or otherwise, with respect to the services or deliverables provided in this report, and Bloctrax specifically disclaims all implied warranties or merchantability, fitness for a particular purpose, noninfringement and those arising from a course of dealing, usage or trade with respect thereto, and all such warranties are hereby excluded to the fullest extent permitted by law.

Bloctrax will not be liable for any lost profits, business, contracts, revenue, goodwill, production, anticipated savings, loss of data, or costs of procurement of substitute goods or services or for any claim or demand against company by any other party. In no event will Bloctrax be liable for consequential, incidental, special, indirect, or exemplary damages arising out of this agreement or any work statement, however caused and (to the fullest extent permitted by law) under any theory of liability (including negligence), even if Bloctrax has been advised of the possibility of such damages.

The scope of this report and review is limited to a review of only the code presented by the Urbit team and only the source code Bloctrax notes as being within the scope of Bloctrax's review within this report. This report does not include an audit of the deployment scripts used to deploy the Solidity contracts in the repository corresponding to this audit. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. In this report you may through hypertext or other computer links, gain access to websites operated by persons other than Bloctrax. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such websites' owners. You agree that Bloctrax is not responsible for the content or operation of such websites, and that Bloctrax shall have no liability to your or any other person or entity for the use of third party websites. Bloctrax assumes no responsibility for the use of third party software and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.