Second Round:

- The second round started on August 2nd with this commit:
    - https://github.com/xBacked-DAO/xbacked-contracts/commit/b95c8697d2631a96a446f9e56181e9770aa438cb
- We received a spec about the contract
    - 8840bddf8790f53bb41cd5910f9be3a10dd0d488287c8464a35bd7078cb82a00 Master_Chef_Staking_Spec.pdf
- First, we reviewed the spec and took notes about it
    - “Amount of rewards ‘deposited’ in the contract should match the ratio set, or stakers will be paid out at the highest amount that satisfies the reward ratios”
        - The phrasing of this suggests that thing after the OR is bad for someone
        - Presumably, it is best for it to be bad for the user, not for xBacked
        - Is that the case?
        - Is it possible to incorporate a concept like “slippage” so that a rewarder can request that their deposit be rejected if it doesn’t behave as they expect it will?
    - The use of `lastConsensusTime`...
        - This means that users need to pay extra for rewards, because they may need to call `updateLastConsensus` before using the reward function.
        - Presumably your UI will deal with this, but it feels bad for the users
        - It is possible in Reach to use `thisConsensusTime` in consensus steps to avoid this problem
    - Re: `withdrawRewards`...
        - I don’t understand what it means to “cache rewards prior to checking the staker’s reward balance”
        - I assume this means that you used to call the `updateRewards` function prior to withdrawing.
        - If you had a version of the code that was failing, and you ran with intermediate files, I would be able to tell you exactly which theorem was hanging it. That may help figure out
            - Whether it is actually wrong
            - Whether there’s an extra check or assertion you could add to speed it up
        - I have some ideas below, in the General Reach Advice section
- Second, we read the diffs on the commit (compared to the version from the first round)
    - https://github.com/xBacked-DAO/xbacked-contracts/compare/46be0a39f9273e49eef2ac5bc4ff7fbfa0f6f5ef..b95c8697d2631a96a446f9e56181e9770aa438cb
    - The diff contains changes to `vault*.rsh` and `stability_pool.rsh`, but we ignored these files
- Next, we studied the new versions directly
- Then we wrote up new grades for each component

Program specification - A

- The specification is excellent, as a specification for a program.
- It is not an A+, because it relies too much on using the program itself as a source of truth (e.g., the code samples as a way to define the reward rate).
- There is a trade-off here. Basically your options are:
    - English prose
    - Reach code
    - Other code
- The problem is that the point of a spec is to have something that you can judge conformance against, i.e. you will ask the question “Does this code do the thing which that spec says it should?”
- The problem with prose is that it is ambiguous and it may be hard to see if the code conforms.
- The problem with Reach code is that it automatically conforms, so there’s no test.
- The problem with Other code may be that it can’t contain all the details and nuances that Reach code can.
- In this case, I think it would be possible to e.g. produce an Excel spreadsheet that shows what the rewards should be for various parties given various scenarios. The Excel would effectively be “Other code”
- In this case, however, I include the testing system as part of the specification and assume that the many many test scenarios there capture the intended behavior of the system.

General Reach Advice - A

- I continue to not like that there is no invariant about what the balance should be.
- This is why you have the check at line 365, instead if you have at the invariant that e.g. `remainingRewards[index] == getAssetBalance[index]`, then you would not need that.
- This would decrease the size of the generated program and increase your confidence that the predictions (e.g. `remainingRewards`) match the actual.
- It could be that this is why the reward updating fails when you withdraw, because there’s kind of a double-update to these values going on
- reward_utils.sh
    - Line 23, I believe that Algorand is going to change the blockrate soon, so you should be a tiny bit nervous about that
    - Line 44, FYI, we just did the hard part of making it so that `?:` is lazy as you expect
- token_utils.rsh
    - Line 12, I don’t understand why you can’t use `distinct`
- distribution_utils.rsh
    - Line 55, in `normalizeAndSelectTotalRewards`, this `check` should be an `assert`
        - It is probably statically dispatched anyways, but you should convert it to be sure
- master_staking.rsh

    - Line 149, I think it is better to pretend they are trustworthy and prove the powerful loop invariants
    - Line 442, You've formatted `! ( x && ( y || z ))` in a confusing way to my taste. I also think that it is a tiny bit confusing to have so many connectives. I would prefer `!x || (!y && !z)` [using De Morgan's Laws], but this is just a taste thing
    - With functions like `updateRewards`, I am very anxious about getting the order of arguments correct and so I like to pass objects as a way to do "by-name" arguments.
    - Line 795 & 800, This might be unnecessary. When the program ends, Reach will close out every account to the creator. If the admin stays the creator the whole time, then this will have the exact same effect.

Test coverage - B+

- In our process, we did not actually run your tests, but we assumed that they run correctly and instead evaluated whether they are good tests.
- The in-Reach verification strategy is inspiring.
- It could be better if there were a larger suite of real contract scenarios, i.e.
    - Create a bunch of users
    - Have them join a pool with certain reward ratios and rates
    - Cause time to pass
    - Harvest the rewards
- The closest you have to this is the "advanced scenario" test, but it is unsatisfying to me, because it does not check that the outcome is as expected. It just checks that there are no errors in operation.

Spec Conformance - A-

- In `unscaledRewardBreakdown` in `distribution_utils.rsh`, there is a comment about rewards lost to rounding. This is not in the spec and I don't example tests that demonstrate this scenario and show what happens.
    - Ideally, you will
        1. Add it to the spec
        2. Include verification tests that demonstrate the scenario
        3. Implement, just in the utility file, the "correct" version.
        4. Demonstrate via verification tests what the correct behavior is
        5. Show what that the effect is small and acceptable
- cont…
    - Next, I don't understand how "lost" these are. Are they lost to the staking pool or are they lost to the user? The first is more serious than the second.
        - I believe the answer is that they are only lost to the user.

First Round:

- We established what code is being audited
    - We were initially told
        - https://github.com/xBacked-DAO/xbacked-contracts/blob/main/src/master_staking.rsh
    - But it was replaced with
        - https://github.com/xBacked-DAO/xbacked-contracts/blob/refactor/stability-and-master-staking/src/master_staking.rsh
    - This file was later removed and we found
        - https://github.com/xBacked-DAO/xbacked-contracts/blob/refactor/master-staking/src/master_staking.rsh
    - Because we are worried about that link changing too, we are fixing the target to
        - https://github.com/xBacked-DAO/xbacked-contracts/blob/46be0a39f9273e49eef2ac5bc4ff7fbfa0f6f5ef/src/master_staking.rsh
- Next, we confirm the deadline
    - End of June is good; We are targeting June 24th, but we do not have a strict deadline.
- Next, we cover what will be reviewed and in what priority.
- First, we will look at general “details” about the Reach code as we try to understand the program.
    - This is to provide general Reach programming advice
- Next, we ask whether they want code changes?
    - That’s fine, but important to stay within the same behavior.
    - Ideally, we can propose new assertions and invariants as well as general improvements.
    - It is risky to do this, because we may not have a perfect understanding of what is “same behavior”
- We solidify the “idea” of the program
    - A “staking program”.
- We interview and ask questions about what this idea means and what the spec is.
    - It is rare for teams to have formal specs, so this is an important, informal step where we find out what their thought process is and what the program is supposed to do.
    - This is a back-and-forth and sometimes the PM on the other side has to get deeper information.
- Scenario: We receive ALGOs from the Algorand Foundation. They want to give xBacked ALGOs to distribute to users and xBacked must also give something (eg. another asset).
    - 1st month is reward of ALGOs, 100/0. 2nd month is 50/50. 3rd month is 75/25. 4th month is 0/100.
    - They can stake xUSD
    - Can’t stake ALGOs and don’t care
    - Rules: Can collect rewards at any time. Unstake at any time and get the claim of rewards.

    - Reward per block --- Reward is % of the stake in the block
    - The admin can control the reward ratio.
    - The admin can change the reward rate.
    - Austin wants the pending rewards to be based on the old ratio and rate, but any future rewards to be based on the new ratio and rate. --- Austin will check?
        - **Answer: Total rewards emitted before update can be updated, but the reward rate is only used when a user claims their rewards**
    - If Austin & Jay both stake the same % at the same time, it doesn't matter if Austin claims multiple times. Do they do the MasterChef thing?
        - **Answer: It doesn't matter, Jay is still entitled to his proportion of emitted rewards since his last interaction.**
    - We will confirm with you if the code doesn't actually do this, so you can change your mind and you won't get a "flaw" in your report.
    - What happens if there's a period with no stakers? Doesn't matter, we reward a specific amount.
        - **Answer: No stakers means no rewards are emitted, however with one staker they will receive all rewards themselves, this makes the APY dynamic as others compete for the rewards.**
    - How do you prove that someone is staking? Local storage. This only hurts them. If someone ClearStates, then this is just like them failing to call Claim, and that's not a problem, but they might get deprecated.
    - When you are deprecated, do you ever steal peoples' stake? We don't want to, but we want to get the rewards out.
        - **Answer: Considering adding a timeout to the deprecated state, where after a period e.g. two weeks the admin can force close and withdraw all unclaimed assets.**
- During the interview, we are asking "Do you understand your own code idea?" and "Is this idea a good one?"
    - Some of my comments so far are like that.
    - We consider it out of scope (and thus prioritized last) to do an exhaustive "lit review" of the problem domain and suggest big changes
    - Jay will prioritize a "staking lit review" last, and this is based on MasterChef and Synthetix
- The main part of the audit will be comparing the given code to the "idea" we've specced out in the interview.
    - This is the meat of the audit and I'll talk about the mismatch if any.
- We will also review the tests
    - So we need to know where they are…
        - https://github.com/xBacked-DAO/xbacked-contracts/blob/refactor/stability-and-master-staking/src/frontend/tests/master_staking.test.ts
        - We are using this specific revision — https://github.com/xBacked-DAO/xbacked-contracts/blob/46be0a39f9273e49eef2ac5bc4ff7fbfa0f6f5ef/src/frontend/tests/master_staking.test.ts
    - Tests are after the code audit and before anything else

- Finally, we discuss how looking at the compiled code is not the main part of the audit, but we will look at it with time
    - Jay will use the TEAL static analyzer and read through the compiled code looking for weirdness.
- In summary, the schedule:
    - Kickoff expectation setting
    - Kickoff interview
    - Program Specification
    - General Reach advice
    - Test coverage
    - Spec conformance
    - Compiled code analysis
    - Literature review

Program specification - D

- During our kickoff interview, we talked about the specification of the program.
- My summary of this section of the analysis:
    - This program's spec fails to live up to a high standard.
    - There's no clear model of what the program should do: no written spec, no spreadsheets indicating the expected formulae, etc.
    - There's a general sense that "It should be like MasterChef or Synthetix" but with no specific claim about which behaviors of those should be replicated.
    - My score comes from this being a "well understood domain"
    - I recommend the following:
        - Write a broad spec about what this project entails
            - This is the text that I found from our pre-kickoff notes:
            - Requirements
            - Can pay out rewards in different tokens. Likley would just need to support Algos + 1 other(?)
            -
            - Can distribute Algos & project tokens, for example
            - Dynamic APY based on supply/demand for rewards
            -
            - Defines reward ratios between the two reward tokens
            -
            - E.g 70/30 algos / project tokens
            - This ratio is dynamic; can be updated to 50/50 down the track
            - Allows for a deadline date / end of rewards to close the contract
            -
            - Allows for more rewards to be deposited by anyone
            -
            - Accounts for untrackedFunds

        - Be clear about what innovations you're offering compared to alternatives (e.g. the two different reward strategies, the reward ratio system, and the ability to change these dynamically)
        - For each innovation, brainstorm about the unique challenges that it introduces and how you're compensating or dealign with them.

General Reach advice - B

- The first half of this was done by June 16th, the comments are archived in this gist:
    - https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9
- They do not include comments on the math formulas in reward_utils.rsh, which will be added separately.
- This one contains the comments on the reward formulas:
    - https://gist.github.com/ac2298c0c1f010f58a8b6e54f7bff92b
- My summary of this section of the analysis:
    - I am extremely critical of code style, but I also have really weird idiosyncratic taste, so I would underweight my rating on your code.
    - Furthermore, I am literally the creator of Reach, so I know the ins-and-outs of what you can do really well, so this gives me a heightened sense of ways to make improvements.
    - With that caveat, I think there are lots of ways that you can improve the readability and performance of this code.
    - There is basically just one big glaring problem with it from a "general Reach program" perspective… it basically ignores the invariant system and performs all checks dynamically.
    - This is not wrong, but it can obscure the correctness of the code.
    - My advice:
        - Use the new Reach features I suggest
        - Build unit tests and add comments for "tricky" things
        - Change functions that take many arguments into functions that take one object with shared keys/variable names.
        - Remove all checks and then try to slowly maximize the invariant and then only afterwards add checks for things you know are specifically about user input validation and not general properties about the state of the program.

General Comments from
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9

Fixes and changes in response to these comments can be found on the branch; audit-jay/master-staking (Jay looked at commit d9f6d4c and left a bunch of comments on the PR)

Link	Note	Resolved

https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L23 https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L41	Agreed that we should create type aliases for the arrays etc that rely on this constant, but I still think that it is a good way of handling this for configurability.	
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L53	Agreed, not using withDisconnect yet but we can disconnect after isInitialized anyway.	ebfa5439581de0fdaa7794f014c78c45bff4ebd5
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L63	Handled by our format rules, we can decide as a team if we want to adjust this, I am indifferent to this change.	Not a priority
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L72	I think an array of type TokenInfo = Tuple(Token, UInt) (Token id, decimals) which contains the staking info it index 0, and the reward assets from index 1 onwards called supportedAssets or something would be a good replacement.	
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L83	Agreed, I am happy to consider renaming these, but it's important to capture the distinction that whilst deprecatedAt represents a block in the future (when set), deprecateTimeout represents a number of blocks e.g. 20, that when added on to the deprecatedAt time represents the time the contract can be closed regardless of the state	
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L99		
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede	The idea is that many of these functions will probably	

51c5fd627838a9#file-202206-audit-review1-diff-L99	end up in shared files, so that we can construct new contracts with independently tested and hopefully validated pieces	
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L151 https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L182	Will be moved to a separate file and have its own dedicated test suite	
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L192	I think this isn't a massive issue but I will bring it up with the team or at least leave a comment explaining this behavior	
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L217	:facepalm:	9ccf96fbb91f7f24bd38ef1690ed11f0105c06fc
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L217	Happy to leave this for now, if you can provide more reasons why this might be bad I am open to discussing this further	
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L250	Good pickup	eb0191cd4c4501836877820a15e1345557cfda66
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L259	Could I get more clarification on this? The invariant would contain a validation function taking the mutable admin properties object, ensuring that at all stages of the parallelReduce they are valid?	
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L269	Similar to above, we would have the invariant contain validateAdminParams(.....) && validateRemainingRewards	

	etc?	
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L277 https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L285		debc81419d266cd7bdd4cfe469fb7c8ec93a84f8
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L293		ce0cc42dbbe8fbb119866223ebdea70601ec531b
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L302		
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L309		7ee396e3c0d4e5270aa7198e8642cee49a774657
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L322	The idea is that in this contract the cold state is for any state that isn't modified after initialization, and as such is never modified in the parallelReduce, so it would seem unnecessary to update this view in the define block?	7ee396e3c0d4e5270aa7198e8642cee49a774657
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L339	Single transfer?! Will do. Agreed but the idea behind the tuples and arrays is that it may be possible in the future to modify the number of supported reward assets by only adjusting the constant `NUM_EXTRA_REWARD_ASSETS`	fce9f2ae4be6218a8252d80c4f3500b1358af815
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L349	Yeh, the idea is in the future a client could read the cold state and adjust the frontend accordingly, e.g. look up the ASA and fetch the name, icon etc.	No changes needed

https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L358	Validating the state of the parallel reduce seems to be something that would make sense to have here based on your other comments	91313b25617f286f32efadb018223a3264d2eb11
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L369	I was unable to use this with the withdrawRewards api endpoint as it lead to verification timeout despite careful inspection to ensure that nothing new was added	d9f6d4c40ad47b16c6396b9d119ff84eeb2b2af1

Comments specifically on reward funcs
https://gist.github.com/ac2298c0c1f010f58a8b6e54f7bff92b

Link	**Note**	**Resolved**
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L121	Massive issue, will fix immediately.	7957769052e540fd4ed2eadb90b38acfcec5b98f
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L137	Agreed, we should process the power calculation and store that separately for later use, since this value is static.	7957769052e540fd4ed2eadb90b38acfcec5b98f
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L156	I have also determined this is flat out wrong, and I have replaced it with a correct implementation.	b199e8c63c69d58fda7ad56da2efd9c5d882e659
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L199	Comments would have made a big difference here, this function is calculating what percentage of rewards the contract can pay, e.g. if a user has 1000 rewards and that equates to 1000 ALGO (micro units), but the contract only has a balance of 8000, but can pay out the other rewards fully, the rewards are scaled down to 80% the lowest amount, and the user would only be deducted 800	8d1ea643fe0230a49d5305f83dde7da5c3190342

	rewards.	
https://gist.github.com/jeapostrophe/e9ed72dc92d27deede51c5fd627838a9#file-202206-audit-review1-diff-L217	Some comments here explaining that all the StakerAPI functions will 'cache' rewards, and the withdrawRewards function will take into account the un-cached rewards, e.g. 1000 + (1000) uncached rewards, user can withdraw 2000.	

Test coverage - D-

- We did an initial investigation of the test suite and found that it is mostly about making sure the interfaces are appropriate and it checks to make sure that the funds at the end are zero.
- But, there are no functional tests that determine that the correct amounts are given to users. For example, there's no test that if Jay stakes 10 and Austin stakes 20, that Austin gets two times what Jay gets and that amount is 50.
- My summary of this section of the analysis:
    - The testing fails to live up to a high standard.
    - The tests essentially amount to "Do we have a library that calls function F on the contract?" and only grossly checks that the contract does the right thing.
    - The main checks for contract safety are:
        - Is the state diagram protected? (i.e. Cannot deposit after the contract is over)
        - Are the admin functions protected?
        - Are the ratios valid?
    - There are no functional tests about expected reward amounts.
    - There are no "exceptional" tests about the kinds of unexpected behaviors we discussed in the interview. For example,
        - How do the rewards of two parties correlate if one harvests often and the other doesn't harvest but once?
        - What is the expected behavior when reward ratios and amounts change mid-staking?
        - Are rounding and saturation math returning sensible answers?
        - Etc
    - I recommend the following:
        - Have the product owner write down some functional requirements independently
        - Have the engineer think about some exceptional scenarios
        - Have the engineer build out both of these sets of tests

        - Have the engineer work through some functional/unit tests of the formulae and compare with product owner's sense
        - Run unit tests on (a) Reach code directly, (b) Reach code extracted in JS, and (c) the actual program in a live situation. [This assumes that B is easier than A is easier than C, so you don't do all for C; or, build a corpus of tests and a mechanism whereby you can run all the tests on all modes.]

Spec conformance - F

- We discovered two important problems during the code walkthrough:
    - The `XOR` problem in `normalizeAssets` and `scaleAssets` is critical.
        - These functions do not do what you think they do.
        - I have multi-faceted recommendations:
            - Don't need to do this at all :)
            - Use pow if you want to stick with decimals
            - Change to not use decimals, but have a general conversion factor
    - The lack of a meaningful invariant is critical.
        - Here is a video about the value of invariants — https://youtu.be/HGUCvIskpCA
        - This is a performance problem, because you do unnecessary checks (i.e. checks that are statically dispatchable).
        - This is a correctness problem, because it means that you do not know what should be invariant about your state.
        - Our recommendation…
            - Enforce a relationship between `deprecateTimeout` and `deprecateAt`
            - Enforce sum-to-100 of `rewardRatios`
            - Enforce relationship between `rewardRate` and `deprecateAt`
            - Enforce that `remainingRewards` is equal to the balances in the contract
            - Enforce relationship between `totalRewards` and `remainingRewards`
            - Enforce that `lastRewardBlock` is in the past
            - Enforce that `totalDeposit` is the balance of the contract
            - Enforce relationship between `rewardPerToken` and other reward variables
            - Enforce that `totalDeposit` is sum of user `amountDeposited`s
- We tried to answer specific questions about the behavior of the contract relative to some spec properties:
    - "If Austin & Jay both stake the same % at the same time, it doesn't matter if Austin claims multiple times."
        - There is no test for this property in the code base.
        - I don't believe that this is true, but I have not found a "violation".

        - I don't believe it because the amount of remaining rewards always goes down and Jay may never harvest, so even though Austin gets 1%, if he continually gets his 1%, he'll eventually take everything and Jay won't be able to get anything at the end.
        - I don't believe that staking systems generally have this property, so I think this is an example of failed/misunderstood expectations
        - The literature review might change my mind about this, but at the very least, there needs to be tests about this.
    - "What happens if there's a period with no stakers?"
        - There is no test for this property in the code
        - The time delta enters into the computation, so I am suspicious that it does not hold
        - My advice is to build tests where there are periods of no staking throughout the process and ensure that the overall results are the same.
    - "When you are deprecated, do you ever steal peoples' stake?"
        - There is no test for this property in the code.
        - It absolutely does not hold… the contract takes from everyone.
        - My advice is to change it so deprecation means that you can't deposit or something and that the remaining rewards are returned to the creator, but withdrawals are still allowed.
- Next, we tried to come up with a general property that could be checked
    - "If Jay stakes N tokens, he will get exactly N tokens out later."
        - We believe this holds
    - "If Jay stakes N tokens and there is a reward rate of R per block per token, then if he harvests after P blocks, then he will get R x P x N rewards"
        - We believe this does not hold, on purpose, because the rewards may be exhausted by the time a user harvests
    - "If Jay stakes N tokens and there is a reward rate of R per block per token, then if he harvests after P blocks, then he will get min(rewardBalance, R x P x N) rewards"
        - We believe this does not hold, because it is missing the bigger picture, because there's a reward rate for the entire system, but not a reward rate for an individual player, because they only get a % based on how much they are staking.
    - "If Jay stakes N tokens and there is a reward rate of R reward points per block per token, then if he harvests after P blocks, then he will have NxRxP reward points. On harvest events, reward points are immediately cashed out for tokens such that the user gets (theirPoints / allPoints) x rewardRemaining?"
        - We believe that this is closer to the expectation, but we're very unsure because there's no clear statement in the code (or tests) about what exactly the expectation is.
        - I investigated MasterChef and its source and could not find a similarly clear statement.

        - We want a clear statement from xBacked about what they think it should do before we can audit if it really does that.
    - “If Jay stakes more than Austin for long, he gets more”
        - There are no tests for this property, but we believe it holds.
- Finally, we have a code/spec recommendation…
    - We think you should combine the staker actions of deposit / withdraw / harvest into a single action
    - stakerUpdate :: (in:Stake) x (out:Stake)
    - This will simultaneously
        - Deposit in (maybe 0)
        - Withdraw out (maybe 0)
        - Harvest bonus (maybe 0)
    - At the very least, you should structure you code this way so that stakerDeposit calls stakerUpdate(in, 0) and stakerWithdraw calls stakerUpdate(0, out) and harvest calls stakerUpdate(0, 0)
    - We think this will make your code easier to test and it will help you ensure that you accurate reward for the portion of the amount that they actually had
    - For example, if you can deposit without being rewarded for what you’ve done so far, then there’s a problem vector of having 10% stake for 9 years and then 90% stake for 1 year and accidentally giving a 90% share of rewards for an 10 year period.
    - MasterChef separates rewarding and withdraw/deposit because ETH is so bad and expensive. You don’t have that problem on ALGO so you can be more accurate and safe.
    - Even if you think you’ve solved this attack vector, you want your code to be as obviously correct as possible.

Compiled code analysis

- I studied the cost analysis that Reach produces
    - https://gist.github.com/jeapostrophe/960a2947586409ba34c32ac8d7ea7b45
    - The `api_StakingUserAPI_withdrawRewards` is about 400 units over the limit and this pushes the rest of the program up
    - I think it is feasible to optimize your program (i.e. for YOU to optimize it) to go below 700 and go to 75% of the costs for your users
- I found an optimization opportunity for Reach to run on your program while I was studying your code. This reduces your costs by a little bit (38 ops) but space by a lot (~900). [You can see it by looking at the revisions.]

Literature review

- MasterChef has no white-paper and the code is very hard to follow for me
- Synthetix is easier to follow for me. One of their main innovations is that they only reward you for the past seven days and it is up to you to come get your percentage for the last

week or it will be lost. This seems like a compromise to try to clarify many of the issues discussed about in the specification conformance section.

- A more successful literature review would be if YOU said “We want to be like these three white papers”. I feel like I must be stupid because there must be some explanation of what MasterChef is supposed to do, but I didn’t find it.