OBELISK

Part of Tibereum Group

AUDITING REPORT

Version Notes

Version	No. Pages	Date	Revised By	Notes
1.0	Total: 56	2022-06-30	DoD4uFN, Donut	Audit Final

Audit Notes

Audit Date	2022-03-16 - 2022-06-30
Auditor/Auditors	DoD4uFN, Plemonade
Auditor/Auditors Contact Information	contact@obeliskauditing.com
Notes	Specified code and contracts are audited for security flaws. UI/UX (website), logic, team, and tokenomics are not audited.
Audit Report Number	OB548632585

Disclaimer

This audit is not financial, investment, or any other kind of advice and is for informational purposes only. This report is not a substitute for doing your own research and due diligence. Obelisk is not responsible or liable for any loss, damage, or otherwise caused by reliance on this report for any purpose. Obelisk has based this audit report solely on the information provided by the audited party and on facts that existed before or during the audit being conducted. Obelisk is not responsible for any outcome, including changes done to the contract/contracts after the audit was published. This audit is fully objective and only discerns what the contract is saying without adding any opinion to it. The audit is paid by the project but neither the auditors nor Obelisk has any other connection to the project and has no obligations other than to publish an objective report. Obelisk will always publish its findings regardless of the outcome of the findings. The audit only covers the subject areas detailed in this report and unless specifically stated, nothing else has been audited. Obelisk assumes that the provided information and material were not altered, suppressed, or misleading. This report is published by Obelisk, and Obelisk has sole ownership of this report. Use of this report for any reason other than for informational purposes on the subjects reviewed in this report including the use of any part of this report is prohibited without the express written consent of Obelisk. In instances where an auditor or team member has a personal connection with the audited project, that auditor or team member will be excluded from viewing or impacting any internal communication regarding the specific audit.

Obelisk Auditing

Defi is a relatively new concept but has seen exponential growth to a point where there is a multitude of new projects created every day. In a fast-paced world like this, there will also be an enormous amount of scams. The scams have become so elaborate that it's hard for the common investor to trust a project, even though it could be legit. We saw a need for creating high-quality audits at a fast phase to keep up with the constantly expanding market. With the Obelisk stamp of approval, a legitimate project can easily grow its user base exponentially in a world where trust means everything. Obelisk Auditing consists of a group of security experts that specialize in security and structural operations, with previous work experience from among other things, PricewaterhouseCoopers. All our audits will always be conducted by at least two independent auditors for maximum security and professionalism.

As a comprehensive security firm, Obelisk provides all kinds of audits and project assistance.

Audit Information

The auditors always conducted a manual visual inspection of the code to find security flaws that automatic tests would not find. Comprehensive tests are also conducted in a specific test environment that utilizes exact copies of the published contract.

While conducting the audit, the Obelisk security team uses best practices to ensure that the reviewed contracts are thoroughly examined against all angles of attack. This is done by evaluating the codebase and whether it gives rise to significant risks. During the audit, Obelisk assesses the risks and assigns a risk level to each section together with an explanatory comment. Take note that the comments from the project team are their opinion and not the opinion of Obelisk.

Table of Contents

Project Information

Name	RaidParty
Description	Join the largest MMO on Ethereum, slaying enemies and meeting comrades along the way.
Website	https://raid.party/
Contact	@raidparty on Twitter
Contact information	@hasan#1111 on Discord
Token Name(s)	N/A
Token Short	N/A
Contract(s)	See Appendix A
Code Language	Solidity
Chain	Ethereum

Audit of RaidParty

Obelisk was commissioned by RaidParty on the 12th of March 2022 to conduct a comprehensive audit of RaidPartys' contracts. The following audit was conducted between the 16th of March 2022 and the 30th of June 2022. Two of Obelisk's security experts went through the related contracts manually using industry standards to find if any vulnerabilities could be exploited either by the project team or users. The reason for the long time between start of the audit and finalisation is that some contracts was worked on during the audit that required some back and forth between the project and auditors.

During the audit of the provided contracts, our auditors found nine issues that could be of some concern. They are ranked from High-Risk->Medium-Risk->Low-Risk and can be seen in the table below. Three of these issues, #4, #6, and #8 was either closed or mitigated, and issue #5 was partly closed during the audit. The remaining issues #1, #2, #3, #7 and #22 is still open. To know more about these open issues and their impact, please refer to the issues in the Findings section.

The informational findings are good to know while interacting with the project but don't directly damage the project in its current state, hence it's up to the project team if they deem that it's worth solving these issues.

The team has not reviewed the UI/UX, logic, team, or tokenomics of the RaidParty project**.**

This document is a summary of the findings that the auditors found. Please read the full document for a complete understanding of the audit.

Summary Table

Code Analysis

Finding	ID	Severity	Status
Request Can Revert Or Never Get Executed	#0001	High Risk	Open
States Gets Out of Sync	#0002	High Risk	Open
Local State Can Get Past Global State	#0003	High Risk	Open
Chainlink Seed Not Always Updated / Frontrunnable Seed	#0022	Medium Risk	Open
Reliance On Chainlink Multisig	#0004	Low Risk	Mitigated
Modulo Of Zero	#0005	Low Risk	Partially Closed
Variables Could Overflow/Underflow In Certain Conditions	#0006	Low Risk	Closed
Claiming Rewards Can Be Halted	#0007	Low Risk	Open
Execute Request Can Be Rendered Unusable	#0008	Low Risk	Mitigated
Code Readability	#0009	Informational	Closed
Unequipping Can Be Halted	#0010	Informational	Open
Callback To User	#0011	Informational	Mitigated
Incorrect Error Message	#0012	Informational	Closed
Centralised Metadata Storage	#0013	Informational	Open
Enhanceable Function Should Be Separate	#0014	Informational	Closed
Unused Contract	#0015	Informational	Closed
Redundant Boolean Statement	#0016	Informational	Open
Make Use Of Protocol Functions	#0017	Informational	Open
Unnecessary Return Value	#0018	Informational	Open

No Events Emitted For Changes To Protocol Values	#0019	Informational	Partially Closed
Deterministic Seed	#0020	Informational	Open
Complex Damage Calculation	#0021	Informational	Open

Findings

Code Analysis

Request Can Revert Or Never Get Executed

FINDING ID	#0001
SEVERITY	High Risk
STATUS	Open
LOCATION	randomness/SeederV2.sol

DESCRIPTION	If the subscription does not have enough link available for 24 hours then, the pending requests will expire. Also, Chainlink's *fulfillRandomWords* could fail due to the amount of Link not being enough to pay the request (Chainlink's *fulfillRandomWords* function in *VRFCoordinatorV2.sol* has a revert for this). There is no on-chain way to ensure that Chainlink will try to recall *fulfillRandomWords* when it reverts with *InsufficientBalance()* thus, Obelisk has to assume the worst. Such a case can also cause the *requestID* to never be fulfilled. If a request is never fulfilled, it would cause the *randomnessId* never to be usable (i.e. no seed for that *randomnessID*).
RECOMMENDATION	Make sure that the Chainlink subscription does not run out of Link.
RESOLUTION	The team introduced a check within the contract to ensure that the coordinator has a balance of at least 20 Link. However, the fees depend on Chainlink's link oracle pricing. **Note:** Fixes to the issue have not been deployed on-chain. The project team has acknowledged this.

States Gets Out of Sync

FINDING ID	#0002
SEVERITY	High Risk
STATUS	Open
LOCATION	core/Raid.sol

DESCRIPTION	Currently, there are two ways to access the state. Globally and on a per-user basis. Users can be ahead or behind the global state (i.e behind, current(same as global), or ahead). Rewards use the variables from the current global state (boss, weights), not from variables at a saved point in time. All of these have an influence on how the next state will proceed. This means that sometimes when a user's state differs from the global state, users might get different rewards than if they were synced. Updating a boss, or adding new bosses will cause people to fall out of a correct state and get different rewards. If they haven't finished that boss they might end up with a different one as the calculation returns to the global state for the boss or weights. Note: upgrading the contract could also cause users to get out of sync. However, a user can sync from his local round to the global round without diverting from his expected rewards as long as there are no changes to the global variables since his last sync.
RECOMMENDATION	Obelisk recommends migrating to a masterchef type system where everyone has a global state without updating each user's local state. More specifically, update the total damage for everyone currently fighting and the % of their allocations when they add or remove elements of their party. This way the global state is preserved, and each user has a predetermined debt to be paid when they withdraw.
RESOLUTION	**Project team comment:** "The system has carefully been designed to not go out of sync, it is also this way to reduce

gas consumption. We invite the auditors to either prove they can break the state, or show that it will most cost effective in terms of gas used."

Obelisk comment:
Getting out of sync with the reward's calculation is still possible. When a boss change occurs and a user has already claimed or cached their rewards to claim (e.g. updateDamage) or by simply not claiming. A careful actor can abuse this to get more rewards.

Local State Can Get Past Global State

FINDING ID	#0003
SEVERITY	High Risk
STATUS	Open
LOCATION	core/Raid.sol -> 372-381

```
    while (block.number > round.finalBlock) {
        _roundId += 1;
        _seed = _rollSeed(_seed);
        round = _rollRound(_seed, round.finalBlock + 1);
        boss = bosses[round.boss];

        if (_roundId >= raider.startRound) {
            rewards += _roundReward(raider, round, boss);
        }
    }
```

DESCRIPTION	Currently a user's local state is able to go past the global state *round.finalBlock*. This makes it harder to securely calculate the rewards as a user can be ahead or behind the global state. Additionally, multiple users will recalculate the same values even when they are in sync. Caching these values may save gas costs. This finding aggravates Finding #2 "States Gets Out of Sync".
RECOMMENDATION	Do not allow users to go past the global state. Update the global state instead.
RESOLUTION	**Project team comment:** "This is by design, The system has carefully been designed to not go out of sync, it is also this way to reduce gas consumption. We invite the auditors to either prove they can break the state, or show that it will most cost effective in terms of gas used." **Obelisk comment:** "When going past the global state, an attack will be able to use their party multiple times per boss (with different addresses). If the attacker can beat the current boss and the global state isn't updated (note: waiting for 1 block is necessary), the attacker can claim

	rewards multiple times on different addresses. At least three claims are possible if an attacker does it at the end of a block and at the start of the next block.”

Chainlink Seed Not Always Updated / Frontrunnable Seed

FINDING ID	#0022
SEVERITY	Medium Risk
STATUS	Open
LOCATION	core/Raid.sol -> 88-99

```
function updateSeed() external onlyRole(DEFAULT_ADMIN_ROLE) {
    if (started) {
        _syncRounds(uint32(block.number));
    }

    seed = seeder.getSeedSafe(address(this), seedId);
}

function requestSeed() external onlyRole(DEFAULT_ADMIN_ROLE) {
    seedId += 1;
    seeder.requestSeed(seedId);
}
```

LOCATION	core/Raid.sol -> 144-151

```
function manualSync(uint32 maxBlock) external {
    require(
        maxBlock > rounds[roundId].finalBlock,
        "Raid::manualSync: CANNOT_SYNC_PAST"
    );

    _syncRounds(maxBlock);
}
```

DESCRIPTION	*updateSeed()* will only update the seed from Chainlink if *requestSeed()* has been called. This means that a user has a time window where the user can read the next seed values before *updateSeed()* is called. That means a user can check if the new seed isn't as beneficial as the current one and act on it before everyone else. The new Chainlink seed is stored in a different transaction, not atomically, and there is no lock state in place to prevent anyone from using the previous/not-updated seed.
RECOMMENDATION	Introducing a locking mechanism will prevent this issue

	without changing the current state mechanism. Such a locking mechanism would activate before a new seed is requested. It would then prevent the use of any function which uses the seed (locally or globally) until the seed has been updated. Also, make sure *updateSeed()* always uses a new seed from Chainlink.
RESOLUTION	**Project team comment:** `This is by design, our model allows people to predict future bosses.`

Reliance On Chainlink Multisig

FINDING ID	#0004
SEVERITY	Low Risk
STATUS	Mitigated
LOCATION	0x271682DEB8C4E0901D1a1550aD2e64D568E69909 VRFCoordinatorV2.sol -> 804-827

```
1   function cancelSubscription(uint64 subId, address to) external
  override onlySubOwner(subId) nonReentrant {
2     if (pendingRequestExists(subId)) {
3       revert PendingRequestExists();
4     }
5     cancelSubscriptionHelper(subId, to);
6   }
7
8   function cancelSubscriptionHelper(uint64 subId, address to) private
  nonReentrant {
9     SubscriptionConfig memory subConfig =
  s_subscriptionConfigs[subId];
10    Subscription memory sub = s_subscriptions[subId];
11    uint96 balance = sub.balance;
12    // Note bounded by MAX_CONSUMERS;
13    // If no consumers, does nothing.
14    for (uint256 i = 0; i < subConfig.consumers.length; i++) {
15      delete s_consumers[subConfig.consumers[i]][subId];
16    }
17    delete s_subscriptionConfigs[subId];
18    delete s_subscriptions[subId];
19    s_totalBalance -= balance;
20    if (!LINK.transfer(to, uint256(balance))) {
21      revert InsufficientBalance();
22    }
23    emit SubscriptionCanceled(subId, to, balance);
24  }
```

LOCATION	0x271682DEB8C4E0901D1a1550aD2e64D568E69909 VRFCoordinatorV2.sol -> 556-602

```
1   function fulfillRandomWords(Proof memory proof, RequestCommitment
  memory rc) external nonReentrant returns (uint96) {
2     uint256 startGas = gasleft();
3     (bytes32 keyHash, uint256 requestId, uint256 randomness) =
  getRandomnessFromProof(proof, rc);
4
5     uint256[] memory randomWords = new uint256[](rc.numWords);
6     for (uint256 i = 0; i < rc.numWords; i++) {
7       randomWords[i] = uint256(keccak256(abi.encode(randomness, i)));
8     }
9
10    delete s_requestCommitments[requestId];
11    VRFConsumerBaseV2 v;
12    bytes memory resp =
  abi.encodeWithSelector(v.rawFulfillRandomWords.selector, requestId,
  randomWords);
13    // [comments omitted....]
14    s_config.reentrancyLock = true;
15    bool success = callWithExactGas(rc.callbackGasLimit, rc.sender,
  resp);
16    s_config.reentrancyLock = false;
17
18    // Increment the req count for fee tier selection.
19    uint64 reqCount = s_subscriptions[rc.subId].reqCount;
20    s_subscriptions[rc.subId].reqCount += 1;
21
22    // [comments omitted....]
23    uint96 payment = calculatePaymentAmount(
24      startGas,
25      s_config.gasAfterPaymentCalculation,
26      getFeeTier(reqCount),
27      tx.gasprice
28    );
29    if (s_subscriptions[rc.subId].balance < payment) {
30      revert InsufficientBalance();
31    }
32    s_subscriptions[rc.subId].balance -= payment;
33    s_withdrawableTokens[s_provingKeys[keyHash]] += payment;
34    // Include payment in the event for tracking costs.
35    emit RandomWordsFulfilled(requestId, randomness, payment,
  success);
36    return payment;
37  }
```

DESCRIPTION	The owner of Chainlink's coordinator is a multisig. This multisig has the permissions to cancel a subscription. If a subscription is canceled before the fulfillment of a

	request, that request will remain unfulfilled because there is no subscription to pay for it. Currently, SeederV2 has no way to create a new Random Word Request for an unfulfilled *requestID*. Chainlink's multisig consists of 19 signers at the time of writing and needs 3 out of 19 signers to pass. Multisig 3 out of 19 owners: 0x0A6c942fA19D83Eb989a9B3CD4A56D682Fff6561 0x19a847AEc97bB3D233D2a1BE429D53e097079Cf6 0x1A068cf4A717deF9735FE1c867f8867302421697 0x23B8D60c73700E524f4d6944126dc690AA73d81A 0x2Da436C56DB0f2B9ac80d852F4133Ea1d6FFD2BE 0x326377a6B92eC69AcbbFe2De1eB1d7c9008E4C89 0x41eAdbc688797a02bfaBE48472995833489ce69D 0x4eAAfcbd16dB53725774f6d97Ec0d40B1A216519 0x7052cB84079905400ea52B635cAb6a275fDA8823 0x7bb13ee9CF27d913414846fC5D8f57f5E515d80A 0x9F3578bea0dc29A0004835df13350416b16eA321 0x9dc6E515207bab22081Ba202a88a6bf351d74772 0xA371265D4e422D3cb858F4D2EA9914E6379C0F51 0xCDf00F2194F166851E51ab45D1BeD160FAB02B5A 0xE062e7D123AC8dF480C56147f911144F55C10f88 0xbae5b1585dEd4d184620f80A858F6118025C3daF 0xdD9F01B1ec0055b48525c56b1ae73118238Cd7bf 0xf2d132f912B30A48a62F27F249c3D4C9B4eA9A23 0xffC5859e47Fa6Ca83456BC7c3Fe9D283A3d20e68
RECOMMENDATION	Ensure that contracts can remain functional even if a request is canceled.
RESOLUTION	**Project team comment:** `This is very unlikely to occur, and if it does we can manually seed the batch.`

Modulo Of Zero

FINDING ID	#0005
SEVERITY	Low Risk
STATUS	Partially Closed
LOCATION	core/Raid.sol -> 251-277

```
1   function _rollRound(uint256 _seed, uint32 startBlock)
2         internal
3         view
4         returns (Round memory round)
5     {
6         // FIXME: check if we will overflow
7         unchecked {
8             uint32 roll = uint32(_seed % weightTotal);
9             uint256 weight = 0;
10            uint32 _bossWeight;
11
12            for (uint16 bossId; bossId < bosses.length; bossId++) {
13                _bossWeight = bosses[bossId].weight;
14
15                if (roll <= weight + _bossWeight) {
16                    round.boss = bossId;
17                    round.roll = roll;
18                    round.startBlock = startBlock;
19                    round.finalBlock = startBlock +
    bosses[bossId].blockHealth;
20
21                    return round;
22                }
23
24                weight += _bossWeight;
25            }
26        }
27    }
```

DESCRIPTION	If the *weightTotal* variable is zero then the modulo calculation will panic and revert. This is because it is not possible to disable the modulo by zero check using the unchecked block. This could be used as a malicious pause instead of pausing the contract.
RECOMMENDATION	Make sure that *weightTotal* is always more than 0 such that it does not revert.
RESOLUTION	Even after the latest patches, it's still possible to replace all the bosses by using the *createBosses()* function. That way

	they have a total weight of 0 (after a *start()* has occurred before).

Variables Could Overflow/Underflow In Certain Conditions

FINDING ID	#0006
SEVERITY	Low Risk
STATUS	Closed
LOCATION	core/Raid.sol

DESCRIPTION	Currently, there are many unchecked statements that technically could overflow in specific conditions. If this occurs, the calculations will overflow or underflow depending on the variable. However, if the contract doesn't use unchecked it will freeze the function. Note: While there are no checks in place to prevent it from happening, this depends on the project team setting the correct values.
RECOMMENDATION	Prevent underflow or overflow with the help of max setting on variables. Ensure that calculations from other contracts are taken into account.
RESOLUTION	Project team has gone through thorough testing of their variables and there are no signs of overflow/underflow risks.

Claiming Rewards Can Be Halted

FINDING ID	#0007
SEVERITY	Low Risk
STATUS	Open
LOCATION	core/Raid.sol -> 227-240

```
1     function claimRewards(address user) external notHalted {
2         Raider storage raider = raiders[user];
3
4         (uint32 _roundId, uint256 rewards) = _fetchRewards(raider);
5
6         raider.startRound = _roundId;
7         raider.pendingRewards = 0;
8         raider.startBlock = uint32(block.number);
9         raider.startSnapshot = uint32(snapshots.length + 1);
10
11        if (rewards > 0) {
12            confetti.mint(user, rewards);
13        }
14    }
```

DESCRIPTION	Claiming rewards can be halted by the admin. Note: Rewards will be accumulated while the contract is halted and they will be claimable once it's un-halted.
RECOMMENDATION	Add a timelock to allow users to react to changes.
RESOLUTION	**Project team comment:** "This is in place to prevent attacks on the Raid contract spiraling out of control. We may in the future add a way to withdraw assets without being able to claim rewards while halted."

Execute Request Can Be Rendered Unusable

FINDING ID	#0008
SEVERITY	Low Risk
STATUS	Mitigated
LOCATION	randomness/SeederV2.sol -> 304-344

```
    function executeRequest(address origin, uint256 identifier)
        external
        payable
    {
        ........

        require(
            msg.value == _fee,
            "Seeder::executeRequest: Transaction value does not match expected fee"
        );

        uint256 linkReqID = _coordinator.requestRandomWords(
            _keyHash,
            _subscriptionId,
            REQUEST_CONFIRMATIONS,
            CALLBACK_GAS_LIMIT,
            NUM_WORDS
        );

        _requestStorage.updateRequest(origin, identifier, bytes32(linkReqID));
    }
```

LOCATION	randomness/Seedable.sol -> 150-152

```
    function setFee(uint256 fee) external onlyRole(DEFAULT_ADMIN_ROLE) {
        _fee = fee;
    }
```

LOCATION	randomness/Seedable.sol -> 357-362

```
1    function setSubscriptionId(uint64 subscriptionId)
2        external
3        onlyRole(DEFAULT_ADMIN_ROLE)
4    {
5        _subscriptionId = subscriptionId;
6    }
```

DESCRIPTION	Currently, the Chainlink subscription is managed by the owner, and a user can pay with Eth. If the owner sets a high fee or ETH price goes up, the user won't be able to use it for a reasonable fee. This occurs if the subscription has no Link anymore, and the user wants to execute their request by manually adding tokens to the subscription.
RECOMMENDATION	Introduce another subscription with a separate request function that sends the maximum amount of LINK necessary to execute the request. This needs to be another subscription as Chainlink doesn't check that there is enough Link to get executed until the fulfillment. Note that this is not a guaranteed solution, because the price of LNK or ETH price could change drastically between the request and fulfillment. To make this unlikely use a sufficient offset, with a price fetched from Chainlink's LNK/ETH oracle.
RESOLUTION	The project team acknowledged it, and they will manually seed the batch as in issue #4, while working on a fix.

Code Readability

FINDING ID	#0009
SEVERITY	Informational
STATUS	Closed
LOCATION	core\Raid.sol -> 200-225

```
1     function updateDamage(address user, uint32 _dpb)
2         external
3         notHalted
4         raidActive
5         partyCaller
6     {
7         Raider storage raider = raiders[user];
8         if (raider.startedAt == 0) {
9             raider.dpb = _dpb;
10            raider.startedAt = uint32(block.number);
11            raider.startBlock = uint32(block.number);
12            raider.startRound =
  _lazyFetchRoundId(uint32(block.number));
13            raider.startSnapshot = uint32(snapshots.length + 1);
14
15            return;
16        }
17
18        (uint32 _roundId, uint256 rewards) = _fetchRewards(raider);
19
20        raider.startRound = _roundId;
21        raider.pendingRewards = rewards;
22        raider.dpb = _dpb;
23        raider.startBlock = uint32(block.number);
24        raider.startSnapshot = uint32(snapshots.length + 1);
25    }
```

LOCATION	core\Raid.sol -> 227-240

```
    function claimRewards(address user) external notHalted {
        Raider storage raider = raiders[user];

        (uint32 _roundId, uint256 rewards) = _fetchRewards(raider);

        raider.startRound = _roundId;
        raider.pendingRewards = 0;
        raider.startBlock = uint32(block.number);
        raider.startSnapshot = uint32(snapshots.length + 1);

        if (rewards > 0) {
            confetti.mint(user, rewards);
        }
    }
```

LOCATION	core\Raid.sol -> 323-345

```
    function _fetchRewards(Raider memory raider)
        internal
        view
        returns (uint32, uint256)
    {
        if (raider.dpb > 0) {
            if (snapshots.length > raider.startSnapshot) {
                (
                    uint32 _roundId,
                    uint256 rewards
                ) = _fetchNewRewardsWithSnapshot(raider);
                rewards += raider.pendingRewards;
                return (_roundId, rewards);
            } else {
                (uint32 _roundId, uint256 rewards) =
_fetchNewRewards(raider);
                rewards += raider.pendingRewards;
                return (_roundId, rewards);
            }
        }

        return (_lazyFetchRoundId(uint32(block.number)),
raider.pendingRewards);
    }
```

DESCRIPTION	Currently, new rewards are fetched and incremented by *pendingRewards*, which then overwrites its value. This results in being excessively complicated.

RECOMMENDATION	Instead of updating *pendingRewards* by overwriting with a local variable, add the rewards to *pendingRewards* for a cleaner and safer code path. This would both reduce code complexity and also optimize the code.
RESOLUTION	The project team implemented the recommended fix.

Unequipping Can Be Halted

FINDING ID	#0010
SEVERITY	Informational
STATUS	Open
LOCATION	core/Party.sol -> 227-266

```
1     function act(
2         Action[] calldata heroActions,
3         Action[] calldata fighterActions
4     ) external override whenNotUpdating {
5         require(heroActions.length <= 1, "Party::act: too many hero
  actions");
6
7         // Update party version if necessary
8         _update(msg.sender);
9
10        uint256[] memory heroesEquipped;
11        uint256[] memory heroesUnequipped;
12        uint256[] memory fightersEquipped;
13        uint256[] memory fightersUnequipped;
14
15        if (heroActions.length > 0) {
16            (heroesEquipped, heroesUnequipped) = _act(
17                Property.HERO,
18                heroActions
19            );
20        }
21
22        if (fighterActions.length > 0) {
23            (fightersEquipped, fightersUnequipped) = _act(
24                Property.FIGHTER,
25                fighterActions
26            );
27        }
28
29        if (heroesEquipped.length > 0) {
30            _validateParty(msg.sender);
31        }
32
33        Damage.DamageComponent[] memory curr = _damageCalculator
34            .getDamageComponents(heroesEquipped, fightersEquipped);
35
36        Damage.DamageComponent[] memory prev = _damageCalculator
37            .getDamageComponents(heroesUnequipped,
  fightersUnequipped);
38
39        _updateDamage(msg.sender, prev, curr);
40    }
```

DESCRIPTION	Unequipping a Hero or Fighter is not possible when the *Party* contract is being updated or the *Raid* contract is halted. Updating the *Party* and halting the *Raid* contracts is possible only by the admin (i.e. the RaidParty team). Note: Once the upgrade/halt is finished Heroes and Fighters can be unequipped normally.
RECOMMENDATION	Confirm that this is the intended behavior.
RESOLUTION	Same as issue #7. **Project team comment:** "This is in place to prevent attacks on the Raid contract spiraling out of control. We may in the future add a way to withdraw assets without being able to claim rewards while halted."

Callback To User

FINDING ID	#0011
SEVERITY	Informational
STATUS	Mitigated
LOCATION	utils/Enhanceable.sol -> 80-114

```
    function _checkOnEnhancement(uint256[] memory tokenIds, uint8[] memory prev)
        internal
        returns (bool)
    {
        require(
            tokenIds.length == prev.length,
            "Enhanceable: update array length mismatch"
        );
        address owner = _token.ownerOf(tokenIds[0]);

        for (uint256 i = 0; i < tokenIds.length; i++) {
            require(
                _token.ownerOf(tokenIds[i]) == owner,
                "Enhanceable: tokens not owned by same owner"
            );
        }

        if (owner.isContract()) {
            try IEnhancer(owner).onEnhancement(tokenIds, prev) returns (
                bytes4 retval
            ) {
                return retval == IEnhancer.onEnhancement.selector;
            } catch (bytes memory reason) {
                if (reason.length == 0) {
                    revert("Enhanceable: transfer to non Enhancer implementer");
                } else {
                    assembly {
                        revert(add(32, reason), mload(reason))
                    }
                }
            }
        } else {
            return true;
        }
    }
```

LOCATION	equipment/Equipment.sol -> 15

```
1 import "@openzeppelin/contracts/token/ERC1155/ERC1155.sol";
```

DESCRIPTION	The contracts using *Enhanceable.sol* have a callback to the owner of the token. This could cause reentrancy issues if it is not properly protected against contracts using those tokens. Similarly, *Equipment.sol* uses the ERC1155 which has a callback to the receiver on mint functions and safeTransfer functions.
RECOMMENDATION	Be careful as it could involve reentrancy in certain cases if the code that calls it is not reentrancy safe. Note that *_safeMint* also has a callback.
RESOLUTION	**Project team comment:** “There was previously a re-entrancy vulnerability here but this has been fixed for some time”

Incorrect Error Message

FINDING ID	#0012
SEVERITY	Informational
STATUS	Closed
LOCATION	randomness/SeederV2.sol -> 316-319

```
1   require(
2       _lastBatchTimestamp + _batchCadence > block.timestamp,
3       "Seeder::executeRequest: Cannot seed individually during batch
  seeding"
4   );
```

DESCRIPTION	The require message is incorrect as it's only callable during the batch period. Currently, the message says “Seeder::executeRequest: Cannot seed individually during batch seeding”.
RECOMMENDATION	Correct the require message.
RESOLUTION	The project team implemented the recommended fix.

Centralised Metadata Storage

FINDING ID	#0013
SEVERITY	Informational
STATUS	Open
LOCATION	equipment/EquipmentURIHandler.sol -> 70-72

```
1    function _baseURI() internal pure returns (string memory) {
2        return "https://api.raid.party/metadata/equipment/";
3    }
```

LOCATION	fighter/FigtherURIHandler.sol -> 186-188

```
1    function _baseURI() internal pure returns (string memory) {
2        return "https://api.raid.party/metadata/fighter/";
3    }
```

LOCATION	hero/HeroURIHandler.sol -> 232-234

```
1    function _baseURI() internal pure returns (string memory) {
2        return "https://api.raid.party/metadata/hero/";
3    }
```

DESCRIPTION	The *_baseURI()* of these contracts use a domain name. If the domain owner stops paying for the domain then this metadata storage could be lost. Note: This might not matter as much for a game. The metadata will exist as long as one person hosts it.
RECOMMENDATION	Consider using a service similar to IPFS. IPNS can also be used if upgradability is a concern, as the mutability is centralized but the hosting isn't.
RESOLUTION	**Project team comment:** “Metadata must be mutable for now.” **Obelisk comment:** “IPNS could still be used. IPNS is

	mutable but doesn't rely on a centralized server (if raid.party goes down, anyone else can host the data for the IPNS link).”

Enhanceable Function Should Be Separate

FINDING ID	#0014
SEVERITY	Informational
STATUS	Closed
LOCATION	utils/Enhanceable.sol -> 72-79

```
1     function reveal(uint256[] calldata ids) public virtual {
2       unchecked {
3           for (uint256 i = 0; i < ids.length; i++) {
4               delete _enhancements[ids[i]];
5           }
6       }
7     }
```

DESCRIPTION	The *reveal()* function of the *Enhanceable* contract is *public* while it is intended to be used internally. This is not an issue currently, as it's overloaded in the derived contract. However, in future implementations, it should be a separate internal function, so that elements of *_enhancements* map are not accidentally deleted.
RECOMMENDATION	Change *reveal()* to an unimplemented function and introduces a separate helper function that can be called internally to delete the mapping.
RESOLUTION	The project team implemented the recommended fix.

Unused Contract

FINDING ID	#0015
SEVERITY	Informational
STATUS	Closed
LOCATION	randomness/Seedable.sol -> 14-18

```
abstract contract Seedable {
    function _validateSeed(uint256 id) internal pure {
        require(id != 0, "Seedable: not seeded");
    }
}
```

DESCRIPTION	The noted contract is not used.
RECOMMENDATION	Remove the contract if it is not supposed to be used later on.
RESOLUTION	The project team implemented the recommended fix.

Redundant Boolean Statement

FINDING ID	#0016
SEVERITY	Informational
STATUS	Open
LOCATION	core/Summon.sol -> 88-122

```
1    function mintHero(uint256 proof, uint256[] calldata burnIds)
2        external
3        whenNotPaused
4    {
5        ...
6        if (proof < 1111 && proof > 0) {
7        ...
8    }
```

DESCRIPTION	The variable *proof* is a *uint256* which is always positive, checking for *proof* > *0* may be unnecessary.
RECOMMENDATION	Use the != operator if checking for non-zero values, otherwise remove the second part of the boolean.
RESOLUTION	The project team implemented the recommended fix. However, the deployed code does not include the fix.

Make Use Of Protocol Functions

FINDING ID	#0017
SEVERITY	Informational
STATUS	Open
LOCATION	core/Summon.sol -> 88-122

```
    function mintHero(uint256 proof, uint256[] calldata burnIds)
        external
        whenNotPaused
    {
        // ...
        for (uint256 i = 0; i < burnIds.length; i++) {
            _fighter.burn(burnIds[i]);
        }
        // ...
    }
```

LOCATION	fighter/Fighter.sol -> 116-127

```
    function burnBatch(uint256[] calldata tokenIds) external {
        unchecked {
            _burnCounter += tokenIds.length;
            for (uint256 i = 0; i < tokenIds.length; i++) {
                require(
                    _isApprovedOrOwner(_msgSender(), tokenIds[i]),
                    "Fighter::burnBatch: caller is not owner nor
approved"
                );
                _burn(tokenIds[i]);
            }
        }
    }
```

DESCRIPTION	Instead of using a for loop inside *mintHero()* function to burn multiple fighters, make use of the already existing function *burnBatch()*.
RECOMMENDATION	Replace the for loop with a call to *burnBatch()* of *Fighter.sol*.
RESOLUTION	No changes made

Unnecessary Return Value

FINDING ID	#0018
SEVERITY	Informational
STATUS	Open
LOCATION	core/DamageCalculator.sol -> 52-55

```
        uint256 idx;

        idx = _getHeroComponents(components, heroIds, idx);
        idx = _getFighterComponents(components, fighterIds, idx);
```

LOCATION	core/DamageCalculator.sol -> 205-229

```
function _getHeroComponents(
    Damage.DamageComponent[] memory components,
    uint256[] memory heroes,
    uint256 idx
) internal view returns (uint256) {
    for (uint256 i = 0; i < heroes.length; i++) {
        components[idx] = _getHeroDamageComponent(heroes[i]);
        idx += 1;
    }

    return idx;
}

function _getFighterComponents(
    Damage.DamageComponent[] memory components,
    uint256[] memory fighters,
    uint256 idx
) internal view returns (uint256) {
    for (uint256 i = 0; i < fighters.length; i++) {
        components[idx] = _getFighterDamageComponent(fighters[i]);
        idx += 1;
    }

    return idx;
}
```

DESCRIPTION	The return value of *_getHeroDamageComponent()* and *_getFighterDamageComponent()* is unnecessary. These functions are used to fill an array with known numbers of components.

RECOMMENDATION	Replace the values of *idx* passed into *_getHeroDamageComponent()* and *_getFighterDamageComponent()* with the values *0* and *heroIds.length respectively*. At the same time, remove the return values of the noted functions. Note: Since there are only two sources of damage components, an optimization could be to not pass in an index at all and instead fill the array from both ends.
RESOLUTION	**Project team comment:** "There will likely be a more major refactor coming in the future but this does not warrant a fix for now."

No Events Emitted For Changes To Protocol Values

FINDING ID	#0019
SEVERITY	Informational
STATUS	Partially Closed
LOCATION	• core/Party.sol -> 92-97: *function setDamageCalculator(IDamageCalculator calculator) external onlyRole(DEFAULT_ADMIN_ROLE) {* • core/Party.sol -> 103-105: *function setRaid(IRaid raid) external onlyRole(DEFAULT_ADMIN_ROLE) {* • core/Party.sol -> 149-151: *function setVersion(uint256 version) external onlyRole(DEFAULT_ADMIN_ROLE) {* • core/Party.sol -> 157-159: *function setUpdating(bool updating) external onlyRole(DEFAULT_ADMIN_ROLE) {* • core/Raid.sol -> 84-86: *function setHalted(bool _halted) external onlyRole(DEFAULT_ADMIN_ROLE) {* • randomness/SeederV2.sol -> 150-152: *function setFee(uint256 fee) external onlyRole(DEFAULT_ADMIN_ROLE) {*

DESCRIPTION	Functions that change protocol variables should emit events such that the changes can be easily monitored.
RECOMMENDATION	Emit events from these functions.
RESOLUTION	Event is now emitted in the *setHalted()* function. The *setVersion()* function has been removed.

Deterministic Seed

FINDING ID	#0020
SEVERITY	Informational
STATUS	Open
LOCATION	core/Raid.sol -> 244-249

```
1     function _rollSeed(uint256 oldSeed) internal pure returns (uint256 rolled) {
2         assembly {
3             mstore(0x00, oldSeed)
4             rolled := keccak256(0x00, 0x20)
5         }
6     }
```

DESCRIPTION	Between Chainlink seed updates, the seed is deterministic. This can be used to gain an advantage for example by buying and selling teams depending on the deterministic seed.
RECOMMENDATION	Switch out the seed with Chainlink often to reduce the risk of this being abused. Make sure users understand that the seed will not always be random, alternatively mitigate it by only giving rewards to users that deposited before the last Chainlink seed.
RESOLUTION	**Project team comment:** “This is by design, our model allows people to predict future bosses.”

Complex Damage Calculation

FINDING ID	#0021
SEVERITY	Informational
STATUS	Open
LOCATION	core/Party.sol -> 227-266

```
function act(
    Action[] calldata heroActions,
    Action[] calldata fighterActions
) external override whenNotUpdating {
    require(heroActions.length <= 1, "Party::act: too many hero actions");

    // Update party version if necessary
    _update(msg.sender);

    uint256[] memory heroesEquipped;
    uint256[] memory heroesUnequipped;
    uint256[] memory fightersEquipped;
    uint256[] memory fightersUnequipped;

    if (heroActions.length > 0) {
        (heroesEquipped, heroesUnequipped) = _act(
            Property.HERO,
            heroActions
        );
    }

    if (fighterActions.length > 0) {
        (fightersEquipped, fightersUnequipped) = _act(
            Property.FIGHTER,
            fighterActions
        );
    }

    if (heroesEquipped.length > 0) {
        _validateParty(msg.sender);
    }

    Damage.DamageComponent[] memory curr = _damageCalculator
        .getDamageComponents(heroesEquipped, fightersEquipped);

    Damage.DamageComponent[] memory prev = _damageCalculator
        .getDamageComponents(heroesUnequipped, fightersUnequipped);

    _updateDamage(msg.sender, prev, curr);
}
```

LOCATION	core/Party.sol -> 542-583

```
function _act(Property item, Action[] calldata actions)
    internal
    returns (uint256[] memory, uint256[] memory)
{
    uint256[] memory equipped = new uint256[](actions.length);
    uint256[] memory unequipped = new uint256[](actions.length);

    uint256 u = 0;

    (uint256 equipCounter, uint256 unequipCounter) = (0, 0);

    // Perform actions
    for (uint256 i = 0; i < actions.length; i++) {
        if (actions[i].action == ActionType.EQUIP) {
            u = _equip(item, actions[i].id, actions[i].slot);

            equipped[equipCounter] = actions[i].id;
            equipCounter += 1;
        } else {
            u = _unequip(item, actions[i].slot);
        }

        if (u != 0) {
            unequipped[unequipCounter] = u;
            unequipCounter += 1;
        }
    }

    // Reset counters and resize arrays
    assembly {
        mstore(
            equipped,
            sub(mload(equipped), sub(actions.length, equipCounter))
        )
        mstore(
            unequipped,
            sub(mload(unequipped), sub(actions.length, unequipCounter))
        )
    }

    return (equipped, unequipped);
}
```

DESCRIPTION	The *act()* function creates 4 arrays: • equipped fighters • equipped heroes • unequipped fighters • unequipped heroes The function then loops through these arrays to create an array of the *prev* damage and an array of the *curr* damage components. The *prev* and *curr* arrays are then looped over and components are then respectively removed and added to the party damage.

	The operation creates local copies of storage structures that can consume a significant amount of gas.
RECOMMENDATION	By fetching the current *DamageComponent* of the user in memory, and passing it down to *_equip*, *_unequip* the changes in the damage can be stored temporarily at the level where each component gets equipped/unequipped and update the storage once after the actions. That way the logic will become simpler and will be storing fewer arrays in memory.
RESOLUTION	No changes made.

On-Chain Analysis

No findings

External Addresses

Externally Owned Accounts

Treasury Signers

ACCOUNT	0x5eB583B39A636bD3Fb5B50e50AFE085309F6E90E 0xE5E4b1ea7cC9e9E8529Cebf0C51DB30748584785 0x50A38a390265a2bc4B78032905226d524d847A42
USAGE	0xcF2D2dA4c2F9B0675A197FEbC6708704834f9c24 *GnosisSafe.getOwners* - Constant
IMPACT	• receives elevated permissions as owner, operator, or other

External Contracts

These contracts are not part of the audit scope.

Treasury - Gnosis Safe

ADDRESS	0xcF2D2dA4c2F9B0675A197FEbC6708704834f9c24
USAGE	0x87E738a3d5E5345d6212D8982205A564289e6324 *Fighter.DEFAULT_ADMIN_ROLE* - Role *Fighter.MINTER_ROLE* - Role 0x5cfEf6B6777aD143e6Edc122632984C87dC6FB40 *FighterURIHandler.admin* - Role 0x966731dFD9b9925DD105FF465687F5aA8f54Ee9f *Hero.DEFAULT_ADMIN_ROLE* - Role *Hero.MINTER_ROLE* - Role 0x8271Ca0DeA5d7c5B3B63A54903383EdaBcb58AE8 *HeroURIHandler.DEFAULT_ADMIN_ROLE* - Role 0x2Ed251752DA7F24F33CFbd38438748BB8eeb44e1 *SeederV2.DEFAULT_ADMIN_ROLE* - Role *SeederV2.INTERNAL_CALLER_ROLE* - Role 0x67283EE31eA17Bb03D958c0386155e6665DE5fbf *Summon.DEFAULT_ADMIN_ROLE* - Role *Summon.getTeam* - Role
IMPACT	• receives transfer of tokens deposited by users • has elevated permissions as owner, operator, or other

Sale Contract

ADDRESS	0x0B5554B0DB95Ad833a36f6802e1e74f1490d810d
USAGE	0x87E738a3d5E5345d6212D8982205A564289e6324 *Fighter.MINTER_ROLE* - Role 0x966731dFD9b9925DD105FF465687F5aA8f54Ee9f *Hero.MINTER_ROLE* - Role

IMPACT	• has elevated permissions as owner, operator, or other

Fighter Claim

ADDRESS	0xfeCE719a38215c6fE07e8979E859955380B965e2
USAGE	0x87E738a3d5E5345d6212D8982205A564289e6324 *Fighter.MINTER_ROLE* - Role
IMPACT	• has elevated permissions as owner, operator, or other

External Tokens

These contracts are not part of the audit scope.

No external tokens

Appendix A - Reviewed Documents

Deployed Contracts

Document	Address
core/DamageCalculator.sol	0x792c933cbA0edd6e8ded3d23d13Cf007E2518878
core/Party.sol	Proxy 0xd311bDACB151b72BddFEE9cBdC414Af22a5E38dc Implementation 0x83d0C1981Be3f7ff6bAc0FB0E766fF7F1AC57fCC
core/Raid.sol	Proxy 0xFa209a705a4DA0A240AA355c889ed0995154D7Eb Implementation 0xfDDa6307192CE1571C25C9036C530a7fF4FB1Bb8
core/Summon.sol	0x67283EE31eA17Bb03D958c0386155e6665DE5fbf
equipment/Equipment.sol	N/A
equipment/EquipmentURIHandler.sol	N/A
fighter/Fighter.sol	0x87E738a3d5E5345d6212D8982205A564289e6324
fighter/FighterURIHandler.sol	Proxy 0x5cfEf6B6777aD143e6Edc122632984C87dC6FB40 Implementation 0x53440476fc79bfF3FF8e6Ac5c964AEecC7Cf2617
hero/Hero.sol	0x966731dFD9b9925DD105FF465687F5aA8f54Ee9f
hero/HeroURIHandler.sol	Proxy 0x8271Ca0DeA5d7c5B3B63A54903383EdaBcb58AE8 Implementation 0x977af44Cb53a855BEaF57E94d06A72e361e82d71
randomness/SeederV2.sol	0x2Ed251752DA7F24F33CFbd38438748BB8eeb44e1
token/Confetti.sol	0xCfef8857E9C80e3440A823971420F7Fa5F62f020

Libraries and Interfaces

interfaces/IConfetti.sol interfaces/IDamageCalculator.sol

interfaces/IEnhanceable.sol interfaces/IEnhancer.sol interfaces/IEquipment.sol interfaces/IEquipmentURIHandler.sol interfaces/IERC20Burnable.sol interfaces/IFighter.sol interfaces/IFighterURIHandler.sol interfaces/IHero.sol interfaces/IHeroURIHandler.sol interfaces/IOldHero.sol interfaces/IParty.sol interfaces/IRaid.sol interfaces/IRaidERC721.sol interfaces/ISeeder.sol interfaces/IValidator.sol lib/Damage.sol lib/Randomness.sol lib/Stats.sol randomness/RequestStorage.sol randomness/Seedable.sol randomness/Seeder.sol randomness/SeedStorage.sol utils/Enhanceable.sol utils/Enhancer.sol utils/ERC721.sol utils/ERC721Enumerable.sol

Revisions

Revision 1	054a3a46609e433a106c5fcc2872a37b1051094c
Revision 2	5692b1e416c3e0fc13f2cb9dfef66d2c3d22fb1b

Imported Contracts

Chainlink	0.8
OpenZeppelin	4.4.2

Appendix B - Risk Ratings

Risk	Description
High Risk	A fatal vulnerability that can cause the loss of all Tokens / Funds.
Medium Risk	A vulnerability that can cause the loss of some Tokens / Funds.
Low Risk	A vulnerability which can cause the loss of protocol functionality.
Informational	Non-security issues such as functionality, style, and convention.

Appendix C - Finding Statuses

Closed	Contracts were modified to permanently resolve the finding.
Mitigated	The finding was resolved on-chain. The issue may require monitoring, for example in the case of a time lock.
Partially Closed	Contracts were modified to partially fix the issue
Partially Mitigated	The finding was resolved by project specific methods which cannot be verified on chain. Examples include compounding at a given frequency, or the use of a multisig wallet.
Open	The finding was not addressed.

Appendix D - Audit Procedure

A typical Obelisk audit uses a combination of the three following methods:

Manual analysis consists of a direct inspection of the contracts to identify any security issues. Obelisk auditors use their experience in software development to spot vulnerabilities. Their familiarity with common contracts allows them to identify a wide range of issues in both forked contracts as well as original code.

Static analysis is software analysis of the contracts. Such analysis is called “static” as it examines the code outside of a runtime environment. Static analysis is a powerful tool used by auditors to identify subtle issues and to verify the results of manual analysis.

On-chain analysis is the audit of the contracts as they are deployed on the block-chain. This procedure verifies that:

- deployed contracts match those which were audited in manual/static analysis;
- contract values are set to reasonable values;
- contracts are connected so that interdependent contract function correctly;
- and the ability to modify contract values is restricted via a timelock or DAO mechanism. (We recommend a timelock value of at least 72 hours)

Each obelisk audit is performed by at least two independent auditors who perform their analysis separately.

After the analysis is complete, the auditors will make recommendations for each issue based on best practice and industry standards. The project team can then resolve the issues, and the auditors will verify that the issues have been resolved with no new issues introduced.

Our auditing method lays a particular focus on the following important concepts:

- Quality code and the use of best practices, industry standards, and thoroughly tested libraries.
- Testing the contract from different angles to ensure that it works under a multitude of circumstances.
- Referencing the contracts through databases of common security flaws.

Follow Obelisk Auditing for the Latest Information

 ObeliskOrg

 ObeliskOrg

OBELISK

Part of Tibereum Group