Fuel Games

Gods Unchained

Solidity Security Review

Version: 1.0

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the smart contracts which make up the *Pack* functionality for *Gods Unchained*. The review focused solely on the security considerations regarding the Solidity implementation of the contracts but also includes general recommendations and informational comments relating to minimisation of gas usage, token functionality, and code quality.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review regarding, the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the pack functionality that falls within the scope of the security review. The discovered vulnerabilities and issues are summarised, followed by a detailed breakdown of each. Each vulnerability is assigned a severity rating (see Vulnerability Severity Classification), an open/closed status, and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as "informational". Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities.

Overview

The Pack functionality of the Gods Unchained project allow users to purchase "bundles", which are ERC20 tokens, using ether. A bundle is a collection of "packs", and is described by the `Bundle.sol` smart contract. Each pack contains 5 cards, the functionality for which exists in the `PackFive.sol` smart contract.

Since a pack is a collection of cards, it relies on a randomness component to determine the properties of these cards. The randomness (entropy) is obtained from a blockhash and is distinct for each card during activation.

When performing payments (i.e. when purchasing bundles or packs) the `processPayment()` function calculates the required amount of funds for a purchase and transfers them to the participating parties. This function and associated logic is contained within the `Processor` contract. The `Referrals` contract is responsible for specifying what percentage of funds goes to the user and the referrer, and how much of a discount is applied during the purchase. All payments for cards ultimately rely on the `CappedVault.sol` contract, which puts a hard limit on the total amount of ether that can be used to purchase cards.

Audit Summary

This review was conducted on the commit 344566e8, which contains the full set of Solidity smart contracts for Gods Unchained. The complete list of smart contracts which are *in scope* for Gods Unchained is as follows:

```
── Packs
   ├── Bundle.sol
   ├── IPack.sol
   ├── IProcessor.sol
   ├── IReferrals.sol
   ├── PackFive.sol
   ├── Pack.sol
   ├── Processor.sol
   ├── RarityProvider.sol
   └── Referrals.sol
```

The following two contracts were out of scope, but were also examined as part of the review.

```
├── Vaults
│   └── CappedVault.sol
└── Zeppelin
    └── ERC20Burnable.sol
```

This security assessment focused on the following contracts:

- `Bundle`
- `PackFive`
- `Processor`
- `RarityProvider`
- `Referrals`

The manual code-review section of this report, focused on identifying any and all issues/vulnerabilities associated with the business logic implementation of the contracts. Specifically, their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout). Additionally, the manual review process focuses on all known Solidity anti-patterns and attack vectors. These include, but are not limited to, the following vectors: re-entrancy, front-running, integer overflow/underflow, and correct visibility specifiers. For a more thorough, but non-exhaustive list of examined vectors, see [1, 2].

To support this review, the testing team used the following automated testing tools:

- Rattle: `https://github.com/trailofbits/rattle`
- Mythril: `https://github.com/ConsenSys/mythril`
- Slither: `https://github.com/trailofbits/slither`
- Surya: `https://github.com/ConsenSys/surya`

Output for these automated tools is available upon request.

Second round of review

Issues raised in this review were addressed by three commits on the repository: 21dd6734, 4c0afad6, and 0fa13700.

The smart contracts in scope were reviewed a second time to ensure that the issues raised in the report have been adequately addressed. The 'status' component of each issue in this report reflects findings from this second round of review.

The tests developed during the first round of review have been updated to ensure they pass.

Per-Contract Vulnerability Summary

Bundle

Some informational notes are given.
No potential vulnerabilities have been identified.
All issues have been resolved or acknowledged.

PackFive

Three low severity issues have been identified, which may have security implications.
Several informational notes are provided, alongside some gas optimisations.
All issues have been resolved or acknowledged.

Processor

One high severity issue has been identified, alongside two low severity issues.
Further, some informational notes and gas optimisations are provided.
All issues have been resolved or acknowledged.

RarityProvider

One medium severity issue has been identified.
Some informational notes are also provided.
All issues have been resolved or acknowledged.

Referrals

Some informational notes and gas optimisations are provided.
All issues have been resolved or acknowledged.

Other Contracts

Some informational notes and gas optimisations are provided.
All issues have been resolved or acknowledged.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Gods Unchained smart contracts. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as “informational”.

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
GU-01	A Non-Zero Discount Prevents All Purchases Of Cards	High	Resolved
GU-02	Modulo Bias In Randomness	Medium	Closed
GU-03	Invalid Packs Can Be Created, And Cannot Be Updated	Low	Resolved
GU-04	`lockup` Variable Overflow Leading To Unexpected State	Low	Resolved
GU-05	Payment Change Returned To Incorrect User In `purchaseFor()`	Low	Resolved
GU-06	Potential For Manipulating Randomness	Low	Closed
GU-07	Unvalidated Input From External Contract	Low	Resolved
GU-08	ERC20 Incorrect Decimals	Informational	Resolved
GU-09	Unnecessary Transfer Events	Informational	Resolved
GU-10	Referrers Set The Discount Of Users	Informational	Closed
GU-11	`PackFive` Informational Issues	Informational	Resolved
GU-12	`Processor` Informational Issues	Informational	Resolved
GU-13	`RarityProvider` Informational Issues	Informational	Resolved
GU-14	`Referrals` Informational Issues	Informational	Resolved
GU-15	Other Informational Issues	Informational	Resolved

GU-01	A Non-Zero Discount Prevents All Purchases Of Cards		
Asset	Processor.sol		
Status	**Resolved:** In commit 21dd6734		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

The `getAllocations()` function on line [60] calculates the allocation percentage of the vault and referrer. On line [66] of the `Processor`, the `vaultPercentage` variable is calculated using both `discount` and `refer`. However, `discount` is not factored into the require statement on line [69]. As a result, configuring a `discount` value that is greater than zero will always cause a revert when the `getAllocations()` function is called.

This issue has been demonstrated in two test files:

- `test_processor_discount.py` line [23]
- `test_pack_five_purchase.py` line [107]

Recommendations

The `getAllocations` function should account for non-zero discounts. Depending on how the percentages are to be interpreted, the `total` in the require on line [69] could be set to the reduced total, i.e. `getPercentage(total, uint(100).sub(discount))`.

Resolution

The `getAllocations()` function has been modified to calculate `discountedTotal`, which considers the discounted amount without reverting.

GU-02	Modulo Bias In Randomness		
Asset	RarityProvider.sol		
Status	**Closed:** Acknowledged by author.		
Rating	Severity: Medium	Impact: Low	Likelihood: High

Description

An ideal randomness function has an equal likelihood of producing each possible outcome. Due to the modulo operations performed between lines [36] and line [39] in the `getComponents()` function, some values are more likely to occur than others. This is known as modulo bias [3].

Although there is bias in the quantities `rarity` , `quality` , and `purity` the bias is the most significant for the `quality` and `purity` randomness components. For these components, values between 0 and 536 each have a likelihood of 0.1007%, with all other numbers having a likelihood of 0.0992%. Hence, there is a bias towards numbers less than or equal to 535.

The `rarity` value also has a bias, but to a lesser extent, due to the larger range of numbers that a 32-bit unsigned integer can represent.

Recommendations

To remove the modulo bias, the random number, n, being modulo divided (by p for example) must be a multiple of the divisor (i.e. `n%p == 0`). This can be resolved in a number of ways.

One way would be to re-sample the random number if it lies outside the modulo range. For example, a 16-bit uint has a range [0, 65536). If we are taking a modular division of 1000, numbers above 65,000 contribute to modulo bias. In this case, if a number greater than 65,000 occurred, one could continually re-sample until a number less than 65,000 resulted.

In the context of this contract, one could consider re-hashing the input randomness repeatedly until the elements provide a number that is less than 65,000. This processes introduces a non-zero probability that no such number will occur, and so this should be capped after a number of iterations, at which point the bias may be accepted (this is to prevent a random number that cannot be calculated within the block gas limit).

Alternatively, one could replace the modulo of 1000 (line [37] and line [38]) with a modulo of 65,536, which will also remove the modulo bias.

Resolution

The bias towards lower numbers has been acknowledged by the authors, and this bias has been factored in to the parameters specified in the `RarityProvider` .

GU-03	Invalid Packs Can Be Created, And Cannot Be Updated		
Asset	PackFive.sol		
Status	**Resolved:** In commit 21dd6734		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

The `setPack()` function on line [85] of `PackFive.sol` allows the creation of packs which are invalid.

For example, a pack could be created with a `size` of zero or an invalid `token` address. Also, any pack that doesn't have a cost that is a multiple of 100 wei cannot be purchased due to the require on line [28] of `Processor.sol`.

In each of the above examples, the `setPack()` function will return normally, however subsequent calls to related functions (such as `purchaseFor()`) will revert.

An example of this erroneous functionality is provided in the test file `test_pack_update.py`, which accompanies this report.

Further, the security of card creation relies on the correct initialisation of `setPack()`. If a malicious token were to be configured in `setPack()`, it could call `openBundle()` (line [191]) and create cards for free, without `processPayment()`.

Recommendations

Correct validation should be performed inside the `setPack()` function. It should:

- Validate the token address
- Ensure the pack size is not zero
- Ensure the cost is a multiple of 100

To address the insecurities around setting arbitrary token addresses, the `setPack()` function could deploy a new instance of `Bundle` directly for each new pack, instead of it being passed in as a reference. This would prevent malicious external contracts from stealing/minting cards for free.

Resolution

Additional checks have been included to prevent invalid packs from being created. Further, the `setPack` function now deploys the bundle contract, rather than relying on external deployment.

GU-04	`lockup` Variable Overflow Leading To Unexpected State		
Asset	PackFive.sol		
Status	**Resolved:** In commit 21dd6734		
Rating	Severity: Low	Impact: Low	Likelihood: Medium

Description

Users with the correct permission can provide a `lockup` when calling `purchaseFor()` on line [130].

As this is an external variable it can be set to any value. User's can therefore set a `lockup` which overflows on line [335] making their purchase avoid the `inLockupPeriod`. This means the purchase cannot be revoked due to the require on line [119]. This also means that `canActivatePurchase()` returns `true` before the commit period. Activation, however, cannot proceed because of the require on line [163] where randomness is required to have been set. In this state, randomness cannot be set (i.e calling the `callback()` function) before the commit period due to the require on line [236].

Recommendations

Apply safe math logic to the `lockup` variable, specifically in `inLockupPeriod()` on line [335].

Resolution

A new `maxLockup` variable has been configured, which restricts values to a sensible range.

GU-05	Payment Change Returned To Incorrect User In `purchaseFor()`		
Asset	Processor.sol		
Status	**Resolved:** In commit 21dd6734		
Rating	Severity: Low	Impact: Low	Likelihood: Medium

Description

When purchasing bundles using the `purchaseFor()` function (`Bundle.sol` line [34]), the user receiving the bundle is specified as an argument. The payment is then processed by the `Processor` contract, and any value sent to the contract that is in excess of the cost of the bundle ("change") is returned to the user receiving the bundle, not the user making the payment.

This is also true for the `purchaseFor()` function in the `PackFive` contract.

This is demonstrated in the test: `test_pack_five_purchase_for_refund`

Recommendations

Modify the `processPayment()` function to return funds to the user who sends the transaction, instead of the user receiving the bundles. This could be done, for example, by adding an extra parameter (`changeUser`) which specifies which address the change should be sent to.

Resolution

Calls to the `processPayment()` function have been modified so that the `user` parameter is always `msg.sender` .

GU-06	Potential For Manipulating Randomness		
Asset	PackFive.sol		
Status	**Closed:** Acknowledged by authors.		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

The randomness (entropy) is obtained by committing to a future block hash when purchasing cards. This entropy is solidified into the contract state when `callback()` on line [227] is called (by anyone). This determines which cards the user will get from their purchase.

Users can predict the outcomes of their cards once the committed block hash is available regardless of whether the `callback()` function has been called. If the `callback()` function doesn't get called within 256 blocks of the committed block number, users can commit to another future block. If the cards are undesirable, the user must wait for only 256 blocks (approximately 1 hour), after which they can call `recommit`, providing them a new random number and potentially a more favourable outcome. Thus, it is in the user's best interest to not call `callback()` until an optimal result is obtained, allowing multiple attempts at cards.

We acknowledge that `callback()` can be called by anyone, and it is likely that the responsibility of calling this function is on the contract maintainers to ensure it is called in the first iteration.

This can be potentially vulnerable if an automated system is set up to call `callback()` which uses semi-static gas prices. A user may purchase cards in periods of high demand, such that the automated system's transactions get delayed due to lower gas prices.

Recommendations

It may be desirable to incentivise users to call `callback()` themselves rather than try to manipulate gas prices and/or hope the automated system's transactions fail, giving them extra chances at getting optimal cards.

One solution, could be to impose a fixed number of times `recommit()` can be called. Thus if `callback()` is not called after a particular number of re-commits, then the user loses their purchase. This adds incentive for users to also call `callback()`, and places a hard limit on the potential number of tries at obtaining optimal cards.

GU-07	Unvalidated Input From External Contract		
Asset	Processor.sol		
Status	**Resolved:** In commit 4c0afad6		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The `getAllocations()` function retrieves external values (`discount` , `refer`) from the `Referrals` contract on line [63] in `Processor.sol` .

These values are subsequently used without safe math and are assumed to be percentages. Since this contract is external, its return values should be validated to ensure they are sensible. Malicious return values can lead to underflows and undesirable states in this context.

Although this validation is done in the assumed external contract, `Referrals` on line [57], it is good practice to validate external inputs in-case of malicious/accidental deployment.

Recommendations

Validate `discount` and `refer` on input (to ensure they are ≤ 100).

Alternatively, consider deploying the `Referrals` contract in the constructor of `Processor` and/or hard-coding the referrals address to ensure the input functions are as expected.

Resolution

An additional `require` statement was included to validate the output from `getAllocations()` .

GU-08	ERC20 Incorrect Decimals
Asset	Bundle.sol
Status	**Resolved:** In commit 21dd6734
Rating	Informational

Description

This section details the compliance with the ERC20 Standard [4]. Non-compliance with the ERC20 standard does not pose any security risk, however may cause issues with third-party applications which expect the standard.

The `Bundle` contract uses a `decimals` value of `1`.

Where the stored token balance value is b, and the decimals value is d, the token balance displayed to the user (t_b) will be: $t_b = \frac{b}{10^d}$

Since the concept of an unopened bundle should be indivisible, d should be 0, so that $t_b = b$.

Single bundles will appear as 0.1 in standard GUI front-ends.

Recommendations

Modify the `decimals` value to be `0`.

Resolution

The decimals value has been updated.

GU-09	Unnecessary Transfer Events
Asset	Bundle.sol
Status	**Resolved:** in commit 21dd6734
Rating	Informational

Description

The `openFor` function on line [51] in `Bundle.sol` can be called with a `value` of zero.

While this correctly does not open any bundles, it will still emit `Transfer` events, which can poison the event log.

Recommendations

Insert a `require` statement which reverts if the `value` parameter is zero.

Resolution

A require has been added, which prevents zero bundles from being opened.

GU-10	Referrers Set The Discount Of Users
Asset	Processor.sol and Referrals.sol
Status	**Closed:** Acknowledged by author.
Rating	Informational

Description

When purchasing cards, users specify a referrer address. The referrer address can call `setSplit()` in `Referrals.sol` to specify how much of the discount is given to the referrer and how much is given to the user. When processing a payment, the `getSplit()` function is called on line [63] of `Processor.sol` which gets the referrers setting and applies it to the purchasing user.

It is thus in the user's best interest to set an address they already own in order to set and receive their own maximum discount.

Recommendations

To change the incentive for users, a whitelist could be constructed that allows specific referrers to call `setSplit()`. This set could then potentially offer higher discounts. If `getSplit()` is called for any address not in this list, the default percentage discount is given to the user without any funds being sent to the non-whitelisted arbitrary referrer.

GU-11	`PackFive` Informational Issues
Asset	PackFive.sol
Status	**Resolved:** See inline comments
Rating	Informational

Description

Several informational issues have been identified with `PackFive`. These are provided as notes and recommendations, fixes for which are not essential.

line [34]**:** The `current` variable in the `Purchase` struct does not appear to be used, and may be removed.
✓ *Resolved in commit [21dd6734]*

line [35]**:** The variables `commit` and `lockup` in the `Purchase` struct are of type `uint64`. The type of these can be changed to `uint128` without any gas penalty, since they can still be tightly packed in the struct[1]. This would provide more head-room for overflows without any disadvantage.
✓*Acknowledged, left as `uint64` to avoid introducing `SafeMath128`.*

line [49]**:** The `packs` mapping uses a `uint8` as the key, which is limited to 256 values. This can be changed to a `uint` (`uint256`) which will be cheaper (in terms of gas) as the EVM prefers 32 byte variables and must perform mask operations to handle lower byte variables.
✓ *Resolved in commit [0fa13700]*

line [81]**:** The function `getActivationLimit` is unnecessary. Because `activationLimit` is a public variable, getters for it are automatically provided by Solidity.
✓ *Resolved in commit [21dd6734]*

line [134]**:** The external call in the `purchaseFor` function should use the check-effects-interactions pattern for safety.
✓ *Resolved in commit [21dd6734]*

line [236]**:** If the admin sets a sufficiently high `commitLag` value, the `getCommitBlock` function (line [264]) could return the largest possible value of a `uint64`. This would cause an overflow in the require on line [236], which indefinitely prevents any callback from succeeding.
We note that an admin can also just set a sufficiently high `commitLag` such that no callbacks can be made in the realistic future.
✓ *Resolved in commit [21dd6734]*

line [268]**:** `getStateSize()` returns a larger number than necessary when `count` is a non-zero multiple of 256.
✓ *Resolved in commit [21dd6734]*

line [327]**:** `predictPacks()` may revert if the number of packs (i.e. `count`) purchased is greater than approximately 180, due to block gas limitations. Since the `count` variable is set externally, this has the potential to form denial-of-service vector.
We note that no internal function calls (from contracts in scope) are made, and this issue is not applicable for external applications calling this function.
✓*Acknowledged, this function is for external applications.*

[1] See the Solidity documentation for details on tightly packed variables.

line [245]: The randomness that determines which cards get chosen is based on a future block hash which is susceptible to miner manipulation. The authors have acknowledged this and may potentially move to another source of entropy in the future.
✓*Acknowledged.*

Recommendations

line [34]: Remove the unused variable. This would slightly improve gas efficiency.

line [35]: Change the variable types to `uint128`.

line [49]: Change the `packs` mapping key type to `uint`.

line [81]: Remove the `getActivationLimit` function and fix references.

line [134]: Refactor so that the external call occurs at the end of the function.

line [236]: Bounds should be placed on setting the `commitLag`, so that can only be configured within a sensible time frame.

line [268]: Change the function to `count.mul(5).sub(1).div(256).add(1);` to account for multiples of 256.

line [327]: Be more explicit about purchase limits.

GU-12	`Processor` Informational Issues
Asset	Processor.sol
Status	**Resolved:** See inline comments
Rating	Informational

Description

Several informational issues have been identified with the `Processor`. These are provided as notes and recommendations, fixes for which are not essential.

line [21]: Since the `processPayment` function is `public`, anyone can call this function. This allows erroneous money to be transferred into the `CappedVault`, which may affect its cap limit. This also enables logs to be spammed with `PaymentProcessed` events.
✓ *Resolved in commit [0fa13700]*

line [60]: The variables `cost` and `items` could be combined into a single variable (e.g. `amount`). This applies to both the `getAllocations()` and `getPrice()` functions on line [60] and line [74] respectively.
✓*Acknowledged, would like to be more clear about how values are calculated.*

line [93]: The `if` statement used here to check whether the amount or percentage is zero is unnecessary. If either of these values were to be zero, then the function will just return zero anyway.
✓ *Resolved in commit [21dd6734]*

Recommendations

line [21]: Consider allowing only specific contracts, such as `Bundle` and `PackFive`, to call this function.

line [60]: Combine the `cost` and `items` variables.

line [93]: Remove the `if` statement block.

GU-13	`RarityProvider` Informational Issues
Asset	RarityProvider.sol
Status	**Resolved:** See inline comments.
Rating	Informational

Description

Several informational issues have been identified with the `RarityProvider`. These are provided as notes and recommendations, fixes for which are not essential.

line [24]: The extract function does not behave as expected, it does not simply extract the expected bytes from the input `num`. It subtracts one bit from the start byte. Perhaps the second part should be `(start -1)*8)`. Please see the test `test_extract_bytes` in the file `test_rarity_provider.py` on line [119].
✓ *Resolved in commit [21dd6734]*

line [54]: The visibility of the `_getShinyCardDetails` is `internal`, which is inconsistent with the other `_get*CardDetails` functions that are `public`.
✓ *Resolved in commit [21dd6734]*

line [179]: The `randTwo` value that is passed to both the `_getPurity` and `_getShinyPurity` functions on line [179] and line [191] respectively, is unnecessary.
✓ *Resolved in commit [21dd6734]*

Recommendations

line [24]: Refactor the `extract` function, or update the comment to reflect its actual functionality or ensure the current functionality is as expected.

line [54]: Either make all the `_get*CardDetails` functions internal (which is suggested due to the leading underscore convention), or make `_getShinyCardDetails` public.

line [179]: Do not include `randTwo` as an argument to these functions, instead simply add the `rc.purity` value in the calling function.

GU-14	`Referrals` Informational Issues
Asset	Referrals.sol
Status	**Resolved:** See inline comments
Rating	Informational

Description

Several informational issues have been identified with `Referrals`. These are provided as notes and recommendations, fixes for which are not essential.

line [10]: `discountLimit`, `defaultDiscount`, and `defaultRefer` can be of `uint` type, instead of `uint8`, which will save gas (as the EVM prefers 32-byte types).[2]
✓ *Resolved in commit [21dd6734]*

line [10]: The `discountLimit` variable is not public, so it can be difficult to determine a valid input when attempting to call `setSplit` on line [32].
✓ *Resolved in commit [21dd6734]*

line [22]: The event `SplitterChanged` is never used.
✓ *Resolved in commit [21dd6734]*

line [70]: If the `discountLimit` is decreased in future, any splits which have been defined before the modification will not be updated.
✓*Acknowledged by author.*

Recommendations

line [10]: Change the type of `discountLimit`, `defaultDiscount`, and `defaultRefer` to `uint`.

line [10]: Make the `discountLimit` variable public, so as to avoid reverts when calling `setSplit`.

line [22]: Remove the `SplitterChanged` event or use it within the contract.

line [70]: While the discounts can be updated manually with `overrideSplit`, this functionality should be more clearly documented. Alternatively, calling `setDiscountLimit` on line [56] could also have functionality to update existing discounts.

[2] `discountPercentage` and `referrerPercentage` are okay to remain as `uint8` types, since they will tightly-pack in the struct.

GU-15	Other Informational Issues
Asset	Other Contracts
Status	**Resolved:** See inline comments
Rating	Informational

Several informational issues have been identified with other contracts in the repository, some of which are out of scope. These are provided as notes and recommendations, fixes for which are not essential.

Description

- `IPack.sol`, `IReferrals.sol`, and `IProcessor.sol`
  The interface contracts are specified as a `contract` whereas they are in fact interfaces.
  ✓ Resolved in commit [21dd6734]

- `ERC20Burnable.sol`
  The `burn` function on line [14] has a `public` visibility, allowing bundles to be destroyed before they are opened.
  ✓ Resolved in commit [21dd6734]

- `CappedVault.sol`
  In the fallback function on line [15], calculation of the incoming funds is doubled. Since the `total` function reads `this.balance`, which already includes `msg.value`, the incoming funds (in `msg.value`) are counted twice.

  Further, the `require` statement on this line does not provide an error, so when the `CappedVault` has reached its limit, the revert occurs without any warning.
  ✓ Resolved in commit [21dd6734]

Recommendations

- `IPack.sol`, `IReferrals.sol`, and `IProcessor.sol`
  Interface contracts should be explicitly specified as `interface`, as described in the Solidity documentation: solidity.readthedocs.io/en/v0.5.0/contracts.html#interfaces

- `ERC20Burnable.sol`
  Make the `burn` function `internal`, so that bundles can only be burned upon opening.

- `CappedVault.sol`
  Modify the require statement to simply say `require(total() <= limit)`, removing `msg.value` entirely.

  A message should also be added to the require statement, to handle errors more gracefully.

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are provided alongside this document. The `pytest` framework was used to perform these tests and the output is given below.

Please note that some tests have been marked with `XFAIL`, indicating that they are *expected* to fail. These tests are intended to demonstrate issues discovered during the review.

```
tests/test_admin_functions.py::test_set_can_lockup                                      PASSED
tests/test_admin_functions.py::test_set_can_revoke                                      PASSED
tests/test_admin_functions.py::test_set_commit_lag                                      PASSED
tests/test_admin_functions.py::test_set_activation_limit                                PASSED
tests/test_admin_functions.py::test_set_pack                                            PASSED
tests/test_admin_functions.py::test_set_activate                                        PASSED
tests/test_admin_functions.py::test_can_activate_purchase                               PASSED
tests/test_admin_functions.py::test_revoke                                              PASSED
tests/test_bundle_open.py::test_bundle_open                                             PASSED
tests/test_bundle_open.py::test_open_max_count                                          PASSED
tests/test_bundle_open.py::test_open_zero                                               XFAIL
tests/test_bundle_open.py::test_open_too_many                                           PASSED
tests/test_bundle_open_for.py::test_bundle_open_for                                     PASSED
tests/test_bundle_open_for.py::test_open_for_max_count                                  PASSED
tests/test_bundle_open_for.py::test_open_for_zero                                       XFAIL
tests/test_bundle_open_for.py::test_open_for_too_many                                   PASSED
tests/test_bundle_open_for.py::test_open_for_too_many_2                                 PASSED
tests/test_bundle_open_for.py::test_open_for_not_approved                               PASSED
tests/test_bundle_open_for.py::test_open_for_not_purchased                              PASSED
tests/test_bundle_open_for.py::test_open_for_not_purchased_enough                       PASSED
tests/test_bundle_purchase.py::test_bundle_purchase                                     PASSED
tests/test_bundle_purchase.py::test_purchase_max_count                                  PASSED
tests/test_bundle_purchase.py::test_bundle_purchase_discounted                          PASSED
tests/test_bundle_purchase.py::test_purchase_referrer_zero                              PASSED
tests/test_bundle_purchase.py::test_purchase_not_enough_eth                             PASSED
tests/test_bundle_purchase.py::test_purchase_buying_0_bundles                           PASSED
tests/test_bundle_purchase.py::test_purchase_refer_self                                 PASSED
tests/test_bundle_purchase.py::test_purchase_exceed_cap                                 PASSED
tests/test_bundle_purchase_for.py::test_bundle_purchase_for                             XFAIL
tests/test_bundle_purchase_for.py::test_purchase_for_max_count                          PASSED
tests/test_bundle_purchase_for.py::test_bundle_purchase_for_discounted                  PASSED
tests/test_bundle_purchase_for.py::test_purchase_for_referrer_zero                      PASSED
tests/test_bundle_purchase_for.py::test_purchase_for_not_enough_eth                     PASSED
tests/test_bundle_purchase_for.py::test_purchase_for_0_bundles                          PASSED
tests/test_bundle_purchase_for.py::test_purchase_for_refer_self                         PASSED
tests/test_bundle_purchase_for.py::test_purchase_for_exceed_cap                         PASSED
tests/test_deploy.py::test_deploy[bundle_sizes0-bundle_caps0]                           PASSED
tests/test_deploy.py::test_deploy[bundle_sizes1-bundle_caps1]                           PASSED
tests/test_deploy.py::test_deploy[bundle_sizes2-bundle_caps2]                           PASSED
tests/test_deploy.py::test_deploy[bundle_sizes3-bundle_caps3]                           PASSED
tests/test_deploy.py::test_deploy[bundle_sizes4-bundle_caps4]                           PASSED
tests/test_deploy.py::test_deploy[bundle_sizes5-bundle_caps5]                           PASSED
tests/test_deploy.py::test_deploy[bundle_sizes6-bundle_caps6]                           PASSED
tests/test_deploy.py::test_deploy[bundle_sizes7-bundle_caps7]                           PASSED
tests/test_deploy.py::test_deploy[bundle_sizes8-bundle_caps8]                           PASSED
tests/test_pack_five_activate.py::test_activate                                         PASSED
tests/test_pack_five_activate.py::test_activate_bad_index                               PASSED
tests/test_pack_five_activate.py::test_activate_already_active                          PASSED
tests/test_pack_five_activate.py::test_activate_no_randomness                           PASSED
tests/test_pack_five_activate_multiple.py::test_activate_multiple                       PASSED
tests/test_pack_five_activate_multiple.py::test_activate_multiple_2                     PASSED
tests/test_pack_five_activate_multiple.py::test_activate_bad_index                      PASSED
tests/test_pack_five_activate_multiple.py::test_activate_already_active                 PASSED
tests/test_pack_five_activate_multiple.py::test_activate_no_randomness                  PASSED
tests/test_pack_five_activate_multiple.py::test_activate_empty                          PASSED
tests/test_pack_five_callback.py::test_callback                                         PASSED
tests/test_pack_five_callback.py::test_bad_index                                        PASSED
tests/test_pack_five_callback.py::test_already_set                                      PASSED
tests/test_pack_five_callback.py::test_purchased_256_blocks_ago                         PASSED
```

```
tests/test_pack_five_predict_packs.py::test_predict_packs                               PASSED
tests/test_pack_five_predict_packs.py::test_predict_packs_gas                           XFAIL
tests/test_pack_five_predict_packs.py::test_bad_index                                   PASSED
tests/test_pack_five_predict_packs.py::test_no_random                                   PASSED
tests/test_pack_five_purchase.py::test_purchase_valid                                   PASSED
tests/test_pack_five_purchase.py::test_purchase_max_count                               PASSED
tests/test_pack_five_purchase.py::test_purchase_discounted                              PASSED
tests/test_pack_five_purchase.py::test_purchase_referrer_zero                           PASSED
tests/test_pack_five_purchase.py::test_purchase_different_count_vs_bundle_size          PASSED
tests/test_pack_five_purchase.py::test_purchase_not_enough_eth                          PASSED
tests/test_pack_five_purchase.py::test_purchase_buying_0_cards                          PASSED
tests/test_pack_five_purchase.py::test_purchase_bad_pack_type                           PASSED
tests/test_pack_five_purchase.py::test_purchase_refer_self                              PASSED
tests/test_pack_five_purchaseFor.py::test_purchase_for                                  PASSED
tests/test_pack_five_purchaseFor.py::test_purchase_for_max_count                        PASSED
tests/test_pack_five_purchaseFor.py::test_purchase_for_referrer_zero                    PASSED
tests/test_pack_five_purchaseFor.py::test_purchase_for_different_count_vs_bundle_size   PASSED
tests/test_pack_five_purchaseFor.py::test_purchase_for_refund                           PASSED
tests/test_pack_five_purchaseFor.py::test_purchase_for_not_enough_eth                   PASSED
tests/test_pack_five_purchaseFor.py::test_purchase_for_buying_0_cards                   PASSED
tests/test_pack_five_purchaseFor.py::test_purchase_bad_pack_type                        PASSED
tests/test_pack_five_purchaseFor.py::test_purchaseFor_refer_self                        PASSED
tests/test_pack_five_recommit.py::test_callback                                         PASSED
tests/test_pack_five_recommit.py::test_bad_index                                        PASSED
tests/test_pack_five_recommit.py::test_already_set                                      PASSED
tests/test_pack_five_recommit.py::test_purchased_256_blocks_ago                         PASSED
tests/test_pack_update.py::test_pack_update                                             PASSED
tests/test_processor_discount.py::test_processor_discount                               PASSED
tests/test_rarity_provider.py::test_card_details[lolcats]                               XFAIL
tests/test_rarity_provider.py::test_card_details[]                                      XFAIL
tests/test_rarity_provider.py::test_card_details[n7DUfYQ5EzwYjhMFXq9]                   XFAIL
tests/test_rarity_provider.py::test_card_details[YRnj76dncgGVBtkMG7CSQi54]              XFAIL
tests/test_rarity_provider.py::test_card_details[0]                                     XFAIL
tests/test_rarity_provider.py::test_extract_bytes                                       PASSED
tests/test_referrals_get_split.py::test_referrals_get_split                             PASSED
tests/test_referrals_get_split.py::test_referrals_get_split_above_discount              PASSED
tests/test_referrals_override_split.py::test_referrals_override_split_valid             PASSED
tests/test_referrals_override_split.py::test_referrals_override_split_invalid           PASSED
tests/test_referrals_set_defaults.py::test_referrals_set_defaults_valid                 PASSED
tests/test_referrals_set_defaults.py::test_referrals_set_defaults_invalid               PASSED
tests/test_referrals_set_discount_limit.py::test_referrals_set_discount_limit_valid     PASSED
tests/test_referrals_set_discount_limit.py::test_referrals_set_discount_limit_invalid   PASSED
tests/test_referrals_set_discount_limit.py::test_referrals_set_discount_limit_decrease  PASSED
tests/test_referrals_set_split.py::test_referrals_set_split_valid                       PASSED
tests/test_referrals_set_split.py::test_referrals_set_split_invalid                     PASSED
```

Appendix B Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

Impact			
High	Medium	High	Critical
Medium	Low	Medium	High
Low	Low	Low	Medium
	Low	Medium	High
		Likelihood	

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] Sigma Prime. Solidity Security. Blog, 2018, Available: https://blog.sigmaprime.io/solidity-security.html. [Accessed 2018].

[2] NCC Group. DASP - Top 10. Website, 2018, Available: http://www.dasp.co/. [Accessed 2018].

[3] Martin Kraft. Modulo Bias. December 2017, Available: https://medium.com/@martinkraft/modulo-bias-c5f985766e3d.

[4] ERC-20 Token Standard. Github, Available: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md.