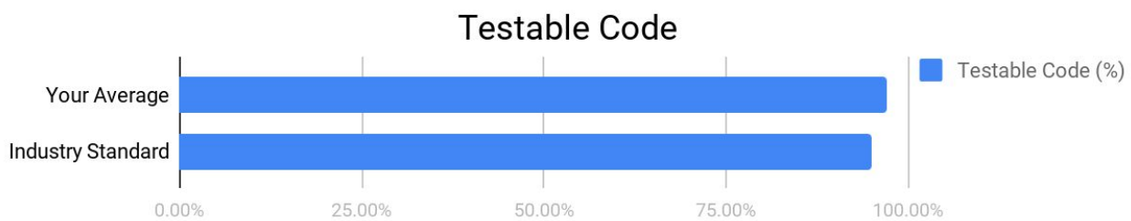
SimplyVitalHealth Contract Audit

by Hosho, April 2018

Executive Summary

This document outlines the overall security of SimplyVitalHealth's smart contract as evaluated by Hosho's Smart Contract auditing team. The scope of this audit was to analyze and document SimplyVitalHealth's token contract codebase for quality, security, and correctness.

All issues and suggestions have been successfully implemented, raising these contracts to a passing status. (See Complete Analysis)



The testable code is 97.03% which is higher than industry standard. (See Coverage Report)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, merely an assessment of its logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Hosho recommend that the SimplyVitalHealth Team put in place a bug bounty program to encourage further and active analysis of the smart contract.

Table Of Contents

1. Auditing Strategy and Techniques Applied

The Hosho Team performed a thorough review of the smart contract code as written and updated on April 5, 2018. All main contract files were reviewed using the following tools and processes. (See All Files Covered)

Throughout the review process, care was taken to ensure that the token contract:

- Implements and adheres to existing ERC20 Token standard appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of gas, without unnecessary waste; and
- Uses methods safe from reentrance attacks.
- Is not affected by the latest vulnerabilities

The Hosho Team has followed best practices and industry-standard techniques to verify the implementation of SimplyVitalHealth's token contract. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as they were discovered. Part of this work included writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

1. Due diligence in assessing the overall code quality of the codebase.
2. Cross-comparison with other, similar smart contracts by industry leaders.
3. Testing contract logic against common and uncommon attack vectors.
4. Thorough, manual review of the codebase, line-by-line.
5. Deploying the smart contract to testnet and production networks using multiple client implementations to run live tests.

2. Structure Analysis and Test Results

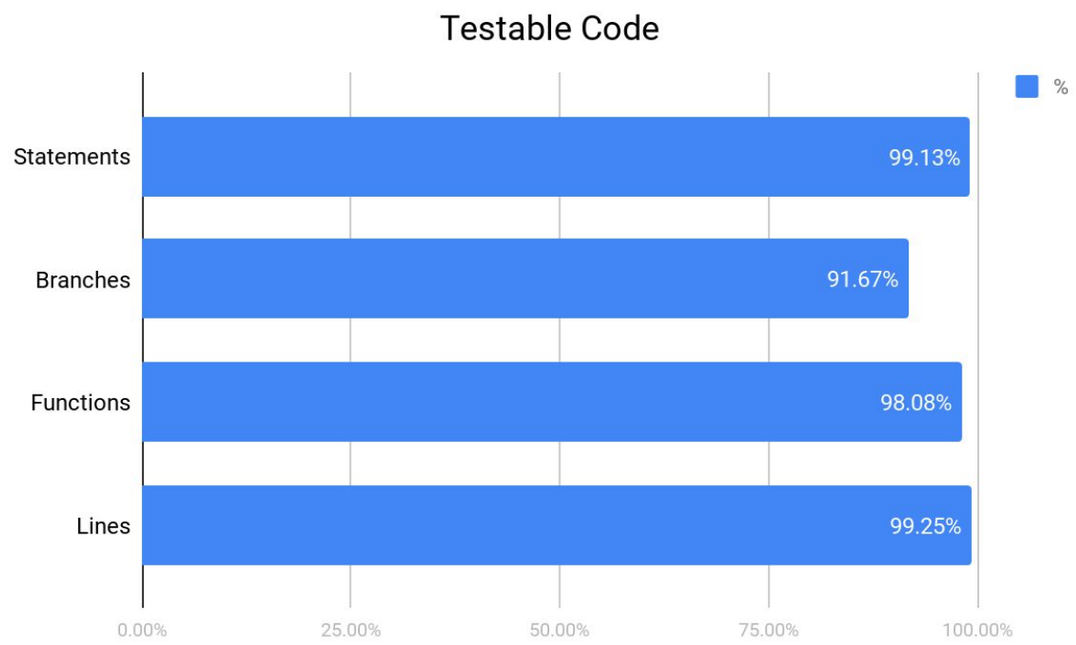
2.1. Summary

The HealthDRS project audited consists of three main smart contracts. The first is a basic ERC-20 token, used as a placeholder. The second is the Transmute contract, which issues events to convert the tokens from the HealthDRS token to their side-chain by burning the ERC-20 token. And, the third is the HealthDRS contract, which makes up the majority of the work. The HealthDRS smart contract serves as a tracking system for URL's (Services), keys that generate events when issued and other useful information.

There has been a small increase in coverage and branching as there are more executable portions of the contract. We're pleased to state that all uncovered paths, those that do not appear in the coverage report, have been manually verified. These cover systems like the address recovery (ecrecover wrapper) and the hard revert on the fallback function that require specific states to be applied.

2.2 Coverage Report

As part of our work assisting SimplyVitalHealth in verifying the correctness of their contract code, our team was responsible for writing a unit test suite using the Truffle testing framework.



For individual files see Additional Coverage Report

2.3 Failing Tests

No failing tests

See Test Suite Results for all tests.

3. Complete Analysis

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

- **Informational** - The issue has no impact on the contract’s ability to operate.
- **Low** - The issue has minimal impact on the contract’s ability to operate.
- **Medium** - The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.
- **High** - The issue affects the ability of the contract to compile or operate in a significant way.
- **Critical** - The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

3.1. Resolved, Critical: Unable to Send Data Using Message Function

Explanation

The service owner is unable to send data via the message function. `isKeyOwner` throws an exception, as the "from" value should never be a valid key, except in the case of a hash collision.

Resolution

A pre-check was added to the key creating the case that it will only check if it is owned if the key is valid, preventing the exception while retaining the ownership check.

3.2. Resolved, Critical: Unable to Send Data Using Log Function

Explanation

The service owner is unable to send data via the log function. `isKeyOwner` throws an exception, as the "from" value should never be a valid key, except in the case of a hash collision.

Resolution

A pre-check was added to the key so that it will only check if it is owned if the key is valid, preventing the exception while retaining the ownership check.

3.3. Resolved, High: createService Function is Unprotected

Explanation

The `createService` function is not protected, and costs nothing to execute. Given that, there is nothing to stop someone from mass-spamming URI's into this function and locking out URI's that may need to be transferred to someone else. There is also nothing stopping someone from simply executing, generating URI's, and then throwing away the account. As the service key is a SHA-3 sum, it is extremely unlikely that a hash collision would occur, but it would be good practice to have some way to clean up this data.

Resolution

The contract is intended for open use by the developer community so needs to be a publicly accessible function. To address the issue of locking out urls, the service id is now calculated using both the url and msg.sender as this will prevent specific urls from becoming unusable.

3.4. Resolved, Medium: Very few protections to ensure key ownership

Explanation

When keys are transferred, they should be locked down and secured, extra owners removed, all valid trade/sell states cleared, etc. As it stands, it's possible to sell/trade/handle keys, then "take" them back if you're fast on the chain and/or your partner doesn't perform due diligence. It's also possible to set up a sale in advance, trade the key, then purchase it back immediately because `cancelSalesOffer` isn't called after a trade is complete. Again, there are multiple paths for this, and the full flow and possible stops need to be mapped out.

Technical Example

I could offer to sell a key, setting the `canTrade` value to true, at which point, it could be sold. Then I could execute `tradeKey` with a prearranged trade that puts the key back under my control.

Resolution

Creating a `salesOffer` would cancel a `tradeOffer` and visa-versa so that you could only have one active at any given time. However, the format of the trade function was not consistent with the sales functions, so that was unclear. To resolve this the following changes were made:

1. Refactored the trade function, creating a `cancelTradeOffer` function to make it clear when this is used.
2. Added an additional requirement to `canSell` and `canTrade` modifiers, namely that the key must not be shared at all.
3. Added `cancelSalesOffer` to the `createTradeOffer` function and to the end of the `purchaseKey` function.

4. Added `cancelTradeOffer` to the `createSalesOffer` function and to the end of the `tradeKey` function.

This should accomplish the following: Only unshared keys can be sold or traded. There can only be an active sales offer or an active trade offer, never both. Completing a sale or trade removes that sale or trade offer.

3.5. Resolved, Medium: No Fallback Function

Explanation

The contract should have a fallback function, even if it contains nothing more than a `revert()` in order to ensure ETH isn't accidentally sent and trapped in the contract.

Resolution

Fallback function with revert added.

3.6. Resolved, Low: No Self-Destruct

Explanation

This contract could potentially become quite large over time, because it is functionally being used as a database. Therefore, we suggest considering the addition of a self-destruct capability, in case you desire to eventually move this data off the block chain and wish to remove the contract.

Resolution

As this contract is intended to be used by multiple service providers - having an `ownerOnly` self-destruct may preclude third-party use as the contract’s availability would not be guaranteed.

---

4. Closing Statement

We are grateful to have been given the opportunity to work with the SimplyVitalHealth Team.

During the initial static analysis, several possible vulnerabilities were noted, and the SVH team was very quick to patch these and return the updated code. As a small team of experts, having backgrounds in all aspects of blockchain, cryptography, and cybersecurity, we can say with confidence that the SVH contract is free of any critical issues.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

We at Hosho recommend that the SimplyVitalHealth Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

5. Test Suite Results

Contract: ERC-20 Compliant Token

✓ Should allocate tokens per the minting function, and validate balances (148ms)

✓ Should transfer tokens from 0x1bbb1269032bfd0b0fe0851235fc798af6bd3c9b to 0x42adbad92ed3e86db13e4f6380223f36df9980ef (66ms)

✓ Should not transfer negative token amounts

✓ Should not transfer more tokens than you have

✓ Should allow 0x3b44fa9f7511113a8c1a1528070d45b1d7cdd101 to authorize 0x341106cb00828c87cd3ac0de55eda7255e04933f to transfer 1000 tokens (50ms)

✓ Should not allow 0x3b44fa9f7511113a8c1a1528070d45b1d7cdd101 to authorize 0x341106cb00828c87cd3ac0de55eda7255e04933f to transfer an additional 1000 tokens once authorized, and authorization balance is > 0

✓ Should allow 0x3b44fa9f7511113a8c1a1528070d45b1d7cdd101 to zero out the 0x341106cb00828c87cd3ac0de55eda7255e04933f authorization

✓ Should allow 0xdaef8d8c30eeb858b8c774a8d7d5e92a552bb0d9 to authorize 0x53353ef6da4bbb18d242b53a17f7a976265878d5 for 1000 token spend, and 0x53353ef6da4bbb18d242b53a17f7a976265878d5 should be able to send these tokens to 0x341106cb00828c87cd3ac0de55eda7255e04933f (98ms)

✓ Should not allow 0x53353ef6da4bbb18d242b53a17f7a976265878d5 to transfer negative tokens from 0xdaef8d8c30eeb858b8c774a8d7d5e92a552bb0d9

Contract: Ownable

✓ Should instantiate contract with owner set to accounts[0]

✓ Should not allow transfer of ownership unless done by owner

✓ Should not allow transfer of ownership to null account by a non-owner

✓ Should not allow transfer of ownership to null account by the owner

✓ Should allow transfer of ownership (40ms)

Contract: Transmute

✓ Should not set the token to BurnableInstance with `setTokenToTransmute` using non-owner

✓ Should set the token to BurnableInstance with `setTokenToTransmute`

✓ Should instantiate contract with `transmuteNonce` set to 0 (39ms)

✓ Should not be able to call enable with non-owner

✓ Should not be able to call `transmuteToken` with value greater than allowance

✓ Should not be able to call `transmuteToken` when enabled is false

✓ Should set boolean enable to true

✓ Should `transmuteToken` when boolean enabled is true (65ms)

Contract: HealthDRS

✓ Should instantiate contract with owner set to `accounts[0]`

✓ Should not allow transfer of ownership unless done by owner

✓ Should not allow transfer of ownership to null account by a non-owner

✓ Should not allow transfer of ownership to null account by the owner

✓ Should allow transfer of ownership (57ms)

✓ Should instantiate contract with Burnable token

✓ Should instantiate contract with `latestContract` set to `hlthDRSInstance.address`

✓ Should instantiate contract with version set to 1

✓ Should not allow `isKeyOwner` to be called with an invalid, null key

✓ Should not allow `isServiceOwner` to be called with an invalid, null service

✓ Should not allow `getUrl` to be called with an invalid, null service

✓ Should not allow `getUrlFromKey` to be called with an invalid, null key

✓ Should not allow `setHealthCashToken` to be called by non-owner with a StandardToken contract

✓ Should allow `setHealthCashToken` to be called by owner with a StandardToken contract (55ms)

✓ Should return the token allowance for msg.sender

✓ Should not allow `setLatestContract` to be called by non-owner

✓ Should allow `setLatestContract` to be called by owner with the latest contract (64ms)

✓ Should allow `createService` to be called (68ms)

✓ Should return the url associated with a service when `getUrl` is called

✓ Should return true when `isKeyOwner` is called for an owned key

✓ Should not allow `createService` to be called for existing url

✓ Should not allow `createKey` to be called by non-owner

✓ Should not allow `issueKey` to be called by non-service owner

✓ Should allow `issueKey` to be called (61ms)

✓ Should allow `createKey` to be called, creating a new key (47ms)

✓ Should return the url associated with a key when `getUrlFromKey` is called

✓ Should allow service owner to update a url with `updateUrl` (121ms)

✓ Should return true when isKeyOwner is called for an owned key

✓ Should not allow for `shareKey` to be called by non-key owner

✓ Should not allow for `shareKey` to be called when boolean `canShare` is false

✓ Should not allow for `shareService` to be called from non-service owner account

✓ Should not allow for `unshareKey` to be called from non-key owner account

✓ Should not allow for `unshareService` to be called from non-service owner account

✓ Should not allow for `createSalesOffer` to be called from non-key owner account

✓ Should not allow for `createSalesOffer` to be called when boolean `canSell` is false

✓ Should not allow for `cancelSalesOffer` to be called from non-key owner account

✓ Should not allow for `purchaseKey` to be called when boolean `canSell` is false

✓ Should not allow for `tradeKey` to be called by non-key owner

✓ Should not allow for `createTradeOffer` to be called by non-key owner

✓ Should not allow for `tradeKey` to be called when boolean `canSell` is false

✓ Should not allow for `createTradeOffer` to be called when boolean `canSell` is false

✓ Should not allow for `setKeyPermissions` to be called with invalid key

✓ Should not allow for `setKeyPermissions` to be called by non-key owner

✓ Should return the length of the `serviceList` array

✓ Should return the length of the `keyList` array

✓ Should not allow for `setKeyPermissions` to be called by non-key owner

✓ Should return service url and owner when getService is called

✓ Should not allow for `setKeyPermissions` to be called by non-key owner

✓ Should return the `service_id` and owner for a key

✓ Should not allow for `setKeyData` to be called for invalid key

✓ Should not allow for `setKeyData` to be called by non-service owner

✓ Should return the length of the `keyList` array (66ms)

✓ Should not allow for `getKeyData` to be called by invalid key

✓ Should not allow for `logAccess` to be called with invalid key

✓ Should not allow for `logAccess` to be called by non-service owner

✓ Should call `logAccess` and record sent data in Access event (50ms)

✓ Should call message and record sent data in Access event (50ms)

✓ Should call message and record sent data in Access event using service_id (50ms)

✓ Should not call message and record sent data in for null address

✓ Should not allow for message to be called by non key and non-service owner (65ms)

✓ Should not allow for log to be called by non key and non-service owner

✓ Should call log and record sent data in Access event

✓ Should call log and record sent data in Access event for a service_id

✓ Should allow key owner to `setKeyPermissions` (100ms)

✓ Should return the number of owners on a key

✓ Should allow key owner to authorize another account with the `shareKey` function (68ms)

✓ Should not allow the same account to be shared with multiple times (64ms)

✓ Should return false for `isKeyOwner` with non-owner account

✓ Should allow service owner to authorize another account with the `shareService` function (59ms)

✓ Should not allow service owner to authorize the same account with the `shareService` function (60ms)

✓ Should return false for `isServiceOwner` with non-owner account

✓ Should allow key owner to deauthorize another account with the `unshareKey` function (313ms)

✓ Should allow service owner to deauthorize another account with the `unshareService` function (251ms)

✓ Should allow key owner to create and cancel a sales offer for an owned key (97ms)

✓ Should allow key owner to create a sales offer and buyer to purchase an owned key (259ms)

✓ Should not allow anyone but the buyer to use a `salesOffer` (104ms)

✓ Should allow `createKey` to be called (52ms)

✓ Should not allow `tradeKey` to be called when `canTrade` is false for the want key

✓ Should not allow `createTradeOffer` to be called when canTrade is false for the want key (53ms)

✓ Should allow key owner to create and execute a trade (215ms)

✓ Should allow `createKey` to be called (57ms)

✓ Should not allow a trade offer where the key is not the wanted one (129ms)

✓ Should call log and record sent data in Access event

✓ Should not accept ETH transfers

✓ Should allow the owner to retrieve tokens that are trapped (64ms)

✓ Should allow the minimum hold to be set

✓ Should require the holding of a number of tokens in order to generate a new key

6. All Contract Files Tested

Updated Files	Fingerprint (SHA256)
contracts/BurnableToken.sol	16cff3b4e53238cf57baaaa95c069a682be96274f00da95de077b14545729b38
contracts/HealthDRS.sol	98d644bd29dbce1f66740f0a7029bd0b9107ad355304ebaedd4eb307fc32916f
contracts/TransmuteAgent.sol	8482b625ded9118b372199d0668449bdc11a2311aebac05c8b958dc9b0cf3477

7. Individual File Coverage Report

File	% Statements	% Branches	% Functions	% Lines
contracts/BurnableToken.sol	100.00%	100.00%	100.00%	100.00%
drs/contracts/HealthDRS.sol	99.00%	92.42%	97.78%	99.15%
transmute/contracts/TransmuteAgent.sol	100.00%	83.33%	100.00%	100.00%
All Files	**99.13%**	**91.67%**	**98.08%**	**99.25%**