Basic Attention Token (BAT) Audit

MAY 22, 2017 | IN SECURITY AUDITS | BY OPENZEPPELIN SECURITY

The Brave team asked us to review and audit their new BAT Token contract code. We looked at their contracts and now publish our results.

The audited contracts can be found in their basic-attention-token-crowdsale repo. The version used for this report is commit 17a5f8440a256a6dc5d8dd894b9615182c2901b2.

Here's our assessment and recommendations, in order of importance.

Update: Brave team followed most of our recommendations in the latest version of their code.

Severe

Brave can get an unfair refund if tokenCreationMin is not reached

A fixed 300 million BAT tokens are assigned to Brave (specifically to the `batFundDeposit` address) when the crowdsale contract is deployed. If the contract doesn't reach the minimum amount of tokens (`tokenCreationMin`), though, it goes into a state where investors can request a refund and receive their ether back.

In this situation, Brave would also be able to request a refund for the 300 million tokens it holds, taking ether that should be available for investors.

Fix this by checking that funds aren't sent to `batFundDeposit` in the `refund` function.

Update: fixed in commit 09a1038c2ac0148b93876057fc056ae2a872fb88.

Potential problems

Correctness of start and end block numbers

Brave has decided to use block numbers instead of timestamps to determine the start and end of the crowdsale, which is a good security practice. However, preconditions about them are missing in the contract constructor.

For example, consider checking `fundingStartBlock` comes before `fundingEndBlock`, and that both are in the future (greater than `block.number`).

Use of Transfer event to log token creation

The crowdsale contract uses the Transfer event to log the creation of tokens, with a *from* field set to the zero address. This is overloading the semantics of the `Transfer` event of the ERC20 standard, which might not be a good idea as other software (such as wallets or exchanges) could be relying on these semantics. This is an ongoing discussion in the ERC20 standardization process, but our stance on it is that a different event should be used.

Update: fixed in commit 09a1038c2ac0148b93876057fc056ae2a872fb88.

Avoid duplicate code

The contracts `Token`, `StandardToken`, and `SafeMath` are very similar to OpenZeppelin library contracts `ERC20Token`, `StandardToken`, and `SafeMath`, respectively. Consider avoiding code repetition, which can bring regression problems and introduce unexpected bugs. Consider using the standard modules from OpenZeppelin.

Several conditional checks, such as `block.number &lt; fundingEndBlock`, are repeated throughout the code (line 51, line 68, line 79). Consider extracting these to reusable internal functions with declarative names (for example `isWithinFundingPeriod`) for better code readability.

Use SafeMath in all math operations

Most math operations are safe, but there's still a few that are unchecked (these two, for instance). It's always better to be safe and perform checked operations.

Update: fixed in commit 09a1038c2ac0148b93876057fc056ae2a872fb88.

Burn of token assignment to BAT fund is not implemented

The amount of BAT assigned to Brave is fixed at deploy time. This seems to be inconsistent with the process document we received that states: "In event of a non-sell out, contract will 'burn' BAT kept in multisig wallet down to a 3/10 ratio of total BAT.". Consider implementing this in the contract, as the specification states, so that the Brave team doesn't have to do this manually after the token sale.

Warnings

Magic constants in token amounts

A few constants in the contract which denote BAT amounts (`batFund`, `tokenCreationCap`, `tokenCreationMin`) are written assuming 18 decimals. This is a hidden case of a magic constant. These amounts should be parameterized by the number of decimals (*e.g.* `batFund = 300 * 10**6 * 10**decimals`).

Update: fixed in commit 09a1038c2ac0148b93876057fc056ae2a872fb88.

Refactor confusing finalization precondition

The condition in line 68, is expressed in a very confusing way. It should be split into two conditional expressions: `totalSupply &lt; tokenCreationMin` and `block.number &lt;= fundingEndBlock &amp;&amp; totalSupply != tokenCreationCap`, each of which throws when true.

Update: fixed in commit 09a1038c2ac0148b93876057fc056ae2a872fb88.

isFunding variable name is not accurate

The variable `isFunding` declared in line 18 doesn't really reflect whether the contract is *funding* or not. This fact depends on other conditions, such as being within the funding period. The variable marks the finalization by the contract owners, so a good name would be **isFinalized** (note that this new name reverses the meaning: `isFinalized == !isFunding`).

There's a comment in line 18 that says the variable is "no longer important", but it is actually a key part of the crowdsale process. Consider removing that comment.

Update: fixed in commit 09a1038c2ac0148b93876057fc056ae2a872fb88.

Avoid using var to define variables

There's two uses of `var`, where the type of the variable is inferred from the expression on the right hand of the definition. We recommend avoiding this feature because in some cases it might infer a smaller integer type than the developer might think. It is best to be explicit regarding types.

Update: fixed in commit 09a1038c2ac0148b93876057fc056ae2a872fb88.

Solidity version

All of the contracts require version 0.4.10 of Solidity. It should also be noted that 0.4.11 was released a few days ago. Consider changing the solidity version pragma to the latest version (`pragma solidity ^0.4.11;`) to enforce latest compiler version to be used.

As an additional note, since the contracts are compiled with a recent version of Solidity, it's possible to use the new `transfer` method instead of `send`(here, and here).

Additional information and notes

- The comment in line 64 is outdated since the `finalize` function no longer issues BAT tokens to the User Growth Fund. These tokens seem to be issued in the constructor.
- The state variable `batFund` in line 21 is only set at deploy time, and never changed, so it could be declared `constant`.
- Consider being consistent in the use of whitespace for a more polished code style. For example see how whitespace is used differently in lines 78 and 79.
- The operational process document appears to be outdated. The Finalization section states that the `finalize` function can be called from any account, but this is neither what the code does, nor what the review document says.
- Consider removing commented code in lines 6–10 from SafeMath.sol.
- Very interesting addition of contract version as part of the token contract in line 11.
- Remove excessive condition from SafeMath (see this pull request in OpenZeppelin).
- Use of send is always risky and should be analyzed in detail. These contracts use it safely, by checking the return value, and calling it only at the end of functions.

- Good job using block numbers instead of timestamps.
- Consider choosing a fail-first approach for precondition in line 55: revert the condition and throw in the *then* branch, instead of the *else* branch.
- Consider reusing the `tmpSupply` variable instead of recalculating the sum on line 56.

Conclusions

One severe security issue was found and explained, along with recommendations on how to fix it. Some additional changes were proposed to follow best practices and reduce potential attack surface.

Update: Brave team followed most of our recommendations in the latest version of their code.

Thanks to Francisco Giordano for helping write this report.

Security Audits

- If you are interested in smart contract security, you can continue the discussion in our forum, or even better, join the team 🚀
- If you are building a project of your own and would like to request a security audit, please do so here.

RELATED POSTS