

Audit Report

DAWN

Summary and Scope

29	5	0	1
Notes	Warnings	Bugs	Vulnerabilities

This audit covers DAWN's EIP-20/EIP-777 token implementation ([DawnTokenImpl.sol](#)), token swapping contract ([TokenSwap.sol](#)), and its first actual application, Staking ([Staking.sol](#)). Also previously unaudited [Recoverable.sol](#) inherited by these contracts was covered. DAWN's official repository <https://github.com/Dawn-Protocol/dawn-erc20> was used, latest commit being [30f9c328fdf4bad4e749bfb93f9606e1a62be665](#).

During the audit, attention was paid to design, overall code quality, best practices, business logic and security.


The DAWN Team fixed all the critical problems encountered. However, these fixes are out-of-scope of this audit.

The findings of the audit are discussed in this document. Some minor findings, such as typos, are omitted from this document, but were reported to the DAWN Team along with the other findings discussed here.

Design

DAWN Token is designed to replace the earlier FirstBlood token ([ST](#)), which was designed in the early days of Ethereum, before the EIP-20 standard, and is therefore out-of-date and not usable to build modern token based smart contract applications. DAWN Token implements the final version of the EIP-20 standard, and also the new EIP-777 suitable for modern smart contract applications.

A swapping contract is provided, so existing ST holders can get an equal amount of DAWNs.



Staking, the first application utilizing the DAWN Token, is included, giving users a possibility to stake their tokens for a certain period of time so they could participate in governance, amongst other things.

The DAWN Team made some, possibly problematic, design choices which are discussed next, with possible solutions.

Token is upgradeable

Although it may not look like a problem, the original ST token is not upgradeable and therefore immutable, unlike the new DAWN Token is.

Although the DAWN Team's motivation is clear (so they wouldn't need to do this kind of swapping again when upgrading the token), it opens the token and its balances to manipulation. This might be an issue to some of the tech savvy investors who did invest in the token back in the days when it was immutable.

Possible remedies include:

- making sure the address capable of upgrade is a multisignature wallet smart contract, with a wide spectrum of participants (such as external lawyers from a reputable company), and
- being open to the public about this design choice and the participants, to prevent any PR damage.

Burning is manual

After a ST token holder swaps their ST tokens to obtain DAWN, the ST tokens are burned and effectively removed from circulation. Burning in ST token's context means transferring tokens to an unreachable address (such as [0x0](#)).

Burning does not happen immediately though, and must be called by a designated operator. The designated operator can also change the address where burned tokens are sent to. It's unclear why there is an option to change the burning address in the first place: 0x0 would be as good as any other unreachable address. Before the designated operator calls the burning function, the tokens are credited to the swapping contract itself.

This design creates two separate technical problems:

- the designated operator can transfer ST tokens to any arbitrary address by changing the burning address, and

- the designated operator can transfer ST tokens to any arbitrary address by using the token recovery option provided by the inherited `Recoverable.sol`.

In theory, the designated operator could confiscate and sell the ST tokens on an exchange, since at the time of writing ST is traded on multiple exchanges.

The current approach may also raise tax and accounting issues for the company issuing DAWN Tokens, since the ST tokens are manually sent to an arbitrary address. This could be seen as selling DAWN for ST. An expert should be consulted on tax, legal, and accounting questions.

A possible remedy would be to redesign the burning process, so that burning would happen immediately to a static unreachable address `0x0`. It would increase the gas price for swapping transactions, but because this is designed to be a one-off operation, that should not be a problem for users.

Implementation

Code is overall of high quality: constant coding style is used, logic is well thought out, the code is well structured and most of the best practices are adhered to.

The Positive

- Well tested, audited and widely used code from OpenZeppelin was used, including their EIP-20 and EIP-777 implementations.
- Consistent coding style is used, and follows Solidity Style Guidelines.
 - In some functions, empty lines are used in the beginning of the function, and/or in the end of the function. Otherwise the coding style is consistent.
- Code is well structured, and easy to understand: variable and function naming is clear and well thought out.
- The code is written in modern Solidity (`0.5.x`).
- Earliest safe Solidity (`0.5.16`) is used, which is a good practice: the newer versions could have regressions. At the time of writing `0.5.16` didn't contain any serious [vulnerabilities](#).
- `require()`s are used often with meaningful error messages to verify function input.

The Negative

- Solidity version is not fixed. **Recommendation:** use `pragma` with fixed Solidity version to prevent accidental compilation using a compiler for which the code hasn't been designed for.
- SafeMath (or comparable) is not used. In this project lack of SafeMath does not pose any issues, but using SafeMath everywhere is considered to be a good practice.

- `assert()`s are never used. This makes symbolic execution tools (such as Mythril) practically useless, since there is no way to differentiate between normal errors (`require()`) and programming errors (`assert()`).
- Ethereum semantic documentation format for comments “NatSpec” is not used, or when used, the usage is incomplete. **Recommendation:** use NatSpec everywhere, because the source code could live forever in places like Etherscan and/or IPFS.
- Comments are often incomplete, misleading and contain typos and bad grammar.
- Use “uint256” instead of “uint” for clarity, like Zeppelin uses in their OpenZeppelin contracts. This project is heavily dependent on OpenZeppelin, so following their coding style is a good practice.
- Initializers are called inconsistently. **Recommendation:** initialize all related contracts consistently from contract to contract for readability.

Vulnerabilities (1)

Vulnerabilities are bugs which have serious consequences, such as loss of capital. Only one vulnerability was found.

Staking.sol	182	Checks-Effects-Interactions pattern is not followed, resulting in a “DAO Hack” style vulnerability which lets anyone drain the contract of tokens. Recommendation: use Checks-Effects-Interactions pattern.
-----------------------------	-----	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Bugs (0)

Bugs are programming errors which are not serious enough to cause serious consequences, such as loss of capital, but can be problematic in other ways. No bugs were found.

Warnings (5)

Warnings are features which work in a way that could differ from its original purpose, but are not bugs. These might have unintended consequences.

Recoverable.sol	36	Memory map padding is not used, making this particular contract not very upgradeable. Recommendation: add
---------------------------------	----	------------------------------------------------------------------------------------------------------------------

		padding, like in Zeppelin's Ownable.sol:80 .
Staking.sol	26	Because the DAWN Token supports also EIP-20, inherited <code>Recoverable</code> gives the designated operator a possibility to confiscate all the staked tokens. Recommendation: implement Recoverable.sol:tokensToBeReturned() here.
TokenSwap.sol	22	Because the original FirstBlood (ST) token supports EIP-20, inherited <code>Recoverable</code> gives the designated operator a possibility to confiscate all the swapped ST tokens prior to burning. Recommendation: redesign the burning process so, that the ST tokens would be burned straight away.
TokenSwap.sol	94	<code>swapTokensForSender()</code> can be called multiple times with different amounts because signature does not contain nonce, nor is flagged as used. This could lead a verified user to funnel tokens from unverified users. It's not clear will server side verification include the amount, and/or should the signature be disposable. However, if this is desired, this should be properly documented to prevent implementation errors off-chain.
TokenSwap.sol	107	<code>getCurrentlySwappedSupply()</code> is useless and misleading: does not report swapped tokens, unlike <code>totalSwapped</code> would. Instead it reports only ST tokens currently in the contract, which is not needed (a standard EIP-20 <code>balanceOf()</code> would suffice when this information is needed). Recommendation: use <code>totalSwapped</code> instead.

Notes (29)

Notes are not problems per se, but instead points to pay attention to.

DawnTokenImpl.sol	60	<code>bytes('')</code> should be used for clarity, like in Staking.sol .
Recoverable.sol	26	This function could be “external” instead of “public” to save gas, because it's not used internally. Recommendation: use “external” always if the function is not used internally.

Recoverable.sol	27	Use <code>require()</code> to check the return value, and handle “false” per EIP-20.
Staking.sol	77	<code>stakes</code> should be public. Recommendation: make it public, and remove redundant <code>getStakeInformation()</code> .
Staking.sol	104	<code>_token</code> should be <code>IERC777</code> instead of address per ConsenSys recommendations on variable naming.
Staking.sol	132	Use “_” consistently when naming internal functions. So <code>stakeInternal()</code> would become <code>_stake()</code> . Recommendation: use consistent naming throughout the project.
Staking.sol	167	This function could be “external” instead of “public” to save gas, because it’s not used internally. Recommendation: use “external” always if the function is not used internally.
Staking.sol	176	This function could be “external” instead of “public” to save gas, because it’s not used internally. Recommendation: use “external” always if the function is not used internally.
Staking.sol	185	By nullifying the whole struct upon unstaking some gas would be saved because of gas refunds. Recommendation: use <code>delete stakes[stakeId];</code>
Staking.sol	193	Misleading comment: only “ <code>stakePriceOracle</code> ” can change the staking parameters. Recommendation: update the comment to match the current code.
Staking.sol	208	Maybe switch from custom made oracle role to RBAC roles like <code>Pausable.sol</code> does? Recommendation: always recycle logic wherever possible.
Staking.sol	209	Inconsistent naming. Recommendation: use either <code>stakePriceOracle</code> , <code>stakePriceOracle</code> OR <code>oracle</code> throughout the code, and name related names accordingly (such as the emitted event and arguments).
Staking.sol	231	Redesign and simplify <code>tokensReceived</code> . Recommendation: the new logic could always presume

		address to stake for. It's clear that the data payload is not meant to be manually generated in the first place. Since it is meant to be machine generated, the address (being the caller or not) can be easily added by the off-chain software interacting with the contract.
TokenSwap.sol	24	<code>oldToken</code> should be public for transparency, and easy verification after deployment. Recommendation: make <code>oldToken</code> a public variable.
TokenSwap.sol	25	<code>newToken</code> should be public for transparency, and easy verification after deployment. Recommendation: make <code>newToken</code> a public variable.
TokenSwap.sol	37	<code>owner</code> should be indexed. Recommendation: add <code>indexed</code> for better off-chain integration.
TokenSwap.sol	47	Disparity between code and comment: the function is named <code>initialize()</code> , not <code>initializeTokenSwap()</code> . Recommendation: change either to match the other.
TokenSwap.sol	71	This check is redundant: already done here . Recommendation: remove redundant check for readability and smaller code footprint.
TokenSwap.sol	72	The explicit check for "true" is not needed, since it's the default operation for <code>require()</code> . Recommendation: drop the redundant condition for simplicity.
TokenSwap.sol	73	The explicit check for "true" is not needed, since it's the default operation for <code>require()</code> . Recommendation: drop the redundant condition for simplicity.
TokenSwap.sol	73	Checks-Effects-Interactions pattern is not followed. In this case it does not pose a problem because <code>totalSwapped</code> is not used inside the contract, but a good practice is to always follow Checks-Effects-Interactions pattern. Recommendation: always follow Checks-Effects-Interactions where possible.
TokenSwap.sol	94	This function could be "external" instead of "public" to save gas, because it's not used internally. Recommendation: use "external" always if the

		function is not used internally.
TokenSwap.sol	112	Misleading comment: this function returns the amount of DAWN Tokens still left to swap. Recommendation: update comments to match the current functionality of the function, for better readability.
TokenSwap.sol	119	This function could be “external” instead of “public” to save gas, because it’s not used internally. Recommendation: use “external” always if the function is not used internally.
TokenSwap.sol	121	Consider emitting an event here for better integration with off-chain applications. Recommendation: use events often, to help off-chain innovation later.
TokenSwap.sol	125	Consider emitting an event here for better integration with off-chain applications. Recommendation: use events often, to help off-chain innovation later.
TokenSwap.sol	129	Consider emitting an event here for better integration with off-chain applications. Recommendation: use events often, to help off-chain innovation later.
TokenSwap.sol	140	This function should be moved to a separate contract to make the code footprint smaller: this function is not specific to this contract, and could live elsewhere. Recommendation: consider providing this as a public community service as a separately deployed contract.
TokenSwap.sol	156	This function should be moved to a separate contract to make the code footprint smaller: this function is not specific to this contract and could live elsewhere. Recommendation: consider providing this as a public community service as a separately deployed contract.

About the Author

Ville Sundell has been involved with smart contracts since Bitcoin’s “Script”, stumbled upon Ethereum in late 2015 and published his first Solidity smart contract in summer 2016. Since his first audit in [2017](#), he has worked full time with smart contracts: developing and auditing for Ethereum and EOSIO based platforms.