FANTOM

Fantom Contract Review

Version: 2.0

June, 2018

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the smart contract `FantomToken`, which governs both the Fantom Initial Coin Offering (ICO) and the dynamics of the Fantom (FTM) token. The review focused solely on the security aspects of the Solidity implementation of the contract, but also includes general recommendations and informational comments relating to minimizing gas usage, token functionality and ERC20 compliance.

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 functionality of the contract (`FantomToken`) contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability 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 within the `FantomToken` contract.

Overview

The `FantomToken` contract serves multiple purposes, namely it

- Dictates the protocols by which Fantom tokens (FTMs or `fantoms`) can be purchased through Fantom's Initial Coin Offering (ICO).
- Governs the dynamics of the `fantom` tokens, which are ERC20 [1] tokens.
- Allows the contract `owner` to mint tokens and assign them to arbitrary recipients. The number of tokens minted is constrained such that the total supply of `fantoms` cannot exceed `TOKEN_TOTAL_SUPPLY`.
- Allows the contract `owner` to time-lock newly minted `fantom` tokens.
- Allows `fantom` holders to migrate their `fantom` holdings when Fantom's DAG-based platform is operational (at an as-yet unspecified date).

A minimum amount of `0.5 ether` is required to participate in Fantom's ICO and, for every `ether` contributed to the ICO, the number of `fantoms` received by participants is specified by the variable `tokensPerEth`.

Technically speaking, the "migration" achieved by the `FantomToken` contract amounts to burning `fantoms`. Token burning events are indexed, allowing Fantom to track them and subsequently issue Fantom's native token (valid on the Fantom platform), to `fantom` holders, in proportion to the amount of `fantoms` burnt by the holder.

It is emphasised that `FantomToken` does not contain any development relating to, or provide any functionality for, Fantom's DAG-based platform. The contract relates solely to the Ethereum based ERC20 `fantom` token. For additional details regarding Fantom's DAG-based platform the reader is directed to the Fantom Whitepaper [2].

Audit Summary

This review was initially conducted on commit fd6ba0c, which contains the sole file FantomToken.sol. This file contains a number of contracts, all of which are inherited (directly or indirectly) by the `FantomToken` contract. The complete list of contracts contained in FantomToken.sol is as follows:

```
└── FantomToken.sol
    ├── SafeMath
    ├── Utils
    ├── Owned
    ├── Wallet
    ├── ERC20Interface
    ├── ERC20Token
    ├── LockSlots
    ├── FantomIcoDates
    └── FantomToken
```

The final version of this review targets commit 1a7313c.

Per-Contract Vulnerability Summary

SafeMath (`FantomToken.sol`)

Some informational notes are given.
No potential vulnerabilities have been identified.

Utils (`FantomToken.sol`)

Some informational notes are given.
No potential vulnerabilities have been identified.

Owned (`FantomToken.sol`)

No potential vulnerabilities have been identified.

Wallet (`FantomToken.sol`)

No potential vulnerabilities have been identified.

ERC20Interface (`FantomToken.sol`)

No potential vulnerabilities have been identified.

ERC20Token (`FantomToken.sol`)

Some gas-saving modifications are suggested.
No potential vulnerabilities have been identified.

LockSlots (`FantomToken.sol`)

Some gas-saving modifications are suggested.
No potential vulnerabilities have been identified.

FantomIcoDates (`FantomToken.sol`)

Some informational notes are given.
No potential vulnerabilities have been identified.

FantomToken (`FantomToken.sol`)

Some informational notes are given.
Some gas-saving modifications are suggested.
All raised issues were resolved.
No further vulnerabilities were identified.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Fantom 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”.

Summary of Findings

ID	Description	Severity	Status
FTM-01	An inactive `owner` can permanently lock tokens.	Medium	Resolved
FTM-02	`owner` or `wallet` recipient can mint tokens for free.	Medium	Resolved
FTM-03	Misleading total supply.	Low	Resolved
FTM-04	Unintended token burning due to invalidation of `_to` in transfer functions.	Low	Resolved
FTM-05	ERC20 standard compliance.	Informational	Resolved
FTM-06	Gas savings.	Informational	Resolved
FTM-07	Miscellaneous notes and comments.	Informational	Resolved

FTM-01	An inactive `owner` can permanently lock tokens.		
Asset	FantomToken.sol		
Status	Closed: Resolved in [259b162]		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

Fantom tokens become tradeable after the boolean state variable `tokensTradeable` is set to `true`. Failure to set this variable to `true` means that no trading/transferring of tokens is possible. Only the contract `owner` can modify `tokensTradeable` by calling the function `makeTradeable()` at the conclusion of the ICO period. However, if the `owner` was unable to access their private key, due to e.g. error or incapacitation, it would not be possible to set `tokensTradeable = true` and `fantom` tokens would remain permanently untransferable. This vulnerability provides a single point of failure that is capable of permanently paralysing all `fantoms`.

Recommendations

There are a number of ways to remove this single point of failure. One technique is to include a mechanism that allows other users to set the `tokensTradeable` variable to `true` after a period of time. For example, the `makeTradeable()` function could remove the `onlyOwner` modifier and add a `require()` of the form `require(msg.sender == owner || atNow() > dateMainEnd + 3 weeks);`.

FTM-02	`owner` or `wallet` recipient can mint tokens for free.		
Asset	FantomToken.sol		
Status	Closed: See the Resolution.		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

The `owner` has the ability to call the `setWallet()` function at any time without restriction. The `wallet` address immediately receives all ether that is deposited into the `FantomToken` contract.

This allows the `owner` or the `wallet` beneficiary to cyclically purchase tokens during the crowdsale. Consider the cycle: `wallet` user buys tokens using `100 ether`. The `100 ether` is immediately returned to the user and the user is credited with tokens. The user repeats this process, accumulating tokens for free.

Equivalently, the `owner` during the crowdsale, can change the `wallet` address to themselves, purchase tokens a number of times, then change the `wallet` address back.

Although the `owner` can already mint tokens for free, they are capped to `TOKEN_TOTAL_SUPPLY - TOKEN_MAIN_CAP`. This vulnerability allows the owner to mint tokens for free, including the amount specified by `TOKEN_MAIN_CAP`.

This is a low likelihood attack as only the `owner` or designated `wallet` address can perform this attack.

Note: The level of trust on the owner may be acceptable for the authors of this crowdsale. This issue is raised as potential investors can be diluted in the event of a malicious owner, or if an attacker gains control of the wallet or owner's private keys.

Recommendations

Only allow the ether purchased in the crowdsale to be withdrawn once the crowdsale has completed.

Resolution

As only a small portion of funds are being raised via the crowdsale, the likelihood and the impact of this issue are low and therefore no action was deemed necessary.

FTM-03	Misleading total supply.		
Asset	FantomToken.sol		
Status	Closed: See the Resolution.		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The `totalSupply()` function does not report tokens which have not yet been minted. This is misleading as the owner may, at any time, mint their total allocation and place them on the market for sale.

Metrics which calculate market capitalisation based upon `totalSupply()` would not be accurate until the moment that the owner mints the tokens, which, as discussed, could be the same moment the tokens are placed on the market. Such a discrepancy in market capitalisation would be significant — in the most extreme case the naive market capitalisation could increase by a factor of two-thirds, an event which may cause investors to hastily re-evaluate their positions.

Recommendations

Ensure the `totalSupply()` function returns the amount of tokens sold during the crowd-sale, plus all tokens available for minting.

Resolution

No action was deemed necessary for this issue.

FTM-04	Unintended token burning due to invalidation of `_to` in transfer functions.		
Asset	FantomToken.sol		
Status	Closed: Resolved in commit [c5339bd]		
Rating	Severity: Low	Impact: Low	Likelihood: Medium

Description

The `_to` field in transfer events is not checked for the `0x0` address. External third-party applications which implement the ERC20 interface may interpret no-input in ERC20 fields as `0`. Thus users can quite easily inadvertently send tokens to the `0x0` address by forgetting to add a `_to` address in their third-party application. This is evident by the large number of tokens currently associated with the `0x0` address.

Recommendations

It is increasingly common for ERC20 tokens to include measures that ensure transfers to the `0x0` address are not possible. Validation of the `_to` field is recommended. This can be implemented in the `transfer()` and `transferFrom()` functions in the `ERC20Token` contract or in the analogous functions in the `Fantomtoken` contract (as the latter call the former).

FTM-05	ERC20 standard compliance.
Asset	FantomToken.sol
Status	Closed: Resolved in commit [211a517]
Rating	Informational

Description

This section details the compliance with the ERC20 Standard [1] and adds any additional ERC20-related notes. 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 `FantomToken` contract complies with the ERC20 token standard with the following discrepancies:

- The `decimals` variable in `FantomToken` is a `uint256` rather than the specified `uint8`.

It should also be noted, that the ERC20 implementation has a known vulnerability to front-running in the `approve()` function [3].

Recommendations

Modify the `decimals` type to comply with the standard.

Be aware of the front-running issues in `approve()`, potentially add extended approve functions which are not vulnerable to the front-running vulnerability for future third-party-applications. See the Open-Zeppelin [4] solution for an example.

FTM-06	Gas savings.
Asset	FantomToken.sol
Status	Closed
Rating	Informational

Description

This section is informational and describes gas savings that could be implemented in the contract. Action need not be taken.

- **Unnecessary Variable Initialisation -** Initializing a variable to its default value is unnecessary and expensive (an extra `5000 gas` for `storage` variables).
  - The `storage` variable, `uint tokensIssuedTotal` is explicitly initialised to zero on line [164].
    ✓ Resolved in commit [e41a1e3]
  - The `uint tokens_to_transfer` is explicitly initialised to zero on line [617].
    ✓ Resolved in commit [e41a1e3]
  - The `uint i` counter is explicitly initialised to zero on lines [233], [286], [445], [618] and [624].
    ✓ Resolved in commit [2e70f75]
- `uint8` **State Variable -** The EVM functions on 32 byte word sizes. It's typically more expensive to perform operations on types smaller than this.
  - The state variable `LOCK_SLOTS` is designated as a `uint8` on line [217]. Specifying this variable as a `uint` (equivalently, `uint256`) will save gas. Reading storage is also expensive. It would be cheaper to store `LOCK_SLOTS` in memory.
    ✓ Resolved in commit [e41a1e3]
- **Redundant** `require` **-** The statement `require(balances[msg.sender] >= _amount)` on line [177] is redundant. The use of `SafeMath` subtraction on line [178] ensures that this condition is satisfied and that failure to satisfy this condition also triggers a `revert`. Removing this `require` would save gas. Similar statements apply to line [191] and line [192]. (An analogous use of `require` appears in the Open Zepplin implementation because they use `assert` in their `SafeMath`, which consumes all transaction gas if the assert fails. However, the FantomToken implementation of SafeMath does not use `assert`.)
  ✓ Resolved in commit [e41a1e3]
- **Redundant Specification of Return Variable -** Adding a name to a return value initialises the variable in memory. Unused return variables waste gas. This is done on lines [172], [176], [184], [190], and [200].
  ✓ Resolved in commit [2e70f75]
- **Calling internal Functions Has Gas Overhead -** Calling functions internally uses more gas than simply executing the code within the function.

- `atNow()` is used extensively and can be replaced by `now` throughout the code to save gas. For testing, it is often convenient to set times in the constructor and shift the timestamp of the testing blockchain (i.e. ganache) during the tests (see this reports accompanying tests for an example).
  ✓ Resolved in commit [e41a1e3]
- `checkDateOrder()` is a function used on lines [311], [326] and [333]. This can be replaced by a modifier of the form:

```
modifier checkDateOrder {
  _;
  require(dateMainStart < dateMainEnd);
  require(dateMainEnd < DATE_LIMIT);
}
```

  ✓ Resolved in commit [e41a1e3]

- **Unnecessary loop during** `transferMultiple()` **-** The loop beginning on [618] is not required to detect insufficient sender balance. A revert will be caused by the first `transfer()` to exceed sender balance, negating all other transfers in that call.
  ✓ Resolved in commit [1eab44b]

FTM-07	Miscellaneous notes and comments.
Asset	FantomToken.sol
Status	Closed: All notes were acknowledged.
Rating	Informational

Description

This section details miscellaneous informational aspects found within the contract. Actions need not be taken, this is mainly for author's reference.

- `balanceOf` **Mapping ignores locked tokens -** Users will experience a discrepancy between the number of tokens they own (displayed by third party applications, i.e. mist, myetherwallet, etc), given by the `balanceOf()` function, and the number of tokens they can transfer. This discrepancy arises because `balanceOf` does not account for locked tokens. Thus, for example, a user may see a balance of '100 tokens' but performing a transfer of this many tokens will result in an unexpected 'revert'. The user has to explicitly lookup `unlockedTokens()` in order to find how many are transferable.
- **Ambiguous Event Index -** The `IcoDateUpdated` event defined on line [307] contains an `id` parameter. The `IcoDateUpdated` event is triggered on line [327] and line [334] but in both cases the value `id=1` is specified. The `id` does not appear to serve any purpose. ✓ Resolved in commit [e599a4f]
- **Clearer Number Representation -** The constants defined on lines [375] and [376] can be represented more clearly using scientific notation, i.e. `1000000000` = `1e9`.
- **Naming of Unit Variables Is Misleading -** The functions `ethToTokens()` and `tokensToEth()` have misleading names, including the parameter `_eth`. One would assume values of dimension `ether` are passed to these functions, when in fact, `wei` is being passed an `18` order of magnitude difference.
- **Decimal mathematics is only valid for** `decimals = 18` **-** The function `ethToTokens()` takes `wei` as a parameter and multiplies by `tokensPerEth`. This only retrieves the correct answer, because `weiPerEth` is `10e18` which gives the correct decimal places for `tokens` (i.e. `10e18`). If `decimals` is changed in the future to a value other than `18` the `ethToTokens()` and `tokensToEth()` functions will return incorrect results.
- **Fallback Uses More Than** `2300 gas` **-** By putting the `buyTokens()` function in the fallback, this restricts any contracts from sending ether to this contract via a `transfer()` call which has a stipend of `2300 gas`.
- **Gas Usage -** The deployment of this contract requires a decent amount of gas, namely ≈ `5M` (unoptimized) and 3M (optimized). The `registerLockedTokens()` function loops through and modifies storage variables and as such is quite expensive. A call to `pMintTokens()` costs around `250,000 gas` (optimized). Purchasing tokens costs around `150,000 gas` (optimized). See Test Suite for further gas estimations.

- **Mint Type -** There is a concept of "mint type", expressed through the `TokensMinted` event and the `balancesMintedByType` mapping. This concept is not documented (e.g., what are valid mint types and what do the integers reference?). Furthermore, storing an extra mapping will consume significant gas and a tally of mints-by-type could be generated by reading only the `mintType` parameter of the `TokensMinted` events. We also question the necessity of some flag which is controlled only by the `owner` and has no effect on the contract logic.
- **Inconsistent use of `p` prefix -** The `buyTokens()` function is `private`, yet it is not prefixed with a `p` like all other `private` functions. E.g., `pBuyTokens()`.

Recommendations

Ensure these are as expected.

Appendix A Test Suite

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

```
Contract: StandardToken
  total supply
    ✓ returns the total amount of tokens immediately after deployment
  balanceOf
    when the requested account has no tokens
      ✓ returns zero
    when the requested account has some tokens
      ✓ returns the total amount of tokens
  transfer
    when the recipient is not the zero address
      when the sender does not have enough balance
        ✓ reverts
      when the sender has enough balance
        ✓ transfers the requested amount (43ms)
        ✓ emits a transfer event (50ms)
    when the recipient is the zero address
      ✓ reverts
  approve
    when the spender is not the zero address
      when the sender has enough balance
        ✓ emits an approval event
        when there was no approved amount before
          ✓ approves the requested amount
        when the spender had an approved amount
          ✓ approves the requested amount and replaces the previous one
      when the sender does not have enough balance
        ✓ emits an approval event
        when there was no approved amount before
          ✓ approves the requested amount
        when the spender had an approved amount
          ✓ approves the requested amount and replaces the previous one
    when the spender is the zero address
      ✓ approves the requested amount
      ✓ emits an approval event
  transfer from
    when the recipient is not the zero address
      when the spender has enough approved balance
        when the owner has enough balance
          ✓ transfers the requested amount (46ms)
          ✓ decreases the spender allowance
          ✓ emits a transfer event
        when the owner does not have enough balance
          ✓ reverts
```

```
        when the spender does not have enough approved balance
          when the owner has enough balance
            ✓ reverts
          when the owner does not have enough balance
            ✓ reverts
      when the recipient is the zero address
        ✓ reverts

  Contract: LockSlots
    ✓ [isAvailableLockSlot] should return true for account no locked slot
   (496ms)
    ✓ [mintTokensLocked] should lock correct number of tokens (560ms)
    ✓ [mintTokensLockedMultiple] mint the correct amount of locked tokens
   (492ms)
    ✓ [mintTokensLockedMultiple] number of lockslots fill correctly (547ms)
    ✓ [mintTokens] should not lock the tokens (461ms)

  Contract: FantomICODates
    ✓ should not allow public to change dates (418ms)
    ✓ should allow owner to change the dates (554ms)
    ✓ should not allow owner to change dates into past (449ms)
    ✓ should not allow owner to set start date after or equal to end date
   (657ms)
    ✓ [mainsale] should detect main period (2785ms)

  Contract: Gas Consumption Tests (optimized-runs = 200)
Deployment Gas Estimate: 2822138
    ✓ Deployment of contract gas estimate (539ms)
Buy Tokens Gas Estimate: 158469
    ✓ should cost less than the block gas limit to buy tokens (optimize-runs
    = 200) (943ms)
Minting Locked Tokens Gas Estimate: 248405
    ✓ should cost less than the block gas limit to mint tokens (optimize-
   runs = 200) (642ms)
Minting locked tokens for 2 accounts.  Gas Estimate: 342063
    ✓ [MintTokensLockedMultiple] should cost less than the block gas limit
   for 2 accounts (684ms)
Minting locked tokens for 5 accounts.  Gas Estimate: 616483
    ✓ [MintTokensLockedMultiple] should cost less than the block gas limit
   for 5 accounts (800ms)
Minting locked tokens for 10 accounts.  Gas Estimate: 1073913
    ✓ [MintTokensLockedMultiple] should cost less than the block gas limit
   for 10 accounts (829ms)
Minting locked tokens for 15 accounts.  Gas Estimate: 1531281
    ✓ [MintTokensLockedMultiple] should cost less than the block gas limit
   for 15 accounts (1093ms)
Minting locked tokens for 20 accounts.  Gas Estimate: 1988586
    ✓ [MintTokensLockedMultiple] should cost less than the block gas limit
   for 20 accounts (1148ms)
```

```
Minting locked tokens for 30 accounts.  Gas Estimate: 2903326
    ✓ [MintTokensLockedMultiple] should cost less than the block gas limit
   for 30 accounts (1286ms)
Minting locked tokens for 50 accounts.  Gas Estimate: 4732688
    ✓ [MintTokensLockedMultiple] should cost less than the block gas limit
   for 50 accounts (1544ms)
Multiple transfer to 2 accounts.  Gas Estimate: 70343
    ✓ [TransferMultiple] should cost less than the block gas limit for 2
   accounts (888ms)
Multiple transfer to 5 accounts.  Gas Estimate: 118703
    ✓ [TransferMultiple] should cost less than the block gas limit for 5
   accounts (1008ms)
Multiple transfer to 10 accounts.  Gas Estimate: 199303
    ✓ [TransferMultiple] should cost less than the block gas limit for 10
   accounts (985ms)
Multiple transfer to 15 accounts.  Gas Estimate: 279903
    ✓ [TransferMultiple] should cost less than the block gas limit for 15
   accounts (1189ms)
Multiple transfer to 20 accounts.  Gas Estimate: 360439
    ✓ [TransferMultiple] should cost less than the block gas limit for 20
   accounts (1282ms)
Multiple transfer to 30 accounts.  Gas Estimate: 521575
    ✓ [TransferMultiple] should cost less than the block gas limit for 30
   accounts (1254ms)
Multiple transfer to 50 accounts.  Gas Estimate: 843783
    ✓ [TransferMultiple] should cost less than the block gas limit for 50
   accounts (1261ms)

  Contract: [FantomToken - Token Math]
    ✓ should have the correct caps (425ms)
    ✓ should have a correct token rate for 1 ether (865ms)
    ✓ should not give more tokens than allowed during first day (1238ms)
    ✓ a user should not be able to purchase more than the token limit in the
     first day (1098ms)
    ✓ should purchase correct amount of tokens if all whitelisted users
   purchase tokens (1319ms)
    ✓ [Scenario.1] should give the correct token amounts for scenario 1
   (1606ms)
    ✓ [Scenario 2] should give the correct token amounts for scenario 1
   (1644ms)

  Contract: Token Sale
    when the tokensale gets completed the first day
      ✓ reverts when there is no more tokens to buy and total cap is
   purchased (3047ms)
    when the tokensale gets completed after the first day
      ✓ reverts when there is no more tokens to buy and total cap is
   purchased (2636ms)
```

```
Contract: StandardToken
  transferMultiple
    when the recipient is not the zero address
      when the sender does not have enough balance
        ✓ reverts
      when the sender has enough balance
        ✓ transfers the requested amount (91ms)
        ✓ emits a transfer event
    when the recipient is the zero address
      ✓ reverts

Contract: StandardToken TransferMultiple Extended
  when tokens are not tradeable
    ✓ reverts
  when the sender has enough unlocked balance
    ✓ transfers the requested amount (152ms)
    ✓ emits a transfer event (77ms)
  when the sender doesn't have  enough unlocked balance
    ✓ reverts (65ms)

66 passing (1m)
```

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.

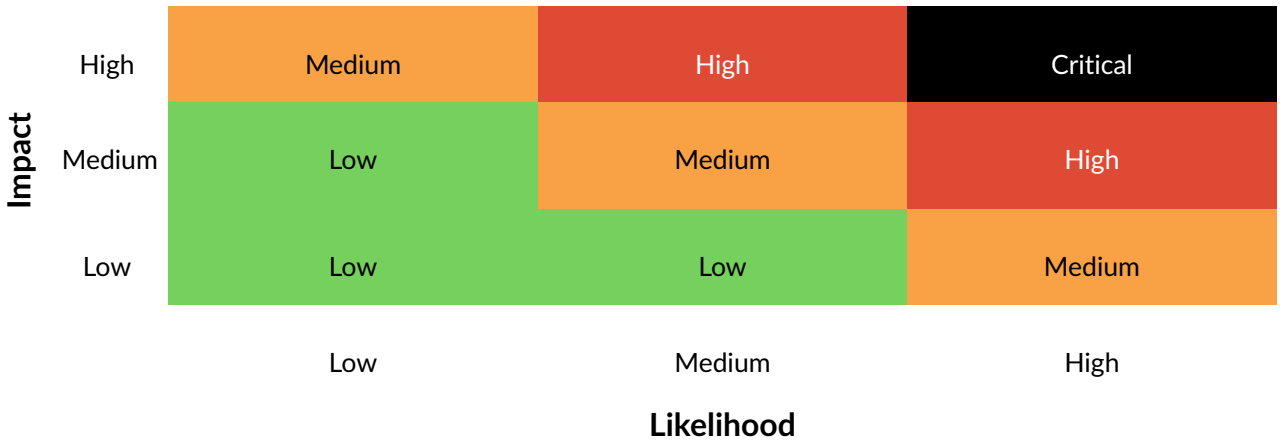

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

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

[2] FANTOM - Whitepaper v1.3. Website, May 2018, Available: http://www.fantom.foundation/data/FANTOM%20Whitepaper%20English%20v1.3.pdf.

[3] ERC20 API: An Attack Vector on Approve/TransferFrom Methods. Google Docs, 2018, Available: https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/edit#heading=h.m9fhqynw2xvt.

[4] OpenZeppelin StandardToken.sol. Github, 2018, Available: https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/token/ERC20/StandardToken.sol.