---

title: Token integration checklist description: A checklist of things to consider when interacting with tokens author: "Trailofbits" lang: en tags: ["solidity", "smart contracts", "security", "tokens"] skill: intermediate published: 2020-08-13 source: Building secure contracts sourceUrl: https://github.com/crytic/building-secure-contracts/blob/master/development-guidelines/token_integration.md

---

Follow this checklist when interacting with arbitrary tokens. Make sure you understand the risks associated with each item, and justify any exceptions to these rules.

For convenience, all Slither utilities can be run directly on a token address, such as:

Using Slither tutorial

```
bash slither-check-erc 0xdac17f958d2ee523a2206206994597c13d831ec7 TetherToken
```

To follow this checklist, you'll want to have this output from Slither for the token:

```
bash - slither-check-erc [target] [contractName] [optional: --erc ERC_NUMBER] - slither [target] --print human-summary - slither [target] --print contract-summary - slither-prop . --contract ContractName # requires configuration, and use of Echidna and Manticore
```

## General considerations {#general-considerations}

- **The contract has a security review.** Avoid interacting with contracts that lack a security review. Check the length of the assessment (aka "level of effort"), the reputation of the security firm, and the number and severity of the findings.
- **You have contacted the developers.** You may need to alert their team to an incident. Look for appropriate contacts on blockchain-security-contacts.
- **They have a security mailing list for critical announcements.** Their team should advise users (like you!) when critical issues are found or when upgrades occur.

## ERC conformity {#erc-conformity}

Slither includes a utility, slither-check-erc, that reviews the conformance of a token to many related ERC standards. Use slither-check-erc to review that:

- **Transfer and transferFrom return a boolean.** Several tokens do not return a boolean on these functions. As a result, their calls in the contract might fail.
- **The name, decimals, and symbol functions are present if used.** These functions are optional in the ERC20 standard and might not be present.
- **Decimals returns a uint8.** Several tokens incorrectly return a uint256. If this is the case, ensure the value returned is below 255.
- **The token mitigates the known ERC20 race condition.** The ERC20 standard has a known ERC20 race condition that must be mitigated to prevent attackers from stealing tokens.
- **The token is not an ERC777 token and has no external function call in transfer and transferFrom.** External calls in the transfer functions can lead to reentrancies.

Slither includes a utility, slither-prop, that generates unit tests and security properties that can discover many common ERC flaws. Use slither-prop to review that:

- **The contract passes all unit tests and security properties from slither-prop.** Run the generated unit tests, then check the properties with Echidna and Manticore.

Finally, there are certain characteristics that are difficult to identify automatically. Review for these conditions by hand:

- **Transfer and transferFrom should not take a fee.** Deflationary tokens can lead to unexpected behavior.
- **Potential interest earned from the token is taken into account.** Some tokens distribute interest to token holders. This interest might be trapped in the contract if not taken into account.

## Contract composition {#contract-composition}

- **The contract avoids unneeded complexity.** The token should be a simple contract; a token with complex code requires a higher standard of review. Use Slither’s human-summary printer to identify complex code.
- **The contract uses SafeMath.** Contracts that do not use SafeMath require a higher standard of review. Inspect the contract by hand for SafeMath usage.
- **The contract has only a few non–token-related functions.** Non–token-related functions increase the likelihood of an issue in the contract. Use Slither’s contract-summary printer to broadly review the code used in the contract.
- **The token only has one address.** Tokens with multiple entry points for balance updates can break internal bookkeeping based on the address (e.g. `balances[token_address][msg.sender]` might not reflect the actual balance).

## Owner privileges {#owner-privileges}

- **The token is not upgradeable.** Upgradeable contracts might change their rules over time. Use Slither’s human-summary printer to determine if the contract is upgradeable.
- **The owner has limited minting capabilities.** Malicious or compromised owners can abuse minting capabilities. Use Slither’s human-summary printer to review minting capabilities, and consider manually reviewing the code.
- **The token is not pausable.** Malicious or compromised owners can trap contracts relying on pausable tokens. Identify pauseable code by hand.
- **The owner cannot blacklist the contract.** Malicious or compromised owners can trap contracts relying on tokens with a blacklist. Identify blacklisting features by hand.
- **The team behind the token is known and can be held responsible for abuse.** Contracts with anonymous development teams, or that reside in legal shelters should require a higher standard of review.

## Token scarcity {#token-scarcity}

Reviews for issues of token scarcity requires manual review. Check for these conditions:

- **No user owns most of the supply.** If a few users own most of the tokens, they can influence operations based on the token's repartition.
- **The total supply is sufficient.** Tokens with a low total supply can be easily manipulated.
- **The tokens are located in more than a few exchanges.** If all the tokens are in one exchange, a compromise of the exchange can compromise the contract relying on the token.
- **Users understand the associated risks of large funds or flash loans.** Contracts relying on the token balance must carefully take in consideration attackers with large funds or attacks through flash loans.
- **The token does not allow flash minting**. Flash minting can lead to substantial swings in the balance and the total supply, which neccessitate strict and comprehensive overflow checks in the operation of the token.