General Recommendations

When developing software, ensuring it functions as intended is usually straightforward. However, preventing unintended usage and vulnerabilities can be more challenging.

In smart contract development, security is very important. A single error can result in the loss of valuable assets or the improper functioning of certain features.

Smart contracts are executed in a public environment where anyone can examine the code and interact with it. Any errors or vulnerabilities in the code can be exploited by malicious actors.

This chapter presents general recommendations for writing secure smart contracts. By incorporating these concepts during development, you can create robust and reliable smart contracts. This reduces the chance of unexpected behavior or vulnerabilities.

Disclaimer

This chapter does not provide an exhaustive list of all possible security issues, and it does not guarantee that your contracts will be completely secure.

If you are developing smart contracts for production use, it is highly recommended to conduct external audits performed by security experts.

Mindset

Cairo is a highly safe language inspired by Rust. It is designed in a way that forces you to cover all possible cases. Security issues on Starknet mostly arise from the way smart contract flows are designed, not much from the language itself.

Adopting a security mindset is the initial step in writing secure smart contracts. Try to always consider all possible scenarios when writing code.

Viewing Smart Contracts as Finite State Machines

Transactions in smart contracts are atomic, meaning they either succeed or fail without making any changes.

Think of smart contracts as state machines: they have a set of initial states defined by the constructor constraints, and external functions represent a set of possible state transitions. A transaction is nothing more than a state transition.

The `assert!` or `panic!` macros can be used to validate conditions before performing specific actions. You can learn more about these on the [Unrecoverable Errors with panic](./ch09-01-unrecoverable-errors-with-panic.md) page.

These validations can include:

- Inputs provided by the caller
- Execution requirements
- Invariants (conditions that must always be true)
- Return values from other function calls

For example, you could use the `assert!` macro to validate that a user has enough funds to perform a withdraw transaction. If the condition is not met, the transaction will fail and the state of the contract will not change.

```cairo,noplayground
{{#include ../listings/ch17-starknet-smart-contracts-security/
no_listing_01_assert_balance/src/lib.cairo:withdraw}}
```

Using these functions to check conditions adds constraints that help clearly define the boundaries of possible state transitions for each function in your smart contract. These checks ensure that the behavior of the contract stays within the expected limits.

Recommendations

Checks Effects Interactions Pattern

The Checks Effects Interactions pattern is a common design pattern used to prevent reentrancy attacks on Ethereum. While reentrancy is harder to achieve in Starknet, it is still recommended to use this pattern in your smart contracts.

<!-- TODO add reference to the reentrancy CairoByExample page -->

The pattern consists of following a specific order of operations in your functions:

1. **Checks**: Validate all conditions and inputs before performing any state changes.
2. **Effects**: Perform all state changes.
3. **Interactions**: All external calls to other contracts should be made at the end of the function.

Access Control

Access control is the process of restricting access to certain features or resources. It is a common security mechanism used to prevent unauthorized access to sensitive information or actions. In smart contracts, some functions may often be restricted to specific users or roles.

You can implement the access control pattern to easily manage permissions. This pattern consists of defining a set of roles and assigning them to specific users. Each function can then be restricted to specific roles.

```cairo,noplayground
{{#include ../listings/ch17-starknet-smart-contracts-security/
no_listing_02_simple_access_control/src/lib.cairo}}
```