Terra

Security Assessment

Sep 4, 2020

For :

Daniel Hong @ Terra

sh@terra.money

Yun Yeo @ Terra

yun@terra.money

By :

Georgios Delkos @ CertiK

georgios.delkos@certik.io

PK: a7c24a98cc68296483251ec87cd8d7a7376e04bd61f68e2ccdda70b8714a8d7a

Camden Smallwood @ CertiK

camden.smallwood@certik.io

Jay Jie @ CertiK

jay.jie@certik.io

Table Of Contents

Executive Summary

Intro

Terra has developed a smart contract solution based on CosmWasm, which is a WebAssembly smart contract system based on the Cosmos SDK and Tendermint BFT consensus.

The main goal of CosmWasm is to provide functionality for creating smart contracts that can be deployed and interact with other smart contracts on different blockchain platforms. For the moment, contracts can be only written in Rust, but more programming languages are currently being looked into for future integration. CosmWasm takes advantage of the Actor model to communicate through messages, which has the advantage of fully encapsulating state and removes classes of bugs such as the infamous Solidity re-entrancy attack.

Terra's CosmWasm integration in the Terra Core repository mostly falls in line with the reference implementation from CosmWasm, which is called Wasmd. Using the Wasmd repository's issues as a source of investigation for existing WebAssembly integration bugs proved to turn up positive and nothing was found in either the Wasmd or Terra Core codebases that presents itself as a direct compromise to the proper functioning of the WebAssembly in the CosmWasm virtual machine or the functionality of standard interactions between smart contracts and the system.

Engagement

CertiK auditing team reviewed the code base of Terra from 10.08.2020 through 28.08.2020. The team conducted a 15 days assessment with 3 engineers working on the repository https://github.com/terra-project/core and more specifically in releases/tag/v0.4.0-rc.2.

The team launched the audit by analyzing the specifications of the project and the key areas of interest, went through the unit testing of the code, and launched fuzzing against targets in the codebase identified in the process, and on cosmo wasm and Terra wasm endpoints.
Fuzzing was conducted in parallel with the manual review and had a 10 day lifespan.
Moving on, the team passed the code through automated tooling, gathered all the output and manually reviewed each one of the issues that were returned from the tooling.
The main process of the audit was the manual review of the key areas of interest and was divided into 3 parts, the language-specific, SDK, and WASM examination of the codebase and target in scope.
The team manually reviewed the codebase written in golang and rust for language-specific problems and proper use of the language itself. In parallel, the team also examined the usage and proper implementation of the Cosmos SDK. Finally, the team reviewed the WASM implementation and targets generated by the codebase in local testnet and latest testnet.

Findings

There have been no major or critical issues related to the codebase and all findings listed here are minor and informational.
The most prominent among our findings were a uint64 underflow that gets wrapped around due to the nature of the type in golang(ring) and the usage of pointers in the codebase that would need a more secure approach to ensure maintainability and code health in the future.

Overview

Overall, the audit has found that the Terra team has done a good job implementing the specifications of the project into code. The usage of language is of a very high standard with good code coverage on unit testing. The SDK specifics are also well-implemented concerning the requirements of the framework and the same applies to the Cosmos WASM implementation. Finally, the audit did all the necessary recommendations to the Terra team, and issues were discussed and addressed.

Project Overview

Project Summary

Projects Name	Terra
Codebase Version	v0.4.0-rc.2
Type	Security Audit
Platform/Framework	Cosmos SDK/WASM

Audit Summary

Date	01.09.2020
Method of audit	Automated and Manual Review
Engineers	3
Timeline	15 days

Vulnerability Summary

Total Critical Issues	0
Total Major Issues	0
Total Minor Issues	2
Total Informational Issues	7
Total Issues	9

Engagement Goals

The engagement was to provide a security review of the Terra updates on the codebase in the core 0.4 rc2 repository for the transition to the new testnet with smart contract implementation based on CosmWASM.

More specifically, the audit was focused on the following areas of interest :

- Msg Authentication.
- Burn Address.
- Msg Swap Send.
- CosmWasm Implementation.
- Wasm Bindings.
- WASM target security evaluation.

Coverage

The audit has focused on the following components of the implementation:

Steps Taken:

- Investigated the official Terra Docs for smart contract composition and interaction:
    - https://docs.terra.money/dapps/tutorial/implementation.html
    - https://docs.terra.money/dapps/tutorial/interacting.html
- Decompiled supplied testing WASM smart contract modules using wabt and wasmdec.
- Verified all imported functions in WASM smart contract modules are only ever valid CosmWasm/Terra functions:
    - db_read
    - db_write
    - db_remove
    - db_scan
    - db_next
    - canonicalize_address
    - humanize_address
    - query_chain

- Verified all required CosmWasm/Terra functions were exported from all WASM smart contract modules:
    - init
    - handle
    - query
- Instrumented supplied testing WASM smart contract modules with instruction hooks via wasabi and attempted to submit to the network, which were accepted but failed to execute correctly.
- Simulated contract interactions based on core unit testing using modified versions of WASM virtual machines wasmer, wasmi and rust-wasm for further inspection of contract message data and changes to internal state.
- Ran code coverage and data flow on supplied testing WASM smart contract modules using twiggy to determine overall memory usage and possibility of sensitive data.
- Built custom WASM smart contract modules in Rust following the official Terra Docs, which were then deployed to instances of localterra and tequila-0002 and tested for valid interaction functionality:
    - Instantiation
    - Initialization
    - Message Handling
    - Querying

Additionally the team performed the following checks:

General

- What are the extensions made on top of the core CosmWasm features.
- Can any possible contract interactions crash or deadlock the system in any way.
- Can malformed contracts be removed or resolved without affecting the running system in any way.

Code Coverage

- When a function is executed, i.e: init, how many other functions execute.
- How many instructions in those functions execute.
- What are the possible code paths and how many of those paths execute for a given input.
- How much code is never executed at all.

Data Flow

- What's the total amount of linear memory allocated.
- How much of that linear memory is statically initialized with a data instruction.
- Where are the spans of those data instructions initializing (offset, length).
- When a function is executed, how much linear memory is accessed from within that function.
- Which instructions in an executed function access which parts of linear memory.

- For all memory initialized with a data expression, how much never gets touched by any executed instruction.
- Is any secure information located in any of the statically allocated linear memory.
- Is any secure information stored in linear memory after executing an instruction without clearing the sensitive data out of memory afterwards.

Recommendations Summary

÷ Recommendations

The recommendations expressed by the audit were mostly regarding the usage of pointers within the codebase. More specifically we recommended that when assigning a pointer to a variable is always a good idea to check if that pointer is nil within the functionality that the variable lives and not somewhere else outside the functionality where something can change in the future and we get introduced a nasty bug out of nowhere.

Findings Summary

#Nr	Title	Type	Severity
TR-001	Integer Underflow	Arithmetic	Minor
TR-002	Pointer Usage	Secure Design	Minor
TR-003	Code Structure	SDK Usage	Info
TR-004	Validity Check	SDK Usage	Info
TR-005	Interface Implementation	SDK Usage	Info
TR-006	Typo	Readability	Info
TR-007	Validation	SDK Usage	Info
TR-008	Redundant Code	Language	Info
TR-009	String Usage	Language	Info

All issues have been addressed by the Terra team.

1. Integer Underflow

Severity: Minor
Type: Arithmetic
File: x/wasm/internal/keeper/api.go
Finding ID: TR-001

Description
In any case that meter.Limit() < of meter.GasConsumed() the value will be a negative number resulting in an integer underflow as the target var is a uint64. The go compiler here will wrap around the value as the uint64 type is a ring type resulting to the max value of a uint64 (18446744073709551615) minus the second of the two numbers (in this case meter.GasConsumed()).

GoPlayground Example

```
// return remaining gas in wasm gas unit
func (k Keeper) getGasRemaining(ctx sdk.Context) uint64 {
	meter := ctx.GasMeter()

	remaining := (meter.Limit() - meter.GasConsumed())

	if maxGas := k.MaxContractGas(ctx); remaining > maxGas {
		remaining = maxGas
	}
	return remaining * types.GasMultiplier
}
```

Exploit Scenario
N/A

Recommendation
N/A

2. Pointer Usage

Severity: Minor
Type: Memory Access
File: x/bank/wasm/interface.go
Finding ID: TR-002

Description
The code assigns a pointer to a variable without checking if the pointer is nil.
Exploit Scenario
N/A

Recommendation
Even if we are 100% percent sure that the pointer will never be nil it is highly recommended that we always check against that inside the assignment of the variable for maintainability and code health issues.

3. Code Structure

Severity: Informational
Type: SDK Usage
File: x/msgauth/client/rest/tx.go
Finding ID: TR-003

Description
As outlined in the Cosmos SDK documentation, it is recommended to have the request types implemented in x/module/client/rest/rest.go. The request types currently reside in x/msgauth/client/rest/tx.go, mixed in with the request handlers.

```
// GrantRequest defines the properties of a grant request's body.
type GrantRequest struct {
	BaseReq rest.BaseReq  `json:"base_req" yaml:"base_req"`
	Period  time.Duration `json:"period"`
	Limit   sdk.Coins     `json:"limit,omitempty"`
}

// RevokeRequest defines the properties of a revoke request's body.
type RevokeRequest struct {
	BaseReq rest.BaseReq `json:"base_req" yaml:"base_req"`
}

// ExecuteRequest defines the properties of a execute request's body.
type ExecuteRequest struct {
	BaseReq rest.BaseReq `json:"base_req" yaml:"base_req"`
	Msgs    []sdk.Msg    `json:"msgs" yaml:"msgs"`
}
```

Exploit Scenario
N/A

Recommendation
Move the structs to the proper location.

4. Validity Check

Severity: Informational
Type: SDK Usage
File: x/msgauth/internal/types/genesis.go
Finding ID: TR-004

Description
Validate Genesis not implemented on module msg auth.

```
// ValidateGenesis check the given genesis state has no integrity issues
func ValidateGenesis(data GenesisState) error {
	return nil
}
```

Exploit Scenario
N/A

Recommendation
Implement the validation.

```
// ValidateGenesis returns nil because accounts are validated by auth
func ValidateGenesis(data GenesisState) error {
	if data.PreviousBlockTime.IsZero() {
		return errors.New("previous block time cannot be zero")
	}
	return nil
}
```

5. Interface Implementation

Severity: Informational
Type: SDK Usage
File: x/msgauth/module.go
Finding ID: TR-005

Description
Incomplete module implementation.
Exploit Scenario
N/A

Recommendation
Fully implement the interface.

```
// Name module name
func (AppModule) Name() string {
	return ModuleName
}
```

6. Typo

Severity: Informational
Type: Readability
File: x/msgauth/internal/keeper/keeper.go
Finding ID: TR-006

Description
Typo in variable name.
Exploit Scenario
N/A

Recommendation
Variable ggmParis to ggmPairs.

7. Validation

Severity: Informational
Type: SDK Usage
File: x/market/internal/types/msgs.go
Finding ID: TR-007

Description
Checking the empty address.

```
// ValidateBasic Implements Msg
func (msg MsgSwapSend) ValidateBasic() error {
	if len(msg.FromAddress) == 0 {
		return sdkerrors.ErrInvalidAddress
	}

	if len(msg.ToAddress) == 0 {
		return sdkerrors.ErrInvalidAddress
	}

	if msg.OfferCoin.Amount.LTE(sdk.ZeroInt()) ||
msg.OfferCoin.Amount.BigInt().BitLen() > 100 {
		return sdkerrors.Wrap(ErrInvalidOfferCoin,
msg.OfferCoin.Amount.String())
	}

	if msg.OfferCoin.Denom == msg.AskDenom {
		return sdkerrors.Wrap(ErrRecursiveSwap, msg.AskDenom)
	}

	return nil
}
```

Exploit Scenario
N/A

Recommendation

Use the SDK Empty() function.

```
// ValidateBasic Implements Msg
func (msg MsgSwapSend) ValidateBasic() error {
	if msg.FromAddress.Empty() {
		return sdkerrors.ErrInvalidAddress
	}

	if msg.ToAddress.Empty() {
		return sdkerrors.ErrInvalidAddress
	}

	if msg.OfferCoin.Amount.LTE(sdk.ZeroInt()) ||
msg.OfferCoin.Amount.BigInt().BitLen() > 100 {
		return sdkerrors.Wrap(ErrInvalidOfferCoin,
msg.OfferCoin.Amount.String())
	}

	if msg.OfferCoin.Denom == msg.AskDenom {
		return sdkerrors.Wrap(ErrRecursiveSwap, msg.AskDenom)
	}

	return nil
}
```

8. Redundant Code

Severity: Informational
Type: Language Usage
File: x/wasm/config/client/utils/utils.go
Finding ID: TR-008

Description
Redundant len().

```
//EncodeKey encode given key with prefix of key's length
func EncodeKey(key string) []byte {
	keyBz := make([]byte, 2, 2+len(key))

	keyLength := uint64(len(key))
	bz := make([]byte, 8)
	binary.LittleEndian.PutUint64(bz, keyLength)

	keyBz[0] = bz[1]
	keyBz[1] = bz[0]

	keyBz = append(keyBz, []byte(key)...)

	return keyBz
}
```

Exploit Scenario
N/A

Recommendation
Start the function with getting the key length as a variable and pass it on accordingly

```
//EncodeKey encode given key with prefix of key's length
func EncodeKey(key string) []byte {
    keyLength := uint64(len(key))
    keyBz := make([]byte, 2, 2+keyLength)
    bz := make([]byte, 8)
    binary.LittleEndian.PutUint64(bz, keyLength)
    keyBz[0] = bz[1]
    keyBz[1] = bz[0]
    keyBz = append(keyBz, []byte(key)...)
    return keyBz
}
```

9. String Usage

Severity: Informational
Type: Language Usage
File: terra-cosmwasm-bindings
Finding ID: TR-009

Description
In the interest of optimization, it may be worthwhile to consider refactoring the route string field in the TerraMsgWrapper and TerraQueryWrapper structures into an enum type, which could then be converted to a string if/when necessary.
Exploit Scenario
N/A

Recommendation

```
#[derive(Clone, Copy, Debug, PartialEq, PartialOrd, Eq, Ord, Hash)]
pub enum TerraRoute {
    Market,
    Treasury,
    // etc...
}

impl ToString for TerraRoute {
    fn to_string(&self) -> String {
        match *self {
            Self::Market => "market".into(),
            Self::Treasury => "treasury".into(),
            // etc...
        }
    }
}
```

So we can essentially do :

```
// from:
pub route: String,
// into:
pub route: TerraRoute,

// from:
route: "market".to_string(),
// into:
route: TerraRoute::Market,

// from:
route: "treasury".to_string(),
```

```
// into:
route: TerraRoute::Treasury,
```