CredShields
Comdex Audit

Aug 5th, 2022 • CONFIDENTIAL

Description

This document details the process and result of the Comdex audit performed by CredShields Technologies PTE. LTD. on behalf of Comdex between July 6th, 2022, and Aug 2nd, 2022 and a retest was performed on 5th Sept, 2022.

Author

Shashank (Co-founder, CredShields)

shashank@CredShields.com

Reviewers

Aditya Dixit (Security Team Lead)

aditya@CredShields.com

Prepared for

Comdex

Table of Contents

1. Executive Summary

Comdex engaged CredShields to perform the Comdex audit from July 6th, 2022, to Aug 2nd, 2022. During this timeframe, six (6) vulnerabilities were identified. **A retest was performed on 5th September 2022 and all bug have been resolved and retested by Credshields team.**

During the audit, two (2) vulnerabilities were found that had a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "Comdex" and should be prioritized for remediation. As of 5th Sept 2022 all the bugs have been resolved.

The table below shows the in-scope assets and breakdown of findings by severity per asset. Section 2.3 contains more information on how severity is calculated.

Assets in Scope	Critical	High	Medium	Low	info	Σ
Comdex	0	2	0	3	1	**6**
	0	**2**	**0**	**3**	**1**	**6**

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in Comdex's scope during the testing window while abiding by the policies set forth by the "Comdex" team.

State of Security

Maintaining a healthy security posture requires constant review and refinement of existing security processes. Running a CredShields continuous audit allows “Comdex” internal security team and development team to not only uncover specific vulnerabilities but gain a better understanding of the current security threat landscape.

We recommend running regular security assessments to identify any vulnerabilities introduced after Comdexintroduces new features or refactored the code.

Reviewing the remaining resolved reports for a root cause analysis can further educate “Comdex” internal development and security teams and allow manual or automated procedures to be put in place to eliminate entire classes of vulnerabilities in the future. This proactive approach helps contribute to future-proofing the security posture of Comdex assets.

2. Methodology

Comdex engaged CredShields to perform the Comdex audit. The following sections cover how the engagement was put together and executed.

2.1 Preparation phase

CredShields team read all the provided documents and comments in the Comdex code to understand the application's features and functionalities. The team reviewed all the functions and prepared a mind map to review for possible security vulnerabilities in the order of the function with more critical and business-sensitive functionalities for the refactored code.

The team deployed a self-hosted version of the Comdex app and nodes to verify the assumptions and validation of the vulnerabilities during the audit phase.

A testing window from July 6th, 2022, to Aug 2nd, 2022, was agreed upon during the preparation phase and retesting was performed on 5th September 2022

2.1.1 Scope

During the preparation phase, the following scopes for the engagement was agreed-upon and was performed in two stages:

IN SCOPE ASSETS
https://github.com/comdex-official/comdex/tree/rc1.0 https://github.com/comdex-official/comdex/tree/audit-stage2

Table: In Scope Assets

2.1.2 Documentation

The following documentations were available to the audit team.

- https://github.com/comdex-official/comdex/blob/rc1.0/README.md
- https://docs.comdex.one
- https://github.com/comdex-official/comdex/blob/audit-stage2/README.md
- Comdex Audit Technical Document.pdf
- Steps_for_adding_app.docx
- https://github.com/comdex-official/technical-supporting-docs/blob/main/comdex-generic-documents/Comdex-custom-modules-diagram.svg

2.1.3 Audit Goals

CredShields' methodology uses individual tools and methods; however, tools are just used for aids. The majority of the audit methods involve manually reviewing the source code. The team followed the standards of the OWASP Application Security Verification Standards for testing. The team focused heavily on understanding the core concept behind all the

functionalities along with preparing test and edge cases. Understanding the business logic and how it could have been exploited.

The audit's focus was to verify that the application is secure, resilient, and working according to its specifications. Breaking the audit activities into the following three categories:

- **Security** - Identifying security-related issues within each contract and the system of contracts.
- **Sound Architecture** - Evaluation of the architecture of this system through the lens of established best practices and general software best practices.
- **Code Correctness and Quality** - A full review of the contract source code. The primary areas of focus include:
    - Correctness
    - Readability
    - Sections of code with high complexity
    - Improving scalability
    - Quantity and quality of test coverage

2.2 Retesting phase

Comdex is actively partnering with CredShields to validate the remediations implemented towards the discovered vulnerabilities.

2.3 Vulnerability classification and severity

Discovering vulnerabilities is important, but estimating the associated risk to the business is just as important.

To adhere to industry guidelines, CredShields follows OWASP's Risk Rating Methodology. This is calculated using two factors - **Likelihood** and **Impact**. Each of these parameters can take three values - **Low**, **Medium**, and **High**.

These depend upon multiple factors such as Threat agents, Vulnerability factors (Ease of discovery and exploitation, etc.), and Technical and Business Impacts. The likelihood and the impact estimate are put together to calculate the overall severity of the risk.

CredShields also define an **Informational** severity level for vulnerabilities that do not align with any of the severity categories and usually have the lowest risk involved.

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

Overall, the categories can be defined as described below -

1. **Informational**

We believe in the importance of technical excellence and pay a great deal of attention to its details. Our coding guidelines, practices, and standards help ensure that our software is stable and reliable.

Informational vulnerabilities should not be a cause for alarm but rather a chance to improve the quality of the codebase by emphasizing readability and good practices.

They do not represent a direct risk to the Contract but rather suggest improvements and the best practices that can not be categorized under any of the other severity categories.

Code maintainers should use their own judgment as to whether to address such issues.

2. Low

Vulnerabilities in this category represent a low risk to the application and the organization. The risk is either relatively small and could not be exploited on a recurring basis, or a risk that the client indicates is not important or significant, given the client's business circumstances.

3. Medium

Medium severity issues are those that are usually introduced due to weak or erroneous logic in the code.

These issues may lead to exfiltration or modification of some of the private information belonging to the end-user, and exploitation would be detrimental to the client’s reputation under certain unexpected circumstances or conditions. These conditions are outside the control of the adversary.

These issues should eventually be fixed under a certain timeframe and remediation cycle.

4. High

High severity vulnerabilities represent a greater risk to the application and the organization. These vulnerabilities may lead to a limited loss of funds for some of the end-users.

They may or may not require external conditions to be met, or these conditions may be manipulated by the attacker, but the complexity of exploitation will be higher.

These vulnerabilities, when exploited, will impact the client’s reputation negatively.

They should be fixed immediately.

5. Critical

Critical issues are directly exploitable bugs or security vulnerabilities. These issues do not require any external conditions to be met.

The majority of vulnerabilities of this type involve a loss of funds from the application and/or from their end-users.

The issue puts the vast majority of, or large numbers of, users' sensitive information at risk of modification or compromise.

The client's reputation will suffer a severe blow, or there will be serious financial repercussions.

2.4 CredShields staff

The following individual at CredShields managed this engagement and produced this report:

- **Shashank, Co-founder CredShields**
    - shashank@CredShields.com

Please feel free to contact this individual with any questions or concerns you have around the engagement or this document.

3. Findings

This chapter contains the results of the security assessment. Findings are sorted by their severity and grouped by the asset and CWE classification. Each asset section will include a summary. The table in the executive summary contains the total number of identified security vulnerabilities per asset per risk indication.

3.1 Findings Overview

3.1.1 Vulnerability Summary

During the security assessment, Six (6) security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC | Vulnerability Type
Go-Yaml v3 Deserialization (CVE-2022-28948)	High	Deserialization of Untrusted Data - CWE-502
Arbitrary File Read	Low	Missing Input Validation - CWE 20
Dead Code	Informational	Dead Code - CWE-561
IDN Homograph Attack	High	Insufficient Visual Distinction of Homoglyphs Presented to User - CWE-1007
Missing Character Length Validation	Low	Missing Character Length Validation - CWE 20

Missing Input Validation on Oracle Price	Low	Missing Input Validation - CWE 20

Table: Findings in Comdex

3.1.2 Findings Summary

The team performed both source code reviews and tested the application using fuzzing methods after making a local deployment. The application was vulnerable to any multiple input validation-related issues.

Since most of the features do not interact with an external user, the impact of these input validations was found to be low except for one that allowed malicious users to create pairs or assets with Cyrillic letters to impersonate other assets.

The team also found outdated packages and software versions. It is recommended to update all third-party libraries to improve the application's overall security.

4. Remediation Status

Comdex is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. The table shows the remediation status of each finding. The retesting was performed on the commit hash mentioned below.

d1872b477944eca3372076d4f527415183197e07

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Go-Yaml v3 Deserialization (CVE-2022-28948)	High	***Fixed [05/09/2022]***
Arbitrary File Read	Low	***Fixed [05/09/2022]***
Dead Code	Informational	***Fixed [05/09/2022]***
IDN Homograph Attack	High	***Fixed [05/09/2022]***
Missing Character Length Validation	Low	***Fixed [05/09/2022]***
Missing Input Validation on Oracle Price	Low	***Fixed [05/09/2022]***

Table: Summary of findings and status of remediation

5. Bug Reports

Bug ID#1 [Fixed]

Go-Yaml v3 Deserialization (CVE-2022-28948)

Vulnerability Type
Deserialization of Untrusted Data - CWE-502

Severity
High

Description
The code was found to be using an outdated version of the library Go-Yaml v3.
An issue in the Unmarshal function in Go-Yaml v3 causes the program to crash when attempting to deserialize invalid input.
The application deserializes untrusted data without sufficiently verifying that the resulting data will be valid.
https://nvd.nist.gov/vuln/detail/CVE-2022-28948

Affected Code
- https://github.com/comdex-official/comdex/blob/rc1.0/go.mod#L257

Impacts
This vulnerability may be exploited to execute system commands through the application and can be abused to generate shells however the likelihood of exploitation is low.

Remediation
- Make fields transient to protect them from deserialization.
- An attempt to serialize and then deserialize a class containing transient fields will result in NULLs where the transient data should be. This is an excellent way to

prevent time, environment-based, or sensitive variables from being carried over and used improperly.

- This vulnerability is remediated in the Go-Yaml package version v3.0.1. The package should be updated to its latest version. (Recommended)

Retest:
This is fixed. The package has been updated to 3.0.1.

Bug ID#2 [Fixed]

Arbitrary File Read

Vulnerability Type
Missing Input Validation

Severity
Low

Description
The "parseAssetMappingFlags()" function on Line 33 inside "/comdex/x/asset/client/cli/parse.go". This function is taking an argument called "addAssetMappingFile" which is then used on Line 41 without any input validation and then it is reading a file using "ReadFile" command which is dangerous without input validation.

Affected Codes

- https://github.com/comdex-official/comdex/blob/rc1.0/x/asset/client/cli/parse.go#L41

```
if addAssetMappingFile == "" {
        return nil, fmt.Errorf("must pass in add asset mapping json using
the --%s flag", FlagAddAssetMappingFile)
    }

    contents, err := ioutil.ReadFile(addAssetMappingFile)
```

- /comdex/x/lend/client/cli/parse.go:78
  https://github.com/comdex-official/comdex/blob/rc1.0/x/lend/client/cli/parse.go#L78

```
    if addLendPairsParamsFile == "" {
          return nil, fmt.Errorf("must pass in a add new lend pairs json
using the --%s flag", FlagNewLendPairFile)
```

```
    }

    contents, err := ioutil.ReadFile(addLendPairsParamsFile)
```

- /comdex/x/lend/client/cli/parse.go:100
  https://github.com/comdex-official/comdex/blob/rc1.0/x/lend/client/cli/parse.go#L100

```
    if addPoolParamsParamsFile == "" {
        return nil, fmt.Errorf("must pass in a add new pool json using the
--%s flag", FlagAddLendPoolFile)
    }

    contents, err := ioutil.ReadFile(addPoolParamsParamsFile)
```

- /comdex/x/lend/client/cli/parse.go:122
  https://github.com/comdex-official/comdex/blob/rc1.0/x/lend/client/cli/parse.go#L122

```
    if addAssetRatesStatsFile == "" {
        return nil, fmt.Errorf("must pass in a add asset rates stats json
using the --%s flag", FlagAddAssetRatesStatsFile)
    }

    contents, err := ioutil.ReadFile(addAssetRatesStatsFile)
```

Impacts

Missing input validation on arguments that go inside ioutil.ReadFile commands may lead to disastrous effects as these may be used to read system files using the privileges with which the application binary is executing. If an arbitrary file path is passed in the argument, there may be a chance to read files in different locations by traversing the directories.

Reference:
https://blog.credshields.com/the-avalanche-blockchain-bug-to-halt-the-chain-4869a19acf7e

Remediation

Implement an input validation on the function arguments that takes the file name as a user-supplied input.

Retest:

A regex match has been added to avoid malicious input and directory traversing.
https://github.com/comdex-official/comdex/blob/d1872b477944eca3372076d4f527415183197e07/x/asset/keeper/asset.go#L198

Bug ID#3 [Fixed]

Dead Code

Vulnerability Type
Dead Code - CWE 561

Severity
Informational

Description
The following code or variables were declared but it was never used anywhere in the code.

Affected Code

- x/esm/client/cli/tx.go:23
  https://github.com/comdex-official/comdex/blob/rc1.0/x/esm/client/cli/tx.go#L23
- x/esm/client/cli/tx.go:24
  https://github.com/comdex-official/comdex/blob/rc1.0/x/esm/client/cli/tx.go#L24

```
const (
   flagPacketTimeoutTimestamp = "packet-timeout-timestamp"
   listSeparator              = ","
)
```

Impacts
This is not a concern but a missing security best practice.

Remediation
Failure to use variables and parameters shows missing or dead code. Remove the variables if they are not being used.

Retest:

This is fixed. Dead code has been removed.

Bug ID#4 [Fixed]

IDN Homograph Attack

Vulnerability Type
Insufficient Visual Distinction of Homoglyphs Presented to User - CWE-1007

Severity
High

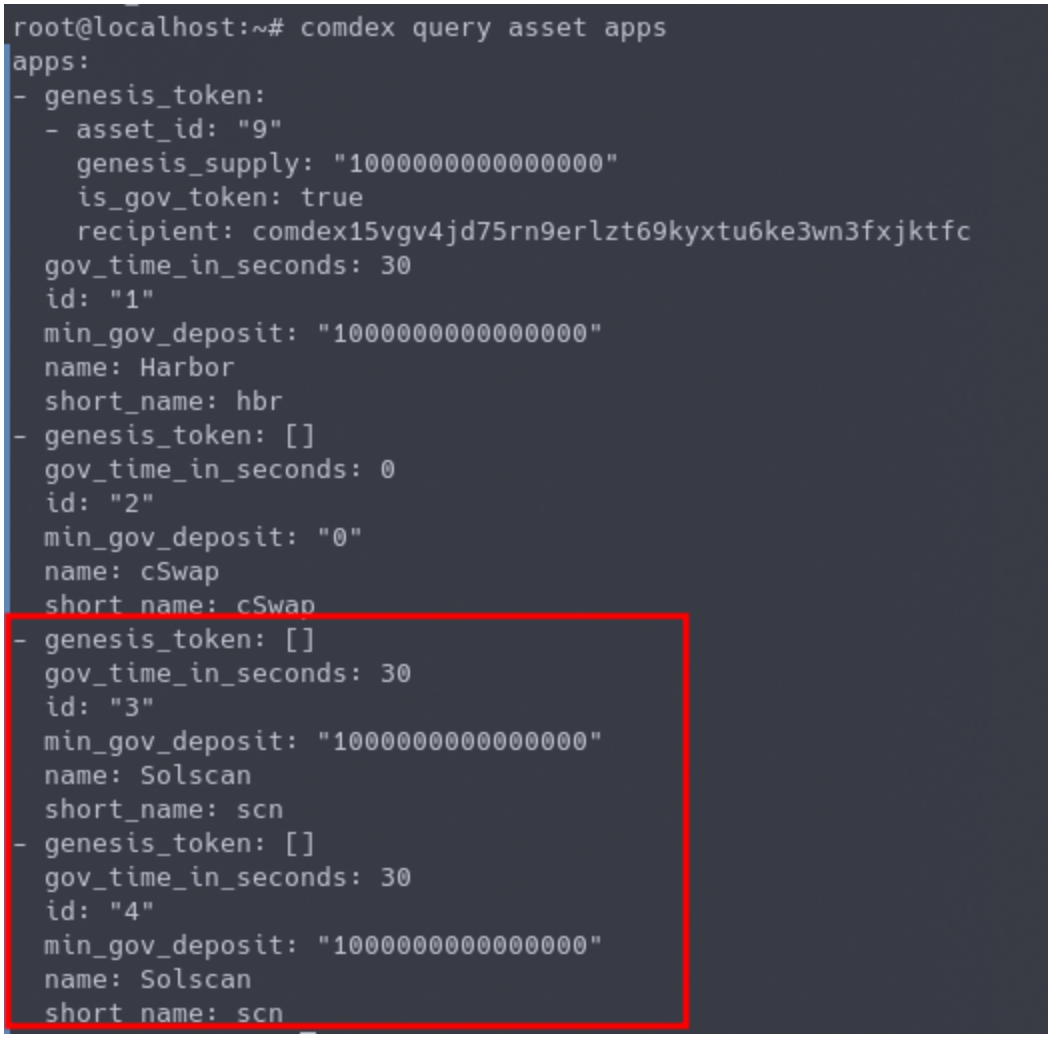
Description
The IDN (Internalized Domain Name) homograph attack, also known by the names "homoglyph" and "script spoofing," is a method in which an attacker deceives victims by making them believe that the site they are visiting is a genuine one.
The same principle can be applied to the blockchain ecosystem by making use of lookalike Unicode characters. Websites such as the Homoglyph Attack Generator can be used to generate payloads for the same.
This can be exploited to create apps and assets with similar-looking names, that can be abused to spoof actual apps and assets.

PoC

- Use the following command to create a similar-looking app to Solscan -

```
comdex tx gov submit-proposal add-app solscan scn 10000000000000000
30000 --title "Solidity Scan" --
description "App for solidityscan" --deposit 10000000ucmdx --from
cooluser --chain-id test-1 --keyring-backend test
```

- Note that the transaction will be successfully completed and another app will be created with the same name.



Affected Code

Impacts

This vulnerability can be used to create apps and assets with similar-looking names but they will be completely different apps. This can be abused to create social engineering scenarios and exploit end-users and organizations.

Remediation

Perform input validation and do not accept unwanted Unicode characters. This is important for critical inputs like app name assets name etc.

Retest:

Now a regex check has been added to allow only [a-z] characters.

https://github.com/comdex-official/comdex/blob/d1872b477944eca3372076d4f527415183197e07/x/asset/keeper/app.go#L211

Bug ID#5 [Fixed]

Missing Character Length Validation

Vulnerability Type

Missing Input Validation - [CWE 20]

Severity

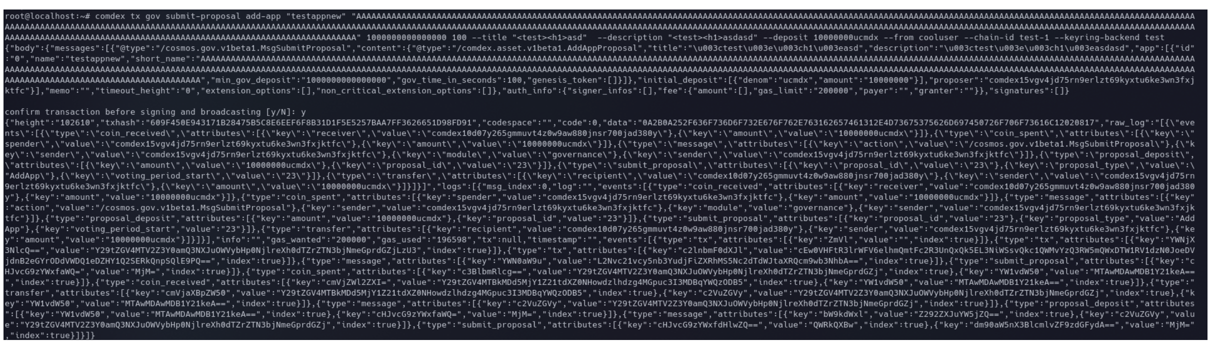
Low

Description

The application does not have input validation on the transaction to create an app. The name and short name parameters are missing a length validation allowing users to have names with very large lengths. This transaction, when executed, will cause an out of gas exception causing the transaction to fail.

PoC

- Enter the following command to submit a proposal for a new app. Check length validation in the name and the short name as shown below.
- Note that the application does not prevent users from using lengthy names, causing out of gas exception and the transaction to fail.



```
comdex tx gov submit-proposal add-app "testappnew"
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAA" 1000000000000000 100 --title "<test><h1>asd"  --description
"<test><h1>asdasd" --deposit 10000000ucmdx --from cooluser
--chain-id test-1 --keyring-backend test
```

Affected User Supplied Inputs:

Command	Inputs
add-app	name, shortName

Impacts

Allowing unlimited characters in the form of any user-supplied input can lead to Denial of Service related attacks.

Remediation

Add a character length limit to all the missing places.

Retest:

Character length has been restricted as well as only [a-z] characters are allowed.
https://github.com/comdex-official/comdex/blob/d1872b477944eca3372076d4f527415183197e07/x/asset/keeper/app.go#L213-L219

Bug ID#6 [Fixed]

Missing Input Validation on Oracle Price

Vulnerability Type
Missing Input Validation - CWE 20

Severity
Low

Description
The band oracle module fetches the prices of assets listed on the cAsset app. A band packet is created containing the list of assets symbols for which price is to be fetched. Then the packet is relayed to the band chain, where the request is acknowledged, and the price is validated. The packets are relayed after every 20 blocks; hence prices are updated after every 20 blocks.
This process lacks an edge case input validation to check if the oracle returns a price for an asset as a 0 value.

Impacts
If the oracle returns an invalid or 0 amount, it will impact the prices for the assets for the apps and in turn affect all the transactions.

Remediation
Add an input validation for invalid values for the prices coming from the oracle. External input should never be trusted as it may have disastrous impact if compromised.

Retest:
The handling in case the oracle return 0 as value has been added
https://github.com/comdex-official/comdex/blob/d1872b477944eca3372076d4f527415183197e07/x/market/keeper/oracle.go#L167-L172

6. Appendix 1

6.1 Disclosure:

The Reports neither endorse nor condemn any specific project or team, nor do they guarantee the security of any specific project. The contents of this report do not, and should not be interpreted as having any bearing on, the economics of tokens, token sales, or any other goods, services, or assets.

Emerging technologies such as Smart Contracts, Bridges and Blockchains carry a high level of technical risk and uncertainty. There is no warranty or representation made by this report to any Third Party in regards to the quality of code, the business model or the proprietors of any such business model, or the legal compliance of any business.
In no way should a third party use these reports to make any decisions about buying or selling a token, product, service, or any other asset. It should be noted that this report is not investment advice, is not intended to be relied on as investment advice, and has no endorsement of this project or team. It does not serve as a guarantee as to the project's absolute security.
CredShields Audit team owes no duty to any third party by virtue of publishing these Reports.