# CredShields
# pStake AVAX Bridge Audit

## June 20th, 2022 • CONFIDENTIAL

**Description**

This document details the process and result of the pStake AVAX Bridge audit performed by CredShields Technologies PTE. LTD. on behalf of pStake finance between May 24th, 2022, and June 14th, 2022.

**Author**

Shashank (Co-founder, CredShields)

shashank@CredShields.com

**Reviewers**

Aditya Dixit (Research Team Lead)

aditya@CredShields.com

**Prepared for**

pStake Finance

# Table of Contents





CRED SHIELDS

# 1. Executive Summary

pStake Finance engaged CredShields to perform the AVAX Bridge audit from May 24th, 2022, to June 14th, 2022. During this timeframe, Six (6) vulnerabilities were identified. **A retest was performed by the CredShields team between 15th June 2022 to 20th June 2022 and all the vulnerabilities were found to be fixed.**

During the audit, zero (0) vulnerability was found that had a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "pStake Finance" and should be prioritized for remediation, and fortunately, none were found.

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 |
|---|---|---|---|---|---|---|
| pStake AVAX Bridge | 0 | 0 | 1 | 2 | 3 | **6** |
| | **0** | **0** | **1** | **2** | **3** | **6** |

*Table: Vulnerabilities Per Asset in Scope*

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

## State of Security

Maintaining a healthy security posture requires constant review and refinement of existing security processes. Running a CredShields continuous audit allows “pStake Finance” 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 pStake Finance introduces new features or refactors the code.

Reviewing the remaining resolved reports for a root cause analysis can further educate “pStake Finance” 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 pStake Finance’s assets.

# 2. Methodology

pStake Finance engaged CredShields to perform the AVAX Bridge 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 AVAX Bridge 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 bridge and nodes to verify the assumptions and validation of the vulnerabilities during the audit phase.

A testing window from May 24th, 2022, to June 14th, 2022, was agreed upon during the preparation phase.

### 2.1.1 Scope

During the preparation phase, the following scope for the engagement was agreed-upon:

| IN SCOPE ASSETS |
| --- |
| https://github.com/persistenceOne/stkAVAX-bridge<br>Commit Hash - b5b0076fdc718ddd7020f33340dc126b7b0aaf68 |

*Table: In Scope Assets*

### 2.1.2 Documentation

The following documentations were available to the audit team.

- https://docs.google.com/document/d/1gIQdHNtutE27Q0Je4ml8CjldtWS4xADb2MqEG5ywDf0/edit
- https://github.com/persistenceOne/stkAVAX-bridge/blob/main/README.md

### 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

pStake Finance 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 |
|---|---|---|
| Outdated Components | Informational | Components with Known Vulnerabilities (CWE-1104) |
| Go-Ethereum (CVE-2022-23328) | Medium | Components with Known Vulnerabilities (CWE-1104) |
| Arbitrary File Write | Low | Missing Input Validation (CWE-20) |
| Missing Input Validation | Low | Missing Input Validation (CWE-20) |
| Redundant Import Statement | Informational | Missing best practices (CWE-1041) |

| | | |
|---|---|---|
| Dead Code | Informational | Dead Code (CWE-561) |

*Table: Findings in AVAX Bridge*

### 3.1.2 Findings Summary

The team performed both source code reviews and tested the application using fuzzing methods after making a local deployment. Although the application was not vulnerable to any critical issues, multiple instances of missing input validation were found.
Since most of the features do not interact with an external user the impact of these input validations was found to be low.
The team also found multiple outdated packages and software versions. It is recommended to update all third-party libraries to improve the overall security of the application.

# 4. Remediation Status

pStake Finance is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. The table shows the remediation status of each finding.

| VULNERABILITY TITLE | SEVERITY | REMEDIATION STATUS |
|---|---|---|
| Outdated Components | Informational | ***Pending Fix*** |
| Go-Ethereum (CVE-2022-23328) | Medium | ***Pending Fix*** |
| Arbitrary File Write | Low | ***Pending Fix*** |
| Missing Input Validation | Low | ***Pending Fix*** |
| Redundant Import Statement | Informational | ***Pending Fix*** |
| Dead Code | Informational | ***Pending Fix*** |

*Table: Summary of findings and status of remediation*

# 5. Bug Reports

**Bug ID#1**

## Outdated Components

**Vulnerability Type**
Components with Known Vulnerabilities

**Severity**
Informational

**Description**
The application uses components with known vulnerabilities that in some cases may allow an attacker to quickly exploit them and gain access to the application.
Multiple go libraries and components were found to be outdated and were affected by publicly disclosed CVEs and exploits.

**Affected Code**
This is added as an attachment due to the bigger size.

**Impacts**
There's a chance of exploitation and application compromise since these outdated libraries may be vulnerable to attacks like injections, and DoS, among other issues.

**Remediation**
Update all the libraries and components to their latest versions.

**Retest:**
-

## Bug ID#2

## Go-Ethereum (CVE-2022-23328)

**Vulnerability Type**
Components with Known Vulnerabilities

**Severity**
Medium

**Description**
The code was found to be using an outdated version of the library Go-Ethereum.
A design flaw in all versions of Go-Ethereum allows an attacker node to send 5120 (CVE-2022-23328) pending transactions of a high gas price from one account that all fully spend the full balance of the account to a victim Geth node, which can purge all of pending transactions in a victim node's memory pool and then occupy the memory pool to prevent new transactions from entering the pool, resulting in a denial of service (DoS).

**Affected Code**

- https://github.com/persistenceOne/stkAVAX-bridge/blob/b5b0076fdc718ddd7020f33340dc126b7b0aaf68/go.mod#L17

**Impacts**
This vulnerability may lead to a Denial of Service attack by preventing new transactions from entering the pool.

**Remediation**
Update all the libraries and components to their latest versions.

**Retest:**
-

## Bug ID#3

## Arbitrary File Write

**Vulnerability Type**
Missing Input Validation

**Severity**
Low

**Description**
The “Write()” function on Line 36 inside “configs/bridge_config.go”. This function is taking an argument called “outputFile” which is then used on Line 37 without any input validation and then it is creating a file using “os.Create” command which is dangerous without input validation.

**Affected Code**

- configs/bridge_config.go - https://github.com/persistenceOne/stkAVAX-bridge/blob/b5b0076fdc718ddd7020f33340dc126b7b0aaf68/configs/bridge_config.go#L36-L53

```
func (cfg *BridgeConfig) Write(outputFile string) error {
   f, err := os.Create(outputFile)
```

- cmd/gen_member_cfg.go https://github.com/persistenceOne/stkAVAX-bridge/blob/b5b0076fdc718ddd7020f33340dc126b7b0aaf68/cmd/gen_member_cfg.go#L52-L55

```
   err = bridgeCfg.Write(ctx.String(outputCfgFile.Name))
   if err != nil {
       return err
   }
```

- cmd/gen_signers.go
  https://github.com/persistenceOne/stkAVAX-bridge/blob/b5b0076fdc718ddd7020f33340dc126b7b0aaf68/cmd/gen_signers.go#L133-L136

```
err = bridgeCfg.Write(bridgeCfgFile)
if err != nil {
    return err
}
```

**Impacts**

Missing input validation on arguments that go inside “os” commands may lead to disastrous effects as these modify and create system files. If an arbitrary file path is passed in the argument, there may be a chance to create 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 argument that takes the file name as a user-supplied input.

**Retest:**

-

## Bug ID#4

# Missing Input Validation

**Vulnerability Type**
Missing Input Validation

**Severity**
Low

**Description**
Input validation is performed to ensure only properly formed data is entering the workflow in an information system, preventing malformed data from persisting in the database and triggering malfunction of various downstream components. Input validation should happen as early as possible in the data flow, preferably as soon as the data is received from the external party.
The bifrost bridge was missing input validation on several function arguments which were directly fed into sensitive system functions.

**Affected Code**

- https://github.com/persistenceOne/stkAVAX-bridge/blob/b5b0076fdc718ddd7020f33340dc126b7b0aaf68/accounts/chain_data.go#L22-L26
- https://github.com/persistenceOne/stkAVAX-bridge/blob/b5b0076fdc718ddd7020f33340dc126b7b0aaf68/cmd/five_nodes.go#L33-L76
- https://github.com/persistenceOne/stkAVAX-bridge/blob/b5b0076fdc718ddd7020f33340dc126b7b0aaf68/cmd/gen_member_cfg.go#L30-L55
- https://github.com/persistenceOne/stkAVAX-bridge/blob/b5b0076fdc718ddd7020f33340dc126b7b0aaf68/cmd/gen_signers.go#L38-L61
- https://github.com/persistenceOne/stkAVAX-bridge/blob/b5b0076fdc718ddd7020f33340dc126b7b0aaf68/cmd/node.go#L40-L114
- https://github.com/persistenceOne/stkAVAX-bridge/blob/b5b0076fdc718ddd7020f33340dc126b7b0aaf68/cmd/sentry_integration.go#L40-L43
- https://github.com/persistenceOne/stkAVAX-bridge/blob/b5b0076fdc718ddd7020f33340dc126b7b0aaf68/cmd/utils.go#L146-L170

- https://github.com/persistenceOne/stkAVAX-bridge/blob/b5b0076fdc718ddd7020f33340dc126b7b0aaf68/cmd/utils.go#L193-L202
- https://github.com/persistenceOne/stkAVAX-bridge/blob/b5b0076fdc718ddd7020f33340dc126b7b0aaf68/configs/bridge_config.go#L161-L180
- https://github.com/persistenceOne/stkAVAX-bridge/blob/b5b0076fdc718ddd7020f33340dc126b7b0aaf68/configs/member_config.go#L22-L26
- https://github.com/persistenceOne/stkAVAX-bridge/blob/b5b0076fdc718ddd7020f33340dc126b7b0aaf68/configs/network_config.go#L52-L98
- https://github.com/persistenceOne/stkAVAX-bridge/blob/b5b0076fdc718ddd7020f33340dc126b7b0aaf68/configs/network_config.go#L135-L149 (maybe add a max fee limit as well)

**Impacts**

Missing input validation on arguments that go inside "os" commands may lead to disastrous effects as these modify and create system files. If an arbitrary file path is passed in the argument, there may be a chance to create files in different locations by traversing the directories.

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

**Remediation**

The bridge does very few input validation checks. Although there are no external interactions it is still recommended to have input validations on all inputs of the functions. Some basic input validation ideas would be as below

***Ports:*** Apart from port 0 validation there should be the validation for ports beyond 65535

***Address:*** All address-related input should do a zero address validation as well as basic address validation.

***File names:*** All fine names should not allow special characters like brackets, dots, and slashes.

***Host:*** All hosts can be checked for respective patterns.

***Amount:*** If there are any inputs related to amount. There should be a check for greater than 0 if applicable and max amounts like in setting fees.

***Strings:*** For all strings, it is suggested that to check for special and dangerous characters like brackets, dots, and slashes

**Retest:**

-

## Bug ID#5

## Redundant Import Statement

**Vulnerability Type**
Missing Best Practices

**Severity**
Informational

**Description**
Go allows importing the same package multiple times given that a different alias is used for each import statement.
However, this is very rarely done on purpose and shows that the code was refactored or modified leading to accidentally adding a duplicate import statement for the same library or file.

**Affected Code**

- https://github.com/persistenceOne/stkAVAX-bridge/blob/b5b0076fdc718ddd7020f33340dc126b7b0aaf68/cmd/gen_signers.go#L10-L11
- https://github.com/persistenceOne/stkAVAX-bridge/blob/b5b0076fdc718ddd7020f33340dc126b7b0aaf68/signer/service.go#L28

```
"github.com/keep-network/keep-core/pkg/net/libp2p"
keeplibp2p "github.com/keep-network/keep-core/pkg/net/libp2p"
```

**Impacts**
Importing the same package twice creates confusion and introduces redundant code which can be achieved with a single import statement as well.

**Remediation**
Go through the code logic to make sure two imports for the same package are not required and either keep the one with the alias or the other import statement.

**Retest:**

## Bug ID#6

# Dead Code

**Vulnerability Type**
Dead Code

**Severity**
Informational

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

**Affected Code**

- https://github.com/persistenceOne/stkAVAX-bridge/blob/b5b0076fdc718ddd7020f33340dc126b7b0aaf68/cmd/errors.go#L8 (**errAddressCountMismatch**)
- https://github.com/persistenceOne/stkAVAX-bridge/blob/b5b0076fdc718ddd7020f33340dc126b7b0aaf68/signer/service.go#L477 (**err**)

```
var (
    ...
    errAddressCountMismatch = errors.New("address count mismatch")
)
```

```
    signedTx, err := tx.WithSignature(signer, avaSig)
```

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

**Remediation**

Failure to use or validate error statements shows missing or dead code. Remove the statement if it is not being used or add input validations to handle errors for those variables.

**Retest:**

-

# 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 and Bridges 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.