



# IOV SAS

## IOV.ONE Report

September 17, 2019

Version: 1.0

Classification: For Public Release

Presented by:

Kudelski Security

Research Team

## Table of Contents

<i>Revision History</i> .....	3
<b>EXECUTIVE SUMMARY</b> .....	<b>4</b>
ENGAGEMENT LIMITATIONS.....	4
ENGAGEMENT ANALYSIS.....	5
GENERAL OBSERVATIONS.....	7
<i>Weave</i> .....	7
<i>IOV Core</i> .....	7
<i>Demonstration web application and Chrome plugin identified problems</i> .....	8
<b>SECURITY POSTURE</b> .....	<b>8</b>
<b>SUMMARY LIST</b> .....	<b>8</b>
<b>TECHNICAL DETAILS</b> .....	<b>9</b>
RANDOM NUMBER GENERATION IS RELIANT ON 3RD PARTY LIBRARY (LIBSODIUM.JS) .....	9
REGULAR EXPRESSION DENIAL OF SERVICE IN ROUTER COMPONENT .....	11
HARDCODED BUG REPORTING SENDS POTENTIALLY SENSITIVE DATA TO A THIRD PARTY .....	12
ETHEREUM CODE GENERALLY COMPLEX AND HARD TO FOLLOW .....	13
CRITICAL FUNCTION PRIVKeyEd25519FromSeed IS MISSING TESTCASE .....	14
<b>APPENDIX A: ABOUT KUDELSKI SECURITY</b> .....	<b>15</b>
<b>APPENDIX B: SEVERITY RATING DEFINITIONS</b> .....	<b>16</b>

## Revision History

Date	Version	Revision
2019-08-29	0.1	First draft
2019-09-01	0.2	First review pass
2019-09-03	0.3	Comments from Brian and Nathan handled
2019-09-05	0.4	Draft ready, delivered to IOV
2019-09-17	1.0	Final delivered to IOV

## Executive Summary

Kudelski Security (“Kudelski”), the cybersecurity division of the Kudelski Group, was engaged by IOV SAS. (“IOV”) defined as the client to conduct an external security assessment in the form of a Code Review of Weave, IOV-Core, and its associated web applications.

The assessment was conducted remotely by the Kudelski Security Team from our secure lab environment. The tests took place from December 19, 2018 to August 29, 2019 and focused on the following objectives:

- To provide a professional opinion on the maturity, adequacy, and efficacy of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the results of our findings.

This report summarizes the tests performed and findings in terms of strengths and weaknesses. It also contains detailed descriptions of the discovered vulnerabilities, steps the Kudelski Security Teams took to exploit each vulnerability, and recommendations for remediation.

## Engagement Limitations

The engagement was a time boxed engagement based on the directions given by the IOV team. Generally, third-party libraries were not included in the source code analysis except for some libraries that were deemed critical to the security of the product. Since the core consensus model is based on Tendermint, any real consensus problems were considered out of scope of the audit.

e.g (<https://forum.cosmos.network/t/critical-cosmossdk-security-advisory-updated/2211>)

The scope of the engagement included detailed audits of the following components:

**IOV-Core:** The client for the IOV Name Service and for Weave based blockchains. It is designed to run in standard JavaScript environments so that third party app developers can build wallets, block-explorers, e-commerce payment apps, distributed exchanges, social media tipping apps, and other programs that run in standard web browsers.

**IOV Weave:** A framework to quickly build your custom ABCI application to power a blockchain based on the Proof-of-stake Tendermint Consensus Engine. We built IOV Name Service using weave.

**IOV Chrome Extension:** A Google Chrome Extension to connect to different chains and generate human friendly addresses It connects to the IOV Wallet (web-based) which demonstrates the different features of IOV-core.

## Engagement Analysis

The engagement was performed by cloning the relevant repositories into the internal GitLab environment utilized by Kudelski Security.

Considerable time and effort of the engagement was spent focusing on the facts deemed most relevant to the client as outlined in the initial document provided by IOV.

Based on input from the development team, the audit followed the basic threat model as posted in the GitHub as a guideline for where to focus the code audit.

Current Checklist as received from IOV Team:

- General Conditions / Permission system design
- x/sigs
- X / multi-sig
- X / gov
- Input parsing and validations (buffer overflow, etc)
- Pieces related to money (and int overflow checks)
- X / cash
- X / distribution
- X / escrow

As a result of our work, we identified zero (0) **High**, three (3) **Medium**, one (1) **Low**, and one (1) **Informational** findings.

The medium findings include reliance on third party libraries to generate key material as well as potential denial of service attack vectors in the router layer, and potential leaks of sensitive data. These findings have the potential to limit availability and functionality as well as well as leak potentially sensitive information that an attacker could use to perform further attacks on the system.

Multiple low and informational findings were identified during the course of our assessment. Although these findings do not represent a significant threat, they can increase the attack surface of the system. For example, the function PrivKeyEd25519FromSeed that is critical for the generation of wallets is not covered by any unit test and could therefore be prone to future errors being implemented without being detected.

Depending on priorities and budget, these issues should be considered for remediation at a future date dictated by scheduling.

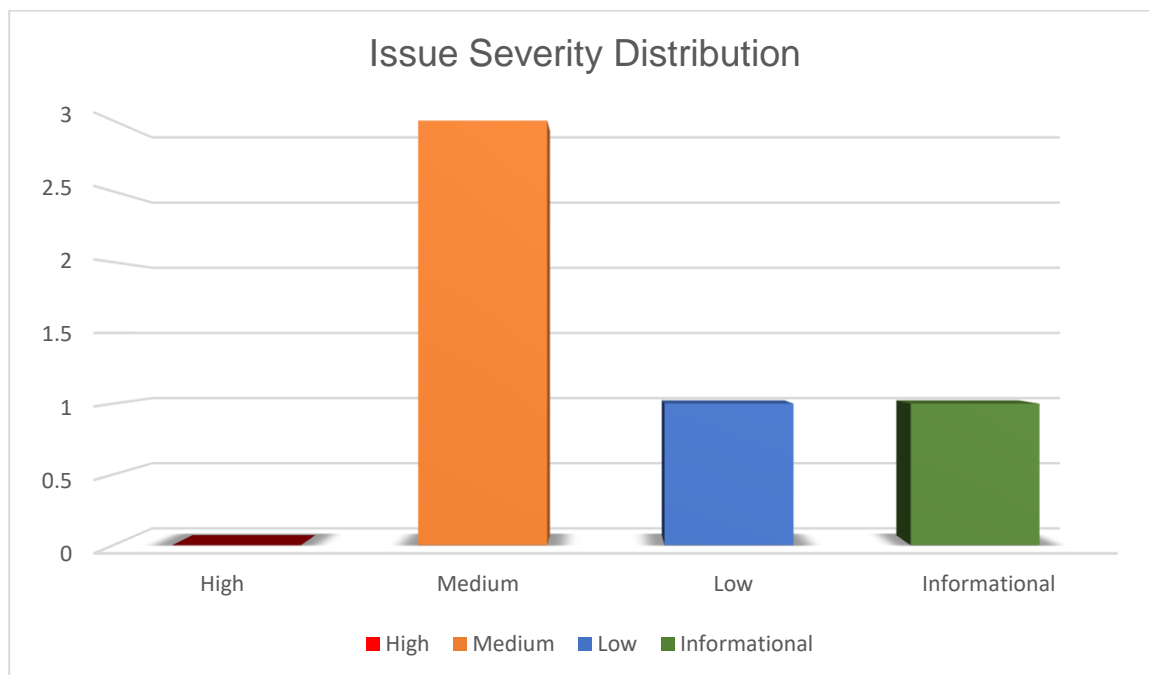


Figure 1: Issue Severity Distribution

## General Observations

### Weave

Our general observations of the code concluded that the overall quality is very high and well structured. The design decision to develop the application layer in Go has had a great impact on security since the inherited security implications of the Go language makes it hard to write unsecure code.

The design decision to use the well proven and heavily reviewed Tendermint Consensus Engine (<https://tendermint.com/>) limits a lot of potential vulnerabilities. Since both the Tendermint core and Weave are written in the Go programming language, the integration code is minimal, and the integration interfaces are clean.

This design decision in conjunction with the strict development environment that requires linters and security checks before building and committing any code has led to an overall code quality that is very good both from both a security as well as from a “code smell” hygiene and readability standpoint.

Furthermore, the use of gogo (<https://github.com/gogo/protobuf>), a protobuf implementation for network serialization, meant that many of the normal security problems (e.g buffer overruns and input validation problems) have been effectively eliminated.

Our analysis was therefore able to focus primarily on the assessment on the potential logical problems of the applications.

### IOV Core

Our general findings support that the code quality is very good and the fact that typescript is used consistently in the codebase indicates that a lot of security problems have been eliminated.

Generally, the code is of very high quality based on its well-defined interface, its correct handling of all errors and error conditions, specifically where necessary, and that general failures (e.g. the default case) are addressed.

In short, from a security audit perspective, there is no “low hanging fruit” that has been uncovered. The developers have excelled in making sure that the code is secure and clean.

Following our analysis of both the logic of the multi-sigs and the atomic swaps, we found that in comparison with other atomic swap implementations publicly available for review, their implementation appears spotless and straight forward.

## Demonstration web application and Chrome plugin identified problems

Our general findings of the web application and Chrome plugin are that the code quality is very good and the fact that typescript is used consistently in the codebase indicates that a lot of security problems have been eliminated.

Generally speaking, the code is of very high quality and the only examples uncovered of less “clearly written” code is in generated React components, which are included and not written by IOV.

Since the application uses the IOV-core for any security-related functions, nearly no new findings have been discovered.

## Security Posture

After analyzing the issues identified and their associated risk and threat levels, Kudelski Security staff ranks the security posture of the application as **Above Average**. This subjective ranking is given based on the number and severity level of the vulnerability as well as the security posture of applications from organizations that are of similar size and industry vertical.

## Summary List

The following is a summary of the issues found during the assessment ranked by severity.

ID	Severity	Finding
KS-IOV-F-01	Medium	Random number generation is reliant on 3rd party library
KS-IOV-F-02	Medium	Regular expression denial of service in router component
KS-IOV-F-03	Medium	Hardcoded bug reporting sends potentially sensitive data to a third party
KS-IOV-F-04	Low	Ethereum code overly complex and hard to follow
KS-IOV-F-05	Informational	Critical function PrivKeyEd25519FromSeed is missing testcase



## Technical Details

This section contains the technical details of our findings as well as recommendations for improvement. Each potential vulnerability is described herein along with recommendations on how to remedy the issue, along with the rated severity, as well as pointers to where in the code the issue has been identified.

### Random number generation is reliant on 3rd party library (Libsodium.js)

Finding ID: KS-IOV-F-01

Severity: **Medium**

Status: **Open**

#### Description

The core implementation of the random number generation (RNG) used for key generation for both wallets, mnemonics and deterministic keys, are obscured and are generated in a third-party library.

The library Libsodium.js is a c library that is compiled to web assembly and “transpiled” to java-script and then included. This makes it very hard to follow and/or track what random sources are actually used in seeding the pseudo-random number generator.

When analyzing the libsodium.js random number generation functions, we do have concerns of some of the potential code paths. Our concerns are that while in a modern browser the library would use web crypto API that would return strong entropy to callers, that is not always the case when the code or the application is running in Node.js.

If the code is run on Node.js, the host entropy pool will be used when available. However, there seem to be cases where the code can generate weak keys seeded with timestamp without throwing an exception.

#### Proof of Issue

Filename: crypto/ed25519.go:68

#### Severity and Impact Summary

Probability of Attack: **Medium**

Ease of Exploitation: **Difficult**

## **Recommendation**

We recommend that the random number generation is rewritten inside of IOV-Core to limit the dependence on libsodium.js.

Additionally, we recommend that the crypto library inside the IOV-Core implements its own fully managed random number generation using best practices and relies directly on crypto API of the browser context or the Node.js crypto module.

## **References**

<https://developer.mozilla.org/en-US/docs/Web/API/Window/crypto>

<https://bitcointechweekly.com/briefs/vulnerabilities-in-numerous-javascript-cryptographic-libraries/>

## Regular expression denial of service in router component

Finding ID: KS-IOV-F-02

Severity: **Medium**

Status: **Open**

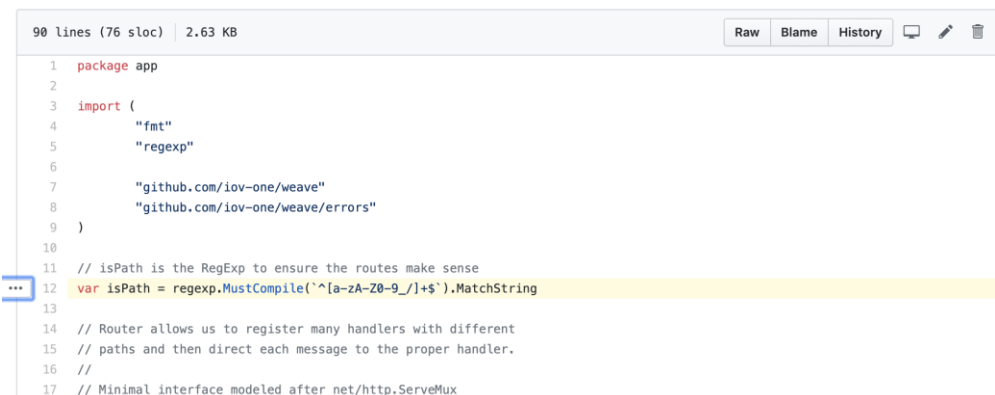
### Description

Performance risks in HTTP router implementation - The HTTP router implementation uses a regular expression to parse the request string at [weave/app/router.go#L12](https://github.com/weaveworks/weave/blob/master/app/router.go#L12).

Regular Expressions potentially expose the application to computational denial of service via specially crafted inputs. This is also known as ReDoS, [https://www.owasp.org/index.php/Regular\\_expression\\_Denial\\_of\\_Service\\_-\\_ReDoS](https://www.owasp.org/index.php/Regular_expression_Denial_of_Service_-_ReDoS).

### **Proof of Issue**

Filename: `weave/app/router.go#L12`



```
90 lines (76 sloc) | 2.63 KB
1 package app
2
3 import (
4     "fmt"
5     "regexp"
6
7     "github.com/iov-one/weave"
8     "github.com/iov-one/weave/errors"
9 )
10
11 // isPath is the RegExp to ensure the routes make sense
12 var isPath = regexp.MustCompile(`^[a-zA-Z0-9_]+$`).MatchString
13
14 // Router allows us to register many handlers with different
15 // paths and then direct each message to the proper handler.
16 //
17 // Minimal interface modeled after net/http.ServeMux
```

### Severity and Impact Summary

Probability of Attack: **Low**

Ease of Exploitation: **Easy**

### Recommendation

To minimize the probability for exploitation of the described vulnerability, we recommend that the router module is refactored so that verification of parsed URLs becomes less resource intensive. A more performant option is to use a prefix tree to route requests as per <https://github.com/julienschmidt/httprouter> (which is mentioned in the code comments).

## Hardcoded bug reporting sends potentially sensitive data to a third party

Finding ID: KS-IOV-F-03

Severity: **Medium**

Status: **Open**

### **Description**

In the Bierzo wallet implementation, the application will always send crash data and bug information to the Sentry.io bug reporting service.

This is a security concern as these reports might contain sensitive data. See data source name: <https://6f1aa71313e14f81b9b663d831705ff6@sentry.io/1374813>

### **Proof of Issue**

Filename: bierzo-wallet/src/index.tsx

### **Severity and Impact Summary**

Probability of Attack: **Low**

Ease of Exploitation: **Difficult**

### **Recommendation**

We recommend that this feature should be disabled by default or at least be configurable.

### **References**

<https://6f1aa71313e14f81b9b663d831705ff6@sentry.io/1374813> in bierzo-wallet/src/index.tsx

## Ethereum code generally complex and hard to follow

Finding ID: KS-IOV-F-04

Severity: **Low**

Status: **Open**

### **Description**

While the majority of the code is well written and easy to follow and test, the Ethereum client that was used is overly complex and riddled with functions that are hundreds of lines long with nested conditions, which makes it more cumbersome to audit and could therefore contain logical flaws as well as state flaws that could be exploited by an attacker. We understand that this is code that is more or less copy-paste in order to integrate with the Ethereum blockchain, however we still recommend that this code be refactored into smaller more well-defined functions that can be tested individually.

One example is `ethereumconnection.ts` where the function `watchBlockHeaders` is more than 120 lines long and contains nested awaits and conditional branches.

### **Proof of Issue**

Filename: `packages/iov-ethereum/src/ethereumconnection.ts:577`

Filename: `iov-core/packages/iov-ethereum/src/ethereumcodec.ts:131:10`

### **Severity and Impact Summary**

Probability of Attack: **Low**

Ease of Exploitation: **Difficult**

### **Recommendation**

We recommend that the code for the Ethereum integration in general is refactored into manageable functions that are easy to follow and that a code complexity review of the component is performed.

## Critical function PrivKeyEd25519FromSeed is missing testcase

Finding ID: KS-IOS-F-05

Severity: **Informational**

Status: **Open**

### **Description**

The function PrivKeyEd25519FromSeed that is critical for the generation of wallets is not covered by any unit test and could therefore be prone to future errors being implemented without being detected.

### **Proof of Issue**

crypto/ed25519.go:68

### **Severity and Impact Summary**

Probability of Attack: **Low**

Ease of Exploitation: **Difficult**

### **Recommendation**

Implement valid unit tests for the function.

## **Appendix A: About Kudelski Security**

Kudelski Security is an innovative, independent Swiss provider of tailored cyber and media security solutions to enterprises and public sector institutions. Our team of security experts delivers end-to-end consulting, technology, managed services, and threat intelligence to help organizations build and run successful security programs. Our global reach and cyber solutions focus is reinforced by key international partnerships.

Kudelski Security is a division of Kudelski Group. For more information, please visit <https://www.kudelskisecurity.com>.

### **Kudelski Security**

route de Genève, 22-24  
1033 Cheseaux-sur-Lausanne  
Switzerland

### **Kudelski Security**

5090 North 40th Street  
Suite 450  
Phoenix, Arizona 85018

This report and its content is copyright (c) Nagravision SA, all rights reserved.

## Appendix B: Severity Rating Definitions

Kudelski Security uses a custom approach when determining criticality of identified issues. This is meant to be simple and fast, as with anything risk related, these findings are situational. We consider multiple factors when assigning a severity level to an identified vulnerability. A few of these include:

- Impact of exploitation
- Ease of exploitation
- Likelihood of attack
- Exposure of attack surface
- Number of instances of identified vulnerability
- Availability of tools and exploits

Severity	Definition
High	The identified issue may be directly exploitable causing an immediate negative impact on the users, data, and availability of the system for multiple users.
Medium	The identified issue is not directly exploitable but combined with other vulnerabilities may allow for exploitation of the system or exploitation may affect singular users. These findings may also increase in severity in the future as techniques change.
Low	The identified issue is not directly exploitable but raises the attack surface of the system. This may be through leaking information that an attacker can use to increase the accuracy of their attacks.
Informational	Informational findings are best practice steps that can be used to harden the application and improve processes.